
FloPy Documentation

Release 3.4.3

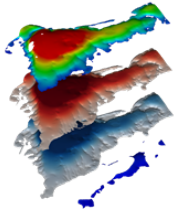
FloPy Team

Oct 04, 2023

CONTENTS

1	Installation	3
2	Resources	5
3	Development Team	7
4	How to Cite	9
5	Tutorials	11
5.1	FloPy	11
5.2	MODFLOW 6	43
5.3	MODFLOW-2005	249
5.4	MODFLOW-LGR	278
5.5	MODFLOW-NWT	294
5.6	MT3DMS	301
5.7	PEST	326
6	Examples Gallery	339
6.1	Preprocessing and Discretization	339
6.2	Postprocessing and Visualization	466
6.3	Exporting data	681
6.4	Other FloPy features	696
6.5	MODFLOW 6 examples	791
6.6	MODFLOW USG examples	881
6.7	MODFLOW-2005/MODFLOW-NWT examples	933
6.8	MODPATH examples	1021
6.9	MT3D and SEAWAT examples	1112
6.10	Examples from Bakker and others (2016)	1202
6.11	Examples from Hughes and others (2023)	1228
6.12	Miscellaneous examples	1252
7	Frequently asked questions	1277
7.1	Executable does not exist	1277
8	API Reference	1279
8.1	MODFLOW 6	1279
8.2	Previous Versions of MODFLOW	1605
8.3	Flopy Utilities	1834
9	Indices and tables	2041

Python Module Index	2043
Index	2047



FloPy₃

a Python package to create, run, and post-process MODFLOW-based models

Documentation for version 3.4.3

Documentation is generated with Sphinx from the [FloPy repository](#).

Contents:

The FloPy package consists of a set of Python scripts to run MODFLOW, MT3D, SEAWAT and other MODFLOW-related groundwater programs. FloPy enables you to run all these programs with Python scripts. The FloPy project started in 2009 and has grown to a fairly complete set of scripts with a growing user base. FloPy3 was released in December 2014 with a few great enhancements that make FloPy3 backwards incompatible. The first significant change is that FloPy3 uses zero-based indexing everywhere, which means that all layers, rows, columns, and stress periods start numbering at zero. This change was made for consistency as all array-indexing was already zero-based (as are all arrays in Python). This may take a little getting-used-to, but hopefully will avoid confusion in the future. A second significant enhancement concerns the ability to specify time-varying boundary conditions that are specified with a sequence of layer-row-column-values, like the WEL and GHB packages. A variety of flexible and readable ways have been implemented to specify these boundary conditions.

Recently, FloPy has been further enhanced to include full support for MODFLOW 6. The majority of recent development has focused on FloPy functionality for MODFLOW 6, helper functions to use GIS shapefiles and raster files to create MODFLOW datasets, and common plotting and export functionality.

FloPy is an open-source project and any assistance is welcomed. Please email the development team if you want to contribute.

Return to the Github [FloPy](#) website.

INSTALLATION

FloPy can be installed using conda (from the conda-forge channel) or pip.

To install with conda:

```
conda install -c conda-forge flopy
```

To install with pip:

```
pip install flopy
```

To install the bleeding edge version of FloPy from the git repository type:

```
pip install git+https://github.com/modflowpy/flopy.git
```

After FloPy is installed, MODFLOW and related programs can be installed using the command:

```
get-modflow :flopy
```

See documentation [get_modflow.md](#) for more information.

RESOURCES

Version history

Supported packages

Model checking capabilities

DEVELOPMENT TEAM

FloPy is developed by a team of MODFLOW users that have switched over to using Python for model development and post-processing. Members of the team currently include:

- Mark Bakker 
- Vincent Post 
- Joseph D. Hughes 
- Christian D. Langevin 
- Jeremy T. White 
- Andrew T. Leaf 
- Scott R. Paulinski 
- Jason C. Bellino 
- Eric D. Morway 
- Michael W. Toews 
- Joshua D. Larsen 
- Michael N. Fienen 
- Jon Jeffrey Starn 
- David A. Brakenhoff 
- Wesley P. Bonelli 
- and others

HOW TO CITE

- Groundwater Paper
- Software Citation

TUTORIALS

The following tutorials demonstrate basic FloPy features and usage with MODFLOW 2005, MODFLOW 6, and related programs.

5.1 FloPy

5.1.1 Formatting ASCII output arrays

Configuring numeric arrays written by FloPy

load and run the Freyberg model

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

import flopy

# Set name of MODFLOW exe
# assumes executable is in users path statement
version = "mf2005"
exe_name = "mf2005"
mfexe = exe_name

# Set the paths
loadpth = os.path.join("../", "..", "examples", "data", "freyberg")
temp_dir = TemporaryDirectory()
modelpth = temp_dir.name

# make sure modelpth directory exists
if not os.path.isdir(modelpth):
    os.makedirs(modelpth)

print(sys.version)
print("numpy version: {}".format(np.__version__))
```

(continues on next page)

(continued from previous page)

```
print("matplotlib version: {}".format(mpl.__version__))
print("flopyp version: {}".format(flopyp.__version__))
```

```
3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopyp version: 3.4.3
```

```
[2]: ml = flopy.modflow.Modflow.load(
    "freyberg.nam", model_ws=loadpth, exe_name=exe_name, version=version
)
ml.model_ws = modelpth
ml.write_input()
success, buff = ml.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

files = ["freyberg.hds", "freyberg.cbc"]
for f in files:
    if os.path.isfile(os.path.join(modelpth, f)):
        msg = "Output file located: {}".format(f)
        print(msg)
    else:
        errmsg = "Error. Output file cannot be found: {}".format(f)
        print(errmsg)
```

```

MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017

Using NAME file: freyberg.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:16

Solving: Stress period:      1   Time step:      1   Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:16
Elapsed run time:  0.016 Seconds

Normal termination of simulation
Output file located: freyberg.hds
Output file located: freyberg.cbc
```

Each Util2d instance now has a `.format` attribute, which is an `ArrayFormat` instance:

```
[3]: print(ml.lpf.hk[0].format)

ArrayFormat: npl:20,format:E,width:15,decimal6,isfree:True,isbinary:False
```

The `ArrayFormat` class exposes each of the attributes seen in the `ArrayFormat.__str__()` call. `ArrayFormat` also exposes `.fortran`, `.py` and `.numpy` attributes, which are the respective format descriptors:

```
[4]: print(ml.dis.botm[0].format.fortran)
      print(ml.dis.botm[0].format.py)
      print(ml.dis.botm[0].format.numpy)
```

```
(FREE)
(20, '{0:15.6E}')
```

```
%15E.6
```

(re)-setting .format

We can reset the format using a standard fortran type format descriptor

```
[5]: ml.dis.botm[0].format.free = False
      ml.dis.botm[0].format.fortran = "(20f10.4)"
      print(ml.dis.botm[0].format.fortran)
      print(ml.dis.botm[0].format.py)
      print(ml.dis.botm[0].format.numpy)
      print(ml.dis.botm[0].format)
```

```
(20F10.4)
(20, '{0:10.4F}')
```

```
%10F.4
```

```
ArrayFormat: npl:20,format:F,width:10,decimal4,isfree:False,isbinary:False
```

```
[6]: ml.write_input()
      success, buff = ml.run_model(silent=True, report=True)
      if success:
          for line in buff:
              print(line)
      else:
          raise ValueError("Failed to run.")
```

```

                                MODFLOW-2005
      U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
                                Version 1.12.00 2/3/2017
```

```
Using NAME file: freyberg.nam
```

```
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:16
```

```
Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
```

```
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:16
```

```
Elapsed run time: 0.009 Seconds
```

```
Normal termination of simulation
```

Let's load the model we just wrote and check that the desired botm[0].format was used:

```
[7]: ml1 = flopy.modflow.Modflow.load("freyberg.nam", model_ws=modelpth)
      print(ml1.dis.botm[0].format)
```

```
ArrayFormat: npl:20,format:E,width:15,decimal6,isfree:True,isbinary:False
```

We can also reset individual format components (we can also generate some warnings):

```
[8]: ml.dis.botm[0].format.width = 9
ml.dis.botm[0].format.decimal = 1
print(ml1.dis.botm[0].format)
```

ArrayFormat warning:setting width less than default of 15
 ArrayFormat warning: setting decimal less than default of 6
 ArrayFormat warning: setting decimal less than current value of 6
 ArrayFormat: npl:20,format:E,width:15,decimal6,isfree:True,isbinary:False

We can also select free format. Note that setting to free format resets the format attributes to the default, max precision:

```
[9]: ml.dis.botm[0].format.free = True
print(ml1.dis.botm[0].format)
```

ArrayFormat: npl:20,format:E,width:15,decimal6,isfree:True,isbinary:False

```
[10]: ml.write_input()
success, buff = ml.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")
```

```

                                MODFLOW-2005
      U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
                                Version 1.12.00 2/3/2017

Using NAME file: freyberg.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:16

Solving:  Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:16
Elapsed run time:  0.009 Seconds

Normal termination of simulation
```

```
[11]: ml1 = flopy.modflow.Modflow.load("freyberg.nam", model_ws=modelpth)
print(ml1.dis.botm[0].format)
```

ArrayFormat: npl:20,format:E,width:15,decimal6,isfree:True,isbinary:False

5.1.2 Exporting to netCDF and shapefile

```
[1]: import datetime
```

```
[2]: import os
import sys
from tempfile import TemporaryDirectory

import flopy
```

(continues on next page)

(continued from previous page)

```
print(sys.version)
print("flopy version: {}".format(flopy.__version__))
```

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
flopy version: 3.4.3

Load our old friend...the Freyberg model

```
[3]: nam_file = "freyberg.nam"
model_ws = os.path.join(
    "..", "..", "examples", "data", "freyberg_multilayer_transient"
)
ml = flopy.modflow.Modflow.load(nam_file, model_ws=model_ws, check=False)
```

We can see the Modelgrid instance has generic entries, as does start_datetime

```
[4]: ml.modelgrid
```

[4]: xll:622241.1904510253; yll:3343617.741737109; rotation:15.0; crs:EPSG:32614; units:
↪meters; lenuni:2

```
[5]: ml.modeltime.start_datetime
```

[5]: '1/1/2015'

Setting the attributes of the ml.modelgrid is easy:

```
[6]: ml.modelgrid.set_coord_info(
    xoff=123456.7, yoff=765432.1, angrot=15.0, crs=3070
)
ml.dis.start_datetime = "7/4/1776"
```

```
[7]: ml.modeltime.start_datetime
```

[7]: '7/4/1776'

Basic netCDF export capabilities

```
[8]: # temporary directory
temp_dir = TemporaryDirectory()
pth = temp_dir.name
```

```
[9]: # export the whole model (inputs and outputs)
fnc = ml.export(os.path.join(pth, ml.name + ".in.nc"))

initialize_geometry::
model crs: EPSG:3070
initialize_geometry::nc_crs = EPSG:4326
transforming coordinates using = proj=pipeline step inv proj=tmerc lat_0=0 lon_0=-90 k=0.
↪9996 x_0=520000 y_0=-4480000 ellps=GRS80 step proj=unitconvert xy_in=rad xy_out=deg
```

```
[10]: # export outputs using spatial reference info
hds = flopy.utils.HeadFile(os.path.join(model_ws, "freyberg.hds"))
flopy.export.utils.output_helper(
    os.path.join(pth, ml.name + ".out.nc"), ml, {"hds": hds}
)

initialize_geometry::
model crs: EPSG:3070
initialize_geometry::nc_crs = EPSG:4326
transforming coordinates using = proj=pipeline step inv proj=tmerc lat_0=0 lon_0=-90 k=0.
↪9996 x_0=520000 y_0=-4480000 ellps=GRS80 step proj=unitconvert xy_in=rad xy_out=deg

[10]: <flopy.export.netcdf.NetCdf at 0x7fd1aebd1670>
```

Export an array to netCDF or shapefile

```
[11]: # export a 2d array
ml.dis.top.export(os.path.join(pth, "top.nc"))
ml.dis.top.export(os.path.join(pth, "top.shp"))

initialize_geometry::
model crs: EPSG:3070
initialize_geometry::nc_crs = EPSG:4326
transforming coordinates using = proj=pipeline step inv proj=tmerc lat_0=0 lon_0=-90 k=0.
↪9996 x_0=520000 y_0=-4480000 ellps=GRS80 step proj=unitconvert xy_in=rad xy_out=deg
```

sparse export of stress period data for a boundary condition package

- excludes cells that aren't in the package (aren't in `package.stress_period_data`)
- by default, stress periods with duplicate parameter values (e.g., stage, conductance, etc.) are omitted (`squeeze=True`); only stress periods with different values are exported
- argue `squeeze=False` to export all stress periods

```
[12]: ml.drn.stress_period_data.export(os.path.join(pth, "drn.shp"), sparse=True)

wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks
```

Export a 3d array

```
[13]: # export a 3d array
ml.upw.hk.export(os.path.join(pth, "hk.nc"))
ml.upw.hk.export(os.path.join(pth, "hk.shp"))

initialize_geometry::
model crs: EPSG:3070
initialize_geometry::nc_crs = EPSG:4326
transforming coordinates using = proj=pipeline step inv proj=tmerc lat_0=0 lon_0=-90 k=0.
↪9996 x_0=520000 y_0=-4480000 ellps=GRS80 step proj=unitconvert xy_in=rad xy_out=deg
```

Export a number of things to the same netCDF file

```
[14]: # export lots of things to the same nc file
fnc = ml.dis.botm.export(os.path.join(pth, "test.nc"))
ml.upw.hk.export(fnc)
ml.dis.top.export(fnc)

# export transient 2d
ml.rch.rech.export(fnc)

initialize_geometry::
model crs: EPSG:3070
initialize_geometry::nc_crs = EPSG:4326
transforming coordinates using = proj=pipeline step inv proj=tmerc lat_0=0 lon_0=-90 k=0.
↪9996 x_0=520000 y_0=-4480000 ellps=GRS80 step proj=unitconvert xy_in=rad xy_out=deg

[14]: <flopy.export.netcdf.NetCdf at 0x7fd1acc449a0>
```

Export a package to netCDF

```
[15]: # export mflist
fnc = ml.wel.export(os.path.join(pth, "packages.nc"))
ml.upw.export(fnc)

initialize_geometry::
model crs: EPSG:3070
initialize_geometry::nc_crs = EPSG:4326
transforming coordinates using = proj=pipeline step inv proj=tmerc lat_0=0 lon_0=-90 k=0.
↪9996 x_0=520000 y_0=-4480000 ellps=GRS80 step proj=unitconvert xy_in=rad xy_out=deg

[15]: <flopy.export.netcdf.NetCdf at 0x7fd1ad339910>
```

Export an entire model to netCDF

```
[16]: fnc = ml.export(os.path.join(pth, "model.nc"))

initialize_geometry::
model crs: EPSG:3070
initialize_geometry::nc_crs = EPSG:4326
transforming coordinates using = proj=pipeline step inv proj=tmerc lat_0=0 lon_0=-90 k=0.
↪9996 x_0=520000 y_0=-4480000 ellps=GRS80 step proj=unitconvert xy_in=rad xy_out=deg
```

Export model outputs to netCDF

FloPy has utilities to export model outputs to a netcdf file. Valid output types for export are MODFLOW binary head files, formatted head files, cell budget files, seawat concentration files, and zonebudget output.

Let's use output from the Freyberg model as an example of these functions

```
[17]: # load binary head and cell budget files
fhead = os.path.join(model_ws, "freyberg.hds")
```

(continues on next page)

(continued from previous page)

```

fcbc = os.path.join(model_ws, "freyberg.cbc")

hds = flopy.utils.HeadFile(fhead)
cbc = flopy.utils.CellBudgetFile(fcbc)

export_dict = {"hds": hds, "cbc": cbc}

# export head and cell budget outputs to netcdf
fnc = flopy.export.utils.output_helper(
    os.path.join(pth, "output.nc"), ml, export_dict
)

initialize_geometry::
model crs: EPSG:3070
initialize_geometry::nc_crs = EPSG:4326
transforming coordinates using = proj=pipeline step inv proj=tmerc lat_0=0 lon_0=-90 k=0.
→ 9996 x_0=5200000 y_0=-44800000 ellps=GRS80 step proj=unitconvert xy_in=rad xy_out=deg
error getting data for cell_by_cell_flowstorage at time 1.0:list index out of range
error getting data for cell_by_cell_flowstorage at time 1097.0:list index out of range

```

```

[18]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass

```

5.1.3 Exporting to VTK

The `Vtk()` class in FloPy allows users to export Structured, Vertex, and Unstructured Grid based models to Visualization Toolkit files for display. This notebook demonstrates how to use FloPy to export to vtk (.vtu) files. This example will cover:

- basic exporting of information for a model, individual package, or array to `Vtk()`
- example usage of the `Vtk()` class object to output data
- exporting heads and model output data
- exporting modpath pathlines to `Vtk()`

```

[1]: import os
import sys
from pathlib import Path
from tempfile import TemporaryDirectory

import numpy as np

import flopy
from flopy.export import vtk

sys.path.append(os.path.join("..", "common"))
import notebook_utils

```

(continues on next page)

(continued from previous page)

```
print(sys.version)
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
flopy version: 3.4.3
```

```
[2]: # load model for examples
nam_file = "freyberg.nam"
prj_root = notebook_utils.get_project_root_path()
model_ws = prj_root / "examples" / "data" / "freyberg_multilayer_transient"
ml = flopy.modflow.Modflow.load(nam_file, model_ws=model_ws, check=False)
```

Create a temporary workspace.

```
[3]: tempdir = TemporaryDirectory()
workspace = Path(tempdir.name)
```

Using the `.export()` method

For all exports a **folder path must be provided** and the `fmt` flag should be set to `'vtk'`.

Exporting FloPy arrays to `.vtu` files

All array exports have the following optional keyword arguments: - `smooth`: True creates a smooth surface, default is False - `point_scalars`: True outputs point scalar values as well as cell values, default is False. - `name`: A name can be specified to use for the output filename and array scalar name, by default the FloPy array name is used - `binary`: argument that can be specified to switch between binary and ASCII, default is True - `xml`: True will write an xml base vtk file, default is False - `masked_values`: list or tuple of values to mask (set to nan) when writing a array - `vertical_exageration`: floating point value that can be used to scale the vertical exaggeration of the vtk points. Default is 1.

Tranient type array exports ("`stress_period_data`"; ex. recharge data, well flux, etc ...) have additional optional keyword arguments: - `pvd`: True will write a paraview data file with simulation time for animations. Default is False - `kper`: a list, tuple, or integer value of specific stress periods to output

Export model top

```
[4]: # create output folder
output_dir = workspace / "arrays_test"
output_dir.mkdir(exist_ok=True)

# export model top
model_top_dir = output_dir / "TOP"
ml.dis.top.export(model_top_dir, fmt="vtk")
```

Export model bottoms

```
[5]: # 3D Array export
# export model bottoms
model_bottom_dir = output_dir / "BOTM"
ml.dis.botm.export(model_bottom_dir, fmt="vtk")
```

Export transient array recharge

```
[6]: # transient 2d array
# export recharge
model_recharge_dir = output_dir / "RECH"
ml.rch.rech.export(model_recharge_dir, fmt="vtk", pvd=True)

Switching to xml, ASCII and standard binary are not supported by Paraview's PVD reader
```

Export HK with point scalars

```
[7]: # 3D Array export
# hk export, with points
model_hk_dir = output_dir / "HK"
ml.upw.hk.export(
    model_hk_dir, smooth=True, fmt="vtk", name="HK", point_scalars=True
)
```

Package export to .vtu files

Package export has the following keyword arguments: - **smooth**: True creates a smooth surface, default is False - **point_scalars**: True outputs point scalar values as well as cell values, default is False. - **name**: A name can be specified to use for the output filename and array scalar name, by default the FloPy array name is used - **binary**: argument that can be specified to switch between binary and ASCII, default is True - **xml**: True will write an xml base vtk file, default is False - **masked_values**: list or tuple of values to mask (set to nan) when writing a array - **vertical_exaggeration**: floating point value that can be used to scale the vertical exaggeration of the vtk points. Default is 1. - **pvd**: True will write a paraview data file with simulation time for animations. Default is False - **kper**: a list, tuple, or integer value of specific stress periods to output

Export dis and upw package

```
[8]: # package export
# set up package export folder
output_dir = workspace / "package_output_test"
output_dir.mkdir(exist_ok=True)

# export dis
dis_output_dir = output_dir / "DIS"
ml.dis.export(dis_output_dir, fmt="vtk")
```

(continues on next page)

(continued from previous page)

```
# export upw with point scalars as a binary xml based vtk file
upw_output_dir = output_dir / "UPW"
ml.upw.export(upw_output_dir, fmt="vtk", point_scalars=True, xml=True)
```

Model export to .vtu files

Model export has the following optional keyword arguments:

- `package_names`: a list of package names to export, default is None and will export all packages in the model.
- `smooth`: True creates a smooth surface, default is False
- `point_scalars`: True outputs point scalar values as well as cell values, default is False.
- `name`: A name can be specified to use for the output filename and array scalar name, by default the FloPy array name is used
- `binary`: argument that can be specified to switch between binary and ASCII, default is True
- `xml`: True will write an xml base vtk file, default is False
- `masked_values`: list or tuple of values to mask (set to nan) when writing a array
- `vertical_exageration`: floating point value that can be used to scale the vertical exageration of the vtk points. Default is 1.
- `pvd`: True will write a paraview data file with simulation time for animations. Default is False
- `kper`: a list, tuple, or integer value of specific stess periods to output

Export model as a binary unstructured vtk file

```
[9]: model_output_dir = workspace / "model_output_test"
ml.export(model_output_dir, fmt="vtk")
```

Using the Vtk class

To export custom arrays, or choose a custom combination of model inputs to view, the user first needs to instantiate a new `Vtk()` object. The `Vtk()` object has a single required parameter and a number of optional parameters that the user can take advantage of. These parameters are as follows:

- `model`: any flopy model object can be supplied to create the vtk geometry. Either the model (recommended!) or `modelgrid` parameter must be supplied to the `Vtk()` object.
- `modelgrid`: any flopy modelgrid object (`StructuredGrid`, `VertexGrid`, `UnstructuredGrid`) can be supplied, in leiu of a model object, to create the vtk geometry.
- `vertical_exageration`: floating point value that can be used to scale the vertical exageration of the vtk points. Default is 1.
- `binary`: boolean flag to switch between binary and ASCII vtk files. Default is True.
- `xml`: boolean flag to write xml based vtk files. Default is False
- `pvd`: boolean flag to write a paraview data file for transient series of vtu files. This file relates model time to vtu file for animations. Default is False. If set to True `Vtk()` will automatically write xml based vtu files.

- `shared_points`: boolean flag to write shared vertices within the vtk file. Default is False.
- `smooth`: boolean flag to interpolate vertex elevations using IDW based on shared cell elevations. Default is False.
- `point_scalars`: boolean flag to write interpolated data at each point.

```
[10]: # create a binary XML VTK object and enable PVD file writing
      vtkobj = vtk.Vtk(ml, xml=True, pvd=True, vertical_exaggeration=10)
      vtkobj
```

```
[10]: <flopy.export.vtk.Vtk at 0x7f9b2988ca90>
```

Adding array data to the Vtk object

The `Vtk()` object has an `add_array()` method that lets the user add array data to the Field data section of the VTK file.

`add_array()` has a few parameters for the user: - `array` : numpy array that has a size equal to the number of cells in the model (`modelgrid.nnodes`). - `name` : array name (string) - `masked_values` : list of array values to mask/set to NaN

```
[11]: # Create a vtk object
      vtkobj = vtk.Vtk(ml, vertical_exaggeration=10)

      ## create some random array data
      r_array = np.random.random(ml.modelgrid.nnodes) * 100

      ## add random data to the VTK object
      vtkobj.add_array(r_array, "random_data")

      ## add the model botom data to the VTK object
      vtkobj.add_array(ml.dis.botm.array, "botm")

      ## write the vtk object to file
      vtkobj.write(output_dir / "Array_example" / "model.vtu")
```

Adding transient array data to the Vtk object

The `Vtk` class has an `add_transient_array()` method that allows the user to create a series of time varying VTK files that can be used for animation in VTK viewers.

The `add_transient_array()` method accepts a dictionary of array2d, array3d, or numpy array objects. Parameters include: - `d`: dictionary of array2d, array3d, or numpy array objects - `name`: parameter name, required when user provides a dictionary of numpy arrays - `masked_values`: optional list of values to set equal to NaN.

```
[12]: # create a vtk object
      vtkobj = vtk.Vtk(ml, xml=True, pvd=True, vertical_exaggeration=10)

      ## add recharge to the VTK object
      recharge = ml.rch.rech.transient_2ds
      vtkobj.add_transient_array(
          recharge,
```

(continues on next page)

(continued from previous page)

```

    "recharge",
    masked_values=[
        0,
    ],
)

## write vtk files
vtkobj.write(output_dir / "tr_array_example" / "recharge.vtu")

```

Adding transient list data to the Vtk object

The Vtk class has an `add_transient_list()` method that allows the user to create a series of time varying VTK files that can be used for animation in VTK viewers.

The `add_transient_list()` method accepts a FloPy mlist (transient list) type object. Parameters include: - `mlist`: flopy transient list object - `masked_values`: list of values to set equal to NaN

```

[13]: # create the vtk object
      vtkobj = vtk.Vtk(ml, xml=True, pvd=True, vertical_exaggeration=10)

      ## add well fluxes to the VTK object
      spd = ml.wel.stress_period_data
      vtkobj.add_transient_list(
          spd,
          masked_values=[
              0,
          ],
      )

      ## write vtk files
      vtkobj.write(output_dir / "tr_list_example" / "wel_flux.vtu")

```

Adding packages to the Vtk object

The Vtk class has a method for adding package data to a VTK file as Field Data. The `add_package()` method allows the user to add packages for subsequent export. `add_package()` takes the following parameters:

- `pkg`: flopy package object
- `masked_values`: optional list of values to set to NaN.

In the following example, a HFB package is added to the existing freyberg model and then exported with the WEL package.

```

[14]: # create a HFB package for the example
      hfb_data = []
      for k in range(3):
          for i in range(20):
              rec = [k, i, 6, i, 7, 1e-06]
              hfb_data.append(rec)

      hfb = flopy.modflow.ModflowHfb(ml, hfb_data=hfb_data)

```

```
[15]: # export HFB and WEL packages using Vtk()
vtkobj = vtk.Vtk(ml, vertical_exageration=10)

vtkobj.add_package(hfb)
vtkobj.add_package(ml.wel)

vtkobj.write(output_dir / "package_example" / "package_export.vtu")
```

Exporting heads to binary .vtu files

Once a Vtk object is instantiated (see above), the `add_heads()` method can be used to add head data. This method has a few parameters: - `hds`: a flopy FormattedHeadFile or HeadFile object. This method also accepts DrawdownFile, and ConcentrationFile objects. - `kstpkper`: optional list of zero based (timestep, stress period) tuples to output. Default is None and will output all data to a series of vtu files - `masked_values`: optional list of values to set to NaN, default is None.

```
[16]: # import the HeadFile reader and read in the head file
from flopy.utils import HeadFile

head_file = model_ws / "freyberg.hds"
hds = HeadFile(head_file)

# create the vtk object and export heads
vtkobj = vtk.Vtk(ml, xml=True, pvd=True, vertical_exageration=10)
vtkobj.add_heads(hds)
vtkobj.write(workspace / "heads_output_test" / "freyberg_head.vtu")
```

Export heads as point scalar arrays

```
[17]: # export heads as point scalars
vtkobj = vtk.Vtk(
    ml, xml=True, pvd=True, point_scalars=True, vertical_exageration=10
)

# export heads for time step 1, stress periods 1, 50, 100, 1000
vtkobj.add_heads(hds, kstpkper=[(0, 0), (0, 49), (0, 99), (0, 999)])
vtkobj.write(workspace / "heads_output_test_parameters" / "freyberg_head.vtu")
```

Export cell budget information

Once a Vtk object is instantiated (see above), the `add_cell_budget()` method can be used to export cell budget data. This method has a few parameters: - `cbc`: flopy CellBudgetFile object - `text`: Optional text identifier for a record type. Examples include 'RIVER LEAKAGE', 'STORAGE', etc... Default is None and will export all cell budget information to vtk files - `kstpkper`: optional list of zero based (timestep, stress period) tuples to output. Default is None and will output all data to a series of vtu files - `masked_values`: optional list of values to set to NaN, default is None.

```
[18]: # import the CellBudgetFile reader and read the CBC file
from flopy.utils import CellBudgetFile

cbc_file = model_ws / "freyberg.cbc"
cbc = CellBudgetFile(cbc_file)

# export the cbc file to a series of Vtu files with a PVD file for animation
vtkobj = vtk.Vtk(ml, xml=True, pvd=True, vertical_exaggeration=10)
vtkobj.add_cell_budget(cbc, kstpker=[(0, 0), (0, 9), (0, 10), (0, 11)])
vtkobj.write(workspace / "cbc_output_test_parameters" / "freyberg_cbc.vtu")
```

Export vectors from the cell budget file

The Vtk class has an `add_vector()` method that allows the user to write vector information to VTK files. This method can be used to export information such as cell centered specific discharge.

The `add_vector()` method accepts a numpy array of vector data. The array size must be 3 * the number of model cells (3 * `modelgrid.nnodes`). Parameters include: - `vector`: numpy array of size 3 * `nnodes` - `name`: name of the vector - `masked_values`: list of values to set equal to NaN

```
[19]: # get frf, fff, flf from the Cell Budget file (or SPDIS when using MF6)
from flopy.utils import postprocessing

frf = cbc.get_data(text="FLOW RIGHT FACE", kstpker=(0, 9), full3D=True)[0]
fff = cbc.get_data(text="FLOW FRONT FACE", kstpker=(0, 9), full3D=True)[0]
flf = cbc.get_data(text="FLOW LOWER FACE", kstpker=(0, 9), full3D=True)[0]

spdis = postprocessing.get_specific_discharge((frf, fff, flf), ml)

# create the Vtk() object
vtkobj = vtk.Vtk(ml, vertical_exaggeration=10)

# add the vector
vtkobj.add_vector(spdis, name="spdis")

# write to file
vtkobj.write(output_dir / "vector_example" / "spdis_vector.vtu")
```

Exporting MODPATH timeseries or pathline data

The Vtk class supports writing MODPATH pathline/timeseries data to a VTK file. To start the example, let's first load and run a MODPATH simulation (see `flopy3_modpath7_unstructured_example` for details) and then add the output to a Vtk object.

```
[20]: # load and run the vertex grid model and modpath7
notebook_utils.run(workspace)

# check if model ran properly
modelpth = workspace / "mp7_ex2" / "mf6"
files = ["mp7p2.hds", "mp7p2.cbb"]
for f in files:
```

(continues on next page)

(continued from previous page)

```

if os.path.isfile(modelpth / f):
    msg = "Output file located: {}".format(f)
    print(msg)
else:
    errmsg = "Error. Output file cannot be found: {}".format(f)
    print(errmsg)

```

WARNING: Unable to resolve dimension of ('gwf6', 'disv', 'cell2d', 'cell2d', 'icvert')
↳ based on shape "ncvert".

writing simulation...

```

writing simulation name file...
writing simulation tdis package...
writing solution package ims...
writing model mp7p2...
writing model name file...
writing package disv...
writing package ic...
writing package npf...
writing package wel_0...

```

INFORMATION: maxbound in ('gwf6', 'wel', 'dimensions') changed to 1 based on size of
↳ stress_period_data

```

writing package rcha_0...
writing package riv_0...

```

INFORMATION: maxbound in ('gwf6', 'riv', 'dimensions') changed to 21 based on size of
↳ stress_period_data

```

writing package oc...

```

MODFLOW 6

U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_0000000

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:09

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

(continues on next page)

(continued from previous page)

Solving: Stress period: 1 Time step: 1

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:09
Elapsed run time: 0.141 Seconds

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'mp7p2.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'mp7p2.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.

Normal termination of simulation.

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.000000E+03 Steady-state flow

Particle Summary:

- 0 particles are pending release.
- 0 particles remain active.
- 16 particles terminated at boundary faces.
- 0 particles terminated at weak sink cells.
- 0 particles terminated at weak source cells.
- 0 particles terminated at strong source/sink cells.
- 0 particles terminated in cells with a specified zone number.
- 0 particles were stranded in inactive or dry cells.
- 0 particles were unreleased.
- 0 particles have an unknown status.

Normal termination.

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.000000E+03 Steady-state flow

Particle Summary:

- 0 particles are pending release.
- 0 particles remain active.
- 416 particles terminated at boundary faces.
- 0 particles terminated at weak sink cells.
- 0 particles terminated at weak source cells.
- 0 particles terminated at strong source/sink cells.
- 0 particles terminated in cells with a specified zone number.
- 0 particles were stranded in inactive or dry cells.

(continues on next page)

(continued from previous page)

```

0 particles were unreleased.
0 particles have an unknown status.

```

Normal termination.

Output file located: mp7p2.hds

Output file located: mp7p2.cbb

```

[21]: # load the simulation and get the model
vertex_sim_name = "mfsim.nam"
vertex_sim = flopy.mf6.MFSimulation.load(
    sim_name=vertex_sim_name, exe_name="mf6", sim_ws=modelpth
)
vertex_ml6 = vertex_sim.get_model("mp7p2")

# load the MODPATH-7 results
mp_namea = "mp7p2a_mp"
fpth = modelpth / f"{mp_namea}.mppth"
p = flopy.utils.PathlineFile(fpth)
p0 = p.get_alldata()

loading simulation...
  loading simulation name file...
  loading tdis package...
  loading model gw6...
  loading package disv...
WARNING: Unable to resolve dimension of ('gw6', 'disv', 'cell2d', 'cell2d', 'icvert')_
↳ based on shape "ncvert".
  loading package ic...
  loading package npf...
  loading package wel...
  loading package rch...
  loading package riv...
  loading package oc...
  loading solution package mp7p2...

```

Create the Vtk() object and add all of the model data to it

```

[22]: vtkobj = vtk.Vtk(vertex_ml6, xml=True, pvd=True, vertical_exageration=10)
      vtkobj.add_model(vertex_ml6)

```

Add modpath data to the Vtk() object.

Note: this will create a second vtk file that has the file signature: myfilename)_pathline.vtu

```

[23]: vtkobj.add_pathline_points(p0, timeseries=False)
      vtkobj.write(output_dir / "mp7_vertex_example" / "vertex_ex.vtu")

```

Clean up the temporary workspace.

```

[24]: try:
      # ignore PermissionError on Windows
      tempdir.cleanup()
    except:
      pass

```

5.1.4 External file handling

```
[1]: import os
import shutil
import sys
from tempfile import TemporaryDirectory

import numpy as np

import flopy
from flopy.utils import flopy_io

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
flopy version: 3.4.3
```

```
[2]: # make a model
nlay, nrow, ncol = 10, 20, 5
temp_dir = TemporaryDirectory()
model_ws = temp_dir.name

# the place for all of your hand made and costly model inputs
array_dir = os.path.join(temp_dir.name, "array_dir")
os.mkdir(array_dir)

ml = flopy.modflow.Modflow(model_ws=model_ws)
dis = flopy.modflow.ModflowDis(
    ml, nlay=nlay, nrow=nrow, ncol=ncol, steady=False, nper=2
)
```

make an `hk` and `vka` array. We'll save `hk` to files - pretend that you spent months making this important model property. Then make an `lpf`

```
[3]: hk = np.zeros((nlay, nrow, ncol)) + 5.0
vka = np.zeros_like(hk)
fnames = []
for i, h in enumerate(hk):
    fname = os.path.join(array_dir, "hk_{0}.ref".format(i + 1))
    fnames.append(fname)
    np.savetxt(fname, h)
    vka[i] = i + 1
lpf = flopy.modflow.ModflowLpf(ml, hk=fnames, vka=vka)
```

Let's also have some recharge with mixed args as well. Pretend the recharge in the second stress period is very important and precise

```
[4]: warmup_recharge = np.ones((nrow, ncol))
important_recharge = np.random.random((nrow, ncol))
fname = os.path.join(array_dir, "important_recharge.ref")
```

(continues on next page)

(continued from previous page)

```
np.savetxt(fname, important_recharge)
rch = flopy.modflow.ModflowRch(ml, rech={0: warmup_recharge, 1: fname})
```

```
[5]: ml.write_input()
```

Let's look at the files that were created

```
[6]: from pprint import pprint
```

```
print("model_ws:", flopy_io.scrub_login(ml.model_ws))
pprint([flopy_io.scrub_login(p) for p in os.listdir(ml.model_ws)])
```

```
model_ws: /tmp/tmpurvegp8kl
['hk_5.ref',
 'hk_2.ref',
 'hk_4.ref',
 'hk_8.ref',
 'hk_10.ref',
 'array_dir',
 'modflowtest.nam',
 'hk_3.ref',
 'modflowtest.lpf',
 'modflowtest.dis',
 'hk_6.ref',
 'modflowtest.rch',
 'hk_9.ref',
 'hk_1.ref',
 'important_recharge.ref',
 'hk_7.ref']
```

We see that a copy of the `hk` files as well as the important recharge file were made in the `model_ws`. Let's look at the `lpf` file

```
[7]: open(os.path.join(ml.model_ws, ml.name + ".lpf"), "r").readlines()[20]
```

[illegible]

(continues on next page)

(continued from previous page)

```
' 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00\n',
' 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00\n',
' 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00\n',
' 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00\n',
' 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00\n',
' 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00\n',
' 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00\n',
' 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00 1.000000E+00\n']
```

We see that the open/close approach was used - this is because `ml.array_free_format` is `True`. Notice that `vka` is written internally

```
[8]: ml.array_free_format
```

```
[8]: True
```

Now change `model_ws`

```
[9]: ml.model_ws = os.path.join(model_ws, "new_external_demo_dir")
```

```
creating model workspace...
```

```
../../../../../../../../tmp/tmpurvegp8k1/new_external_demo_dir
```

Now when we call `write_input()`, a copy of external files are made in the current `model_ws`

```
[10]: ml.write_input()
```

```
[11]: # list the files in model_ws that have 'hk' in the name
```

```
print(
    "\n".join(
        [
            name
            for name in os.listdir(ml.model_ws)
            if "hk" in name or "impor" in name
        ]
    )
)
```

```
hk_5.ref
hk_2.ref
hk_4.ref
hk_8.ref
hk_10.ref
hk_3.ref
hk_6.ref
hk_9.ref
hk_1.ref
important_recharge.ref
hk_7.ref
```

Now we see that the external files were copied to the new `model_ws`

Using external_path

It is sometimes useful when first building a model to write the model arrays as external files for processing and parameter estimation. The model attribute `external_path` triggers this behavior

```
[12]: # make a model - same code as before except for the model constructor
nlay, nrow, ncol = 10, 20, 5
model_ws = os.path.join(model_ws, "external_demo")
os.mkdir(model_ws)

# the place for all of your hand made and costly model inputs
array_dir = os.path.join(model_ws, "array_dir")
os.mkdir(array_dir)

# lets make an external path relative to the model_ws
ml = flopy.modflow.Modflow(
    model_ws=model_ws, external_path=os.path.join(model_ws, "ref")
)
dis = flopy.modflow.ModflowDis(
    ml, nlay=nlay, nrow=nrow, ncol=ncol, steady=False, nper=2
)

hk = np.zeros((nlay, nrow, ncol)) + 5.0
vka = np.zeros_like(hk)
fnames = []
for i, h in enumerate(hk):
    fname = os.path.join(array_dir, "hk_{0}.ref".format(i + 1))
    fnames.append(fname)
    np.savetxt(fname, h)
    vka[i] = i + 1
lpf = flopy.modflow.ModflowLpf(ml, hk=fnames, vka=vka)

warmup_recharge = np.ones((nrow, ncol))
important_recharge = np.random.random((nrow, ncol))
fname = os.path.join(array_dir, "important_recharge.ref")
np.savetxt(fname, important_recharge)
rch = flopy.modflow.ModflowRch(ml, rech={0: warmup_recharge, 1: fname})
```

We can see that the model constructor created both `model_ws` and `external_path` which is *relative to the model_ws*

```
[13]: os.listdir(ml.model_ws)
```

```
[13]: ['array_dir', 'ref']
```

Now, when we call `write_input()`, any array properties that were specified as `np.ndarray` will be written externally. If a scalar was passed as the argument, the value remains internal to the model input files

```
[14]: ml.write_input()
# open(os.path.join(ml.model_ws, ml.name + ".lpf"), "r").readlines()[:20]

Util2d:delr: resetting 'how' to external
Util2d:delc: resetting 'how' to external
Util2d:model_top: resetting 'how' to external
Util2d:botm layer 1: resetting 'how' to external
Util2d:botm layer 2: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```

Util2d:botm layer 3: resetting 'how' to external
Util2d:botm layer 4: resetting 'how' to external
Util2d:botm layer 5: resetting 'how' to external
Util2d:botm layer 6: resetting 'how' to external
Util2d:botm layer 7: resetting 'how' to external
Util2d:botm layer 8: resetting 'how' to external
Util2d:botm layer 9: resetting 'how' to external
Util2d:botm layer 10: resetting 'how' to external
Util2d:ss layer 1: resetting 'how' to external
Util2d:ss layer 2: resetting 'how' to external
Util2d:ss layer 3: resetting 'how' to external
Util2d:ss layer 4: resetting 'how' to external
Util2d:ss layer 5: resetting 'how' to external
Util2d:ss layer 6: resetting 'how' to external
Util2d:ss layer 7: resetting 'how' to external
Util2d:ss layer 8: resetting 'how' to external
Util2d:ss layer 9: resetting 'how' to external
Util2d:ss layer 10: resetting 'how' to external

```

Now, vka was also written externally, but not the storage properties. Let's verify the contents of the external path directory. We see our hard-fought hk and important_recharge arrays, as well as the vka arrays.

```

[15]: ml.lpf.ss.how = "internal"
ml.write_input()
# open(os.path.join(ml.model_ws, ml.name + ".lpf"), "r").readlines()[:20]

```

```

Util2d:delr: resetting 'how' to external
Util2d:delc: resetting 'how' to external
Util2d:model_top: resetting 'how' to external
Util2d:botm layer 1: resetting 'how' to external
Util2d:botm layer 2: resetting 'how' to external
Util2d:botm layer 3: resetting 'how' to external
Util2d:botm layer 4: resetting 'how' to external
Util2d:botm layer 5: resetting 'how' to external
Util2d:botm layer 6: resetting 'how' to external
Util2d:botm layer 7: resetting 'how' to external
Util2d:botm layer 8: resetting 'how' to external
Util2d:botm layer 9: resetting 'how' to external
Util2d:botm layer 10: resetting 'how' to external
Util2d:ss layer 1: resetting 'how' to external
Util2d:ss layer 2: resetting 'how' to external
Util2d:ss layer 3: resetting 'how' to external
Util2d:ss layer 4: resetting 'how' to external
Util2d:ss layer 5: resetting 'how' to external
Util2d:ss layer 6: resetting 'how' to external
Util2d:ss layer 7: resetting 'how' to external
Util2d:ss layer 8: resetting 'how' to external
Util2d:ss layer 9: resetting 'how' to external
Util2d:ss layer 10: resetting 'how' to external

```

```

[16]: print("\n".join(os.listdir(os.path.join(ml.model_ws, ml.external_path))))

```

```
hk_5.ref
hk_2.ref
rech_0.ref
hk_4.ref
hk_8.ref
hk_10.ref
botm_layer_2.ref
vka6.ref
ss_layer_5.ref
botm_layer_7.ref
botm_layer_4.ref
ss_layer_10.ref
botm_layer_8.ref
hk_3.ref
model_top.ref
botm_layer_1.ref
ss_layer_8.ref
vka4.ref
ss_layer_2.ref
ss_layer_7.ref
ss_layer_4.ref
ss_layer_9.ref
vka9.ref
vka7.ref
botm_layer_10.ref
vka5.ref
vka3.ref
vka2.ref
hk_6.ref
botm_layer_3.ref
ss_layer_6.ref
vka10.ref
ss_layer_1.ref
vka1.ref
botm_layer_9.ref
hk_9.ref
delr.ref
delc.ref
hk_1.ref
ss_layer_3.ref
important_recharge.ref
hk_7.ref
botm_layer_5.ref
botm_layer_6.ref
vka8.ref
```

Fixed format

All of this behavior also works for fixed-format type models (really, really old models - I mean OLD!)

```
[17]: # make a model - same code as before except for the model constructor
nlay, nrow, ncol = 10, 20, 5

# lets make an external path relative to the model_ws
ml = flopy.modflow.Modflow(model_ws=model_ws, external_path="ref")

# explicitly reset the free_format flag BEFORE ANY PACKAGES ARE MADE!!!
ml.array_free_format = False

dis = flopy.modflow.ModflowDis(
    ml, nlay=nlay, nrow=nrow, ncol=ncol, steady=False, nper=2
)

hk = np.zeros((nlay, nrow, ncol)) + 5.0
vka = np.zeros_like(hk)
fnames = []
for i, h in enumerate(hk):
    fname = os.path.join(array_dir, "hk_{0}.ref".format(i + 1))
    fnames.append(fname)
    np.savetxt(fname, h)
    vka[i] = i + 1
lpf = flopy.modflow.ModflowLpf(ml, hk=fnames, vka=vka)
ml.lpf.ss.how = "internal"
warmup_recharge = np.ones((nrow, ncol))
important_recharge = np.random.random((nrow, ncol))
fname = os.path.join(array_dir, "important_recharge.ref")
np.savetxt(fname, important_recharge)
rch = flopy.modflow.ModflowRch(ml, rech={0: warmup_recharge, 1: fname})

ml.write_input()
```

```
Note: external_path ref already exists
Util2d:delr: resetting 'how' to external
Util2d:delc: resetting 'how' to external
Util2d:model_top: resetting 'how' to external
Util2d:botm layer 1: resetting 'how' to external
Util2d:botm layer 2: resetting 'how' to external
Util2d:botm layer 3: resetting 'how' to external
Util2d:botm layer 4: resetting 'how' to external
Util2d:botm layer 5: resetting 'how' to external
Util2d:botm layer 6: resetting 'how' to external
Util2d:botm layer 7: resetting 'how' to external
Util2d:botm layer 8: resetting 'how' to external
Util2d:botm layer 9: resetting 'how' to external
Util2d:botm layer 10: resetting 'how' to external
Util2d hk layer 1: can't be free format...resetting
Util2d:ss layer 1: resetting 'how' to external
Util2d hk layer 2: can't be free format...resetting
Util2d:ss layer 2: resetting 'how' to external
Util2d hk layer 3: can't be free format...resetting
```

(continues on next page)

(continued from previous page)

```

Util2d:ss layer 3: resetting 'how' to external
Util2d:hk layer 4: can't be free format...resetting
Util2d:ss layer 4: resetting 'how' to external
Util2d:hk layer 5: can't be free format...resetting
Util2d:ss layer 5: resetting 'how' to external
Util2d:hk layer 6: can't be free format...resetting
Util2d:ss layer 6: resetting 'how' to external
Util2d:hk layer 7: can't be free format...resetting
Util2d:ss layer 7: resetting 'how' to external
Util2d:hk layer 8: can't be free format...resetting
Util2d:ss layer 8: resetting 'how' to external
Util2d:hk layer 9: can't be free format...resetting
Util2d:ss layer 9: resetting 'how' to external
Util2d:hk layer 10: can't be free format...resetting
Util2d:ss layer 10: resetting 'how' to external
Util2d:rech_2: can't be free format...resetting

```

We see that now the external arrays are being handled through the name file. Let's look at the name file

```

[18]: open(os.path.join(ml.model_ws, ml.name + ".nam"), "r").readlines()
[18]: ['# Name file for MODFLOW-2005, generated by FloPy version 3.4.3.\n',
      '#x11:0.0; y11:0.0; rotation:0.0; units:meters; lenuni:2; start_datetime:1-1-1970\n',
      'LIST          2  modflowtest.list\n',
      'DIS            11  modflowtest.dis\n',
      'LPF            15  modflowtest.lpf\n',
      'RCH            19  modflowtest.rch\n',
      'DATA          1001  ref/delr.ref\n',
      'DATA          1002  ref/delc.ref\n',
      'DATA          1003  ref/model_top.ref\n',
      'DATA          1004  ref/botm_layer_1.ref\n',
      'DATA          1005  ref/botm_layer_2.ref\n',
      'DATA          1006  ref/botm_layer_3.ref\n',
      'DATA          1007  ref/botm_layer_4.ref\n',
      'DATA          1008  ref/botm_layer_5.ref\n',
      'DATA          1009  ref/botm_layer_6.ref\n',
      'DATA          1010  ref/botm_layer_7.ref\n',
      'DATA          1011  ref/botm_layer_8.ref\n',
      'DATA          1012  ref/botm_layer_9.ref\n',
      'DATA          1013  ref/botm_layer_10.ref\n',
      'DATA          1014  ref/hk_1.ref\n',
      'DATA          1015  ref/vka1.ref\n',
      'DATA          1016  ref/ss_layer_1.ref\n',
      'DATA          1017  ref/hk_2.ref\n',
      'DATA          1018  ref/vka2.ref\n',
      'DATA          1019  ref/ss_layer_2.ref\n',
      'DATA          1020  ref/hk_3.ref\n',
      'DATA          1021  ref/vka3.ref\n',
      'DATA          1022  ref/ss_layer_3.ref\n',
      'DATA          1023  ref/hk_4.ref\n',
      'DATA          1024  ref/vka4.ref\n',
      'DATA          1025  ref/ss_layer_4.ref\n',
      'DATA          1026  ref/hk_5.ref\n',

```

(continues on next page)

(continued from previous page)

```
'DATA      1027  ref/vka5.ref\n',
'DATA      1028  ref/ss_layer_5.ref\n',
'DATA      1029  ref/hk_6.ref\n',
'DATA      1030  ref/vka6.ref\n',
'DATA      1031  ref/ss_layer_6.ref\n',
'DATA      1032  ref/hk_7.ref\n',
'DATA      1033  ref/vka7.ref\n',
'DATA      1034  ref/ss_layer_7.ref\n',
'DATA      1035  ref/hk_8.ref\n',
'DATA      1036  ref/vka8.ref\n',
'DATA      1037  ref/ss_layer_8.ref\n',
'DATA      1038  ref/hk_9.ref\n',
'DATA      1039  ref/vka9.ref\n',
'DATA      1040  ref/ss_layer_9.ref\n',
'DATA      1041  ref/hk_10.ref\n',
'DATA      1042  ref/vka10.ref\n',
'DATA      1043  ref/ss_layer_10.ref\n',
'DATA      1044  ref/rech_0.ref\n',
'DATA      1045  ref/important_recharge.ref\n']
```

Free and binary format

```
[19]: ml.dis.botm[0].format.binary = True
ml.write_input()
```

```
Util2d:delr: resetting 'how' to external
Util2d:delc: resetting 'how' to external
Util2d:model_top: resetting 'how' to external
Util2d:botm layer 1: resetting 'how' to external
Util2d:botm layer 2: resetting 'how' to external
Util2d:botm layer 3: resetting 'how' to external
Util2d:botm layer 4: resetting 'how' to external
Util2d:botm layer 5: resetting 'how' to external
Util2d:botm layer 6: resetting 'how' to external
Util2d:botm layer 7: resetting 'how' to external
Util2d:botm layer 8: resetting 'how' to external
Util2d:botm layer 9: resetting 'how' to external
Util2d:botm layer 10: resetting 'how' to external
Util2d:ss layer 1: resetting 'how' to external
Util2d:ss layer 2: resetting 'how' to external
Util2d:ss layer 3: resetting 'how' to external
Util2d:ss layer 4: resetting 'how' to external
Util2d:ss layer 5: resetting 'how' to external
Util2d:ss layer 6: resetting 'how' to external
Util2d:ss layer 7: resetting 'how' to external
Util2d:ss layer 8: resetting 'how' to external
Util2d:ss layer 9: resetting 'how' to external
Util2d:ss layer 10: resetting 'how' to external
```

```
[20]: open(os.path.join(ml.model_ws, ml.name + ".nam"), "r").readlines()
```

```
[20]: ['# Name file for MODFLOW-2005, generated by Flopy version 3.4.3.\n',
      '#xll:0.0; yll:0.0; rotation:0.0; units:meters; lenuni:2; start_datetime:1-1-1970\n',
      'LIST          2  modflowtest.list\n',
      'DIS            11  modflowtest.dis\n',
      'LPF             15  modflowtest.lpf\n',
      'RCH             19  modflowtest.rch\n',
      'DATA            1046 ref/delr.ref\n',
      'DATA            1047 ref/delc.ref\n',
      'DATA            1048 ref/model_top.ref\n',
      'DATA(BINARY)    1049 ref/botm_layer_1.ref\n',
      'DATA            1050 ref/botm_layer_2.ref\n',
      'DATA            1051 ref/botm_layer_3.ref\n',
      'DATA            1052 ref/botm_layer_4.ref\n',
      'DATA            1053 ref/botm_layer_5.ref\n',
      'DATA            1054 ref/botm_layer_6.ref\n',
      'DATA            1055 ref/botm_layer_7.ref\n',
      'DATA            1056 ref/botm_layer_8.ref\n',
      'DATA            1057 ref/botm_layer_9.ref\n',
      'DATA            1058 ref/botm_layer_10.ref\n',
      'DATA            1059 ref/hk_1.ref\n',
      'DATA            1060 ref/vka1.ref\n',
      'DATA            1061 ref/ss_layer_1.ref\n',
      'DATA            1062 ref/hk_2.ref\n',
      'DATA            1063 ref/vka2.ref\n',
      'DATA            1064 ref/ss_layer_2.ref\n',
      'DATA            1065 ref/hk_3.ref\n',
      'DATA            1066 ref/vka3.ref\n',
      'DATA            1067 ref/ss_layer_3.ref\n',
      'DATA            1068 ref/hk_4.ref\n',
      'DATA            1069 ref/vka4.ref\n',
      'DATA            1070 ref/ss_layer_4.ref\n',
      'DATA            1071 ref/hk_5.ref\n',
      'DATA            1072 ref/vka5.ref\n',
      'DATA            1073 ref/ss_layer_5.ref\n',
      'DATA            1074 ref/hk_6.ref\n',
      'DATA            1075 ref/vka6.ref\n',
      'DATA            1076 ref/ss_layer_6.ref\n',
      'DATA            1077 ref/hk_7.ref\n',
      'DATA            1078 ref/vka7.ref\n',
      'DATA            1079 ref/ss_layer_7.ref\n',
      'DATA            1080 ref/hk_8.ref\n',
      'DATA            1081 ref/vka8.ref\n',
      'DATA            1082 ref/ss_layer_8.ref\n',
      'DATA            1083 ref/hk_9.ref\n',
      'DATA            1084 ref/vka9.ref\n',
      'DATA            1085 ref/ss_layer_9.ref\n',
      'DATA            1086 ref/hk_10.ref\n',
      'DATA            1087 ref/vka10.ref\n',
      'DATA            1088 ref/ss_layer_10.ref\n',
      'DATA            1089 ref/rech_0.ref\n',
      'DATA            1090 ref/important_recharge.ref']
```



```
[21]: open(os.path.join(ml.model_ws, ml.name + ".dis"), "r").readlines()
[21]: ['# DIS package for MODFLOW-2005 generated by FloPy 3.4.3\n',
      '      10      20      5      2      4      2\n',
      ' 0 0 0 0 0 0 0 0 0 0\n',
      '    1046      1      (5E15.6)      -1 #delr\n',
      '    1047      1      (20E15.6)     -1 #delc\n',
      '    1048      1      (5E15.6)     -1 #model_top\n',
      '   -1049      1      (BINARY)     -1 #botm layer 1\n',
      '    1050      1      (5E15.6)     -1 #botm layer 2\n',
      '    1051      1      (5E15.6)     -1 #botm layer 3\n',
      '    1052      1      (5E15.6)     -1 #botm layer 4\n',
      '    1053      1      (5E15.6)     -1 #botm layer 5\n',
      '    1054      1      (5E15.6)     -1 #botm layer 6\n',
      '    1055      1      (5E15.6)     -1 #botm layer 7\n',
      '    1056      1      (5E15.6)     -1 #botm layer 8\n',
      '    1057      1      (5E15.6)     -1 #botm layer 9\n',
      '    1058      1      (5E15.6)     -1 #botm layer 10\n',
      '    1.000000      1  1.000000  TR\n',
      '    1.000000      1  1.000000  TR']
```

The .how attribute

Util2d includes a .how attribute that gives finer grained control of how arrays will be written

```
[22]: ml.lpf.hk[0].how
[22]: 'external'
```

This will raise an error since our model does not support free format...

```
[23]: try:
      ml.lpf.hk[0].how = "openclose"
      ml.lpf.hk[0].how
      ml.write_input()
except Exception as e:
    print("\n", e, "\n")

Util2d:delr: resetting 'how' to external
Util2d:delc: resetting 'how' to external
Util2d:model_top: resetting 'how' to external
Util2d:botm layer 1: resetting 'how' to external
Util2d:botm layer 2: resetting 'how' to external
Util2d:botm layer 3: resetting 'how' to external
Util2d:botm layer 4: resetting 'how' to external
Util2d:botm layer 5: resetting 'how' to external
Util2d:botm layer 6: resetting 'how' to external
Util2d:botm layer 7: resetting 'how' to external
Util2d:botm layer 8: resetting 'how' to external
Util2d:botm layer 9: resetting 'how' to external
Util2d:botm layer 10: resetting 'how' to external

Util2d error: 'how' is openclose, but model doesn't support free fmt
```

So let's reset hk layer 1 back to external...

```
[24]: ml.lpf.hk[0].how = "external"
      ml.lpf.hk[0].how
```

```
[24]: 'external'
```

```
[25]: ml.dis.top.how = "external"
```

```
[26]: ml.write_input()

Util2d:delr: resetting 'how' to external
Util2d:delc: resetting 'how' to external
Util2d:botm layer 1: resetting 'how' to external
Util2d:botm layer 2: resetting 'how' to external
Util2d:botm layer 3: resetting 'how' to external
Util2d:botm layer 4: resetting 'how' to external
Util2d:botm layer 5: resetting 'how' to external
Util2d:botm layer 6: resetting 'how' to external
Util2d:botm layer 7: resetting 'how' to external
Util2d:botm layer 8: resetting 'how' to external
Util2d:botm layer 9: resetting 'how' to external
Util2d:botm layer 10: resetting 'how' to external
Util2d:ss layer 1: resetting 'how' to external
Util2d:ss layer 2: resetting 'how' to external
Util2d:ss layer 3: resetting 'how' to external
Util2d:ss layer 4: resetting 'how' to external
Util2d:ss layer 5: resetting 'how' to external
Util2d:ss layer 6: resetting 'how' to external
Util2d:ss layer 7: resetting 'how' to external
Util2d:ss layer 8: resetting 'how' to external
Util2d:ss layer 9: resetting 'how' to external
Util2d:ss layer 10: resetting 'how' to external
```

```
[27]: open(os.path.join(ml.model_ws, ml.name + ".dis"), "r").readlines()
```

```
[27]: ['# DIS package for MODFLOW-2005 generated by FloPy 3.4.3\n',
      '    10    20    5    2    4    2\n',
      ' 0 0 0 0 0 0 0 0 0 0\n',
      '    1104    1    (5E15.6)    -1 #delr\n',
      '    1105    1    (20E15.6)   -1 #delc\n',
      '    1106    1    (5E15.6)   -1 #model_top\n',
      '   -1107    1    (BINARY)    -1 #botm layer 1\n',
      '    1108    1    (5E15.6)    -1 #botm layer 2\n',
      '    1109    1    (5E15.6)    -1 #botm layer 3\n',
      '    1110    1    (5E15.6)    -1 #botm layer 4\n',
      '    1111    1    (5E15.6)    -1 #botm layer 5\n',
      '    1112    1    (5E15.6)    -1 #botm layer 6\n',
      '    1113    1    (5E15.6)    -1 #botm layer 7\n',
      '    1114    1    (5E15.6)    -1 #botm layer 8\n',
      '    1115    1    (5E15.6)    -1 #botm layer 9\n',
      '    1116    1    (5E15.6)    -1 #botm layer 10\n',
      '    1.000000    1 1.000000 TR\n',
      '    1.000000    1 1.000000 TR']
```

```
open(os.path.join(ml.model_ws, ml.name + ".lpf"), "r").readlines()
```

```
[ '# LPF package for MODFLOW-2005 generated by Flopy 3.4.3\n',  
    '      0      -1E+30      0      \n',  
    '      0      0      0      0      0      0      0      0  
→ 0      0\n',  
    '      0      0      0      0      0      0      0      0  
→ 0      0\n',  
    '      1.000000E+00      1.000000E+00      1.000000E+00      1.000000E+00      1.000000E+00      1.  
→ 000000E+00      1.000000E+00      1.000000E+00      1.000000E+00      1.000000E+00\n',  
    '      0      0      0      0      0      0      0      0  
→ 0      0\n',  
    '      0      0      0      0      0      0      0      0  
→ 0      0\n',  
    '      1117      1      (5E15.6)      -1 #hk layer 1\n',  
    '      1118      1      (5E15.6)      -1 #vka1\n',  
    '      1119      1      (5E15.6)      -1 #ss layer 1\n',  
    '      1120      1      (5E15.6)      -1 #hk layer 2\n',  
    '      1121      1      (5E15.6)      -1 #vka2\n',  
    '      1122      1      (5E15.6)      -1 #ss layer 2\n',  
    '      1123      1      (5E15.6)      -1 #hk layer 3\n',  
    '      1124      1      (5E15.6)      -1 #vka3\n',  
    '      1125      1      (5E15.6)      -1 #ss layer 3\n',  
    '      1126      1      (5E15.6)      -1 #hk layer 4\n',  
    '      1127      1      (5E15.6)      -1 #vka4\n',  
    '      1128      1      (5E15.6)      -1 #ss layer 4\n',  
    '      1129      1      (5E15.6)      -1 #hk layer 5\n',  
    '      1130      1      (5E15.6)      -1 #vka5\n',  
    '      1131      1      (5E15.6)      -1 #ss layer 5\n',  
    '      1132      1      (5E15.6)      -1 #hk layer 6\n',  
    '      1133      1      (5E15.6)      -1 #vka6\n',  
    '      1134      1      (5E15.6)      -1 #ss layer 6\n',  
    '      1135      1      (5E15.6)      -1 #hk layer 7\n',  
    '      1136      1      (5E15.6)      -1 #vka7\n',  
    '      1137      1      (5E15.6)      -1 #ss layer 7\n',  
    '      1138      1      (5E15.6)      -1 #hk layer 8\n',  
    '      1139      1      (5E15.6)      -1 #vka8\n',  
    '      1140      1      (5E15.6)      -1 #ss layer 8\n',  
    '      1141      1      (5E15.6)      -1 #hk layer 9\n',  
    '      1142      1      (5E15.6)      -1 #vka9\n',  
    '      1143      1      (5E15.6)      -1 #ss layer 9\n',  
    '      1144      1      (5E15.6)      -1 #hk layer 10\n',  
    '      1145      1      (5E15.6)      -1 #vka10\n',  
    '      1146      1      (5E15.6)      -1 #ss layer 10\n']
```

```
open(os.path.join(ml.model_ws, ml.name + ".nam"), "r").readlines()
```

```
[ '# Name file for MODFLOW-2005, generated by Flopy version 3.4.3.\n',  
  '#xll:0.0; yll:0.0; rotation:0.0; units:meters; lenuni:2; start_datetime:1-1-1970\n',  
  'LIST          2  modflowtest.list\n',  
  'DIS          11  modflowtest.dis\n',  
  'LPF          15  modflowtest.lpf\n',  
  'RCH          19  modflowtest.rch\n',
```

(continues on next page)

(continued from previous page)

```

'DATA          1104 ref/delr.ref\n',
'DATA          1105 ref/delc.ref\n',
'DATA          1106 ref/model_top.ref\n',
'DATA(BINARY)  1107 ref/botm_layer_1.ref\n',
'DATA          1108 ref/botm_layer_2.ref\n',
'DATA          1109 ref/botm_layer_3.ref\n',
'DATA          1110 ref/botm_layer_4.ref\n',
'DATA          1111 ref/botm_layer_5.ref\n',
'DATA          1112 ref/botm_layer_6.ref\n',
'DATA          1113 ref/botm_layer_7.ref\n',
'DATA          1114 ref/botm_layer_8.ref\n',
'DATA          1115 ref/botm_layer_9.ref\n',
'DATA          1116 ref/botm_layer_10.ref\n',
'DATA          1117 ref/hk_1.ref\n',
'DATA          1118 ref/vka1.ref\n',
'DATA          1119 ref/ss_layer_1.ref\n',
'DATA          1120 ref/hk_2.ref\n',
'DATA          1121 ref/vka2.ref\n',
'DATA          1122 ref/ss_layer_2.ref\n',
'DATA          1123 ref/hk_3.ref\n',
'DATA          1124 ref/vka3.ref\n',
'DATA          1125 ref/ss_layer_3.ref\n',
'DATA          1126 ref/hk_4.ref\n',
'DATA          1127 ref/vka4.ref\n',
'DATA          1128 ref/ss_layer_4.ref\n',
'DATA          1129 ref/hk_5.ref\n',
'DATA          1130 ref/vka5.ref\n',
'DATA          1131 ref/ss_layer_5.ref\n',
'DATA          1132 ref/hk_6.ref\n',
'DATA          1133 ref/vka6.ref\n',
'DATA          1134 ref/ss_layer_6.ref\n',
'DATA          1135 ref/hk_7.ref\n',
'DATA          1136 ref/vka7.ref\n',
'DATA          1137 ref/ss_layer_7.ref\n',
'DATA          1138 ref/hk_8.ref\n',
'DATA          1139 ref/vka8.ref\n',
'DATA          1140 ref/ss_layer_8.ref\n',
'DATA          1141 ref/hk_9.ref\n',
'DATA          1142 ref/vka9.ref\n',
'DATA          1143 ref/ss_layer_9.ref\n',
'DATA          1144 ref/hk_10.ref\n',
'DATA          1145 ref/vka10.ref\n',
'DATA          1146 ref/ss_layer_10.ref\n',
'DATA          1147 ref/rech_0.ref\n',
'DATA          1148 ref/important_recharge.ref\n']

```

```

[30]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
except:
    pass

```

5.2 MODFLOW 6

5.2.1 MODFLOW 6: Accessing Simulation Settings, Models, and Packages

This tutorial shows how to view, access, and change the underlying package variables for MODFLOW 6 objects in FloPy. Interaction with a FloPy MODFLOW 6 model is different from other models, such as MODFLOW-2005, MT3D, and SEAWAT, for example.

The MODFLOW 6 simulation structure is arranged in the following generalized way:

```
SIMULATION --> PACKAGE --> Data

SIMULATION --> MODEL --> PACKAGE (--> PACKAGE) --> Data
```

This tutorial focuses on accessing simulation-wide FloPy settings and how to create and access models and packages. Tutorial 3, 4, and 5 offer a more in depth look at observation, time series, and time array series packages, and tutorial 6, 7, 8, and 9 offer a more in depth look at the data.

Create Simple Demonstration Model

This tutorial uses a simple demonstration simulation with one GWF Model. The model has 3 layers, 4 rows, and 5 columns. The model is set up to use multiple model layers in order to demonstrate some of the layered functionality in FloPy.

```
[1]: # package import
    from pathlib import Path
    from tempfile import TemporaryDirectory

[2]: import flopy

[3]: temp_dir = TemporaryDirectory()
    workspace = temp_dir.name
    name = "tutorial01_mf6_data"

[4]: # set up simulation and basic packages
    sim = flopy.mf6.MFSimulation(sim_name=name, sim_ws=workspace)
    flopy.mf6.ModflowTdis(
        sim, nper=10, perioddata=[[365.0, 1, 1.0] for _ in range(10)]
    )
    flopy.mf6.ModflowIms(sim)
    gwf = flopy.mf6.ModflowGwf(sim, modelname=name, save_flows=True)
    flopy.mf6.ModflowGwfdis(gwf, nlay=3, nrow=4, ncol=5)
    flopy.mf6.ModflowGwfic(gwf)
    flopy.mf6.ModflowGwfnpf(gwf, save_specific_discharge=True)
    flopy.mf6.ModflowGwfchd(
        gwf, stress_period_data=[[(0, 0, 0), 1.0], [(2, 3, 4), 0.0]]
    )
    budget_file = f"{name}.bud"
    head_file = f"{name}.hds"
    flopy.mf6.ModflowGwfoc(
        gwf,
        budget_filerecord=budget_file,
```

(continues on next page)

(continued from previous page)

```

    head_filerecord=head_file,
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
)
print("Done creating simulation.")
Done creating simulation.

```

Accessing Simulation-Level Settings

FloPy has a number of settings that can be set for the entire simulation. These include how much information FloPy writes to the console, how to format the MODFLOW package files, and whether to verify MODFLOW data.

The verbosity level, which determines how much FloPy writes to command line output. The options are 1 for quiet, 2 for normal, and 3 for verbose. Below we set the verbosity level to verbose.

```
[5]: sim.simulation_data.verbosity_level = 3
```

We can also set the number of spaces to indent data when writing package files by setting the indent string.

```
[6]: sim.simulation_data.indent_string = "    "
```

Next we set the precision and number of characters written for floating point variables.

```
[7]: sim.float_precision = 8
     sim.float_characters = 15
```

Lastly, we disable verify_data and auto_set_sizes for faster performance. With these options disabled FloPy will not do any checking or autocorrecting of your data.

```
[8]: sim.verify_data = False
     sim.auto_set_sizes = False
```

Accessing Models and Packages

At this point a simulation is available in memory. In this particular case the simulation was created directly using Python code; however, the simulation might also have been loaded from existing model files using the FloPy.mf6.MFSimulation.load() function.

Once a MODFLOW 6 simulation is available in memory, the contents of the simulation object can be listed using a simple print command.

```
[9]: print(sim)

sim_name = tutorial01_mf6_data
sim_path = /tmp/tmp69lfx6si
exe_name = mf6

#####
Package mfsim.nam
#####

package_name = mfsim.nam

```

(continues on next page)

(continued from previous page)

```
filename = mfsim.nam
package_type = nam
model_or_simulation_package = simulation
simulation_name = tutorial01_mf6_data
```

```
#####
```

```
Package tutorial01_mf6_data.tdis
```

```
#####
```

```
package_name = tutorial01_mf6_data.tdis
filename = tutorial01_mf6_data.tdis
package_type = tdis
model_or_simulation_package = simulation
simulation_name = tutorial01_mf6_data
```

```
#####
```

```
Package ims_-1
```

```
#####
```

```
package_name = ims_-1
filename = tutorial01_mf6_data.ims
package_type = ims
model_or_simulation_package = simulation
simulation_name = tutorial01_mf6_data
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
Model tutorial01_mf6_data
```

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
name = tutorial01_mf6_data
model_type = gwf6
version = mf6
model_relative_path = .
```

```
#####
```

```
Package dis
```

```
#####
```

```
package_name = dis
filename = tutorial01_mf6_data.dis
package_type = dis
model_or_simulation_package = model
model_name = tutorial01_mf6_data
```

```
#####
```

```
Package ic
```

```
#####
```

(continues on next page)

(continued from previous page)

```
package_name = ic
filename = tutorial01_mf6_data.ic
package_type = ic
model_or_simulation_package = model
model_name = tutorial01_mf6_data
```

```
#####
Package npf
#####
```

```
package_name = npf
filename = tutorial01_mf6_data.npf
package_type = npf
model_or_simulation_package = model
model_name = tutorial01_mf6_data
```

```
#####
Package chd_0
#####
```

```
package_name = chd_0
filename = tutorial01_mf6_data.chd
package_type = chd
model_or_simulation_package = model
model_name = tutorial01_mf6_data
```

```
#####
Package oc
#####
```

```
package_name = oc
filename = tutorial01_mf6_data.oc
package_type = oc
model_or_simulation_package = model
model_name = tutorial01_mf6_data
```

Simulation-level packages, models, and model packages can be shown by printing the simulation object. In this case, you should see the all of the contents of simulation and some information about each FloPy object that is part of simulation.

To get the TDIS package and print the contents, we can do the following

```
[10]: tdis = sim.tdis
      print(tdis)

      package_name = tutorial01_mf6_data.tdis
```

(continues on next page)

(continued from previous page)

```
filename = tutorial01_mf6_data.tdis
package_type = tdis
model_or_simulation_package = simulation
simulation_name = tutorial01_mf6_data

Block dimensions
-----
nper
{internal}
(10)

Block perioddata
-----
perioddata
{internal}
([(365., 1, 1.) (365., 1, 1.) (365., 1, 1.) (365., 1, 1.) (365., 1, 1.)
 (365., 1, 1.) (365., 1, 1.) (365., 1, 1.) (365., 1, 1.) (365., 1, 1.)])
```

To get the Iterative Model Solution (IMS) object, we use the following syntax

```
[11]: ims = sim.get_package("ims_-1")
print(ims)

package_name = ims_-1
filename = tutorial01_mf6_data.ims
package_type = ims
model_or_simulation_package = simulation
simulation_name = tutorial01_mf6_data
```

Or because there is only one IMS object for this simulation, we can access it as

```
[12]: ims = sim.get_package("ims")
print(ims)

package_name = ims_-1
filename = tutorial01_mf6_data.ims
package_type = ims
model_or_simulation_package = simulation
simulation_name = tutorial01_mf6_data
```

When printing the sim object, there is also a simulation package called nam. This package contains the information that is written to the mfsim.nam file, which is the primary file that MODFLOW 6 reads when it first starts. The nam package is automatically updated for you by FloPy and does not require modification.

```
[13]: nam = sim.get_package("nam")
print(nam)
```

```
package_name = mfsim.nam
filename = mfsim.nam
package_type = nam
model_or_simulation_package = simulation
simulation_name = tutorial01_mf6_data

Block timing
-----
tdis6
{internal}
(tutorial01_mf6_data.tdis)

Block models
-----
models
{internal}
([('gwf6', 'tutorial01_mf6_data.nam', 'tutorial01_mf6_data')])

Block solutiongroup
-----
solutiongroup
{internal}
([('ims6', 'tutorial01_mf6_data.ims', 'tutorial01_mf6_data')])
```

To see the models that are contained within the simulation, we can get a list of their names as follows

```
[14]: print(sim.model_names)

['tutorial01_mf6_data']
```

`sim.model_names` returns the keys of an ordered dictionary, which isn't very useful to us, but we can convert that to a list and then go through that list and print information about each model in the simulation. In this case there is only one model, but had there been more models, we would see them listed here

```
[15]: model_names = list(sim.model_names)
      for mname in model_names:
          print(mname)

tutorial01_mf6_data
```

If we want to get a model from a simulation, then we use the `get_model()` method of the `sim` object. Here we go through all the models in the simulation and print the model name and the model type.

```
[16]: model_names = list(sim.model_names)
      for mname in model_names:
          m = sim.get_model(mname)
          print(m.name, m.model_type)

tutorial01_mf6_data gwf6
```

For this simple case here with only one GWF model, we can very easily get the FloPy representation of the GWF model as

```
[17]: gwf = sim.get_model(name)
```

Now that we have the GWF object, we can print it, and see what's it contains.

```
[18]: print(gwf)

name = tutorial01_mf6_data
model_type = gwf6
version = mf6
model_relative_path = .

#####
Package dis
#####

package_name = dis
filename = tutorial01_mf6_data.dis
package_type = dis
model_or_simulation_package = model
model_name = tutorial01_mf6_data

#####
Package ic
#####

package_name = ic
filename = tutorial01_mf6_data.ic
package_type = ic
model_or_simulation_package = model
model_name = tutorial01_mf6_data

#####
Package npf
#####

package_name = npf
filename = tutorial01_mf6_data.npf
package_type = npf
model_or_simulation_package = model
model_name = tutorial01_mf6_data

#####
Package chd_0
#####

package_name = chd_0
filename = tutorial01_mf6_data.chd
package_type = chd
```

(continues on next page)

(continued from previous page)

```
model_or_simulation_package = model
model_name = tutorial01_mf6_data

#####
Package oc
#####

package_name = oc
filename = tutorial01_mf6_data.oc
package_type = oc
model_or_simulation_package = model
model_name = tutorial01_mf6_data
```

What we see here is the information that we saw when we printed the sim object.

One of the most common operations on a model is to see what packages are in it and then get packages of interest. A list of packages in a model can be obtained as

```
[19]: package_list = gwf.get_package_list()
print(package_list)

['DIS', 'IC', 'NPF', 'CHD_0', 'OC']
```

As you might expect we can access each package in this list with `gwf.get_package()`. Thus, the following syntax can be used to obtain and print the contents of the DIS Package

```
[20]: dis = gwf.get_package("dis")
print(dis)

package_name = dis
filename = tutorial01_mf6_data.dis
package_type = dis
model_or_simulation_package = model
model_name = tutorial01_mf6_data

Block dimensions
-----
nlay
{internal}
(3)

nrow
{internal}
(4)

ncol
{internal}
(5)
```

(continues on next page)

(continued from previous page)

```

Block griddata
-----
delr
{constant 1.0}

delc
{constant 1.0}

top
{constant 1.0}

botm
{constant 0.0}

```

The Python type for this dis package is simply

```
[21]: print(type(dis))

<class 'flopy.mf6.modflow.mfgwfdis.ModflowGwfdis'>
```

```
[22]: try:
        temp_dir.cleanup()
    except PermissionError:
        # can occur on windows: https://docs.python.org/3/library/tempfile.html#tempfile.TemporaryDirectory
        pass
```

5.2.2 MODFLOW 6: Observation packages

Introduction to Observations

Observations can be set for any package through the `package.obs` object, and each `package.obs` object has several attributes that can be set:

Attribute	Type	Description
<code>package.obs.filename</code>	str	Name of observations file to create. The default is <code>packagename + '.obs'</code> .
<code>package.obs.continuous</code>	dict	A dictionary that has file names as keys and a list of observations as the dictionary values.
<code>package.obs.digits</code>	int	Number of digits to write the observation values. Default is 10.
<code>package.obs.print_input</code>	bool	Flag indicating whether or not observations are written to listing file.

The following code sets up a simulation used in the observation examples.

```
[1]: # package import
import os
```

(continues on next page)

(continued from previous page)

```
from pathlib import Path
from tempfile import TemporaryDirectory
```

```
[2]: import numpy as np
```

```
[3]: import flopy
```

```
[4]: # set up where simulation workspace will be stored
temp_dir = TemporaryDirectory()
workspace = temp_dir.name
name = "tutorial02_mf6_data"
```

```
[5]: # create the flopy simulation and tdis objects
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws=workspace
)
tdis_rc = [(1.0, 1, 1.0), (10.0, 5, 1.0), (10.0, 5, 1.0), (10.0, 1, 1.0)]
tdis_package = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim, time_units="DAYS", nper=4, perioddata=tdis_rc
)
# create the flopy groundwater flow (gwf) model object
model_name_file = f"{name}.nam"
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, model_name_file=model_name_file)
# create the flopy iterative model solver (ims) package object
ims = flopy.mf6.modflow.mfims.ModflowIms(sim, pname="ims", complexity="SIMPLE")
# create the discretization package
bot = np.linspace(-3.0, -50.0 / 3.0, 3)
delrow = delcol = 4.0
dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf,
    pname="dis",
    nogrb=True,
    nlay=3,
    nrow=101,
    ncol=101,
    delr=delrow,
    delc=delcol,
    top=0.0,
    botm=bot,
)
# create the initial condition (ic) and node property flow (npf) packages
ic_package = flopy.mf6.modflow.mfgwfic.ModflowGwfic(gwf, strt=50.0)
npf_package = flopy.mf6.modflow.mfgwnpf.ModflowGwnpf(
    gwf,
    save_flows=True,
    icelltype=[1, 0, 0],
    k=[5.0, 0.1, 4.0],
    k33=[0.5, 0.005, 0.1],
)
```

Observation Example 1

One method to build the observation package is to pass a dictionary with the observations containing “observations” parameters of the parent package.

This example uses the observation package in a GHB package. First the stress period data for a ghb package is built.

```
[6]: # build ghb stress period data
ghb_spd = {}
ghb_period = []
for layer, cond in zip(range(1, 3), [15.0, 1500.0]):
    for row in range(0, 15):
        ghb_period.append((layer, row, 9), 1.0, cond, "Estuary-L2"))
ghb_spd[0] = ghb_period
```

The next step is to build the observation data in a dictionary. The dictionary key is the filename of the observation output file and optionally a “binary” keyword to make the file binary. When the optional “binary” keyword is used the dictionary key is a tuple, otherwise it is a string. The dictionary value is a list of tuples containing the contents of the observation package’s continuous block, with each tuple containing one line of information.

```
[7]: # build obs data
ghb_obs = {
    ("ghb_obs.csv", "binary"): [
        ("ghb-2-6-10", "GHB", (1, 5, 9)),
        ("ghb-3-6-10", "GHB", (2, 5, 9)),
    ],
    "ghb_flows.csv": [
        ("Estuary2", "GHB", "Estuary-L2"),
        ("Estuary3", "GHB", "Estuary-L3"),
    ],
}
```

The ghb package is now constructed with observations by setting the observations parameter to ghb_obs on construction of the ghb package.

```
[8]: # build ghb package passing obs dictionary to package constructor
ghb = flopy.mf6.modflow.mfgwfgghb.ModflowGwfgghb(
    gwf,
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    observations=ghb_obs,
    pname="ghb",
    maxbound=30,
    stress_period_data=ghb_spd,
)
```

Observation information such as the print_input option can then be set using the package’s obs parameter.

```
[9]: ghb.obs.print_input = True
```

```
[10]: # clean up for next example
gwf.remove_package("ghb")
```

Observation Example 2

Alternatively, an obs package can be built by initializing obs through `ghb.obs.initialize`.

First, a GHB package is built without defining observations.

```
[11]: # build ghb package
ghb = flopy.mf6.modflow.mfgwfghb.ModflowGwfghb(
    gwf,
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    maxbound=30,
    stress_period_data=ghb_spd,
    pname="ghb",
)
```

Then the ghb observations are defined in a dictionary similar to example 1.

```
[12]: # build obs data
ghb_obs = {
    ("ghb_obs.csv", "binary"): [
        ("ghb-2-6-10", "GHB", (1, 5, 9)),
        ("ghb-3-6-10", "GHB", (2, 5, 9)),
    ],
    "ghb_flows.csv": [
        ("Estuary2", "GHB", "Estuary-L2"),
        ("Estuary3", "GHB", "Estuary-L3"),
    ],
}
```

The observations can then be added to the ghb package using the obs attribute's initialize method. The observation package's file name, digits, and print_input options, along with the continuous block data are set in the initialize method.

```
[13]: # initialize obs package
ghb.obs.initialize(
    filename="child_pkgs_test.ghb.obs",
    digits=9,
    print_input=True,
    continuous=ghb_obs,
)
```

```
[14]: try:
    temp_dir.cleanup()
except PermissionError:
    # can occur on windows: https://docs.python.org/3/library/tempfile.html#tempfile.
    ↪ TemporaryDirectory
    pass
```


5.2.3 MODFLOW 6: Time Series Packages

Introduction to Time Series

Time series can be set for any package through the `package.ts` object, and each `package.ts` object has several attributes that can be set:

Attribute	Type	Description
<code>package.ts.filename</code>	str	Name of time series file to create. The default is <code>packagename + ".ts"</code> .
<code>package.ts.timeseries</code>	recarray	Array containing the time series information
<code>package.ts.time_series_namere</code>	str or list	List of names of the time series data columns. Default is to use names from <code>timeseries.dtype.names[1:]</code> .
<code>package.ts.interpolation_metho</code>	str	Interpolation method. Must be only one time series record. If there are multiple time series records, then the <code>methods</code> attribute must be used.
<code>package.ts.interpolation_metho</code>	float	Scale factor to multiply the time series data column. Can only be used if there is one time series data column.
<code>package.ts.sfacrecord_single</code>	float	Scale factor to multiply the time series data column. Can only be used if there is one time series data column.
<code>package.ts.sfacrecord</code>	list (of floats)	Scale factors to multiply the time series data columns.

The following code sets up a simulation used in the time series examples.

```
[1]: # package import
import os
from pathlib import Path
from tempfile import TemporaryDirectory

[2]: import numpy as np

[3]: import flopy

[4]: # set up where simulation workspace will be stored
temp_dir = TemporaryDirectory()
workspace = temp_dir.name
name = "tutorial03_mf6_data"

[5]: # create the flopy simulation and tdis objects
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws=workspace
)
tdis_rc = [(1.0, 1, 1.0), (10.0, 5, 1.0), (10.0, 5, 1.0), (10.0, 1, 1.0)]
tdis_package = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim, time_units="DAYS", nper=4, perioddata=tdis_rc
)
# create the Flopy groundwater flow (gwf) model object
model_name_file = f"{name}.nam"
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, model_name_file=model_name_file)
# create the flopy iterative model solver (ims) package object
ims = flopy.mf6.modflow.mfims.ModflowIms(sim, pname="ims", complexity="SIMPLE")
```

(continues on next page)

(continued from previous page)

```
# create the discretization package
bot = np.linspace(-3.0, -50.0 / 3.0, 3)
delrow = delcol = 4.0
dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf,
    pname="dis",
    nogrb=True,
    nlay=3,
    nrow=101,
    ncol=101,
    delr=delrow,
    delc=delcol,
    top=0.0,
    botm=bot,
)
# create the initial condition (ic) and node property flow (npf) packages
ic_package = flopy.mf6.modflow.mfgwfic.ModflowGwfic(gwf, strt=50.0)
npf_package = flopy.mf6.modflow.mfgwnpf.ModflowGwnpf(
    gwf,
    save_flows=True,
    icelltype=[1, 0, 0],
    k=[5.0, 0.1, 4.0],
    k33=[0.5, 0.005, 0.1],
)
```

Time Series Example 1

One way to construct a time series is to pass the time series data to the parent package constructor.

This example uses time series data in a GHB package. First the GHB `stress_period_data` is built.

```
[6]: # build ghb stress period data
ghb_spd_ts = {}
ghb_period = []
for layer, cond in zip(range(1, 3), [15.0, 1500.0]):
    for row in range(0, 15):
        ghb_period.append((layer, row, 9), "tides", cond, "Estuary-L2"))
ghb_spd_ts[0] = ghb_period
```

Next the time series data is built. The time series data is constructed as a list of tuples, with each tuple containing a time and the value (or values) at that time. The time series data is put in a dictionary along with additional time series information including `filename`, `time_series_namerecord`, `interpolation_methodrecord`, and `sfacrecord`.

```
[7]: # build ts data
ts_data = []
for n in range(0, 365):
    time = float(n / 11.73)
    val = float(n / 60.0)
    ts_data.append((time, val))
ts_dict = {
    "filename": "tides.ts",
    "time_series_namerecord": "tide",
```

(continues on next page)

(continued from previous page)

```

    "timeseries": ts_data,
    "interpolation_methodrecord": "linearend",
    "sfacrecord": 1.1,
}

```

The GHB package is then constructed, passing the time series data into the `timeseries` parameter.

```

[8]: # build ghb package
ghb = flopy.mf6.modflow.mfgwfghb.ModflowGwfghb(
    gwf,
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    timeseries=ts_dict,
    pname="ghb",
    maxbound=30,
    stress_period_data=ghb_spd_ts,
)

```

```

WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".

```

Time series attributes, like `time_series_namerecord`, can be modified using the `ghb.ts` object.

```

[9]: # set required time series attributes
ghb.ts.time_series_namerecord = "tides"

```

```

[10]: # clean up for next example
gwf.remove_package("ghb")

```

Time Series Example 2

Another way to construct a time series is to initialize the time series through the `ghb.ts.initialize` method. Additional time series can then be appended using the `append_package` method.

First the GHB stress period data is built.

```

[11]: # build ghb stress period data
ghb_spd_ts = {}
ghb_period = []
for layer, cond in zip(range(1, 3), [15.0, 1500.0]):
    for row in range(0, 15):
        if row < 10:
            ghb_period.append(((layer, row, 9), "tides", cond, "Estuary-L2"))
        else:
            ghb_period.append(((layer, row, 9), "w1", cond, "Estuary-L2"))
ghb_spd_ts[0] = ghb_period

```

Next the time series data is built. The time series data is constructed as a list of tuples, with each tuple containing a time and the value (or values) at that time.

```
[12]: # build ts data
ts_data = []
for n in range(0, 365):
    time = float(n / 11.73)
    val = float(n / 60.0)
    ts_data.append((time, val))
ts_data2 = []
for n in range(0, 365):
    time = float(1.0 + (n / 12.01))
    val = float(n / 60.0)
    ts_data2.append((time, val))
ts_data3 = []
for n in range(0, 365):
    time = float(10.0 + (n / 12.01))
    val = float(n / 60.0)
    ts_data3.append((time, val))
```

A ghb package is constructed without the time series data

```
[13]: # build ghb package
ghb = flopy.mf6.modflow.mfgwfghb.ModflowGwfghb(
    gwf,
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    pname="ghb",
    maxbound=30,
    stress_period_data=ghb_spd_ts,
)
```

The first time series data are added by calling the initialize method from the `ghb.ts` object. The times series package's file name, name record, method record, and sfac record, along with the time series data are set in the initialize method.

```
[14]: # initialize first time series
ghb.ts.initialize(
    filename="tides.ts",
    timeseries=ts_data,
    time_series_namerecord="tides",
    interpolation_methodrecord="linearend",
    sfacrecord=1.1,
)
```

```
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
```

The remaining time series data are added using the `append_package` method. The `append_package` method takes the same parameters as the initialize method.

```
[15]: # append additional time series
ghb.ts.append_package(
    filename="wls.ts",
```

(continues on next page)

(continued from previous page)

```

    timeseries=ts_data2,
    time_series_namerecord="w1",
    interpolation_methodrecord="stepwise",
    sfacrecord=1.2,
)
# append additional time series
ghb.ts.append_package(
    filename="wls2.ts",
    timeseries=ts_data3,
    time_series_namerecord="w12",
    interpolation_methodrecord="stepwise",
    sfacrecord=1.3,
)

WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".

```

Information can be retrieved from time series packages using the `ts` attribute of its parent package. Below the interpolation method record for each of the three time series are retrieved.

```

[16]: print(
    "{} is using {} interpolation".format(
        ghb.ts[0].filename,
        ghb.ts[0].interpolation_methodrecord.get_data()[0][0],
    )
)
print(
    "{} is using {} interpolation".format(
        ghb.ts[1].filename,
        ghb.ts[1].interpolation_methodrecord.get_data()[0][0],
    )
)
print(
    "{} is using {} interpolation".format(
        ghb.ts[2].filename,
        ghb.ts[2].interpolation_methodrecord.get_data()[0][0],
    )
)

```

```

tides.ts is using linearend interpolation
wls.ts is using stepwise interpolation
wls2.ts is using stepwise interpolation

```

```

[17]: try:
    temp_dir.cleanup()
except PermissionError:
    # can occur on windows: https://docs.python.org/3/library/tempfile.html#tempfile.

```

(continues on next page)

(continued from previous page)

```
↪ TemporaryDirectory
pass
```

5.2.4 MODFLOW 6: Time Array Series Packages

Introduction to Time Array Series

Time array series can be set for any package through the `package.tas` object, and each `package.tas` object has several attributes that can be set:

Attribute	Type	Description
<code>package.tas.filename</code>	str	Name of time series file to create. The default is packagename + “.tas”.
<code>package.tas.tas_array</code>	{double:double}	Array containing the time array series information for specific times.
<code>package.tas.time_series_name</code>	str	Name by which a package references a particular time-array series. The name must be unique among all time-array series used in a package.
<code>package.tas.interpolation_methods</code>	list (of strings)	List of interpolation methods to use for time array series. Method must be either “stepwise” or “linear”.
<code>package.tas.scale_factor</code>	float	Scale factor to multiply the time array series data column. Can only be used if there is one time series data column.

The following code sets up a simulation used in the time array series examples.

```
[1]: # package import
import os
from pathlib import Path
from tempfile import TemporaryDirectory
```

```
[2]: import numpy as np
```

```
[3]: import flopy
```

```
[4]: # set up where simulation workspace will be stored
temp_dir = TemporaryDirectory()
workspace = temp_dir.name
name = "tutorial04_mf6_data"
```

```
[5]: # create the Flopy simulation and tdis objects
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws=workspace
)
tdis_rc = [(1.0, 1, 1.0), (10.0, 5, 1.0), (10.0, 5, 1.0), (10.0, 1, 1.0)]
tdis_package = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim, time_units="DAYS", nper=4, perioddata=tdis_rc
)
# create the Flopy groundwater flow (gwf) model object
model_name_file = f"{name}.nam"
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, model_name_file=model_name_file)
```

(continues on next page)

(continued from previous page)

```

# create the flopy iterative model solver (ims) package object
ims = flopy.mf6.modflow.mfims.ModflowIms(sim, pname="ims", complexity="SIMPLE")
# create the discretization package
bot = np.linspace(-3.0, -50.0 / 3.0, 3)
delrow = delcol = 4.0
dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf,
    pname="dis",
    nogrb=True,
    nlay=3,
    nrow=101,
    ncol=101,
    delr=delrow,
    delc=delcol,
    top=0.0,
    botm=bot,
)
# create the initial condition (ic) and node property flow (npf) packages
ic_package = flopy.mf6.modflow.mfgwfic.ModflowGwfic(gwf, strt=50.0)
npf_package = flopy.mf6.modflow.mfgwfnpf.ModflowGwfnpf(
    gwf,
    save_flows=True,
    icelltype=[1, 0, 0],
    k=[5.0, 0.1, 4.0],
    k33=[0.5, 0.005, 0.1],
)

```

Time Array Series Example 1

Time array series data can be passed into the `timearrayseries` parameter on construction of any package that supports time array series. This example uses the `timearrayseries` parameter to create a time array series and then uses the package's `tas` property to finish the time array series setup.

This example uses time array series data in a RCHA package. The time array series is built as a dictionary with the times as the keys and the recharge values as the values.

```
[6]: tas = {0.0: 0.0000002, 200.0: 0.0000001}
```

The time array series data is then passed into the `timearrayseries` parameter when the RCHA package is constructed.

```
[7]: rcha = flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha(
    gwf, timearrayseries=tas, recharge="TIMEARRAYSERIES rchararray_1"
)
```

```
WARNING: Time array series name rchararray_1 not found in any time series file
```

Time array series attributes can be set by access the time array series package object through the `rcha.tas` attribute.

```
[8]: # finish defining the time array series properties
rcha.tas.time_series_namerecord = "rchararray_1"
rcha.tas.interpolation_methodrecord = "LINEAR"
```

The simulation is then written to files and run.

```
[9]: sim.write_simulation()
sim.run_simulation()

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims...
  writing model tutorial04_mf6_data...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
    writing package rcha_0...
    writing package tas_0...
FloPy is using the following executable to run the model: ../../home/runner/.local/bin/
↪modflow/mf6

                                MODFLOW 6
                U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
                        VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
                        Build 20220726_0000000

This software has been approved for release by the U.S. Geological
Survey (USGS). Although the software has been subjected to rigorous
review, the USGS reserves the right to update the software as needed
pursuant to further analysis and review. No warranty, expressed or
implied, is made by the USGS or the U.S. Government as to the
functionality of the software and related material nor shall the
fact of release constitute any such warranty. Furthermore, the
software is released on condition that neither the USGS nor the U.S.
Government shall be held liable for any damages resulting from its
authorized or unauthorized use. Also refer to the USGS Water
Resources Software User Rights Notice for complete use, copyright,
and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:43

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

ERROR REPORT:

  1. Invalid model name: TUTORIAL04_MF6_DATA
  2. Name length of 19 exceeds maximum length of 16

[9]: (False, [])

[10]: # clean up for next example
gwf.remove_package("rcha")
```


Time Array Series Example 2

A time array series can be added after a package is created by calling the package's `tas` attribute's `initialize` method. Initialize allows you to define all time array series attributes including file name, the time array series data, name record, and method record.

First a recharge package is built.

```
[11]: # create recharge package with recharge pointing to a time array series
# not yet defined. FloPy will generate a warning that there is not yet a
# time series name record for rchararray_1
rcha = flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha(
    gwf, recharge="TIMEARRAYSERIES rchararray_1"
)

WARNING: Time array series name rchararray_1 not found in any time series file
```

Then a time array series dictionary is created as done in example 1.

```
[12]: tas = {0.0: 0.0000002, 200.0: 0.0000001}
```

The time array series data are added by calling the `initialize` method from the 'RCHA' package's `tas` attribute. The time array series file name, name record, and method record, along with the time series data are set in the `initialize` method.

```
[13]: # initialize the time array series
rcha.tas.initialize(
    filename="method2.tas",
    tas_array=tas,
    time_series_namerecord="rchararray_1",
    interpolation_methodrecord="LINEAR",
)
```

```
[14]: try:
    temp_dir.cleanup()
except PermissionError:
    # can occur on windows: https://docs.python.org/3/library/tempfile.html#tempfile.
    ↪ TemporaryDirectory
    pass
```

5.2.5 MODFLOW 6: Working with MODFLOW Scalar Data

This tutorial shows how to view, access, and change the underlying data variables for MODFLOW 6 objects in FloPy. Interaction with a FloPy MODFLOW 6 model is different from other models, such as MODFLOW-2005, MT3D, and SEAWAT, for example.

FloPy stores model data in data objects (`MFDdataArray`, `MFDdataList`, `MFDdataScalar` objects) that are accessible from packages. Data can be added to a package by using the appropriate parameters when the package is constructed and through package attributes.

The MODFLOW 6 simulation structure is arranged in the following generalized way:

```
Simulation --> Package --> DATA

Simulation --> Model --> Package (--> Package) --> DATA
```

This tutorial focuses on MODFLOW Data that is a single integer or string, or consists of boolean flag(s). These data are stored by FloPy in a MFScalar object and are referred to as MODFLOW scalar data.

Introduction to MODFLOW Scalar Data

MODFLOW single integer, strings, or boolean flag(s) are stored by FloPy as scalar data in MFScalar objects. The different types of scalar data are described below.

1. Single integer values. Examples include `nrow`, `ncol`, `nlay`, and `nper`.
2. Single string values. Examples include `time_units` and `length_units`.
3. Boolean flags. These can be found in the options section of most packages. These include `perched`, `nogrb`, `print_input`, and `save_flows`.
4. Boolean flags with an additional optional flag. These include `newton under_relaxation` and `xt3d rhs`.

In the following all four types of scalar data will be added to a model. Before adding data to your model first create a simulation (MFSimulation) and a model (MFModel) object in FloPy.

```
[1]: # package import
import os
from pathlib import Path
from tempfile import TemporaryDirectory
```

```
[2]: import numpy as np
```

```
[3]: import flopy
```

```
[4]: # set up where simulation workspace will be stored
temp_dir = TemporaryDirectory()
workspace = temp_dir.name
name = "tutorial05_mf6_data"
```

```
[5]: # create the flopy simulation object
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws=workspace
)
```

```
[6]: # create the flopy groundwater flow (gwf) model object
model_nam_file = f"{name}.nam"
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, model_nam_file=model_nam_file)
# create the flopy iterative model solver (ims) package object
# (both pname and complexity are scalar data)
ims = flopy.mf6.modflow.mfims.ModflowIms(sim, pname="ims", complexity="SIMPLE")
```

Adding MODFLOW Single Integer and String Values

Single integer and string values can be assigned on construction of the MFScalar data object, and can be assigned or changed after construction.

Below, a TDIS package is constructed with the `time_units` and `nper` parameters being assigned “DAYS” and “2”, respectively.

```
[7]: # create the FloPy temporal discretization object
tdis = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim,
    pname="tdis",
    time_units="DAYS",
    nper=2,
    perioddata=[(1.0, 1, 1.0), (1.0, 1, 1.0)],
)
```

Next, `time_units` is reassigned a value after construction by using TDIS’s `time_units` attribute.

```
[8]: tdis.time_units = "MONTHS"
```

Setting MODFLOW Boolean Flags

Boolean flags can be assigned a True or False value. In the example below `nogrb` is assigned a value of True and then changed to false.

For this example, first some values are first defined for the discretization package

```
[9]: nlay = 3
h = 50.0
length = 400.0
n = 10
bot = np.linspace(-h / nlay, -h, nlay)
delrow = delcol = length / (n - 1)
```

Below the discretization package is created. The MODFLOW `nogrb` option assigned a value of True, switching this option on.

```
[10]: dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf,
    pname="dis",
    nogrb=True,
    nlay=nlay,
    nrow=n,
    ncol=n,
    delr=delrow,
    delc=delcol,
    top=0.0,
    botm=bot,
)
```

The `nogrb` option is then switched off by setting the DIS package’s `nogrb` attribute to False.

```
[11]: dis.nogrb = False
```

Boolean flags with an additional optional flag can either be specified by:

1. Specifying the entire line as it would be displayed in the package file as a string (`xt3doptions="xt3d rhs"`)
2. Specifying each flag name in a list (`xt3doptions=["xt3d", "rhs"]`)

To turn off both flags use an empty string (`xt3doptions=""`) or an empty list (`xt3doptions=[]`).

First, an NPF package is created. `xt3doptions` can either be turned on or off, and if it is on `rhs` can optionally be turned on. `xt3doptions` is set to the string “`xt3d rhs`”, turning both options on.

```
[12]: # create the node property flow package with xt3doptions as single
npf = flopy.mf6.modflow.mfgwnpf.ModflowGwnpf(
    gwf,
    rewet_record="REWET WETFCT 1.0 IWETIT 1 IHDWET 0",
    pname="npf",
    icelltype=1,
    k=1.0,
    save_flows=True,
    xt3doptions="xt3d rhs",
)

<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7f494a067ac0>
<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7f494a067af0>
```

Next, the `rhs` option is turned off by setting `xt3doptions` to the string “`xt3d`”.

```
[13]: npf.xt3doptions = "xt3d"
```

Finally, both `xt3d` and `rhs` are turned off by setting `xt3doptions` to an empty string.

```
[14]: npf.xt3doptions = ""
```

Retrieving MODFLOW Scalar Data

MODFLOW scalar data can be retrieved with `get_data`, `repr/str`, or `get_file_entry`.

Re-trieval Method	Description
<code>get_data</code>	Returns scalar value
<code>repr/str</code>	Returns string with a header describing how data is stored (internal, external) with a string representation of the data on the next line
<code>get_file_e</code>	Returns string with the scalar keyword (if any) followed by a space and a string representation of the scalar value (if any). This is the format used by the MODFLOW-6 package file. This is the format used by the MODFLOW-6 package file.

The IMS package’s `complexity` option and the NPF package’s `xt3doptions` are printed below using the different data retrieval methods highlighted above.

First the complexity data is printed using the `get_data` method.

```
[15]: print(ims.complexity.get_data())
```

```
simple
```

The xt3doptions data can also be printed with `get_data`.

```
[16]: print(npf.xt3doptions.get_data())
[]
```

The complexity data is then printed with `repr`

```
[17]: print(repr(ims.complexity))
{internal}
('simple')
```

The xt3doptions data is printed with `repr`

```
[18]: print(str(npf.xt3doptions))
{internal}
([])
```

The complexity data is printed as it would appear in a MODFLOW 6 file using the `get_file_entry` method.

```
[19]: print(ims.complexity.get_file_entry())
COMPLEXITY  simple
```

The xt3doptions data is printed as it would appear in a MODFLOW 6 file using the `get_file_entry` method.

```
[20]: print(npf.xt3doptions.get_file_entry())
```

```
[21]: try:
        temp_dir.cleanup()
    except PermissionError:
        # can occur on windows: https://docs.python.org/3/library/tempfile.html#tempfile.
        ↳ TemporaryDirectory
        pass
```

5.2.6 MODFLOW 6: Working with MODFLOW List Data.

This tutorial shows how to view, access, and change the underlying data variables for MODFLOW 6 objects in FloPy. Interaction with a FloPy MODFLOW 6 model is different from other models, such as MODFLOW-2005, MT3D, and SEAWAT, for example.

FloPy stores model data in data objects (`MFDdataArray`, `MFDdataList`, `MFDdataScalar` objects) that are accessible from packages. Data can be added to a package by using the appropriate parameters when the package is constructed and through package attributes.

The MODFLOW 6 simulation structure is arranged in the following generalized way:

```
Simulation --> Package --> DATA
```

```
Simulation --> Model --> Package (--> Package) --> DATA
```

This tutorial focuses on MODFLOW Data from the `PackageData`, `ConnectionData`, `StressPeriodData`, and other similar blocks. These blocks contain data with columns, data that fits into a numpy recarray, pandas data frame, or a spreadsheet with column headers. These data are stored by FloPy in a `MFList` or `MFTransientList` object and are referred to as MODFLOW list data.

Introduction to MODFLOW List Data

MODFLOW contains list data that can be conveniently stored in a numpy recarray or a pandas dataframe. These data are either a single or multiple row of data, with each column containing the same data type.

Some MODFLOW list data only contains a single row, like the OC package's `head print_format` option and the NPF package's `rewet_record`. Other MODFLOW list data can contain multiple rows, like the MAW package's `packagedata` and `connectiondata`. FloPy stores both single row and multiple row list data in `MFList` objects.

MODFLOW stress period data can contain lists of data for one or more stress periods. FloPy stores stress period list data in `MFTransientList` objects. Note that not all MODFLOW stress period data is “list” data that fits neatly in a recarray or a panda's dataframe. Some packages including RCH and EVT have a `READASARRAYS` option that allows stress period data to be inputted as an array. When `READASARRAYS` is selected FloPy stores stress period array data in an `MFTransientArray` object (see tutorial 8).

Examples of using FloPy to store, update, and retrieve different types of MODFLOW list data are given below. The examples start by first creating a simulation (`MFSimulation`) and a model (`MFModel`) object in FloPy.

```
[1]: # package import
import os
from pathlib import Path
from tempfile import TemporaryDirectory
```

```
[2]: import numpy as np
```

```
[3]: import flopy
```

```
[4]: # set up where simulation workspace will be stored
temp_dir = TemporaryDirectory()
workspace = temp_dir.name
name = "tutorial06_mf6_data"
```

```
[5]: # create the Flopy simulation and tdis objects
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws=workspace
)
tdis = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim,
    pname="tdis",
    time_units="DAYS",
    nper=2,
    perioddata=[(1.0, 1, 1.0), (1.0, 1, 1.0)],
)
```

(continues on next page)

(continued from previous page)

```

# create the Flopy groundwater flow (gwf) model object
model_nam_file = f"{name}.nam"
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, model_nam_file=model_nam_file)
# create the flopy iterative model solver (ims) package object
ims = flopy.mf6.modflow.mfims.ModflowIms(sim, pname="ims", complexity="SIMPLE")
# create the discretization package
bot = np.linspace(-50.0 / 3.0, -3.0, 3)
delrow = delcol = 4.0
dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf,
    pname="dis",
    nogrb=True,
    nlay=3,
    nrow=10,
    ncol=10,
    delr=delrow,
    delc=delcol,
    top=0.0,
    botm=bot,
)

```

Adding MODFLOW Package Data, Connection Data, and Option Lists

MODFLOW Package data, connection data, and option lists are stored by FloPy as numpy recarrays. FloPy does accept numpy recarrays as input, but does has other supported formats discussed below.

MODFLOW option lists that only contain a single row or data can be either specified by:

1. Specifying a string containing the entire line as it would be displayed in the package file (`rewet_record="REWET WETFCT 1.0 IWETIT 1 IHDWET 0"`)
2. Specifying the data in a tuple within a list (`rewet_record=[("WETFCT", 1.0, "IWETIT", 1, "IHDWET", 0)]`)

In the example below the npf package is created setting the `rewet_record` option to a string of text as would be typed into the package file.

```

[6]: npf = flopy.mf6.modflow.mfgwfnpf.ModflowGwfnpf(
    gwf,
    rewet_record="REWET WETFCT 1.0 IWETIT 1 IHDWET 0",
    pname="npf",
    icelltype=1,
    k=1.0,
    save_flows=True,
    xt3doptions="xt3d rhs",
)

<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7fe55fe84100>
<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7fe55fe84130>

```

`rewet_record` is then set using the npf package's `rewet_record` property. This time 'rewet_record' is defined using a tuple within a list.

```
[7]: npf.rewet_record = [("WETFCT", 1.1, "IWETIT", 0, "IHDWET", 1)]
```

MODFLOW multirow lists, like package data and connection data, can be specified:

1. As a list of tuples where each tuple represents a row in the list (stress_period_data = [(1, 2, 3), 20.0), ((1, 7, 3), 25.0)])
2. As a numpy recarray. Building a numpy recarray is more complicated and is beyond the scope of this guide.

In the example below the chd package is created, setting stress_period_data as a list of tuples.

We build the chd package using an array of tuples for stress_period_data stress_period_data = [(first_chd_cell, head), (second_chd_cell, head), ...] Note that the cellid information (layer, row, column) is encapsulated in a tuple.

```
[8]: stress_period_data = [((1, 8, 8), 100.0), ((1, 9, 9), 105.0)]
# build chd package
chd = flopy.mf6.modflow.mfgwfchd.ModflowGwfchd(
    gwf,
    pname="chd",
    maxbound=len(stress_period_data),
    stress_period_data=stress_period_data,
    save_flows=True,
)
```

Adding Stress Period List Data

MODFLOW stress period data is stored by FloPy as a dictionary of numpy recarrays, where each dictionary key is a zero-based stress period and each dictionary value is a recarray containing the stress period data for that stress period. FloPy keeps this stress period data in a MFTransientList object and this data type is referred to as a transient list.

FloPy accepts stress period data as a dictionary of numpy recarrays, but also supports replacing the recarrays with lists of tuples discussed above. Stress period data spanning multiple stress periods must be specified as a dictionary of lists where the dictionary key is the stress period expressed as a zero-based integer.

The example below creates stress_period_data for the wel package with the first stress period containing a single well and the second stress period empty. When empty stress period data is entered FloPy writes an empty stress period block to the package file.

First we create wel package with stress_period_data dictionary keys as zero-based integers so key "0" is stress period 1

```
[9]: stress_period_data = {
    0: [((2, 3, 1), -25.0)], # stress period 1 well data
    1: [],
} # stress period 2 well data is empty
```

Then, using the dictionary created above, we build the wel package.

```
[10]: wel = flopy.mf6.ModflowGwfwel(
    gwf,
    print_input=True,
    print_flows=True,
    stress_period_data=stress_period_data,
    save_flows=False,
    pname="WEL-1",
)
```


Retrieving MODFLOW Package Data, Connection Data, and Option Lists

MODFLOW package data, connection data, and option lists can be retrieved with `get_data`, `array`, `repr/str`, or `get_file_entry`.

Retrieval Method	Description
<code>get_data</code>	Returns recarray
<code>array</code>	Return recarray
<code>repr/str</code>	Returns string with storage information followed by recarray's <code>repr/str</code>
<code>get_file_entry</code>	Returns string containing data formatted for the MODFLOW-6 package file. Certain zero-based numbers, like layer, row, column, are converted to one-based numbers.

The NPF package's `rewet_record` is printed below using the different data retrieval methods highlighted above.

First we use the `get_data` method to get the `rewet_record` as a recarray.

```
[11]: print(npf.rewet_record.get_data())
[('WETFCT', 1.1, 'IWETIT', 0, 'IHDWET', 1)]
```

Next we use the `array` method, which also returns a recarray.

```
[12]: print(npf.rewet_record.array)
[('WETFCT', 1.1, 'IWETIT', 0, 'IHDWET', 1)]
```

Then we use `repr` to print a string representation of `rewet_record`.

```
[13]: print(repr(npf.rewet_record))
{internal}
(rec.array([('WETFCT', 1.1, 'IWETIT', 0, 'IHDWET', 1)],
          dtype=[('wetfct_label', '0'), ('wetfct', '<f8'), ('iwetit_label', '0'), (
→ 'iwetit', '<i8'), ('ihdwet_label', '0'), ('ihdwet', '<i8')]))
```

Using `str` prints a similar string representation of `rewet_record`.

```
[14]: print(str(npf.rewet_record))
{internal}
[('WETFCT', 1.1, 'IWETIT', 0, 'IHDWET', 1)]
```

Last, using the `get_file_entry` method the data is printed as it would appear in a MODFLOW 6 file.

```
[15]: print(npf.rewet_record.get_file_entry())
REWET WETFCT      1.10000000 IWETIT 0 IHDWET 1
```

Retrieving MODFLOW Stress Period List Data

Stress period data can be retrieved with `get_data`, `array`, `repr/str`, or `get_file_entry`.

Retrieval Method	Description
<code>get_data</code>	Returns dictionary of recarrays
<code>array</code>	Return single recarray for all stress periods
<code>repr/str</code>	Returns string with storage information followed by recarray <code>repr/str</code> for each recarray
<code>get_file_entry(key)</code>	Returns string containing data formatted for the MODFLOW-6 package file for the stress period specified by key

The WEL package's `stress_period_data` is printed below using the different data retrieval methods highlighted above.

First we use the `get_data` method to get the stress period data as a dictionary of recarrays.

```
[16]: print(wel.stress_period_data.get_data())
{0: rec.array([(2, 3, 1), -25.]),
      dtype=[('cellid', '0'), ('q', '<f8')])}
```

Next we use the `array` attribute to get the stress period data as a single recarray.

```
[17]: print(wel.stress_period_data.array)
[rec.array([(2, 3, 1), -25.]),
      dtype=[('cellid', '0'), ('q', '<f8')]), None]
```

`repr` can be used to generate a string representation of stress period data.

```
[18]: print(repr(wel.stress_period_data))
{internal}
(rec.array([(2, 3, 1), -25.]),
      dtype=[('cellid', '0'), ('q', '<f8')])}
```

`str` produces a similar string representation of stress period data.

```
[19]: print(str(wel.stress_period_data))
{internal}
([(2, 3, 1), -25.)]
```

The `get_file_entry` method prints the stress period data as it would appear in a MODFLOW 6 file.

```
[20]: print(wel.stress_period_data.get_file_entry(0))
3 4 2      -25.00000000
```

```
[21]: try:
      temp_dir.cleanup()
```

(continues on next page)

(continued from previous page)

```
except PermissionError:
    # can occur on windows: https://docs.python.org/3/library/tempfile.html#tempfile.
    ↪ TemporaryDirectory
    pass
```

5.2.7 MODFLOW 6: Working with MODFLOW Grid Array Data

This tutorial shows how to view, access, and change the underlying data variables for MODFLOW 6 objects in FloPy. Interaction with a FloPy MODFLOW 6 model is different from other models, such as MODFLOW-2005, MT3D, and SEAWAT, for example.

FloPy stores model data in data objects (MFDataArray, MFDataList, MFDataScalar objects) that are accessible from packages. Data can be added to a package by using the appropriate parameters when the package is constructed and through package attributes.

The MODFLOW 6 simulation structure is arranged in the following generalized way:

```
Simulation --> Package --> DATA

Simulation --> Model --> Package (--> Package) --> DATA
```

This tutorial focuses on MODFLOW grid array data from the GridData and other similar blocks. These blocks contain data in a one or more dimensional array format organized by dimensions, which can include layer, row, column, and stress period. These data are stored by FloPy in a MFArray or MFTransientArray object and are referred to as array data.

Introduction to MODFLOW Array Data

MODFLOW array data use the MFArray or MFTransientArray FloPy classes and are stored in numpy ndarrays. Most MODFLOW array data are two (row, column) or three (layer, row, column) dimensional and represent data on the model grid. Other MODFLOW array data contain data by stress period. The following list summarizes the different types of MODFLOW array data.

- Time-invariant multi-dimensional array data. This includes:
 1. One and two dimensional arrays that do not have a layer dimension. Examples include `top`, `delc`, and `delr`.
 2. Three dimensional arrays that can contain a layer dimension. Examples include `botm`, `idomain`, and `k`.
- Transient arrays that can change with time and therefore contain arrays of data for one or more stress periods. Examples include `irch` and `recharge` in the RCHA package.

In the example below a three dimensional ndarray is constructed for the DIS package's `botm` array. First, the a simulation and groundwater-flow model are set up.

```
[1]: # package import
import os
from pathlib import Path
from tempfile import TemporaryDirectory
```

```
[2]: import numpy as np
```

```
[3]: import flopy
```

```
[4]: # set up where simulation workspace will be stored
temp_dir = TemporaryDirectory()
workspace = temp_dir.name
name = "tutorial07_mf6_data"
```

```
[5]: # create the FloPy simulation and tdis objects
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws=workspace
)
tdis = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim,
    pname="tdis",
    time_units="DAYS",
    nper=2,
    perioddata=[(1.0, 1, 1.0), (1.0, 1, 1.0)],
)
# create the FloPy groundwater flow (gwf) model object
model_name_file = f"{name}.nam"
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, model_name_file=model_name_file)
# create the flopy iterative model solver (ims) package object
ims = flopy.mf6.modflow.mfims.ModflowIms(sim, pname="ims", complexity="SIMPLE")
```

Then a three-dimensional ndarray of floating point values is created using numpy's linspace method.

```
[6]: bot = np.linspace(-50.0 / 3.0, -3.0, 3)
delrow = delcol = 4.0
```

The DIS package is then created passing the three-dimensional array to the botm parameter. The botm array defines the model's cell bottom elevations.

```
[7]: dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf,
    pname="dis",
    nogrb=True,
    nlay=3,
    nrow=10,
    ncol=10,
    delr=delrow,
    delc=delcol,
    top=0.0,
    botm=bot,
)
```

Adding MODFLOW Grid Array Data

MODFLOW grid array data, like the data found in the NPF package's GridData block, can be specified as:

1. A constant value
2. A n-dimensional list
3. A numpy ndarray

Additionally, layered grid data (generally arrays with a layer dimension) can be specified by layer.

In the example below `icelltype` is specified as constants by layer, `k` is specified as a numpy ndarray, `k22` is specified as an array by layer, and `k33` is specified as a constant.

First `k` is set up as a 3 layer, by 10 row, by 10 column array with all values set to 10.0 using numpy's `full` method.

```
[8]: k = np.full((3, 10, 10), 10.0)
```

Next `k22` is set up as a three dimensional list of nested lists. This option can be useful for those that are familiar with python lists but are not familiar with the numpy library.

```
[9]: k22_row = []
    for row in range(0, 10):
        k22_row.append(8.0)
    k22_layer = []
    for col in range(0, 10):
        k22_layer.append(k22_row)
    k22 = [k22_layer, k22_layer, k22_layer]
```

`K33` is set up as a single constant value. Whenever an array has all the same values the easiest and most efficient way to set it up is as a constant value. Constant values also take less space to store.

```
[10]: k33 = 1.0
```

The `k`, `k22`, and `k33` values defined above are then passed in on construction of the npf package.

```
[11]: npf = flopy.mf6.ModflowGwfnpf(
    gwf,
    pname="npf",
    save_flows=True,
    icelltype=[1, 1, 1],
    k=k,
    k22=k22,
    k33=k33,
    xt3doptions="xt3d rhs",
    rewet_record="REWET WETFCT 1.0 IWETIT 1 IHDWET 0",
)

<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7f6288cb1730>
<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7f6288cb1760>
```

Layered Data

When we look at what will be written to the npf input file, we see that the entire `npf.k22` array is written as one long array with the number of values equal to `nlay * nrow * ncol`. And this whole-array specification may be of use in some cases. Often times, however, it is easier to work with each layer separately. An `MFArrray` object, such as `npf.k22` can be converted to a layered array as follows.

```
[12]: npf.k22.make_layered()
```

By changing `npf.k22` to layered, we are then able to manage each layer separately. Before doing so, however, we need to pass in data that can be separated into three layers. An array of the correct size is one option.

```
[13]: shp = npf.k22.array.shape
a = np.arange(shp[0] * shp[1] * shp[2]).reshape(shp)
npf.k22.set_data(a)
```

Now that `npf.k22` has been set to be layered, if we print information about it, we see that each layer is stored separately, however, `npf.k22.array` will still return a full three-dimensional array.

```
[14]: print(type(npf.k22))
print(npf.k22)

<class 'flopy.mf6.data.mfdataarray.MFArrray'>
Layer_1{internal}
([[ 0  1  2  3  4  5  6  7  8  9]
 [10 11 12 13 14 15 16 17 18 19]
 [20 21 22 23 24 25 26 27 28 29]
 [30 31 32 33 34 35 36 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]
 [50 51 52 53 54 55 56 57 58 59]
 [60 61 62 63 64 65 66 67 68 69]
 [70 71 72 73 74 75 76 77 78 79]
 [80 81 82 83 84 85 86 87 88 89]
 [90 91 92 93 94 95 96 97 98 99]])
Layer_2{internal}
([[100 101 102 103 104 105 106 107 108 109]
 [110 111 112 113 114 115 116 117 118 119]
 [120 121 122 123 124 125 126 127 128 129]
 [130 131 132 133 134 135 136 137 138 139]
 [140 141 142 143 144 145 146 147 148 149]
 [150 151 152 153 154 155 156 157 158 159]
 [160 161 162 163 164 165 166 167 168 169]
 [170 171 172 173 174 175 176 177 178 179]
 [180 181 182 183 184 185 186 187 188 189]
 [190 191 192 193 194 195 196 197 198 199]])
Layer_3{internal}
([[200 201 202 203 204 205 206 207 208 209]
 [210 211 212 213 214 215 216 217 218 219]
 [220 221 222 223 224 225 226 227 228 229]
 [230 231 232 233 234 235 236 237 238 239]
 [240 241 242 243 244 245 246 247 248 249]
 [250 251 252 253 254 255 256 257 258 259]
 [260 261 262 263 264 265 266 267 268 269]
 [270 271 272 273 274 275 276 277 278 279])
```

(continues on next page)

(continued from previous page)

```
[280 281 282 283 284 285 286 287 288 289]
[290 291 292 293 294 295 296 297 298 299]])
```

We also see that each layer is printed separately to the npf Package input file, and that the LAYERED keyword is activated:

```
[15]: print(npf.k22.get_file_entry())
```

```
k22  LAYERED
      INTERNAL  FACTOR  1.0
          0.00000000    1.00000000    2.00000000    3.00000000    4.
↪00000000    5.00000000    6.00000000    7.00000000    8.00000000    9.
↪00000000
      10.00000000    11.00000000    12.00000000    13.00000000    14.
↪00000000    15.00000000    16.00000000    17.00000000    18.00000000    19.
↪00000000
      20.00000000    21.00000000    22.00000000    23.00000000    24.
↪00000000    25.00000000    26.00000000    27.00000000    28.00000000    29.
↪00000000
      30.00000000    31.00000000    32.00000000    33.00000000    34.
↪00000000    35.00000000    36.00000000    37.00000000    38.00000000    39.
↪00000000
      40.00000000    41.00000000    42.00000000    43.00000000    44.
↪00000000    45.00000000    46.00000000    47.00000000    48.00000000    49.
↪00000000
      50.00000000    51.00000000    52.00000000    53.00000000    54.
↪00000000    55.00000000    56.00000000    57.00000000    58.00000000    59.
↪00000000
      60.00000000    61.00000000    62.00000000    63.00000000    64.
↪00000000    65.00000000    66.00000000    67.00000000    68.00000000    69.
↪00000000
      70.00000000    71.00000000    72.00000000    73.00000000    74.
↪00000000    75.00000000    76.00000000    77.00000000    78.00000000    79.
↪00000000
      80.00000000    81.00000000    82.00000000    83.00000000    84.
↪00000000    85.00000000    86.00000000    87.00000000    88.00000000    89.
↪00000000
      90.00000000    91.00000000    92.00000000    93.00000000    94.
↪00000000    95.00000000    96.00000000    97.00000000    98.00000000    99.
↪00000000
      INTERNAL  FACTOR  1.0
      100.00000000    101.00000000    102.00000000    103.00000000    104.
↪00000000    105.00000000    106.00000000    107.00000000    108.00000000    109.
↪00000000
      110.00000000    111.00000000    112.00000000    113.00000000    114.
↪00000000    115.00000000    116.00000000    117.00000000    118.00000000    119.
↪00000000
      120.00000000    121.00000000    122.00000000    123.00000000    124.
↪00000000    125.00000000    126.00000000    127.00000000    128.00000000    129.
↪00000000
      130.00000000    131.00000000    132.00000000    133.00000000    134.
↪00000000    135.00000000    136.00000000    137.00000000    138.00000000    139.
```

(continues on next page)

(continued from previous page)

```

→000000000
    140.00000000    141.00000000    142.00000000    143.00000000    144.
→000000000    145.00000000    146.00000000    147.00000000    148.00000000    149.
→000000000
    150.00000000    151.00000000    152.00000000    153.00000000    154.
→000000000    155.00000000    156.00000000    157.00000000    158.00000000    159.
→000000000
    160.00000000    161.00000000    162.00000000    163.00000000    164.
→000000000    165.00000000    166.00000000    167.00000000    168.00000000    169.
→000000000
    170.00000000    171.00000000    172.00000000    173.00000000    174.
→000000000    175.00000000    176.00000000    177.00000000    178.00000000    179.
→000000000
    180.00000000    181.00000000    182.00000000    183.00000000    184.
→000000000    185.00000000    186.00000000    187.00000000    188.00000000    189.
→000000000
    190.00000000    191.00000000    192.00000000    193.00000000    194.
→000000000    195.00000000    196.00000000    197.00000000    198.00000000    199.
→000000000
    INTERNAL FACTOR 1.0
    200.00000000    201.00000000    202.00000000    203.00000000    204.
→000000000    205.00000000    206.00000000    207.00000000    208.00000000    209.
→000000000
    210.00000000    211.00000000    212.00000000    213.00000000    214.
→000000000    215.00000000    216.00000000    217.00000000    218.00000000    219.
→000000000
    220.00000000    221.00000000    222.00000000    223.00000000    224.
→000000000    225.00000000    226.00000000    227.00000000    228.00000000    229.
→000000000
    230.00000000    231.00000000    232.00000000    233.00000000    234.
→000000000    235.00000000    236.00000000    237.00000000    238.00000000    239.
→000000000
    240.00000000    241.00000000    242.00000000    243.00000000    244.
→000000000    245.00000000    246.00000000    247.00000000    248.00000000    249.
→000000000
    250.00000000    251.00000000    252.00000000    253.00000000    254.
→000000000    255.00000000    256.00000000    257.00000000    258.00000000    259.
→000000000
    260.00000000    261.00000000    262.00000000    263.00000000    264.
→000000000    265.00000000    266.00000000    267.00000000    268.00000000    269.
→000000000
    270.00000000    271.00000000    272.00000000    273.00000000    274.
→000000000    275.00000000    276.00000000    277.00000000    278.00000000    279.
→000000000
    280.00000000    281.00000000    282.00000000    283.00000000    284.
→000000000    285.00000000    286.00000000    287.00000000    288.00000000    289.
→000000000
    290.00000000    291.00000000    292.00000000    293.00000000    294.
→000000000    295.00000000    296.00000000    297.00000000    298.00000000    299.
→000000000

```

Working with a layered array provides lots of flexibility. For example, constants can be set for some layers, but arrays

for others:

```
[16]: npf.k22.set_data([1, a[2], 200])
print(npf.k22.get_file_entry())
```

```
k22  LAYERED
      CONSTANT      1.000000000
      INTERNAL FACTOR 1.0
          200.00000000    201.00000000    202.00000000    203.00000000    204.
→ 000000000    205.00000000    206.00000000    207.00000000    208.00000000    209.
→ 000000000
          210.00000000    211.00000000    212.00000000    213.00000000    214.
→ 000000000    215.00000000    216.00000000    217.00000000    218.00000000    219.
→ 000000000
          220.00000000    221.00000000    222.00000000    223.00000000    224.
→ 000000000    225.00000000    226.00000000    227.00000000    228.00000000    229.
→ 000000000
          230.00000000    231.00000000    232.00000000    233.00000000    234.
→ 000000000    235.00000000    236.00000000    237.00000000    238.00000000    239.
→ 000000000
          240.00000000    241.00000000    242.00000000    243.00000000    244.
→ 000000000    245.00000000    246.00000000    247.00000000    248.00000000    249.
→ 000000000
          250.00000000    251.00000000    252.00000000    253.00000000    254.
→ 000000000    255.00000000    256.00000000    257.00000000    258.00000000    259.
→ 000000000
          260.00000000    261.00000000    262.00000000    263.00000000    264.
→ 000000000    265.00000000    266.00000000    267.00000000    268.00000000    269.
→ 000000000
          270.00000000    271.00000000    272.00000000    273.00000000    274.
→ 000000000    275.00000000    276.00000000    277.00000000    278.00000000    279.
→ 000000000
          280.00000000    281.00000000    282.00000000    283.00000000    284.
→ 000000000    285.00000000    286.00000000    287.00000000    288.00000000    289.
→ 000000000
          290.00000000    291.00000000    292.00000000    293.00000000    294.
→ 000000000    295.00000000    296.00000000    297.00000000    298.00000000    299.
→ 000000000
      CONSTANT      200.00000000
```

To gain full control over an individual layers, layer information can be provided as a dictionary:

```
[17]: a0 = {"factor": 0.5, "iprn": 1, "data": 100 * np.ones((10, 10))}
a1 = 50
a2 = {"factor": 1.0, "iprn": 14, "data": 30 * np.ones((10, 10))}
npf.k22.set_data([a0, a1, a2])
print(npf.k22.get_file_entry())
```

```
k22  LAYERED
      INTERNAL FACTOR 0.5 IPRN 1
          100.00000000    100.00000000    100.00000000    100.00000000    100.
→ 000000000    100.00000000    100.00000000    100.00000000    100.00000000    100.
→ 000000000
          100.00000000    100.00000000    100.00000000    100.00000000    100.
→ 000000000
```

(continues on next page)

(continued from previous page)

[illegible]

(continues on next page)

(continued from previous page)

```

30.000000000 30.000000000 30.000000000 30.000000000 30.
↪000000000 30.000000000 30.000000000 30.000000000 30.000000000 30.
↪000000000
30.000000000 30.000000000 30.000000000 30.000000000 30.
↪000000000 30.000000000 30.000000000 30.000000000 30.000000000 30.
↪000000000

```

Here we say that the FACTOR has been set to 0.5 for the first layer and an alternative print flag is set for the last layer.

Because we are specifying a factor for the top layer, we can also see that the `get_data()` method returns the array without the factor applied

```
[18]: print(npf.k22.get_data())
```

```

[[[100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]
  [100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]
  [100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]
  [100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]
  [100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]
  [100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]
  [100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]
  [100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]
  [100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]
  [100. 100. 100. 100. 100. 100. 100. 100. 100. 100.]]

[[ 50.  50.  50.  50.  50.  50.  50.  50.  50.  50.]
 [ 50.  50.  50.  50.  50.  50.  50.  50.  50.  50.]
 [ 50.  50.  50.  50.  50.  50.  50.  50.  50.  50.]
 [ 50.  50.  50.  50.  50.  50.  50.  50.  50.  50.]
 [ 50.  50.  50.  50.  50.  50.  50.  50.  50.  50.]
 [ 50.  50.  50.  50.  50.  50.  50.  50.  50.  50.]
 [ 50.  50.  50.  50.  50.  50.  50.  50.  50.  50.]
 [ 50.  50.  50.  50.  50.  50.  50.  50.  50.  50.]
 [ 50.  50.  50.  50.  50.  50.  50.  50.  50.  50.]
 [ 50.  50.  50.  50.  50.  50.  50.  50.  50.  50.]]

[[ 30.  30.  30.  30.  30.  30.  30.  30.  30.  30.]
 [ 30.  30.  30.  30.  30.  30.  30.  30.  30.  30.]
 [ 30.  30.  30.  30.  30.  30.  30.  30.  30.  30.]
 [ 30.  30.  30.  30.  30.  30.  30.  30.  30.  30.]
 [ 30.  30.  30.  30.  30.  30.  30.  30.  30.  30.]
 [ 30.  30.  30.  30.  30.  30.  30.  30.  30.  30.]
 [ 30.  30.  30.  30.  30.  30.  30.  30.  30.  30.]
 [ 30.  30.  30.  30.  30.  30.  30.  30.  30.  30.]
 [ 30.  30.  30.  30.  30.  30.  30.  30.  30.  30.]
 [ 30.  30.  30.  30.  30.  30.  30.  30.  30.  30.]]]

```

whereas the array property returns the array with the factor applied

```
[19]: print(npf.k22.array)
```

```

[[[50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
  [50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]

```

(continues on next page)

(continued from previous page)

```
[50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
[50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
[50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
[50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
[50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
[50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
[50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
[50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]]

[[50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
 [50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
 [50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
 [50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
 [50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
 [50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
 [50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
 [50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
 [50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]
 [50. 50. 50. 50. 50. 50. 50. 50. 50. 50.]]

[[30. 30. 30. 30. 30. 30. 30. 30. 30. 30.]
 [30. 30. 30. 30. 30. 30. 30. 30. 30. 30.]
 [30. 30. 30. 30. 30. 30. 30. 30. 30. 30.]
 [30. 30. 30. 30. 30. 30. 30. 30. 30. 30.]
 [30. 30. 30. 30. 30. 30. 30. 30. 30. 30.]
 [30. 30. 30. 30. 30. 30. 30. 30. 30. 30.]
 [30. 30. 30. 30. 30. 30. 30. 30. 30. 30.]
 [30. 30. 30. 30. 30. 30. 30. 30. 30. 30.]
 [30. 30. 30. 30. 30. 30. 30. 30. 30. 30.]
 [30. 30. 30. 30. 30. 30. 30. 30. 30. 30.]]
```

Adding MODFLOW Stress Period Array Data

Transient array data spanning multiple stress periods must be specified as a dictionary of arrays, where the dictionary key is the stress period, expressed as a zero-based integer, and the dictionary value is the grid data for that stress period.

In the following example a RCHA package is created. First a dictionary is created that contains recharge for the model's two stress periods. Recharge is specified as a constant value in this example, though it could also be specified as a 3-dimensional ndarray or list of lists.

```
[20]: rch_sp1 = 0.01
      rch_sp2 = 0.03
      rch_spd = {0: rch_sp1, 1: rch_sp2}
```

The RCHA package is created and the dictionary constructed above is passed in as the `recharge` parameter.

```
[21]: rch = flopy.mf6.ModflowGwfrcha(
      gwf, readasarrays=True, pname="rch", print_input=True, recharge=rch_spd
      )
```

Retrieving Grid Array Data

Grid data can be retrieved with `get_data`, `array`, `[]`, `repr/str`, or `get_file_entry`.

Retrieval Method	Description
<code>get_data</code>	Returns ndarray of data without multiplier applied, unless the <code>apply_mult</code> parameter is set to True
<code>array</code>	Returns ndarray of data with multiplier applied
<code>[]</code>	Returns a particular layer of data if data is layered, otherwise is an array index
<code>repr/str</code>	Returns string with storage information followed by ndarray repr/str
<code>get_file_entry</code>	Returns string containing data formatted for the MODFLOW-6 package file. If layer is not specified returns all layers, otherwise returns just the layer specified.

Below the NPF k array is retrieved using the various methods highlighted above.

First, we use the `get_data` method to get k as an ndarray.

```
[22]: print(npf.k.get_data())

[[[10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]]

[[[10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]]

[[[10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]]]
```

Next, we use the `array` attribute which also gets k as an ndarray.

```
[23]: print(npf.k.array)

[[[10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]]

[[[10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]]

[[[10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
  [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]]]
```

We can also use the `[]` to get a single layer of data as an ndarray.

```
[24]: print(npf.k[0])

[[10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]]
```

`repr` gives a string representation of the data.

```
[25]: print(repr(npf.k))
```

```

Layer_1{internal}
(array([[10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.]])
Layer_2{internal}
(array([[10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.]])
Layer_3{internal}
(array([[10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.],
       [10., 10., 10., 10., 10., 10., 10., 10., 10., 10.]])

```

str gives a similar string representation of the data.

```
[26]: print(str(npf.k))
```

```

Layer_1{internal}
([[10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]])
Layer_2{internal}
([[10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]

```

(continues on next page)

(continued from previous page)

```

[10. 10. 10. 10. 10. 10. 10. 10. 10.]
[10. 10. 10. 10. 10. 10. 10. 10. 10.]
[10. 10. 10. 10. 10. 10. 10. 10. 10.]
[10. 10. 10. 10. 10. 10. 10. 10. 10.]
[10. 10. 10. 10. 10. 10. 10. 10. 10.]
[10. 10. 10. 10. 10. 10. 10. 10. 10.]
[10. 10. 10. 10. 10. 10. 10. 10. 10.]
[10. 10. 10. 10. 10. 10. 10. 10. 10.]]
Layer_3{internal}
([[10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10.]])

```

The method `get_file_entry` prints the data as it would appear in a MODFLOW 6 file.

```
[27]: print(npf.k.get_file_entry())
```

```

k  LAYERED
INTERNAL  FACTOR  1.0
10.000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000
10.000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000
10.000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000
10.000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000
10.000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000
10.000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000
10.000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000
10.000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000
10.000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000  10.000000000  10.000000000  10.000000000  10.
↪000000000

```

(continues on next page)

[illegible]

(continues on next page)

(continued from previous page)

```

→000000000 10.00000000 10.00000000 10.00000000 10.00000000 10.
→000000000
10.00000000 10.00000000 10.00000000 10.00000000 10.
→000000000 10.00000000 10.00000000 10.00000000 10.00000000 10.
→000000000
10.00000000 10.00000000 10.00000000 10.00000000 10.
→000000000 10.00000000 10.00000000 10.00000000 10.00000000 10.
→000000000
10.00000000 10.00000000 10.00000000 10.00000000 10.
→000000000 10.00000000 10.00000000 10.00000000 10.00000000 10.
→000000000
10.00000000 10.00000000 10.00000000 10.00000000 10.
→000000000 10.00000000 10.00000000 10.00000000 10.00000000 10.
→000000000

```

Retrieving MODFLOW Stress Period Array Data

Transient array data can be retrieved with `get_data`, `array`, `repr/str`, or `get_file_entry`.

Retrieval Method	Description
<code>get_data</code>	Returns dictionary of ndarrays without multiplier applied. The dictionary key is the stress period as a zero-based integer.
<code>array</code>	Returns ndarray of data for all stress periods (stress period is an added dimension)
<code>repr/str</code>	Returns string with storage information followed by ndarray repr/str for all stress periods
<code>get_file_entry(la</code>	Returns string containing data formatted for the MODFLOW-6 package file. Use to specify a stress period (zero-based integer).

Below the RCHA recharge array is retrieved using the various methods highlighted above.

First, we use the `get_data` method to get the recharge data as a dictionary of ndarrays. The dictionary key is a the stress period (zero based).

```

[28]: print(rch.recharge.get_data())
{0: array([[0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
          [0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
          [0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
          [0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
          [0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
          [0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
          [0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
          [0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01],
          [0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]]), 1: array([[0.03,
→0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03],
          [0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03],
          [0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03],
          [0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03],
          [0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03],

```

(continues on next page)

(continued from previous page)

```
[0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03],
[0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03],
[0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03],
[0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03],
[0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03, 0.03]]])}
```

Next, we use the array attribute to get the data as an 4-dimensional ndarray.

```
[29]: print(rch.recharge.array)
```

```
[[[0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]
  [0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]
  [0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]
  [0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]
  [0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]
  [0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]
  [0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]
  [0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]
  [0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]
  [0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.01]]]

[[[0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03]
  [0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03]
  [0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03]
  [0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03]
  [0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03]
  [0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03]
  [0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03]
  [0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03]
  [0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03]
  [0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03 0.03]]]]
```

repr gives a string representation of the data.

```
[30]: print(repr(rch.recharge))
```

```
{constant 0.03}
```

str gives a similar representation of the data.

```
[31]: print(str(rch.recharge))
```

```
{constant 0.03}
```

We can use the `get_file_entry` method to get the data as it would appear in a MODFLOW 6 file for a specific stress period. In this case we are getting the data for stress period 2 (stress periods are treated as 0-based in FloPy).

```
[32]: print(rch.recharge.get_file_entry(1))
```

```
recharge
  CONSTANT      0.03000000
```

```
[33]: try:
      temp_dir.cleanup()
    except PermissionError:
      # can occur on windows: https://docs.python.org/3/library/tempfile.html#tempfile.
      ↪ TemporaryDirectory
      pass
```

5.2.8 MODFLOW 6: Data Storage Information and Performance Optimization

This tutorial shows the different options for storing MODFLOW data in FloPy. Interaction with a FloPy MODFLOW 6 model is different from other models, such as MODFLOW-2005, MT3D, and SEAWAT, for example.

FloPy stores model data in data objects (MFDataArray, MFDataList, MFDataScalar objects) that are accessible from packages. Data can be added to a package by using the appropriate parameters when the package is constructed and through package attributes.

The MODFLOW 6 simulation structure is arranged in the following generalized way:

```
Simulation --> Package --> DATA

Simulation --> Model --> Package (--> Package) --> DATA
```

This tutorial focuses on the different storage options for MODFLOW data and how to optimize data storage read/write speed.

Introduction to Data Storage Information

MODFLOW array and list data can either be stored internally or externally in text or binary files. Additionally array data can have a factor applied to them and can have a format flag/code to define how these data will be formatted. This data storage information is specified within a python dictionary. The python dictionary must contain a “data” key where the data is stored and supports several other keys that determine how and where the data is stored.

The following code sets up a basic simulation with a groundwater flow model. for the example below.

```
[1]: # package import
import os
from pathlib import Path
from tempfile import TemporaryDirectory

[2]: import numpy as np

[3]: import flopy

[4]: # set up where simulation workspace will be stored
temp_dir = TemporaryDirectory()
workspace = temp_dir.name
name = "tutorial08_mf6_data"

[5]: # create the FloPy simulation and tdis objects
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws=workspace
```

(continues on next page)

(continued from previous page)

```

)
tdis = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim,
    pname="tdis",
    time_units="DAYS",
    nper=2,
    perioddata=[(1.0, 1, 1.0), (1.0, 1, 1.0)],
)
# create the Flopy groundwater flow (gwf) model object
model_nam_file = f"{name}.nam"
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, model_nam_file=model_nam_file)
# create the flopy iterative model solver (ims) package object
ims = flopy.mf6.modflow.mfims.ModflowIms(sim, pname="ims", complexity="SIMPLE")
# create the discretization package
bot = np.linspace(-50.0 / 3.0, -3.0, 3)
delrow = delcol = 4.0
dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf,
    pname="dis",
    nogrb=True,
    nlay=3,
    nrow=10,
    ncol=10,
    delr=delrow,
    delc=delcol,
    top=0.0,
    botm=bot,
)

```

Setting up a Data Storage Information Dictionary

To store data externally add a `filename` key to the dictionary whose value is the file where you want to store the data. Add a `binary` key with value `True` to make the file a binary file. Add a `prn` key whose value is a format code to set the format code for the data. For array data add a `factor` key whose value is a positive floating point number to add a factor/multiplier for the array.

Below a dictionary is created that defines how a `k33` array will be stored. The dictionary specifies that the `k33` array be stored in the binary file `k33.txt` with a factor of 1.0 applied to the array and a print code of 1. The `k33` array data is constructed as a numpy array.

```

[6]: k33_values = np.full((3, 10, 10), 1.1)
k33 = {
    "filename": "k33.txt",
    "factor": 1.0,
    "data": k33_values,
    "iprn": 1,
    "binary": "True",
}

```

The NPF package is then created with the `k33` array in an external binary file. This binary file is created when the simulation method `write_simulation` is called.

```
[7]: npf = flopy.mf6.ModflowGwfnf(
    gwf,
    pname="npf",
    save_flows=True,
    icelltype=[1, 1, 1],
    k=10.0,
    k22=5.0,
    k33=k33,
    xt3doptions="xt3d rhs",
    rewet_record="REWET WETFCT 1.0 IWETIT 1 IHDWET 0",
)

<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7f9568dc3100>
<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7f9568dc3130>
```

External files can be set for specific layers of data. If we want to store the bottom elevations for the third model layer to an external file, then the dictionary that we pass in for the third layer can be given a “filename” key.

```
[8]: a0 = {"factor": 0.5, "iprn": 1, "data": np.ones((10, 10))}
a1 = -100
a2 = {
    "filename": "dis.botm.3.txt",
    "factor": 2.0,
    "iprn": 1,
    "data": -100 * np.ones((10, 10)),
}
```

A list containing data for the three specified layers is then passed in to the botm object’s set_record method.

```
[9]: dis.botm.set_record([a0, a1, a2])
print(dis.botm.get_file_entry())
```

```
botm LAYERED
INTERNAL FACTOR 0.5 IPRN 1
1.000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000
1.000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000
1.000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000
1.000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000
1.000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000
1.000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000 1.000000000 1.000000000 1.000000000 1.
↪ 000000000
```

(continues on next page)

(continued from previous page)

[illegible]

The botm record retrieved can be modified and then saved with `set_record`. For example, the array data's "factor" can be modified and saved.

```
[11]: botm_record[0]["factor"] = 0.6
      dis.botm.set_record(botm_record)
```

The updated value can then be retrieved.

```
[12]: botm_record = dis.botm.get_record()
      print(f"botm layer 1 factor: {botm_record[0]['factor']}")

      botm layer 1 factor: 0.6
```

The `get_record` and `set_record` methods can also be used with list data to get and set the data and its “filename” and “binary” attributes. This is demonstrated with the `wel` package. First, a `wel` package is constructed.

```
[13]: welspdict = {
        0: {"filename": "well_sp1.txt", "data": [[(0, 0, 0), 0.25]]},
        1: [[(0, 0, 0), 0.1]],
    }
    wel = flopy.mf6.ModflowGwfwel(
        gwf,
        print_input=True,
```

(continues on next page)

(continued from previous page)

```

    print_flows=True,
    stress_period_data=welspdict,
    save_flows=False,
)

```

The wel stress period data and associated “filename” and “binary” attributes can be retrieved with `get_record`.

```

[14]: spd_record = wel.stress_period_data.get_record()
print("Stress period 1 record:")
print(spd_record[0])
print("\nStress period 2 record:")
print(spd_record[1])

Stress period 1 record:
{'filename': 'well_sp1.txt', 'binary': False, 'data': rec.array([[(0, 0, 0), 0.25]],
      dtype=[('cellid', 'O'), ('q', '<f8')])}

Stress period 2 record:
{'data': rec.array([[(0, 0, 0), 0.1]],
      dtype=[('cellid', 'O'), ('q', '<f8')])}

```

The wel data and associated attributes can be changed by modifying the record and then saving it with the `set_record` method.

```

[15]: spd_record[0]["filename"] = "well_package_sp1.txt"
      spd_record[0]["binary"] = True
      spd_record[1]["filename"] = "well_package_sp2.bin"
      wel.stress_period_data.set_record(spd_record)

```

The changes can be verified by calling `get_record` again.

```

[16]: spd_record = wel.stress_period_data.get_record()
print(f"New filename for stress period 1: {spd_record[0]['filename']}")
print(f"New binary flag for stress period 1: {spd_record[0]['binary']}")
print(f"New filename for stress period 2: {spd_record[1]['filename']}")

New filename for stress period 1: well_package_sp1.txt
New binary flag for stress period 1: True
New filename for stress period 2: well_package_sp2.bin

```

Optimizing FloPy Performance

By default FloPy will perform a number of verification checks on your data when FloPy loads or saves that data. For large datasets turning these verification checks off can significantly improve FloPy’s performance. Additionally, storing files externally can help minimize the amount of data FloPy reads/writes when loading and saving a simulation. The following steps will help you optimize FloPy’s performance for large datasets.

- 1) Turn off FloPy verification checks and FloPy’s option to automatically update “maxbound”. This can be turned off on an existing simulation by either individually turning off each of these settings.

```

[17]: sim.simulation_data.auto_set_sizes = False
      sim.simulation_data.verify_data = False
      sim.write_simulation()

```

```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims...
  writing model tutorial08_mf6_data...
    writing model name file...
    writing package dis...
    writing package npf...
    writing package wel_0...
```

or by setting `lazy_io` to `True`.

```
[18]: sim.simulation_data.lazy_io = True
sim.write_simulation()

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims...
  writing model tutorial08_mf6_data...
    writing model name file...
    writing package dis...
    writing package npf...
    writing package wel_0...
```

These options can also be turned off when loading an existing simulation or creating a new simulation by setting `lazy_io` to `True`.

```
[19]: sim2 = flopy.mf6.MFSimulation.load(
        sim_ws=workspace,
        lazy_io=True,
    )

loading simulation...
  loading simulation name file...
  loading tdis package...
  loading model gw6...
    loading package dis...
    loading package npf...
    loading package wel...
  loading solution package tutorial08_mf6_data...
```

```
[20]: sim3 = flopy.mf6.MFSimulation(lazy_io=True)
```

- 2) Whenever possible save large datasets to external binary files. Binary files are more compact and the data will be read and written significantly faster. See MODFLOW-6 documentation for which packages and data support binary files.

```
[21]: # store all well period data in external binary files
spd_record[0]["binary"] = True
spd_record[1]["binary"] = True
wel.stress_period_data.set_record(spd_record)
```

- 3) For datasets that do not support binary files, save them to external text files. When loading a simulation, FloPy will always parse MODFLOW-6 package files, but will not parse external text files when `auto_set_sizes` and `verify`

data are both set to False. Additionally, if write_simulation is later called, external files will not be re-written unless either the data or the file path has changed.

```
[22]: # store lak period data in external text files
period = {
    0: {"filename": "lak_sp1.txt", "data": [(0, "STAGE", 10.0)]},
    1: {"filename": "lak_sp2.txt", "data": [(0, "STAGE", 15.0)]},
}
lakpd = [(0, -2.0, 1)]
lakecn = [(0, 0, (0, 1, 0), "HORIZONTAL", 1.0, -5.0, 0.0, 10.0, 10.0)]
lak = flopy.mf6.ModflowGwflak(
    gwf,
    pname="lak-1",
    nlakes=1,
    noutlets=0,
    ntables=0,
    packagedata=lakpd,
    connectiondata=lakecn,
    perioddata=period,
)
```

```
[23]: try:
    temp_dir.cleanup()
except PermissionError:
    # can occur on windows: https://docs.python.org/3/library/tempfile.html#tempfile.
    ↪ TemporaryDirectory
    pass
```

5.2.9 MODFLOW 6: Multiple Models - How to create multiple models in a simulation

This tutorial shows a simulation using two models, demonstrating how to use exchanges and exchange subpackages.

Introduction to Multiple Models

MODFLOW-6 simulations can contain multiple models, which can be linked through the groundwater exchange package, which can contain mover and ghost node correction subpackages.

The following code sets up a basic simulation.

```
[1]: # package import
from pathlib import Path
from tempfile import TemporaryDirectory
```

```
[2]: import flopy
```

```
[3]: # set up where simulation workspace will be stored
temp_dir = TemporaryDirectory()
workspace = temp_dir.name
name = "tutorial09_mf6_data"
```

```
[4]: # create the FloPy simulation and tdis objects
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws=workspace
)
tdis = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim,
    pname="tdis",
    time_units="DAYS",
    nper=2,
    perioddata=[(1.0, 1, 1.0), (1.0, 1, 1.0)],
)
```

Groundwater Flow Model Setup

We will start by setting up two groundwater flow models that are part of the same simulation.

```
[5]: # set up first groundwater flow model
name_1 = "ex_1_mod_1"
model_nam_file = f"{name_1}.nam"
gwf = flopy.mf6.ModflowGwf(
    sim, modelname=name_1, model_nam_file=model_nam_file
)
# create the discretization package
bot = [-10.0, -50.0, -200.0]
delrow = delcol = 4.0
nlay = 3
nrow = 10
ncol = 10
dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf,
    pname="dis-1",
    nogrb=True,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=delrow,
    delc=delcol,
    top=0.0,
    botm=bot,
)
# create npf package
npf = flopy.mf6.ModflowGwnpf(
    gwf,
    pname="npf-1",
    save_flows=True,
    icelltype=[1, 1, 1],
    k=10.0,
    k33=5.0,
    xt3doptions="xt3d rhs",
    # rewet_record="REWET WETFCT 1.0 IWETIT 1 IHDWET 0",
)
# create ic package
```

(continues on next page)

(continued from previous page)

```
ic_package = flopy.mf6.modflow.mfgwfic.ModflowGwfic(gwf, strt=0.0)
```

```
[6]: # create ghb package
ghb_spd = {0: [((0, 0, 0), -1.0, 1000.0)]}
ghb = flopy.mf6.modflow.mfgwfgghb.ModflowGwfgghb(
    gwf,
    print_input=True,
    print_flows=True,
    pname="ghb-1",
    maxbound=1,
    stress_period_data=ghb_spd,
)
```

```
[7]: # create wel package
welspd = {0: [((0, 5, ncol - 1), -100.0)]}
wel = flopy.mf6.ModflowGwfwel(
    gwf,
    print_input=True,
    print_flows=True,
    mover=True,
    stress_period_data=welspd,
    save_flows=False,
    pname="WEL-1",
)
```

```
[8]: # set up second groundwater flow model with a finer grid
name_1 = "ex_1_mod_2"
model_nam_file = f"{name_1}.nam"
gwf_2 = flopy.mf6.ModflowGwf(
    sim, modelname=name_1, model_nam_file=model_nam_file
)
# create the flopy iterative model solver (ims) package object
# by default flopy will register both models with the ims package.
ims = flopy.mf6.modflow.mfims.ModflowIms(
    sim, pname="ims", complexity="SIMPLE", linear_acceleration="BICGSTAB"
)
# no need to create a new ims package. flopy will automatically register
# create the discretization package
bot = [-10.0, -50.0, -200.0]
dis_2 = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf_2,
    pname="dis-2",
    nogrb=True,
    nlay=nlay,
    nrow=nrow * 2,
    ncol=ncol * 2,
    delr=delrow / 2.0,
    delc=delcol / 2.0,
    top=0.0,
    botm=bot,
)
```

(continues on next page)

(continued from previous page)

```
# create npf package
npf_2 = flopy.mf6.ModflowGwfnpf(
    gwf_2,
    pname="npf-2",
    save_flows=True,
    icelltype=[1, 1, 1],
    k=10.0,
    k33=5.0,
    xt3doptions="xt3d rhs",
    # rewet_record="REWET WETFCT 1.0 IWETIT 1 IHDWET 0",
)
# create ic package
ic_package_2 = flopy.mf6.modflow.mfgwfic.ModflowGwfic(gwf_2, strt=0.0)
```

```
[9]: # create ghb package
ghb_spd = {0: [(0, 0, 19), -10.0, 1000.0]}
ghb_2 = flopy.mf6.modflow.mfgwfgbh.ModflowGwfgbh(
    gwf_2,
    print_input=True,
    print_flows=True,
    pname="ghb-2",
    maxbound=1,
    stress_period_data=ghb_spd,
)
```

```
[10]: # create lak package
lakpd = [(0, -2.0, 1)]
lakecn = [(0, 0, (0, 5, 0), "HORIZONTAL", 1.0, -5.0, 0.0, 10.0, 10.0)]
lak_2 = flopy.mf6.ModflowGwflak(
    gwf_2,
    pname="lak-2",
    print_input=True,
    mover=True,
    nlakes=1,
    noutlets=0,
    ntables=0,
    packagedata=lakpd,
    connectiondata=lakecn,
)
```

Connecting the Flow Models

The two groundwater flow models created above are now part of the same simulation, but they are not connected. To connect them we will use the gwfgwf package and two of its subpackages, gnc and mvr.

Use exchangedata to define how the two models are connected. In this example we are connecting the right edge of the first model to the left edge of the second model. The second model is 2x the discretization, so each cell in the first model is connected to two cells in the second model.

```
[11]: gwfgwf_data = []
row_2 = 0
```

(continues on next page)

(continued from previous page)

```

for row in range(0, nrow):
    gwfgwf_data.append([(0, ncol - 1, row), (0, 0, row_2), 1, 2.03, 1.01, 2.0])
    row_2 += 1
    gwfgwf_data.append([(0, ncol - 1, row), (0, 0, row_2), 1, 2.03, 1.01, 2.0])
    row_2 += 1

```

```

[12]: # create the gwfgwf package
gwfgwf = flopy.mf6.ModflowGwfgwf(
    sim,
    exgtype="GWF6-GWF6",
    nexg=len(gwfgwf_data),
    exgmnamea=gwf.name,
    exgmnameb=gwf_2.name,
    exchangedata=gwfgwf_data,
    filename="mod1_mod2.gwfgwf",
)

```

Due to the two model's different cell sizes, the cell centers of the first model do not align with the cell centers in the second model. To correct for this we will use the ghost node correction package (gnc).

```

[13]: gnc_data = []
col_2 = 0
weight_close = 1.0 / 1.25
weight_far = 0.25 / 1.25
for col in range(0, ncol):
    if col == 0:
        gnc_data.append(
            (
                (0, nrow - 1, col),
                (0, 0, col_2),
                (0, nrow - 1, col),
                (0, nrow - 1, 0),
                1.00,
                0.0,
            )
        )
    else:
        gnc_data.append(
            (
                (0, nrow - 1, col),
                (0, 0, col_2),
                (0, nrow - 1, col),
                (0, nrow - 1, col - 1),
                weight_close,
                weight_far,
            )
        )
    col_2 += 1
    if col == ncol - 1:
        gnc_data.append(
            (
                (0, nrow - 1, col),

```

(continues on next page)

(continued from previous page)

```

        (0, 0, col_2),
        (0, nrow - 1, col),
        (0, nrow - 1, 0),
        1.00,
        0.0,
    )
)
)
else:
    gnc_data.append(
        (
            (0, nrow - 1, col),
            (0, 0, col_2),
            (0, nrow - 1, col),
            (0, nrow - 1, col + 1),
            weight_close,
            weight_far,
        )
    )
)
col_2 += 1

```

```

[14]: # set up gnc package
fname = "gwfgwf.input.gnc"
gwfgwf.gnc.initialize(
    filename=fname,
    print_input=True,
    print_flows=True,
    numgnc=ncol * 2,
    numalphaj=2,
    gncdata=gnc_data,
)

```

The extraction well at the right-hand side of the first model is pumping the water it extracts into a nearby lake at the left-hand side of the second model. Using the mover (mvr) package, water extracted from the first model's wel package is moved to the second model's lak package.

```

[15]: package_data = [(gwf.name, "WEL-1"), (gwf_2.name, "lak-2")]
period_data = [(gwf.name, "WEL-1", 0, gwf_2.name, "lak-2", 0, "FACTOR", 1.0)]
fname = "gwfgwf.input.mvr"
gwfgwf.mvr.initialize(
    filename=fname,
    modelnames=True,
    print_input=True,
    print_flows=True,
    maxpackages=2,
    maxmvr=1,
    packages=package_data,
    perioddata=period_data,
)

```

```

[16]: sim.write_simulation()
sim.run_simulation()

```



```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims...
  writing package mod1_mod2.gwfgwf...
  writing package gwfgwf.input.gnc...
  writing package gwfgwf.input.mvr...
  writing model ex_1_mod_1...
    writing model name file...
    writing package dis-1...
    writing package npf-1...
    writing package ic...
    writing package ghb-1...
    writing package wel-1...
INFORMATION: maxbound in ('gw6', 'wel', 'dimensions') changed to 1 based on size of
↳ stress_period_data
  writing model ex_1_mod_2...
    writing model name file...
    writing package dis-2...
    writing package npf-2...
    writing package ic...
    writing package ghb-2...
    writing package lak-2...
FloPy is using the following executable to run the model: ../../home/runner/.local/bin/
↳ modflow/mf6

                        MODFLOW 6
        U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
                VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
                Build 20220726_000000

This software has been approved for release by the U.S. Geological
Survey (USGS). Although the software has been subjected to rigorous
review, the USGS reserves the right to update the software as needed
pursuant to further analysis and review. No warranty, expressed or
implied, is made by the USGS or the U.S. Government as to the
functionality of the software and related material nor shall the
fact of release constitute any such warranty. Furthermore, the
software is released on condition that neither the USGS nor the U.S.
Government shall be held liable for any damages resulting from its
authorized or unauthorized use. Also refer to the USGS Water
Resources Software User Rights Notice for complete use, copyright,
and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:30

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

        Solving:  Stress period:      1      Time step:      1
```

(continues on next page)

(continued from previous page)

```

Solving:  Stress period:      2    Time step:      1

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:30
Elapsed run time:  0.209 Seconds

Normal termination of simulation.

```

```
[16]: (True, [])
```

```

[17]: try:
        temp_dir.cleanup()
    except PermissionError:
        # can occur on windows: https://docs.python.org/3/library/tempfile.html#tempfile.
        ↪ TemporaryDirectory
        pass

```

5.2.10 Observations, time series and time array series

This code sets up a simulation and creates some data for the simulation.

```

[1]: import os
import sys
from tempfile import TemporaryDirectory

import numpy as np

try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

# init paths
exe_name = "mf6"

# temporary directory
temp_dir = TemporaryDirectory()
sim_path = os.path.join(temp_dir.name, "obs_ts_tas_ex")
# make the directory if it does not exist
if not os.path.isdir(sim_path):
    os.makedirs(sim_path, exist_ok=True)

# init paths
test_ex_name = "child_pkgs_test"
model_name = "child_pkgs"

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("flopy version: {}".format(flopy.__version__))

```

```
3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
flopy version: 3.4.3
```

```
[2]: # create simulation
sim = flopy.mf6.MFSimulation(
    sim_name=test_ex_name, version="mf6", exe_name="mf6", sim_ws=sim_path
)

tdis_rc = [(1.0, 1, 1.0), (10.0, 120, 1.0), (10.0, 120, 1.0), (10.0, 120, 1.0)]
tdis_package = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim, time_units="DAYS", nper=4, perioddata=tdis_rc
)

model = flopy.mf6.ModflowGwf(
    sim, modelname=model_name, model_nam_file="{}.nam".format(model_name)
)

ims_package = flopy.mf6.modflow.mfims.ModflowIms(
    sim,
    print_option="SUMMARY",
    complexity="SIMPLE",
    outer_hclose=0.0001,
    outer_maximum=500,
    under_relaxation="NONE",
    inner_maximum=100,
    inner_hclose=0.0001,
    rcloserecord=0.001,
    linear_acceleration="CG",
    scaling_method="NONE",
    reordering_method="NONE",
    relaxation_factor=0.97,
)

sim.register_ims_package(ims_package, [model.name])
bot_data = [-100 for x in range(150)]
dis_package = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    model,
    nlay=3,
    nrow=15,
    ncol=10,
    delr=500.0,
    delc=500.0,
    top=50.0,
    botm=[5.0, -10.0, {"factor": 1.0, "data": bot_data}],
    filename="{}.dis".format(model_name),
)

ic_package = flopy.mf6.modflow.mfgwfic.ModflowGwfic(
    model, strt=50.0, filename="{}.ic".format(model_name)
)

npf_package = flopy.mf6.modflow.mfgwfnpf.ModflowGwfnpf(
    model,
    save_flows=True,
    icelltype=[1, 0, 0],
```

(continues on next page)

(continued from previous page)

```

    k=[5.0, 0.1, 4.0],
    k33=[0.5, 0.005, 0.1],
)
oc_package = flopy.mf6.modflow.mfgwfoc.ModflowGwfoc(
    model,
    budget_filerecord="child_pkgs.cbc",
    head_filerecord="child_pkgs.hds",
    headprintrecord=["COLUMNS", 10, "WIDTH", 15, "DIGITS", 6, "GENERAL"],
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
    printrecord=[("HEAD", "FIRST"), ("HEAD", "LAST"), ("BUDGET", "LAST")],
)
sto_package = flopy.mf6.modflow.mfgwfsto.ModflowGwfsto(
    model,
    save_flows=True,
    iconvert=1,
    ss=0.0000001,
    sy=0.2,
    steady_state={0: True},
    transient={1: True},
)

```

Observations

Observations can be set for any package through the package.obs object, and each package.obs object has several attributes that can be set:

package.obs.filename : str Name of observations file to create. The default is packagename + '.obs', e.g. my-model.ghb.obs.

package.obs.continuous : dict A dictionary that has file names as keys and a list of observations as the dictionary values. Default should probably be None. package.obs.observations = { 'fname1': [(obsname, obstype, cellid), ...], 'fname2': [(obsname, obstype, cellid), ...]}

package.obs.digits : int Number of digits to write the observation values. Default is 10.

package.obs.print_input : bool Flag indicating whether or not observations are written to listing file.

Method 1: Pass obs to package constructor

```

[3]: # build ghb stress period data
ghb_spd = {}
ghb_period = []
for layer, cond in zip(range(1, 3), [15.0, 1500.0]):
    for row in range(0, 15):
        ghb_period.append((layer, row, 9), 1.0, cond, "Estuary-L2")
ghb_spd[0] = ghb_period

# build obs data
ghb_obs = {
    ("ghb_obs.csv", "binary"): [
        ("ghb-2-6-10", "GHB", (1, 5, 9)),
    ]
}

```

(continues on next page)

(continued from previous page)

```

        ("ghb-3-6-10", "GHB", (2, 5, 9)),
    ],
    "ghb_flows.csv": [
        ("Estuary2", "GHB", "Estuary-L2"),
        ("Estuary3", "GHB", "Estuary-L3"),
    ],
}
# build ghb package
ghb = flopy.mf6.modflow.mfgwfghb.ModflowGwfghb(
    model,
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    observations=ghb_obs,
    pname="ghb",
    maxbound=30,
    stress_period_data=ghb_spd,
)
ghb.obs.print_input = True

sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

# clean up for next example
model.remove_package("ghb")

```

```

writing simulation...
writing simulation name file...
writing simulation tdis package...
writing solution package ims_1...
writing model child_pkgs...
writing model name file...
writing package dis...
writing package ic...
writing package npf...
writing package oc...
writing package sto...
writing package ghb...
writing package obs_0...

```

```

MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

```

```

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

```

(continues on next page)

(continued from previous page)

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:34

Writing simulation list file: mfsim.lst

Using Simulation name file: mfsim.nam

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5
Solving:	Stress period:	2	Time step:	6
Solving:	Stress period:	2	Time step:	7
Solving:	Stress period:	2	Time step:	8
Solving:	Stress period:	2	Time step:	9
Solving:	Stress period:	2	Time step:	10
Solving:	Stress period:	2	Time step:	11
Solving:	Stress period:	2	Time step:	12
Solving:	Stress period:	2	Time step:	13
Solving:	Stress period:	2	Time step:	14
Solving:	Stress period:	2	Time step:	15
Solving:	Stress period:	2	Time step:	16
Solving:	Stress period:	2	Time step:	17
Solving:	Stress period:	2	Time step:	18
Solving:	Stress period:	2	Time step:	19
Solving:	Stress period:	2	Time step:	20
Solving:	Stress period:	2	Time step:	21
Solving:	Stress period:	2	Time step:	22
Solving:	Stress period:	2	Time step:	23
Solving:	Stress period:	2	Time step:	24
Solving:	Stress period:	2	Time step:	25
Solving:	Stress period:	2	Time step:	26
Solving:	Stress period:	2	Time step:	27
Solving:	Stress period:	2	Time step:	28
Solving:	Stress period:	2	Time step:	29
Solving:	Stress period:	2	Time step:	30
Solving:	Stress period:	2	Time step:	31

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	32
Solving:	Stress period:	2	Time step:	33
Solving:	Stress period:	2	Time step:	34
Solving:	Stress period:	2	Time step:	35
Solving:	Stress period:	2	Time step:	36
Solving:	Stress period:	2	Time step:	37
Solving:	Stress period:	2	Time step:	38
Solving:	Stress period:	2	Time step:	39
Solving:	Stress period:	2	Time step:	40
Solving:	Stress period:	2	Time step:	41
Solving:	Stress period:	2	Time step:	42
Solving:	Stress period:	2	Time step:	43
Solving:	Stress period:	2	Time step:	44
Solving:	Stress period:	2	Time step:	45
Solving:	Stress period:	2	Time step:	46
Solving:	Stress period:	2	Time step:	47
Solving:	Stress period:	2	Time step:	48
Solving:	Stress period:	2	Time step:	49
Solving:	Stress period:	2	Time step:	50
Solving:	Stress period:	2	Time step:	51
Solving:	Stress period:	2	Time step:	52
Solving:	Stress period:	2	Time step:	53
Solving:	Stress period:	2	Time step:	54
Solving:	Stress period:	2	Time step:	55
Solving:	Stress period:	2	Time step:	56
Solving:	Stress period:	2	Time step:	57
Solving:	Stress period:	2	Time step:	58
Solving:	Stress period:	2	Time step:	59
Solving:	Stress period:	2	Time step:	60
Solving:	Stress period:	2	Time step:	61
Solving:	Stress period:	2	Time step:	62
Solving:	Stress period:	2	Time step:	63
Solving:	Stress period:	2	Time step:	64
Solving:	Stress period:	2	Time step:	65
Solving:	Stress period:	2	Time step:	66
Solving:	Stress period:	2	Time step:	67
Solving:	Stress period:	2	Time step:	68
Solving:	Stress period:	2	Time step:	69
Solving:	Stress period:	2	Time step:	70
Solving:	Stress period:	2	Time step:	71
Solving:	Stress period:	2	Time step:	72
Solving:	Stress period:	2	Time step:	73
Solving:	Stress period:	2	Time step:	74
Solving:	Stress period:	2	Time step:	75
Solving:	Stress period:	2	Time step:	76
Solving:	Stress period:	2	Time step:	77
Solving:	Stress period:	2	Time step:	78
Solving:	Stress period:	2	Time step:	79
Solving:	Stress period:	2	Time step:	80
Solving:	Stress period:	2	Time step:	81
Solving:	Stress period:	2	Time step:	82
Solving:	Stress period:	2	Time step:	83

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	84
Solving:	Stress period:	2	Time step:	85
Solving:	Stress period:	2	Time step:	86
Solving:	Stress period:	2	Time step:	87
Solving:	Stress period:	2	Time step:	88
Solving:	Stress period:	2	Time step:	89
Solving:	Stress period:	2	Time step:	90
Solving:	Stress period:	2	Time step:	91
Solving:	Stress period:	2	Time step:	92
Solving:	Stress period:	2	Time step:	93
Solving:	Stress period:	2	Time step:	94
Solving:	Stress period:	2	Time step:	95
Solving:	Stress period:	2	Time step:	96
Solving:	Stress period:	2	Time step:	97
Solving:	Stress period:	2	Time step:	98
Solving:	Stress period:	2	Time step:	99
Solving:	Stress period:	2	Time step:	100
Solving:	Stress period:	2	Time step:	101
Solving:	Stress period:	2	Time step:	102
Solving:	Stress period:	2	Time step:	103
Solving:	Stress period:	2	Time step:	104
Solving:	Stress period:	2	Time step:	105
Solving:	Stress period:	2	Time step:	106
Solving:	Stress period:	2	Time step:	107
Solving:	Stress period:	2	Time step:	108
Solving:	Stress period:	2	Time step:	109
Solving:	Stress period:	2	Time step:	110
Solving:	Stress period:	2	Time step:	111
Solving:	Stress period:	2	Time step:	112
Solving:	Stress period:	2	Time step:	113
Solving:	Stress period:	2	Time step:	114
Solving:	Stress period:	2	Time step:	115
Solving:	Stress period:	2	Time step:	116
Solving:	Stress period:	2	Time step:	117
Solving:	Stress period:	2	Time step:	118
Solving:	Stress period:	2	Time step:	119
Solving:	Stress period:	2	Time step:	120
Solving:	Stress period:	3	Time step:	1
Solving:	Stress period:	3	Time step:	2
Solving:	Stress period:	3	Time step:	3
Solving:	Stress period:	3	Time step:	4
Solving:	Stress period:	3	Time step:	5
Solving:	Stress period:	3	Time step:	6
Solving:	Stress period:	3	Time step:	7
Solving:	Stress period:	3	Time step:	8
Solving:	Stress period:	3	Time step:	9
Solving:	Stress period:	3	Time step:	10
Solving:	Stress period:	3	Time step:	11
Solving:	Stress period:	3	Time step:	12
Solving:	Stress period:	3	Time step:	13
Solving:	Stress period:	3	Time step:	14
Solving:	Stress period:	3	Time step:	15

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	16
Solving:	Stress period:	3	Time step:	17
Solving:	Stress period:	3	Time step:	18
Solving:	Stress period:	3	Time step:	19
Solving:	Stress period:	3	Time step:	20
Solving:	Stress period:	3	Time step:	21
Solving:	Stress period:	3	Time step:	22
Solving:	Stress period:	3	Time step:	23
Solving:	Stress period:	3	Time step:	24
Solving:	Stress period:	3	Time step:	25
Solving:	Stress period:	3	Time step:	26
Solving:	Stress period:	3	Time step:	27
Solving:	Stress period:	3	Time step:	28
Solving:	Stress period:	3	Time step:	29
Solving:	Stress period:	3	Time step:	30
Solving:	Stress period:	3	Time step:	31
Solving:	Stress period:	3	Time step:	32
Solving:	Stress period:	3	Time step:	33
Solving:	Stress period:	3	Time step:	34
Solving:	Stress period:	3	Time step:	35
Solving:	Stress period:	3	Time step:	36
Solving:	Stress period:	3	Time step:	37
Solving:	Stress period:	3	Time step:	38
Solving:	Stress period:	3	Time step:	39
Solving:	Stress period:	3	Time step:	40
Solving:	Stress period:	3	Time step:	41
Solving:	Stress period:	3	Time step:	42
Solving:	Stress period:	3	Time step:	43
Solving:	Stress period:	3	Time step:	44
Solving:	Stress period:	3	Time step:	45
Solving:	Stress period:	3	Time step:	46
Solving:	Stress period:	3	Time step:	47
Solving:	Stress period:	3	Time step:	48
Solving:	Stress period:	3	Time step:	49
Solving:	Stress period:	3	Time step:	50
Solving:	Stress period:	3	Time step:	51
Solving:	Stress period:	3	Time step:	52
Solving:	Stress period:	3	Time step:	53
Solving:	Stress period:	3	Time step:	54
Solving:	Stress period:	3	Time step:	55
Solving:	Stress period:	3	Time step:	56
Solving:	Stress period:	3	Time step:	57
Solving:	Stress period:	3	Time step:	58
Solving:	Stress period:	3	Time step:	59
Solving:	Stress period:	3	Time step:	60
Solving:	Stress period:	3	Time step:	61
Solving:	Stress period:	3	Time step:	62
Solving:	Stress period:	3	Time step:	63
Solving:	Stress period:	3	Time step:	64
Solving:	Stress period:	3	Time step:	65
Solving:	Stress period:	3	Time step:	66
Solving:	Stress period:	3	Time step:	67

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	68
Solving:	Stress period:	3	Time step:	69
Solving:	Stress period:	3	Time step:	70
Solving:	Stress period:	3	Time step:	71
Solving:	Stress period:	3	Time step:	72
Solving:	Stress period:	3	Time step:	73
Solving:	Stress period:	3	Time step:	74
Solving:	Stress period:	3	Time step:	75
Solving:	Stress period:	3	Time step:	76
Solving:	Stress period:	3	Time step:	77
Solving:	Stress period:	3	Time step:	78
Solving:	Stress period:	3	Time step:	79
Solving:	Stress period:	3	Time step:	80
Solving:	Stress period:	3	Time step:	81
Solving:	Stress period:	3	Time step:	82
Solving:	Stress period:	3	Time step:	83
Solving:	Stress period:	3	Time step:	84
Solving:	Stress period:	3	Time step:	85
Solving:	Stress period:	3	Time step:	86
Solving:	Stress period:	3	Time step:	87
Solving:	Stress period:	3	Time step:	88
Solving:	Stress period:	3	Time step:	89
Solving:	Stress period:	3	Time step:	90
Solving:	Stress period:	3	Time step:	91
Solving:	Stress period:	3	Time step:	92
Solving:	Stress period:	3	Time step:	93
Solving:	Stress period:	3	Time step:	94
Solving:	Stress period:	3	Time step:	95
Solving:	Stress period:	3	Time step:	96
Solving:	Stress period:	3	Time step:	97
Solving:	Stress period:	3	Time step:	98
Solving:	Stress period:	3	Time step:	99
Solving:	Stress period:	3	Time step:	100
Solving:	Stress period:	3	Time step:	101
Solving:	Stress period:	3	Time step:	102
Solving:	Stress period:	3	Time step:	103
Solving:	Stress period:	3	Time step:	104
Solving:	Stress period:	3	Time step:	105
Solving:	Stress period:	3	Time step:	106
Solving:	Stress period:	3	Time step:	107
Solving:	Stress period:	3	Time step:	108
Solving:	Stress period:	3	Time step:	109
Solving:	Stress period:	3	Time step:	110
Solving:	Stress period:	3	Time step:	111
Solving:	Stress period:	3	Time step:	112
Solving:	Stress period:	3	Time step:	113
Solving:	Stress period:	3	Time step:	114
Solving:	Stress period:	3	Time step:	115
Solving:	Stress period:	3	Time step:	116
Solving:	Stress period:	3	Time step:	117
Solving:	Stress period:	3	Time step:	118
Solving:	Stress period:	3	Time step:	119

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	120
Solving:	Stress period:	4	Time step:	1
Solving:	Stress period:	4	Time step:	2
Solving:	Stress period:	4	Time step:	3
Solving:	Stress period:	4	Time step:	4
Solving:	Stress period:	4	Time step:	5
Solving:	Stress period:	4	Time step:	6
Solving:	Stress period:	4	Time step:	7
Solving:	Stress period:	4	Time step:	8
Solving:	Stress period:	4	Time step:	9
Solving:	Stress period:	4	Time step:	10
Solving:	Stress period:	4	Time step:	11
Solving:	Stress period:	4	Time step:	12
Solving:	Stress period:	4	Time step:	13
Solving:	Stress period:	4	Time step:	14
Solving:	Stress period:	4	Time step:	15
Solving:	Stress period:	4	Time step:	16
Solving:	Stress period:	4	Time step:	17
Solving:	Stress period:	4	Time step:	18
Solving:	Stress period:	4	Time step:	19
Solving:	Stress period:	4	Time step:	20
Solving:	Stress period:	4	Time step:	21
Solving:	Stress period:	4	Time step:	22
Solving:	Stress period:	4	Time step:	23
Solving:	Stress period:	4	Time step:	24
Solving:	Stress period:	4	Time step:	25
Solving:	Stress period:	4	Time step:	26
Solving:	Stress period:	4	Time step:	27
Solving:	Stress period:	4	Time step:	28
Solving:	Stress period:	4	Time step:	29
Solving:	Stress period:	4	Time step:	30
Solving:	Stress period:	4	Time step:	31
Solving:	Stress period:	4	Time step:	32
Solving:	Stress period:	4	Time step:	33
Solving:	Stress period:	4	Time step:	34
Solving:	Stress period:	4	Time step:	35
Solving:	Stress period:	4	Time step:	36
Solving:	Stress period:	4	Time step:	37
Solving:	Stress period:	4	Time step:	38
Solving:	Stress period:	4	Time step:	39
Solving:	Stress period:	4	Time step:	40
Solving:	Stress period:	4	Time step:	41
Solving:	Stress period:	4	Time step:	42
Solving:	Stress period:	4	Time step:	43
Solving:	Stress period:	4	Time step:	44
Solving:	Stress period:	4	Time step:	45
Solving:	Stress period:	4	Time step:	46
Solving:	Stress period:	4	Time step:	47
Solving:	Stress period:	4	Time step:	48
Solving:	Stress period:	4	Time step:	49
Solving:	Stress period:	4	Time step:	50
Solving:	Stress period:	4	Time step:	51

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	52
Solving:	Stress period:	4	Time step:	53
Solving:	Stress period:	4	Time step:	54
Solving:	Stress period:	4	Time step:	55
Solving:	Stress period:	4	Time step:	56
Solving:	Stress period:	4	Time step:	57
Solving:	Stress period:	4	Time step:	58
Solving:	Stress period:	4	Time step:	59
Solving:	Stress period:	4	Time step:	60
Solving:	Stress period:	4	Time step:	61
Solving:	Stress period:	4	Time step:	62
Solving:	Stress period:	4	Time step:	63
Solving:	Stress period:	4	Time step:	64
Solving:	Stress period:	4	Time step:	65
Solving:	Stress period:	4	Time step:	66
Solving:	Stress period:	4	Time step:	67
Solving:	Stress period:	4	Time step:	68
Solving:	Stress period:	4	Time step:	69
Solving:	Stress period:	4	Time step:	70
Solving:	Stress period:	4	Time step:	71
Solving:	Stress period:	4	Time step:	72
Solving:	Stress period:	4	Time step:	73
Solving:	Stress period:	4	Time step:	74
Solving:	Stress period:	4	Time step:	75
Solving:	Stress period:	4	Time step:	76
Solving:	Stress period:	4	Time step:	77
Solving:	Stress period:	4	Time step:	78
Solving:	Stress period:	4	Time step:	79
Solving:	Stress period:	4	Time step:	80
Solving:	Stress period:	4	Time step:	81
Solving:	Stress period:	4	Time step:	82
Solving:	Stress period:	4	Time step:	83
Solving:	Stress period:	4	Time step:	84
Solving:	Stress period:	4	Time step:	85
Solving:	Stress period:	4	Time step:	86
Solving:	Stress period:	4	Time step:	87
Solving:	Stress period:	4	Time step:	88
Solving:	Stress period:	4	Time step:	89
Solving:	Stress period:	4	Time step:	90
Solving:	Stress period:	4	Time step:	91
Solving:	Stress period:	4	Time step:	92
Solving:	Stress period:	4	Time step:	93
Solving:	Stress period:	4	Time step:	94
Solving:	Stress period:	4	Time step:	95
Solving:	Stress period:	4	Time step:	96
Solving:	Stress period:	4	Time step:	97
Solving:	Stress period:	4	Time step:	98
Solving:	Stress period:	4	Time step:	99
Solving:	Stress period:	4	Time step:	100
Solving:	Stress period:	4	Time step:	101
Solving:	Stress period:	4	Time step:	102
Solving:	Stress period:	4	Time step:	103

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 4 Time step: 104
Solving: Stress period: 4 Time step: 105
Solving: Stress period: 4 Time step: 106
Solving: Stress period: 4 Time step: 107
Solving: Stress period: 4 Time step: 108
Solving: Stress period: 4 Time step: 109
Solving: Stress period: 4 Time step: 110
Solving: Stress period: 4 Time step: 111
Solving: Stress period: 4 Time step: 112
Solving: Stress period: 4 Time step: 113
Solving: Stress period: 4 Time step: 114
Solving: Stress period: 4 Time step: 115
Solving: Stress period: 4 Time step: 116
Solving: Stress period: 4 Time step: 117
Solving: Stress period: 4 Time step: 118
Solving: Stress period: 4 Time step: 119
Solving: Stress period: 4 Time step: 120

```

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:34

Elapsed run time: 0.382 Seconds

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
 2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.
- Normal termination of simulation.

Method 2: Initialize obs through ghb.obs.initialize

```

[4]: # build ghb stress period data
ghb_spd = {}
ghb_period = []
for layer, cond in zip(range(1, 3), [15.0, 1500.0]):
    for row in range(0, 15):
        ghb_period.append(((layer, row, 9), 1.0, cond, "Estuary-L2"))
ghb_spd[0] = ghb_period

# build ghb package
ghb = flopy.mf6.modflow.mfgwfghb.ModflowGwfghb(
    model,
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    maxbound=30,
    stress_period_data=ghb_spd,

```

(continues on next page)

(continued from previous page)

```

    pname="ghb",
)

# build obs data
ghb_obs = {
    ("ghb_obs.csv", "binary"): [
        ("ghb-2-6-10", "GHB", (1, 5, 9)),
        ("ghb-3-6-10", "GHB", (2, 5, 9)),
    ],
    "ghb_flows.csv": [
        ("Estuary2", "GHB", "Estuary-L2"),
        ("Estuary3", "GHB", "Estuary-L3"),
    ],
}

# initialize obs package
ghb_obs.initialize(
    filename="child_pkgs_test.ghb_obs",
    digits=9,
    print_input=True,
    continuous=ghb_obs,
)

sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

# clean up for next example
model.remove_package("ghb")

```

```

writing simulation...
writing simulation name file...
writing simulation tdis package...
writing solution package ims_1...
writing model child_pkgs...
writing model name file...
writing package dis...
writing package ic...
writing package npf...
writing package oc...
writing package sto...
writing package ghb...
writing package obs_0...

MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64

```

(continues on next page)

(continued from previous page)

Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:34

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5
Solving:	Stress period:	2	Time step:	6
Solving:	Stress period:	2	Time step:	7
Solving:	Stress period:	2	Time step:	8
Solving:	Stress period:	2	Time step:	9
Solving:	Stress period:	2	Time step:	10
Solving:	Stress period:	2	Time step:	11
Solving:	Stress period:	2	Time step:	12
Solving:	Stress period:	2	Time step:	13
Solving:	Stress period:	2	Time step:	14
Solving:	Stress period:	2	Time step:	15
Solving:	Stress period:	2	Time step:	16
Solving:	Stress period:	2	Time step:	17
Solving:	Stress period:	2	Time step:	18
Solving:	Stress period:	2	Time step:	19
Solving:	Stress period:	2	Time step:	20
Solving:	Stress period:	2	Time step:	21
Solving:	Stress period:	2	Time step:	22
Solving:	Stress period:	2	Time step:	23
Solving:	Stress period:	2	Time step:	24
Solving:	Stress period:	2	Time step:	25
Solving:	Stress period:	2	Time step:	26
Solving:	Stress period:	2	Time step:	27
Solving:	Stress period:	2	Time step:	28
Solving:	Stress period:	2	Time step:	29

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	30
Solving:	Stress period:	2	Time step:	31
Solving:	Stress period:	2	Time step:	32
Solving:	Stress period:	2	Time step:	33
Solving:	Stress period:	2	Time step:	34
Solving:	Stress period:	2	Time step:	35
Solving:	Stress period:	2	Time step:	36
Solving:	Stress period:	2	Time step:	37
Solving:	Stress period:	2	Time step:	38
Solving:	Stress period:	2	Time step:	39
Solving:	Stress period:	2	Time step:	40
Solving:	Stress period:	2	Time step:	41
Solving:	Stress period:	2	Time step:	42
Solving:	Stress period:	2	Time step:	43
Solving:	Stress period:	2	Time step:	44
Solving:	Stress period:	2	Time step:	45
Solving:	Stress period:	2	Time step:	46
Solving:	Stress period:	2	Time step:	47
Solving:	Stress period:	2	Time step:	48
Solving:	Stress period:	2	Time step:	49
Solving:	Stress period:	2	Time step:	50
Solving:	Stress period:	2	Time step:	51
Solving:	Stress period:	2	Time step:	52
Solving:	Stress period:	2	Time step:	53
Solving:	Stress period:	2	Time step:	54
Solving:	Stress period:	2	Time step:	55
Solving:	Stress period:	2	Time step:	56
Solving:	Stress period:	2	Time step:	57
Solving:	Stress period:	2	Time step:	58
Solving:	Stress period:	2	Time step:	59
Solving:	Stress period:	2	Time step:	60
Solving:	Stress period:	2	Time step:	61
Solving:	Stress period:	2	Time step:	62
Solving:	Stress period:	2	Time step:	63
Solving:	Stress period:	2	Time step:	64
Solving:	Stress period:	2	Time step:	65
Solving:	Stress period:	2	Time step:	66
Solving:	Stress period:	2	Time step:	67
Solving:	Stress period:	2	Time step:	68
Solving:	Stress period:	2	Time step:	69
Solving:	Stress period:	2	Time step:	70
Solving:	Stress period:	2	Time step:	71
Solving:	Stress period:	2	Time step:	72
Solving:	Stress period:	2	Time step:	73
Solving:	Stress period:	2	Time step:	74
Solving:	Stress period:	2	Time step:	75
Solving:	Stress period:	2	Time step:	76
Solving:	Stress period:	2	Time step:	77
Solving:	Stress period:	2	Time step:	78
Solving:	Stress period:	2	Time step:	79
Solving:	Stress period:	2	Time step:	80
Solving:	Stress period:	2	Time step:	81

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	82
Solving:	Stress period:	2	Time step:	83
Solving:	Stress period:	2	Time step:	84
Solving:	Stress period:	2	Time step:	85
Solving:	Stress period:	2	Time step:	86
Solving:	Stress period:	2	Time step:	87
Solving:	Stress period:	2	Time step:	88
Solving:	Stress period:	2	Time step:	89
Solving:	Stress period:	2	Time step:	90
Solving:	Stress period:	2	Time step:	91
Solving:	Stress period:	2	Time step:	92
Solving:	Stress period:	2	Time step:	93
Solving:	Stress period:	2	Time step:	94
Solving:	Stress period:	2	Time step:	95
Solving:	Stress period:	2	Time step:	96
Solving:	Stress period:	2	Time step:	97
Solving:	Stress period:	2	Time step:	98
Solving:	Stress period:	2	Time step:	99
Solving:	Stress period:	2	Time step:	100
Solving:	Stress period:	2	Time step:	101
Solving:	Stress period:	2	Time step:	102
Solving:	Stress period:	2	Time step:	103
Solving:	Stress period:	2	Time step:	104
Solving:	Stress period:	2	Time step:	105
Solving:	Stress period:	2	Time step:	106
Solving:	Stress period:	2	Time step:	107
Solving:	Stress period:	2	Time step:	108
Solving:	Stress period:	2	Time step:	109
Solving:	Stress period:	2	Time step:	110
Solving:	Stress period:	2	Time step:	111
Solving:	Stress period:	2	Time step:	112
Solving:	Stress period:	2	Time step:	113
Solving:	Stress period:	2	Time step:	114
Solving:	Stress period:	2	Time step:	115
Solving:	Stress period:	2	Time step:	116
Solving:	Stress period:	2	Time step:	117
Solving:	Stress period:	2	Time step:	118
Solving:	Stress period:	2	Time step:	119
Solving:	Stress period:	2	Time step:	120
Solving:	Stress period:	3	Time step:	1
Solving:	Stress period:	3	Time step:	2
Solving:	Stress period:	3	Time step:	3
Solving:	Stress period:	3	Time step:	4
Solving:	Stress period:	3	Time step:	5
Solving:	Stress period:	3	Time step:	6
Solving:	Stress period:	3	Time step:	7
Solving:	Stress period:	3	Time step:	8
Solving:	Stress period:	3	Time step:	9
Solving:	Stress period:	3	Time step:	10
Solving:	Stress period:	3	Time step:	11
Solving:	Stress period:	3	Time step:	12
Solving:	Stress period:	3	Time step:	13

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	14
Solving:	Stress period:	3	Time step:	15
Solving:	Stress period:	3	Time step:	16
Solving:	Stress period:	3	Time step:	17
Solving:	Stress period:	3	Time step:	18
Solving:	Stress period:	3	Time step:	19
Solving:	Stress period:	3	Time step:	20
Solving:	Stress period:	3	Time step:	21
Solving:	Stress period:	3	Time step:	22
Solving:	Stress period:	3	Time step:	23
Solving:	Stress period:	3	Time step:	24
Solving:	Stress period:	3	Time step:	25
Solving:	Stress period:	3	Time step:	26
Solving:	Stress period:	3	Time step:	27
Solving:	Stress period:	3	Time step:	28
Solving:	Stress period:	3	Time step:	29
Solving:	Stress period:	3	Time step:	30
Solving:	Stress period:	3	Time step:	31
Solving:	Stress period:	3	Time step:	32
Solving:	Stress period:	3	Time step:	33
Solving:	Stress period:	3	Time step:	34
Solving:	Stress period:	3	Time step:	35
Solving:	Stress period:	3	Time step:	36
Solving:	Stress period:	3	Time step:	37
Solving:	Stress period:	3	Time step:	38
Solving:	Stress period:	3	Time step:	39
Solving:	Stress period:	3	Time step:	40
Solving:	Stress period:	3	Time step:	41
Solving:	Stress period:	3	Time step:	42
Solving:	Stress period:	3	Time step:	43
Solving:	Stress period:	3	Time step:	44
Solving:	Stress period:	3	Time step:	45
Solving:	Stress period:	3	Time step:	46
Solving:	Stress period:	3	Time step:	47
Solving:	Stress period:	3	Time step:	48
Solving:	Stress period:	3	Time step:	49
Solving:	Stress period:	3	Time step:	50
Solving:	Stress period:	3	Time step:	51
Solving:	Stress period:	3	Time step:	52
Solving:	Stress period:	3	Time step:	53
Solving:	Stress period:	3	Time step:	54
Solving:	Stress period:	3	Time step:	55
Solving:	Stress period:	3	Time step:	56
Solving:	Stress period:	3	Time step:	57
Solving:	Stress period:	3	Time step:	58
Solving:	Stress period:	3	Time step:	59
Solving:	Stress period:	3	Time step:	60
Solving:	Stress period:	3	Time step:	61
Solving:	Stress period:	3	Time step:	62
Solving:	Stress period:	3	Time step:	63
Solving:	Stress period:	3	Time step:	64
Solving:	Stress period:	3	Time step:	65

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	66
Solving:	Stress period:	3	Time step:	67
Solving:	Stress period:	3	Time step:	68
Solving:	Stress period:	3	Time step:	69
Solving:	Stress period:	3	Time step:	70
Solving:	Stress period:	3	Time step:	71
Solving:	Stress period:	3	Time step:	72
Solving:	Stress period:	3	Time step:	73
Solving:	Stress period:	3	Time step:	74
Solving:	Stress period:	3	Time step:	75
Solving:	Stress period:	3	Time step:	76
Solving:	Stress period:	3	Time step:	77
Solving:	Stress period:	3	Time step:	78
Solving:	Stress period:	3	Time step:	79
Solving:	Stress period:	3	Time step:	80
Solving:	Stress period:	3	Time step:	81
Solving:	Stress period:	3	Time step:	82
Solving:	Stress period:	3	Time step:	83
Solving:	Stress period:	3	Time step:	84
Solving:	Stress period:	3	Time step:	85
Solving:	Stress period:	3	Time step:	86
Solving:	Stress period:	3	Time step:	87
Solving:	Stress period:	3	Time step:	88
Solving:	Stress period:	3	Time step:	89
Solving:	Stress period:	3	Time step:	90
Solving:	Stress period:	3	Time step:	91
Solving:	Stress period:	3	Time step:	92
Solving:	Stress period:	3	Time step:	93
Solving:	Stress period:	3	Time step:	94
Solving:	Stress period:	3	Time step:	95
Solving:	Stress period:	3	Time step:	96
Solving:	Stress period:	3	Time step:	97
Solving:	Stress period:	3	Time step:	98
Solving:	Stress period:	3	Time step:	99
Solving:	Stress period:	3	Time step:	100
Solving:	Stress period:	3	Time step:	101
Solving:	Stress period:	3	Time step:	102
Solving:	Stress period:	3	Time step:	103
Solving:	Stress period:	3	Time step:	104
Solving:	Stress period:	3	Time step:	105
Solving:	Stress period:	3	Time step:	106
Solving:	Stress period:	3	Time step:	107
Solving:	Stress period:	3	Time step:	108
Solving:	Stress period:	3	Time step:	109
Solving:	Stress period:	3	Time step:	110
Solving:	Stress period:	3	Time step:	111
Solving:	Stress period:	3	Time step:	112
Solving:	Stress period:	3	Time step:	113
Solving:	Stress period:	3	Time step:	114
Solving:	Stress period:	3	Time step:	115
Solving:	Stress period:	3	Time step:	116
Solving:	Stress period:	3	Time step:	117

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	118
Solving:	Stress period:	3	Time step:	119
Solving:	Stress period:	3	Time step:	120
Solving:	Stress period:	4	Time step:	1
Solving:	Stress period:	4	Time step:	2
Solving:	Stress period:	4	Time step:	3
Solving:	Stress period:	4	Time step:	4
Solving:	Stress period:	4	Time step:	5
Solving:	Stress period:	4	Time step:	6
Solving:	Stress period:	4	Time step:	7
Solving:	Stress period:	4	Time step:	8
Solving:	Stress period:	4	Time step:	9
Solving:	Stress period:	4	Time step:	10
Solving:	Stress period:	4	Time step:	11
Solving:	Stress period:	4	Time step:	12
Solving:	Stress period:	4	Time step:	13
Solving:	Stress period:	4	Time step:	14
Solving:	Stress period:	4	Time step:	15
Solving:	Stress period:	4	Time step:	16
Solving:	Stress period:	4	Time step:	17
Solving:	Stress period:	4	Time step:	18
Solving:	Stress period:	4	Time step:	19
Solving:	Stress period:	4	Time step:	20
Solving:	Stress period:	4	Time step:	21
Solving:	Stress period:	4	Time step:	22
Solving:	Stress period:	4	Time step:	23
Solving:	Stress period:	4	Time step:	24
Solving:	Stress period:	4	Time step:	25
Solving:	Stress period:	4	Time step:	26
Solving:	Stress period:	4	Time step:	27
Solving:	Stress period:	4	Time step:	28
Solving:	Stress period:	4	Time step:	29
Solving:	Stress period:	4	Time step:	30
Solving:	Stress period:	4	Time step:	31
Solving:	Stress period:	4	Time step:	32
Solving:	Stress period:	4	Time step:	33
Solving:	Stress period:	4	Time step:	34
Solving:	Stress period:	4	Time step:	35
Solving:	Stress period:	4	Time step:	36
Solving:	Stress period:	4	Time step:	37
Solving:	Stress period:	4	Time step:	38
Solving:	Stress period:	4	Time step:	39
Solving:	Stress period:	4	Time step:	40
Solving:	Stress period:	4	Time step:	41
Solving:	Stress period:	4	Time step:	42
Solving:	Stress period:	4	Time step:	43
Solving:	Stress period:	4	Time step:	44
Solving:	Stress period:	4	Time step:	45
Solving:	Stress period:	4	Time step:	46
Solving:	Stress period:	4	Time step:	47
Solving:	Stress period:	4	Time step:	48
Solving:	Stress period:	4	Time step:	49

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	50
Solving:	Stress period:	4	Time step:	51
Solving:	Stress period:	4	Time step:	52
Solving:	Stress period:	4	Time step:	53
Solving:	Stress period:	4	Time step:	54
Solving:	Stress period:	4	Time step:	55
Solving:	Stress period:	4	Time step:	56
Solving:	Stress period:	4	Time step:	57
Solving:	Stress period:	4	Time step:	58
Solving:	Stress period:	4	Time step:	59
Solving:	Stress period:	4	Time step:	60
Solving:	Stress period:	4	Time step:	61
Solving:	Stress period:	4	Time step:	62
Solving:	Stress period:	4	Time step:	63
Solving:	Stress period:	4	Time step:	64
Solving:	Stress period:	4	Time step:	65
Solving:	Stress period:	4	Time step:	66
Solving:	Stress period:	4	Time step:	67
Solving:	Stress period:	4	Time step:	68
Solving:	Stress period:	4	Time step:	69
Solving:	Stress period:	4	Time step:	70
Solving:	Stress period:	4	Time step:	71
Solving:	Stress period:	4	Time step:	72
Solving:	Stress period:	4	Time step:	73
Solving:	Stress period:	4	Time step:	74
Solving:	Stress period:	4	Time step:	75
Solving:	Stress period:	4	Time step:	76
Solving:	Stress period:	4	Time step:	77
Solving:	Stress period:	4	Time step:	78
Solving:	Stress period:	4	Time step:	79
Solving:	Stress period:	4	Time step:	80
Solving:	Stress period:	4	Time step:	81
Solving:	Stress period:	4	Time step:	82
Solving:	Stress period:	4	Time step:	83
Solving:	Stress period:	4	Time step:	84
Solving:	Stress period:	4	Time step:	85
Solving:	Stress period:	4	Time step:	86
Solving:	Stress period:	4	Time step:	87
Solving:	Stress period:	4	Time step:	88
Solving:	Stress period:	4	Time step:	89
Solving:	Stress period:	4	Time step:	90
Solving:	Stress period:	4	Time step:	91
Solving:	Stress period:	4	Time step:	92
Solving:	Stress period:	4	Time step:	93
Solving:	Stress period:	4	Time step:	94
Solving:	Stress period:	4	Time step:	95
Solving:	Stress period:	4	Time step:	96
Solving:	Stress period:	4	Time step:	97
Solving:	Stress period:	4	Time step:	98
Solving:	Stress period:	4	Time step:	99
Solving:	Stress period:	4	Time step:	100
Solving:	Stress period:	4	Time step:	101

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 4 Time step: 102
Solving: Stress period: 4 Time step: 103
Solving: Stress period: 4 Time step: 104
Solving: Stress period: 4 Time step: 105
Solving: Stress period: 4 Time step: 106
Solving: Stress period: 4 Time step: 107
Solving: Stress period: 4 Time step: 108
Solving: Stress period: 4 Time step: 109
Solving: Stress period: 4 Time step: 110
Solving: Stress period: 4 Time step: 111
Solving: Stress period: 4 Time step: 112
Solving: Stress period: 4 Time step: 113
Solving: Stress period: 4 Time step: 114
Solving: Stress period: 4 Time step: 115
Solving: Stress period: 4 Time step: 116
Solving: Stress period: 4 Time step: 117
Solving: Stress period: 4 Time step: 118
Solving: Stress period: 4 Time step: 119
Solving: Stress period: 4 Time step: 120

```

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:34

Elapsed run time: 0.330 Seconds

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
 2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.
- Normal termination of simulation.

Method 3: Pass observations a dictionary of anything that could be passed to `ghb.obs.initialize`

```

[5]: # build ghb stress period data
ghb_spd = {}
ghb_period = []
for layer, cond in zip(range(1, 3), [15.0, 1500.0]):
    for row in range(0, 15):
        ghb_period.append((layer, row, 9), 1.0, cond, "Estuary-L2")
ghb_spd[0] = ghb_period

# build obs data
ghb_obs = {
    ("ghb_obs.csv", "binary"): [
        ("ghb-2-6-10", "GHB", (1, 5, 9)),
        ("ghb-3-6-10", "GHB", (2, 5, 9)),
    ],
    "ghb_flows.csv": [

```

(continues on next page)

(continued from previous page)

```

        ("Estuary2", "GHB", "Estuary-L2"),
        ("Estuary3", "GHB", "Estuary-L3"),
    ],
}
# append additional obs attributes to obs dictionary
ghb_obs["digits"] = 7
ghb_obs["print_input"] = False
ghb_obs["filename"] = "method_3.obs"

# build ghb package
ghb_package = flopy.mf6.modflow.mfgwfghb.ModflowGwfghb(
    model,
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    observations=ghb_obs,
    pname="ghb",
    maxbound=30,
    stress_period_data=ghb_spd,
)

sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

# clean up for next example
model.remove_package("ghb")

```

```

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims_1...
  writing model child_pkgs...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
    writing package oc...
    writing package sto...
    writing package ghb...
    writing package obs_0...

                                MODFLOW 6
      U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
      VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0

```

(continues on next page)

(continued from previous page)

Build 20220726_000000

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:34

Writing simulation list file: mfsim.lst

Using Simulation name file: mfsim.nam

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5
Solving:	Stress period:	2	Time step:	6
Solving:	Stress period:	2	Time step:	7
Solving:	Stress period:	2	Time step:	8
Solving:	Stress period:	2	Time step:	9
Solving:	Stress period:	2	Time step:	10
Solving:	Stress period:	2	Time step:	11
Solving:	Stress period:	2	Time step:	12
Solving:	Stress period:	2	Time step:	13
Solving:	Stress period:	2	Time step:	14
Solving:	Stress period:	2	Time step:	15
Solving:	Stress period:	2	Time step:	16
Solving:	Stress period:	2	Time step:	17
Solving:	Stress period:	2	Time step:	18
Solving:	Stress period:	2	Time step:	19
Solving:	Stress period:	2	Time step:	20
Solving:	Stress period:	2	Time step:	21
Solving:	Stress period:	2	Time step:	22
Solving:	Stress period:	2	Time step:	23
Solving:	Stress period:	2	Time step:	24
Solving:	Stress period:	2	Time step:	25
Solving:	Stress period:	2	Time step:	26
Solving:	Stress period:	2	Time step:	27
Solving:	Stress period:	2	Time step:	28
Solving:	Stress period:	2	Time step:	29
Solving:	Stress period:	2	Time step:	30

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	31
Solving:	Stress period:	2	Time step:	32
Solving:	Stress period:	2	Time step:	33
Solving:	Stress period:	2	Time step:	34
Solving:	Stress period:	2	Time step:	35
Solving:	Stress period:	2	Time step:	36
Solving:	Stress period:	2	Time step:	37
Solving:	Stress period:	2	Time step:	38
Solving:	Stress period:	2	Time step:	39
Solving:	Stress period:	2	Time step:	40
Solving:	Stress period:	2	Time step:	41
Solving:	Stress period:	2	Time step:	42
Solving:	Stress period:	2	Time step:	43
Solving:	Stress period:	2	Time step:	44
Solving:	Stress period:	2	Time step:	45
Solving:	Stress period:	2	Time step:	46
Solving:	Stress period:	2	Time step:	47
Solving:	Stress period:	2	Time step:	48
Solving:	Stress period:	2	Time step:	49
Solving:	Stress period:	2	Time step:	50
Solving:	Stress period:	2	Time step:	51
Solving:	Stress period:	2	Time step:	52
Solving:	Stress period:	2	Time step:	53
Solving:	Stress period:	2	Time step:	54
Solving:	Stress period:	2	Time step:	55
Solving:	Stress period:	2	Time step:	56
Solving:	Stress period:	2	Time step:	57
Solving:	Stress period:	2	Time step:	58
Solving:	Stress period:	2	Time step:	59
Solving:	Stress period:	2	Time step:	60
Solving:	Stress period:	2	Time step:	61
Solving:	Stress period:	2	Time step:	62
Solving:	Stress period:	2	Time step:	63
Solving:	Stress period:	2	Time step:	64
Solving:	Stress period:	2	Time step:	65
Solving:	Stress period:	2	Time step:	66
Solving:	Stress period:	2	Time step:	67
Solving:	Stress period:	2	Time step:	68
Solving:	Stress period:	2	Time step:	69
Solving:	Stress period:	2	Time step:	70
Solving:	Stress period:	2	Time step:	71
Solving:	Stress period:	2	Time step:	72
Solving:	Stress period:	2	Time step:	73
Solving:	Stress period:	2	Time step:	74
Solving:	Stress period:	2	Time step:	75
Solving:	Stress period:	2	Time step:	76
Solving:	Stress period:	2	Time step:	77
Solving:	Stress period:	2	Time step:	78
Solving:	Stress period:	2	Time step:	79
Solving:	Stress period:	2	Time step:	80
Solving:	Stress period:	2	Time step:	81
Solving:	Stress period:	2	Time step:	82

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	83
Solving:	Stress period:	2	Time step:	84
Solving:	Stress period:	2	Time step:	85
Solving:	Stress period:	2	Time step:	86
Solving:	Stress period:	2	Time step:	87
Solving:	Stress period:	2	Time step:	88
Solving:	Stress period:	2	Time step:	89
Solving:	Stress period:	2	Time step:	90
Solving:	Stress period:	2	Time step:	91
Solving:	Stress period:	2	Time step:	92
Solving:	Stress period:	2	Time step:	93
Solving:	Stress period:	2	Time step:	94
Solving:	Stress period:	2	Time step:	95
Solving:	Stress period:	2	Time step:	96
Solving:	Stress period:	2	Time step:	97
Solving:	Stress period:	2	Time step:	98
Solving:	Stress period:	2	Time step:	99
Solving:	Stress period:	2	Time step:	100
Solving:	Stress period:	2	Time step:	101
Solving:	Stress period:	2	Time step:	102
Solving:	Stress period:	2	Time step:	103
Solving:	Stress period:	2	Time step:	104
Solving:	Stress period:	2	Time step:	105
Solving:	Stress period:	2	Time step:	106
Solving:	Stress period:	2	Time step:	107
Solving:	Stress period:	2	Time step:	108
Solving:	Stress period:	2	Time step:	109
Solving:	Stress period:	2	Time step:	110
Solving:	Stress period:	2	Time step:	111
Solving:	Stress period:	2	Time step:	112
Solving:	Stress period:	2	Time step:	113
Solving:	Stress period:	2	Time step:	114
Solving:	Stress period:	2	Time step:	115
Solving:	Stress period:	2	Time step:	116
Solving:	Stress period:	2	Time step:	117
Solving:	Stress period:	2	Time step:	118
Solving:	Stress period:	2	Time step:	119
Solving:	Stress period:	2	Time step:	120
Solving:	Stress period:	3	Time step:	1
Solving:	Stress period:	3	Time step:	2
Solving:	Stress period:	3	Time step:	3
Solving:	Stress period:	3	Time step:	4
Solving:	Stress period:	3	Time step:	5
Solving:	Stress period:	3	Time step:	6
Solving:	Stress period:	3	Time step:	7
Solving:	Stress period:	3	Time step:	8
Solving:	Stress period:	3	Time step:	9
Solving:	Stress period:	3	Time step:	10
Solving:	Stress period:	3	Time step:	11
Solving:	Stress period:	3	Time step:	12
Solving:	Stress period:	3	Time step:	13
Solving:	Stress period:	3	Time step:	14

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	15
Solving:	Stress period:	3	Time step:	16
Solving:	Stress period:	3	Time step:	17
Solving:	Stress period:	3	Time step:	18
Solving:	Stress period:	3	Time step:	19
Solving:	Stress period:	3	Time step:	20
Solving:	Stress period:	3	Time step:	21
Solving:	Stress period:	3	Time step:	22
Solving:	Stress period:	3	Time step:	23
Solving:	Stress period:	3	Time step:	24
Solving:	Stress period:	3	Time step:	25
Solving:	Stress period:	3	Time step:	26
Solving:	Stress period:	3	Time step:	27
Solving:	Stress period:	3	Time step:	28
Solving:	Stress period:	3	Time step:	29
Solving:	Stress period:	3	Time step:	30
Solving:	Stress period:	3	Time step:	31
Solving:	Stress period:	3	Time step:	32
Solving:	Stress period:	3	Time step:	33
Solving:	Stress period:	3	Time step:	34
Solving:	Stress period:	3	Time step:	35
Solving:	Stress period:	3	Time step:	36
Solving:	Stress period:	3	Time step:	37
Solving:	Stress period:	3	Time step:	38
Solving:	Stress period:	3	Time step:	39
Solving:	Stress period:	3	Time step:	40
Solving:	Stress period:	3	Time step:	41
Solving:	Stress period:	3	Time step:	42
Solving:	Stress period:	3	Time step:	43
Solving:	Stress period:	3	Time step:	44
Solving:	Stress period:	3	Time step:	45
Solving:	Stress period:	3	Time step:	46
Solving:	Stress period:	3	Time step:	47
Solving:	Stress period:	3	Time step:	48
Solving:	Stress period:	3	Time step:	49
Solving:	Stress period:	3	Time step:	50
Solving:	Stress period:	3	Time step:	51
Solving:	Stress period:	3	Time step:	52
Solving:	Stress period:	3	Time step:	53
Solving:	Stress period:	3	Time step:	54
Solving:	Stress period:	3	Time step:	55
Solving:	Stress period:	3	Time step:	56
Solving:	Stress period:	3	Time step:	57
Solving:	Stress period:	3	Time step:	58
Solving:	Stress period:	3	Time step:	59
Solving:	Stress period:	3	Time step:	60
Solving:	Stress period:	3	Time step:	61
Solving:	Stress period:	3	Time step:	62
Solving:	Stress period:	3	Time step:	63
Solving:	Stress period:	3	Time step:	64
Solving:	Stress period:	3	Time step:	65
Solving:	Stress period:	3	Time step:	66

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	67
Solving:	Stress period:	3	Time step:	68
Solving:	Stress period:	3	Time step:	69
Solving:	Stress period:	3	Time step:	70
Solving:	Stress period:	3	Time step:	71
Solving:	Stress period:	3	Time step:	72
Solving:	Stress period:	3	Time step:	73
Solving:	Stress period:	3	Time step:	74
Solving:	Stress period:	3	Time step:	75
Solving:	Stress period:	3	Time step:	76
Solving:	Stress period:	3	Time step:	77
Solving:	Stress period:	3	Time step:	78
Solving:	Stress period:	3	Time step:	79
Solving:	Stress period:	3	Time step:	80
Solving:	Stress period:	3	Time step:	81
Solving:	Stress period:	3	Time step:	82
Solving:	Stress period:	3	Time step:	83
Solving:	Stress period:	3	Time step:	84
Solving:	Stress period:	3	Time step:	85
Solving:	Stress period:	3	Time step:	86
Solving:	Stress period:	3	Time step:	87
Solving:	Stress period:	3	Time step:	88
Solving:	Stress period:	3	Time step:	89
Solving:	Stress period:	3	Time step:	90
Solving:	Stress period:	3	Time step:	91
Solving:	Stress period:	3	Time step:	92
Solving:	Stress period:	3	Time step:	93
Solving:	Stress period:	3	Time step:	94
Solving:	Stress period:	3	Time step:	95
Solving:	Stress period:	3	Time step:	96
Solving:	Stress period:	3	Time step:	97
Solving:	Stress period:	3	Time step:	98
Solving:	Stress period:	3	Time step:	99
Solving:	Stress period:	3	Time step:	100
Solving:	Stress period:	3	Time step:	101
Solving:	Stress period:	3	Time step:	102
Solving:	Stress period:	3	Time step:	103
Solving:	Stress period:	3	Time step:	104
Solving:	Stress period:	3	Time step:	105
Solving:	Stress period:	3	Time step:	106
Solving:	Stress period:	3	Time step:	107
Solving:	Stress period:	3	Time step:	108
Solving:	Stress period:	3	Time step:	109
Solving:	Stress period:	3	Time step:	110
Solving:	Stress period:	3	Time step:	111
Solving:	Stress period:	3	Time step:	112
Solving:	Stress period:	3	Time step:	113
Solving:	Stress period:	3	Time step:	114
Solving:	Stress period:	3	Time step:	115
Solving:	Stress period:	3	Time step:	116
Solving:	Stress period:	3	Time step:	117
Solving:	Stress period:	3	Time step:	118

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	119
Solving:	Stress period:	3	Time step:	120
Solving:	Stress period:	4	Time step:	1
Solving:	Stress period:	4	Time step:	2
Solving:	Stress period:	4	Time step:	3
Solving:	Stress period:	4	Time step:	4
Solving:	Stress period:	4	Time step:	5
Solving:	Stress period:	4	Time step:	6
Solving:	Stress period:	4	Time step:	7
Solving:	Stress period:	4	Time step:	8
Solving:	Stress period:	4	Time step:	9
Solving:	Stress period:	4	Time step:	10
Solving:	Stress period:	4	Time step:	11
Solving:	Stress period:	4	Time step:	12
Solving:	Stress period:	4	Time step:	13
Solving:	Stress period:	4	Time step:	14
Solving:	Stress period:	4	Time step:	15
Solving:	Stress period:	4	Time step:	16
Solving:	Stress period:	4	Time step:	17
Solving:	Stress period:	4	Time step:	18
Solving:	Stress period:	4	Time step:	19
Solving:	Stress period:	4	Time step:	20
Solving:	Stress period:	4	Time step:	21
Solving:	Stress period:	4	Time step:	22
Solving:	Stress period:	4	Time step:	23
Solving:	Stress period:	4	Time step:	24
Solving:	Stress period:	4	Time step:	25
Solving:	Stress period:	4	Time step:	26
Solving:	Stress period:	4	Time step:	27
Solving:	Stress period:	4	Time step:	28
Solving:	Stress period:	4	Time step:	29
Solving:	Stress period:	4	Time step:	30
Solving:	Stress period:	4	Time step:	31
Solving:	Stress period:	4	Time step:	32
Solving:	Stress period:	4	Time step:	33
Solving:	Stress period:	4	Time step:	34
Solving:	Stress period:	4	Time step:	35
Solving:	Stress period:	4	Time step:	36
Solving:	Stress period:	4	Time step:	37
Solving:	Stress period:	4	Time step:	38
Solving:	Stress period:	4	Time step:	39
Solving:	Stress period:	4	Time step:	40
Solving:	Stress period:	4	Time step:	41
Solving:	Stress period:	4	Time step:	42
Solving:	Stress period:	4	Time step:	43
Solving:	Stress period:	4	Time step:	44
Solving:	Stress period:	4	Time step:	45
Solving:	Stress period:	4	Time step:	46
Solving:	Stress period:	4	Time step:	47
Solving:	Stress period:	4	Time step:	48
Solving:	Stress period:	4	Time step:	49
Solving:	Stress period:	4	Time step:	50

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	51
Solving:	Stress period:	4	Time step:	52
Solving:	Stress period:	4	Time step:	53
Solving:	Stress period:	4	Time step:	54
Solving:	Stress period:	4	Time step:	55
Solving:	Stress period:	4	Time step:	56
Solving:	Stress period:	4	Time step:	57
Solving:	Stress period:	4	Time step:	58
Solving:	Stress period:	4	Time step:	59
Solving:	Stress period:	4	Time step:	60
Solving:	Stress period:	4	Time step:	61
Solving:	Stress period:	4	Time step:	62
Solving:	Stress period:	4	Time step:	63
Solving:	Stress period:	4	Time step:	64
Solving:	Stress period:	4	Time step:	65
Solving:	Stress period:	4	Time step:	66
Solving:	Stress period:	4	Time step:	67
Solving:	Stress period:	4	Time step:	68
Solving:	Stress period:	4	Time step:	69
Solving:	Stress period:	4	Time step:	70
Solving:	Stress period:	4	Time step:	71
Solving:	Stress period:	4	Time step:	72
Solving:	Stress period:	4	Time step:	73
Solving:	Stress period:	4	Time step:	74
Solving:	Stress period:	4	Time step:	75
Solving:	Stress period:	4	Time step:	76
Solving:	Stress period:	4	Time step:	77
Solving:	Stress period:	4	Time step:	78
Solving:	Stress period:	4	Time step:	79
Solving:	Stress period:	4	Time step:	80
Solving:	Stress period:	4	Time step:	81
Solving:	Stress period:	4	Time step:	82
Solving:	Stress period:	4	Time step:	83
Solving:	Stress period:	4	Time step:	84
Solving:	Stress period:	4	Time step:	85
Solving:	Stress period:	4	Time step:	86
Solving:	Stress period:	4	Time step:	87
Solving:	Stress period:	4	Time step:	88
Solving:	Stress period:	4	Time step:	89
Solving:	Stress period:	4	Time step:	90
Solving:	Stress period:	4	Time step:	91
Solving:	Stress period:	4	Time step:	92
Solving:	Stress period:	4	Time step:	93
Solving:	Stress period:	4	Time step:	94
Solving:	Stress period:	4	Time step:	95
Solving:	Stress period:	4	Time step:	96
Solving:	Stress period:	4	Time step:	97
Solving:	Stress period:	4	Time step:	98
Solving:	Stress period:	4	Time step:	99
Solving:	Stress period:	4	Time step:	100
Solving:	Stress period:	4	Time step:	101
Solving:	Stress period:	4	Time step:	102

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 4 Time step: 103
Solving: Stress period: 4 Time step: 104
Solving: Stress period: 4 Time step: 105
Solving: Stress period: 4 Time step: 106
Solving: Stress period: 4 Time step: 107
Solving: Stress period: 4 Time step: 108
Solving: Stress period: 4 Time step: 109
Solving: Stress period: 4 Time step: 110
Solving: Stress period: 4 Time step: 111
Solving: Stress period: 4 Time step: 112
Solving: Stress period: 4 Time step: 113
Solving: Stress period: 4 Time step: 114
Solving: Stress period: 4 Time step: 115
Solving: Stress period: 4 Time step: 116
Solving: Stress period: 4 Time step: 117
Solving: Stress period: 4 Time step: 118
Solving: Stress period: 4 Time step: 119
Solving: Stress period: 4 Time step: 120

```

```

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:35
Elapsed run time: 0.580 Seconds

```

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
 2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.
- Normal termination of simulation.

Time Series

Time series can be set for any package through the package.ts object, and each package.ts object has several attributes that can be set:

package.ts.filename : str Name of time series file to create. The default is packagename + '.ts', e.g. mymodel.ghb.ts.

package.ts.timeseries : recarray Array containing the time series information. timeseries = [(t, np.sin(t)) for t in np.linspace(0, 100., 10)]

package.ts.time_series_namerecord : str or list (of strings) List of names of the time series data columns. Default is to use names from timeseries.dtype.names[1:].

package.ts.interpolation_methodrecord_single : str Interpolation method. Must be only one time series record. If there are multiple time series records, then the methods attribute must be used. Default is 'linear'.

package.ts.interpolation_methodrecord : list (of strings) List of interpolation methods to use for each time series data column. Method must be either 'stepwise', 'linear', or 'linearend'.

package.ts.sfarecord_single : float Scale factor to multiply the time series data column. Can only be used if there is one time series data column.

package.ts.sfarecord : list (of floats) Scale factors to multiply the time series data columns.

Method 1: Pass time series to package constructor

```
[6]: # build ghb stress period data
ghb_spd_ts = {}
ghb_period = []
for layer, cond in zip(range(1, 3), [15.0, 1500.0]):
    for row in range(0, 15):
        ghb_period.append((layer, row, 9), "tides", cond, "Estuary-L2")
ghb_spd_ts[0] = ghb_period

# build ts data
ts_data = []
for n in range(0, 365):
    ts_data.append((float(n / 11.73), float(n / 60.0)))

# build obs data
ghb_obs = {
    ("ghb_obs.csv", "binary"): [
        ("ghb-2-6-10", "GHB", (1, 5, 9)),
        ("ghb-3-6-10", "GHB", (2, 5, 9)),
    ],
    "ghb_flows.csv": [
        ("Estuary2", "GHB", "Estuary-L2"),
        ("Estuary3", "GHB", "Estuary-L3"),
    ],
}

# build ghb package
ghb = flopy.mf6.modflow.mfgwfghb.ModflowGwfghb(
    model,
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    timeseries=ts_data,
    pname="ghb",
    maxbound=30,
    stress_period_data=ghb_spd_ts,
)

# set required time series attributes
ghb.ts.time_series_namerecord = "tides"
ghb.ts.interpolation_methodrecord = "stepwise"

sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

# clean up for next example
model.remove_package("ghb")
```



```

WARNING: Unable to resolve dimension of ('ts', 'timeseries', 'timeseries', 'ts_array')_
↳based on shape "time_series_names".
WARNING: Unable to resolve dimension of ('ts', 'timeseries', 'timeseries', 'ts_array')_
↳based on shape "time_series_names".
WARNING: Unable to resolve dimension of ('ts', 'timeseries', 'timeseries', 'ts_array')_
↳based on shape "time_series_names".

```

```
writing simulation...
```

```

writing simulation name file...
writing simulation tdis package...
writing solution package ims_-1...
writing model child_pkgs...
writing model name file...
writing package dis...
writing package ic...
writing package npf...
writing package oc...
writing package sto...
writing package ghb...
writing package ts_0...

```

MODFLOW 6

U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_0000000

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:35

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5
Solving:	Stress period:	2	Time step:	6

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	7
Solving:	Stress period:	2	Time step:	8
Solving:	Stress period:	2	Time step:	9
Solving:	Stress period:	2	Time step:	10
Solving:	Stress period:	2	Time step:	11
Solving:	Stress period:	2	Time step:	12
Solving:	Stress period:	2	Time step:	13
Solving:	Stress period:	2	Time step:	14
Solving:	Stress period:	2	Time step:	15
Solving:	Stress period:	2	Time step:	16
Solving:	Stress period:	2	Time step:	17
Solving:	Stress period:	2	Time step:	18
Solving:	Stress period:	2	Time step:	19
Solving:	Stress period:	2	Time step:	20
Solving:	Stress period:	2	Time step:	21
Solving:	Stress period:	2	Time step:	22
Solving:	Stress period:	2	Time step:	23
Solving:	Stress period:	2	Time step:	24
Solving:	Stress period:	2	Time step:	25
Solving:	Stress period:	2	Time step:	26
Solving:	Stress period:	2	Time step:	27
Solving:	Stress period:	2	Time step:	28
Solving:	Stress period:	2	Time step:	29
Solving:	Stress period:	2	Time step:	30
Solving:	Stress period:	2	Time step:	31
Solving:	Stress period:	2	Time step:	32
Solving:	Stress period:	2	Time step:	33
Solving:	Stress period:	2	Time step:	34
Solving:	Stress period:	2	Time step:	35
Solving:	Stress period:	2	Time step:	36
Solving:	Stress period:	2	Time step:	37
Solving:	Stress period:	2	Time step:	38
Solving:	Stress period:	2	Time step:	39
Solving:	Stress period:	2	Time step:	40
Solving:	Stress period:	2	Time step:	41
Solving:	Stress period:	2	Time step:	42
Solving:	Stress period:	2	Time step:	43
Solving:	Stress period:	2	Time step:	44
Solving:	Stress period:	2	Time step:	45
Solving:	Stress period:	2	Time step:	46
Solving:	Stress period:	2	Time step:	47
Solving:	Stress period:	2	Time step:	48
Solving:	Stress period:	2	Time step:	49
Solving:	Stress period:	2	Time step:	50
Solving:	Stress period:	2	Time step:	51
Solving:	Stress period:	2	Time step:	52
Solving:	Stress period:	2	Time step:	53
Solving:	Stress period:	2	Time step:	54
Solving:	Stress period:	2	Time step:	55
Solving:	Stress period:	2	Time step:	56
Solving:	Stress period:	2	Time step:	57
Solving:	Stress period:	2	Time step:	58

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	59
Solving:	Stress period:	2	Time step:	60
Solving:	Stress period:	2	Time step:	61
Solving:	Stress period:	2	Time step:	62
Solving:	Stress period:	2	Time step:	63
Solving:	Stress period:	2	Time step:	64
Solving:	Stress period:	2	Time step:	65
Solving:	Stress period:	2	Time step:	66
Solving:	Stress period:	2	Time step:	67
Solving:	Stress period:	2	Time step:	68
Solving:	Stress period:	2	Time step:	69
Solving:	Stress period:	2	Time step:	70
Solving:	Stress period:	2	Time step:	71
Solving:	Stress period:	2	Time step:	72
Solving:	Stress period:	2	Time step:	73
Solving:	Stress period:	2	Time step:	74
Solving:	Stress period:	2	Time step:	75
Solving:	Stress period:	2	Time step:	76
Solving:	Stress period:	2	Time step:	77
Solving:	Stress period:	2	Time step:	78
Solving:	Stress period:	2	Time step:	79
Solving:	Stress period:	2	Time step:	80
Solving:	Stress period:	2	Time step:	81
Solving:	Stress period:	2	Time step:	82
Solving:	Stress period:	2	Time step:	83
Solving:	Stress period:	2	Time step:	84
Solving:	Stress period:	2	Time step:	85
Solving:	Stress period:	2	Time step:	86
Solving:	Stress period:	2	Time step:	87
Solving:	Stress period:	2	Time step:	88
Solving:	Stress period:	2	Time step:	89
Solving:	Stress period:	2	Time step:	90
Solving:	Stress period:	2	Time step:	91
Solving:	Stress period:	2	Time step:	92
Solving:	Stress period:	2	Time step:	93
Solving:	Stress period:	2	Time step:	94
Solving:	Stress period:	2	Time step:	95
Solving:	Stress period:	2	Time step:	96
Solving:	Stress period:	2	Time step:	97
Solving:	Stress period:	2	Time step:	98
Solving:	Stress period:	2	Time step:	99
Solving:	Stress period:	2	Time step:	100
Solving:	Stress period:	2	Time step:	101
Solving:	Stress period:	2	Time step:	102
Solving:	Stress period:	2	Time step:	103
Solving:	Stress period:	2	Time step:	104
Solving:	Stress period:	2	Time step:	105
Solving:	Stress period:	2	Time step:	106
Solving:	Stress period:	2	Time step:	107
Solving:	Stress period:	2	Time step:	108
Solving:	Stress period:	2	Time step:	109
Solving:	Stress period:	2	Time step:	110

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	111
Solving:	Stress period:	2	Time step:	112
Solving:	Stress period:	2	Time step:	113
Solving:	Stress period:	2	Time step:	114
Solving:	Stress period:	2	Time step:	115
Solving:	Stress period:	2	Time step:	116
Solving:	Stress period:	2	Time step:	117
Solving:	Stress period:	2	Time step:	118
Solving:	Stress period:	2	Time step:	119
Solving:	Stress period:	2	Time step:	120
Solving:	Stress period:	3	Time step:	1
Solving:	Stress period:	3	Time step:	2
Solving:	Stress period:	3	Time step:	3
Solving:	Stress period:	3	Time step:	4
Solving:	Stress period:	3	Time step:	5
Solving:	Stress period:	3	Time step:	6
Solving:	Stress period:	3	Time step:	7
Solving:	Stress period:	3	Time step:	8
Solving:	Stress period:	3	Time step:	9
Solving:	Stress period:	3	Time step:	10
Solving:	Stress period:	3	Time step:	11
Solving:	Stress period:	3	Time step:	12
Solving:	Stress period:	3	Time step:	13
Solving:	Stress period:	3	Time step:	14
Solving:	Stress period:	3	Time step:	15
Solving:	Stress period:	3	Time step:	16
Solving:	Stress period:	3	Time step:	17
Solving:	Stress period:	3	Time step:	18
Solving:	Stress period:	3	Time step:	19
Solving:	Stress period:	3	Time step:	20
Solving:	Stress period:	3	Time step:	21
Solving:	Stress period:	3	Time step:	22
Solving:	Stress period:	3	Time step:	23
Solving:	Stress period:	3	Time step:	24
Solving:	Stress period:	3	Time step:	25
Solving:	Stress period:	3	Time step:	26
Solving:	Stress period:	3	Time step:	27
Solving:	Stress period:	3	Time step:	28
Solving:	Stress period:	3	Time step:	29
Solving:	Stress period:	3	Time step:	30
Solving:	Stress period:	3	Time step:	31
Solving:	Stress period:	3	Time step:	32
Solving:	Stress period:	3	Time step:	33
Solving:	Stress period:	3	Time step:	34
Solving:	Stress period:	3	Time step:	35
Solving:	Stress period:	3	Time step:	36
Solving:	Stress period:	3	Time step:	37
Solving:	Stress period:	3	Time step:	38
Solving:	Stress period:	3	Time step:	39
Solving:	Stress period:	3	Time step:	40
Solving:	Stress period:	3	Time step:	41
Solving:	Stress period:	3	Time step:	42

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	43
Solving:	Stress period:	3	Time step:	44
Solving:	Stress period:	3	Time step:	45
Solving:	Stress period:	3	Time step:	46
Solving:	Stress period:	3	Time step:	47
Solving:	Stress period:	3	Time step:	48
Solving:	Stress period:	3	Time step:	49
Solving:	Stress period:	3	Time step:	50
Solving:	Stress period:	3	Time step:	51
Solving:	Stress period:	3	Time step:	52
Solving:	Stress period:	3	Time step:	53
Solving:	Stress period:	3	Time step:	54
Solving:	Stress period:	3	Time step:	55
Solving:	Stress period:	3	Time step:	56
Solving:	Stress period:	3	Time step:	57
Solving:	Stress period:	3	Time step:	58
Solving:	Stress period:	3	Time step:	59
Solving:	Stress period:	3	Time step:	60
Solving:	Stress period:	3	Time step:	61
Solving:	Stress period:	3	Time step:	62
Solving:	Stress period:	3	Time step:	63
Solving:	Stress period:	3	Time step:	64
Solving:	Stress period:	3	Time step:	65
Solving:	Stress period:	3	Time step:	66
Solving:	Stress period:	3	Time step:	67
Solving:	Stress period:	3	Time step:	68
Solving:	Stress period:	3	Time step:	69
Solving:	Stress period:	3	Time step:	70
Solving:	Stress period:	3	Time step:	71
Solving:	Stress period:	3	Time step:	72
Solving:	Stress period:	3	Time step:	73
Solving:	Stress period:	3	Time step:	74
Solving:	Stress period:	3	Time step:	75
Solving:	Stress period:	3	Time step:	76
Solving:	Stress period:	3	Time step:	77
Solving:	Stress period:	3	Time step:	78
Solving:	Stress period:	3	Time step:	79
Solving:	Stress period:	3	Time step:	80
Solving:	Stress period:	3	Time step:	81
Solving:	Stress period:	3	Time step:	82
Solving:	Stress period:	3	Time step:	83
Solving:	Stress period:	3	Time step:	84
Solving:	Stress period:	3	Time step:	85
Solving:	Stress period:	3	Time step:	86
Solving:	Stress period:	3	Time step:	87
Solving:	Stress period:	3	Time step:	88
Solving:	Stress period:	3	Time step:	89
Solving:	Stress period:	3	Time step:	90
Solving:	Stress period:	3	Time step:	91
Solving:	Stress period:	3	Time step:	92
Solving:	Stress period:	3	Time step:	93
Solving:	Stress period:	3	Time step:	94

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	95
Solving:	Stress period:	3	Time step:	96
Solving:	Stress period:	3	Time step:	97
Solving:	Stress period:	3	Time step:	98
Solving:	Stress period:	3	Time step:	99
Solving:	Stress period:	3	Time step:	100
Solving:	Stress period:	3	Time step:	101
Solving:	Stress period:	3	Time step:	102
Solving:	Stress period:	3	Time step:	103
Solving:	Stress period:	3	Time step:	104
Solving:	Stress period:	3	Time step:	105
Solving:	Stress period:	3	Time step:	106
Solving:	Stress period:	3	Time step:	107
Solving:	Stress period:	3	Time step:	108
Solving:	Stress period:	3	Time step:	109
Solving:	Stress period:	3	Time step:	110
Solving:	Stress period:	3	Time step:	111
Solving:	Stress period:	3	Time step:	112
Solving:	Stress period:	3	Time step:	113
Solving:	Stress period:	3	Time step:	114
Solving:	Stress period:	3	Time step:	115
Solving:	Stress period:	3	Time step:	116
Solving:	Stress period:	3	Time step:	117
Solving:	Stress period:	3	Time step:	118
Solving:	Stress period:	3	Time step:	119
Solving:	Stress period:	3	Time step:	120
Solving:	Stress period:	4	Time step:	1
Solving:	Stress period:	4	Time step:	2
Solving:	Stress period:	4	Time step:	3
Solving:	Stress period:	4	Time step:	4
Solving:	Stress period:	4	Time step:	5
Solving:	Stress period:	4	Time step:	6
Solving:	Stress period:	4	Time step:	7
Solving:	Stress period:	4	Time step:	8
Solving:	Stress period:	4	Time step:	9
Solving:	Stress period:	4	Time step:	10
Solving:	Stress period:	4	Time step:	11
Solving:	Stress period:	4	Time step:	12
Solving:	Stress period:	4	Time step:	13
Solving:	Stress period:	4	Time step:	14
Solving:	Stress period:	4	Time step:	15
Solving:	Stress period:	4	Time step:	16
Solving:	Stress period:	4	Time step:	17
Solving:	Stress period:	4	Time step:	18
Solving:	Stress period:	4	Time step:	19
Solving:	Stress period:	4	Time step:	20
Solving:	Stress period:	4	Time step:	21
Solving:	Stress period:	4	Time step:	22
Solving:	Stress period:	4	Time step:	23
Solving:	Stress period:	4	Time step:	24
Solving:	Stress period:	4	Time step:	25
Solving:	Stress period:	4	Time step:	26

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	27
Solving:	Stress period:	4	Time step:	28
Solving:	Stress period:	4	Time step:	29
Solving:	Stress period:	4	Time step:	30
Solving:	Stress period:	4	Time step:	31
Solving:	Stress period:	4	Time step:	32
Solving:	Stress period:	4	Time step:	33
Solving:	Stress period:	4	Time step:	34
Solving:	Stress period:	4	Time step:	35
Solving:	Stress period:	4	Time step:	36
Solving:	Stress period:	4	Time step:	37
Solving:	Stress period:	4	Time step:	38
Solving:	Stress period:	4	Time step:	39
Solving:	Stress period:	4	Time step:	40
Solving:	Stress period:	4	Time step:	41
Solving:	Stress period:	4	Time step:	42
Solving:	Stress period:	4	Time step:	43
Solving:	Stress period:	4	Time step:	44
Solving:	Stress period:	4	Time step:	45
Solving:	Stress period:	4	Time step:	46
Solving:	Stress period:	4	Time step:	47
Solving:	Stress period:	4	Time step:	48
Solving:	Stress period:	4	Time step:	49
Solving:	Stress period:	4	Time step:	50
Solving:	Stress period:	4	Time step:	51
Solving:	Stress period:	4	Time step:	52
Solving:	Stress period:	4	Time step:	53
Solving:	Stress period:	4	Time step:	54
Solving:	Stress period:	4	Time step:	55
Solving:	Stress period:	4	Time step:	56
Solving:	Stress period:	4	Time step:	57
Solving:	Stress period:	4	Time step:	58
Solving:	Stress period:	4	Time step:	59
Solving:	Stress period:	4	Time step:	60
Solving:	Stress period:	4	Time step:	61
Solving:	Stress period:	4	Time step:	62
Solving:	Stress period:	4	Time step:	63
Solving:	Stress period:	4	Time step:	64
Solving:	Stress period:	4	Time step:	65
Solving:	Stress period:	4	Time step:	66
Solving:	Stress period:	4	Time step:	67
Solving:	Stress period:	4	Time step:	68
Solving:	Stress period:	4	Time step:	69
Solving:	Stress period:	4	Time step:	70
Solving:	Stress period:	4	Time step:	71
Solving:	Stress period:	4	Time step:	72
Solving:	Stress period:	4	Time step:	73
Solving:	Stress period:	4	Time step:	74
Solving:	Stress period:	4	Time step:	75
Solving:	Stress period:	4	Time step:	76
Solving:	Stress period:	4	Time step:	77
Solving:	Stress period:	4	Time step:	78

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	79
Solving:	Stress period:	4	Time step:	80
Solving:	Stress period:	4	Time step:	81
Solving:	Stress period:	4	Time step:	82
Solving:	Stress period:	4	Time step:	83
Solving:	Stress period:	4	Time step:	84
Solving:	Stress period:	4	Time step:	85
Solving:	Stress period:	4	Time step:	86
Solving:	Stress period:	4	Time step:	87
Solving:	Stress period:	4	Time step:	88
Solving:	Stress period:	4	Time step:	89
Solving:	Stress period:	4	Time step:	90
Solving:	Stress period:	4	Time step:	91
Solving:	Stress period:	4	Time step:	92
Solving:	Stress period:	4	Time step:	93
Solving:	Stress period:	4	Time step:	94
Solving:	Stress period:	4	Time step:	95
Solving:	Stress period:	4	Time step:	96
Solving:	Stress period:	4	Time step:	97
Solving:	Stress period:	4	Time step:	98
Solving:	Stress period:	4	Time step:	99
Solving:	Stress period:	4	Time step:	100
Solving:	Stress period:	4	Time step:	101
Solving:	Stress period:	4	Time step:	102
Solving:	Stress period:	4	Time step:	103
Solving:	Stress period:	4	Time step:	104
Solving:	Stress period:	4	Time step:	105
Solving:	Stress period:	4	Time step:	106
Solving:	Stress period:	4	Time step:	107
Solving:	Stress period:	4	Time step:	108
Solving:	Stress period:	4	Time step:	109
Solving:	Stress period:	4	Time step:	110
Solving:	Stress period:	4	Time step:	111
Solving:	Stress period:	4	Time step:	112
Solving:	Stress period:	4	Time step:	113
Solving:	Stress period:	4	Time step:	114
Solving:	Stress period:	4	Time step:	115
Solving:	Stress period:	4	Time step:	116
Solving:	Stress period:	4	Time step:	117
Solving:	Stress period:	4	Time step:	118
Solving:	Stress period:	4	Time step:	119
Solving:	Stress period:	4	Time step:	120

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:36

Elapsed run time: 0.495 Seconds

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.

(continues on next page)

(continued from previous page)

2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.
Normal termination of simulation.

Method 2: Initialize time series through ghb.ts.initialize

```
[7]: # build ghb stress period data
ghb_spd_ts = {}
ghb_period = []
for layer, cond in zip(range(1, 3), [15.0, 1500.0]):
    for row in range(0, 15):
        ghb_period.append((layer, row, 9), "tides", cond, "Estuary-L2")
ghb_spd_ts[0] = ghb_period

# build ts data
ts_data = []
for n in range(0, 365):
    ts_data.append((float(n / 11.73), float(n / 60.0)))

# build obs data
ghb_obs = {
    ("ghb_obs.csv", "binary"): [
        ("ghb-2-6-10", "GHB", (1, 5, 9)),
        ("ghb-3-6-10", "GHB", (2, 5, 9)),
    ],
    "ghb_flows.csv": [
        ("Estuary2", "GHB", "Estuary-L2"),
        ("Estuary3", "GHB", "Estuary-L3"),
    ],
}

# build ghb package
ghb = flopy.mf6.modflow.mfgwfgghb.ModflowGwfgghb(
    model,
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    pname="ghb",
    maxbound=30,
    stress_period_data=ghb_spd_ts,
)

# initialize time series
ghb.ts.initialize(
    filename="method2.ts",
    timeseries=ts_data,
    time_series_namerecord="tides",
    interpolation_methodrecord="linearend",
    sfacrecord=1.1,
)
```

(continues on next page)

(continued from previous page)

```

sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

# clean up for next example
model.remove_package("ghb")

```

```

WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".

```

```
writing simulation...
```

```

writing simulation name file...
writing simulation tdis package...
writing solution package ims_1...
writing model child_pkgs...
writing model name file...
writing package dis...
writing package ic...
writing package npf...
writing package oc...
writing package sto...
writing package ghb...
writing package ts_0...

```

```

WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".

```

```

MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

```

```

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

(continues on next page)

(continued from previous page)

```
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:36
```

```
Writing simulation list file: mfsim.lst
```

```
Using Simulation name file: mfsim.nam
```

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5
Solving:	Stress period:	2	Time step:	6
Solving:	Stress period:	2	Time step:	7
Solving:	Stress period:	2	Time step:	8
Solving:	Stress period:	2	Time step:	9
Solving:	Stress period:	2	Time step:	10
Solving:	Stress period:	2	Time step:	11
Solving:	Stress period:	2	Time step:	12
Solving:	Stress period:	2	Time step:	13
Solving:	Stress period:	2	Time step:	14
Solving:	Stress period:	2	Time step:	15
Solving:	Stress period:	2	Time step:	16
Solving:	Stress period:	2	Time step:	17
Solving:	Stress period:	2	Time step:	18
Solving:	Stress period:	2	Time step:	19
Solving:	Stress period:	2	Time step:	20
Solving:	Stress period:	2	Time step:	21
Solving:	Stress period:	2	Time step:	22
Solving:	Stress period:	2	Time step:	23
Solving:	Stress period:	2	Time step:	24
Solving:	Stress period:	2	Time step:	25
Solving:	Stress period:	2	Time step:	26
Solving:	Stress period:	2	Time step:	27
Solving:	Stress period:	2	Time step:	28
Solving:	Stress period:	2	Time step:	29
Solving:	Stress period:	2	Time step:	30
Solving:	Stress period:	2	Time step:	31
Solving:	Stress period:	2	Time step:	32
Solving:	Stress period:	2	Time step:	33
Solving:	Stress period:	2	Time step:	34
Solving:	Stress period:	2	Time step:	35
Solving:	Stress period:	2	Time step:	36
Solving:	Stress period:	2	Time step:	37
Solving:	Stress period:	2	Time step:	38
Solving:	Stress period:	2	Time step:	39
Solving:	Stress period:	2	Time step:	40
Solving:	Stress period:	2	Time step:	41
Solving:	Stress period:	2	Time step:	42
Solving:	Stress period:	2	Time step:	43
Solving:	Stress period:	2	Time step:	44
Solving:	Stress period:	2	Time step:	45
Solving:	Stress period:	2	Time step:	46

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	47
Solving:	Stress period:	2	Time step:	48
Solving:	Stress period:	2	Time step:	49
Solving:	Stress period:	2	Time step:	50
Solving:	Stress period:	2	Time step:	51
Solving:	Stress period:	2	Time step:	52
Solving:	Stress period:	2	Time step:	53
Solving:	Stress period:	2	Time step:	54
Solving:	Stress period:	2	Time step:	55
Solving:	Stress period:	2	Time step:	56
Solving:	Stress period:	2	Time step:	57
Solving:	Stress period:	2	Time step:	58
Solving:	Stress period:	2	Time step:	59
Solving:	Stress period:	2	Time step:	60
Solving:	Stress period:	2	Time step:	61
Solving:	Stress period:	2	Time step:	62
Solving:	Stress period:	2	Time step:	63
Solving:	Stress period:	2	Time step:	64
Solving:	Stress period:	2	Time step:	65
Solving:	Stress period:	2	Time step:	66
Solving:	Stress period:	2	Time step:	67
Solving:	Stress period:	2	Time step:	68
Solving:	Stress period:	2	Time step:	69
Solving:	Stress period:	2	Time step:	70
Solving:	Stress period:	2	Time step:	71
Solving:	Stress period:	2	Time step:	72
Solving:	Stress period:	2	Time step:	73
Solving:	Stress period:	2	Time step:	74
Solving:	Stress period:	2	Time step:	75
Solving:	Stress period:	2	Time step:	76
Solving:	Stress period:	2	Time step:	77
Solving:	Stress period:	2	Time step:	78
Solving:	Stress period:	2	Time step:	79
Solving:	Stress period:	2	Time step:	80
Solving:	Stress period:	2	Time step:	81
Solving:	Stress period:	2	Time step:	82
Solving:	Stress period:	2	Time step:	83
Solving:	Stress period:	2	Time step:	84
Solving:	Stress period:	2	Time step:	85
Solving:	Stress period:	2	Time step:	86
Solving:	Stress period:	2	Time step:	87
Solving:	Stress period:	2	Time step:	88
Solving:	Stress period:	2	Time step:	89
Solving:	Stress period:	2	Time step:	90
Solving:	Stress period:	2	Time step:	91
Solving:	Stress period:	2	Time step:	92
Solving:	Stress period:	2	Time step:	93
Solving:	Stress period:	2	Time step:	94
Solving:	Stress period:	2	Time step:	95
Solving:	Stress period:	2	Time step:	96
Solving:	Stress period:	2	Time step:	97
Solving:	Stress period:	2	Time step:	98

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	99
Solving:	Stress period:	2	Time step:	100
Solving:	Stress period:	2	Time step:	101
Solving:	Stress period:	2	Time step:	102
Solving:	Stress period:	2	Time step:	103
Solving:	Stress period:	2	Time step:	104
Solving:	Stress period:	2	Time step:	105
Solving:	Stress period:	2	Time step:	106
Solving:	Stress period:	2	Time step:	107
Solving:	Stress period:	2	Time step:	108
Solving:	Stress period:	2	Time step:	109
Solving:	Stress period:	2	Time step:	110
Solving:	Stress period:	2	Time step:	111
Solving:	Stress period:	2	Time step:	112
Solving:	Stress period:	2	Time step:	113
Solving:	Stress period:	2	Time step:	114
Solving:	Stress period:	2	Time step:	115
Solving:	Stress period:	2	Time step:	116
Solving:	Stress period:	2	Time step:	117
Solving:	Stress period:	2	Time step:	118
Solving:	Stress period:	2	Time step:	119
Solving:	Stress period:	2	Time step:	120
Solving:	Stress period:	3	Time step:	1
Solving:	Stress period:	3	Time step:	2
Solving:	Stress period:	3	Time step:	3
Solving:	Stress period:	3	Time step:	4
Solving:	Stress period:	3	Time step:	5
Solving:	Stress period:	3	Time step:	6
Solving:	Stress period:	3	Time step:	7
Solving:	Stress period:	3	Time step:	8
Solving:	Stress period:	3	Time step:	9
Solving:	Stress period:	3	Time step:	10
Solving:	Stress period:	3	Time step:	11
Solving:	Stress period:	3	Time step:	12
Solving:	Stress period:	3	Time step:	13
Solving:	Stress period:	3	Time step:	14
Solving:	Stress period:	3	Time step:	15
Solving:	Stress period:	3	Time step:	16
Solving:	Stress period:	3	Time step:	17
Solving:	Stress period:	3	Time step:	18
Solving:	Stress period:	3	Time step:	19
Solving:	Stress period:	3	Time step:	20
Solving:	Stress period:	3	Time step:	21
Solving:	Stress period:	3	Time step:	22
Solving:	Stress period:	3	Time step:	23
Solving:	Stress period:	3	Time step:	24
Solving:	Stress period:	3	Time step:	25
Solving:	Stress period:	3	Time step:	26
Solving:	Stress period:	3	Time step:	27
Solving:	Stress period:	3	Time step:	28
Solving:	Stress period:	3	Time step:	29
Solving:	Stress period:	3	Time step:	30

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	31
Solving:	Stress period:	3	Time step:	32
Solving:	Stress period:	3	Time step:	33
Solving:	Stress period:	3	Time step:	34
Solving:	Stress period:	3	Time step:	35
Solving:	Stress period:	3	Time step:	36
Solving:	Stress period:	3	Time step:	37
Solving:	Stress period:	3	Time step:	38
Solving:	Stress period:	3	Time step:	39
Solving:	Stress period:	3	Time step:	40
Solving:	Stress period:	3	Time step:	41
Solving:	Stress period:	3	Time step:	42
Solving:	Stress period:	3	Time step:	43
Solving:	Stress period:	3	Time step:	44
Solving:	Stress period:	3	Time step:	45
Solving:	Stress period:	3	Time step:	46
Solving:	Stress period:	3	Time step:	47
Solving:	Stress period:	3	Time step:	48
Solving:	Stress period:	3	Time step:	49
Solving:	Stress period:	3	Time step:	50
Solving:	Stress period:	3	Time step:	51
Solving:	Stress period:	3	Time step:	52
Solving:	Stress period:	3	Time step:	53
Solving:	Stress period:	3	Time step:	54
Solving:	Stress period:	3	Time step:	55
Solving:	Stress period:	3	Time step:	56
Solving:	Stress period:	3	Time step:	57
Solving:	Stress period:	3	Time step:	58
Solving:	Stress period:	3	Time step:	59
Solving:	Stress period:	3	Time step:	60
Solving:	Stress period:	3	Time step:	61
Solving:	Stress period:	3	Time step:	62
Solving:	Stress period:	3	Time step:	63
Solving:	Stress period:	3	Time step:	64
Solving:	Stress period:	3	Time step:	65
Solving:	Stress period:	3	Time step:	66
Solving:	Stress period:	3	Time step:	67
Solving:	Stress period:	3	Time step:	68
Solving:	Stress period:	3	Time step:	69
Solving:	Stress period:	3	Time step:	70
Solving:	Stress period:	3	Time step:	71
Solving:	Stress period:	3	Time step:	72
Solving:	Stress period:	3	Time step:	73
Solving:	Stress period:	3	Time step:	74
Solving:	Stress period:	3	Time step:	75
Solving:	Stress period:	3	Time step:	76
Solving:	Stress period:	3	Time step:	77
Solving:	Stress period:	3	Time step:	78
Solving:	Stress period:	3	Time step:	79
Solving:	Stress period:	3	Time step:	80
Solving:	Stress period:	3	Time step:	81
Solving:	Stress period:	3	Time step:	82

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	83
Solving:	Stress period:	3	Time step:	84
Solving:	Stress period:	3	Time step:	85
Solving:	Stress period:	3	Time step:	86
Solving:	Stress period:	3	Time step:	87
Solving:	Stress period:	3	Time step:	88
Solving:	Stress period:	3	Time step:	89
Solving:	Stress period:	3	Time step:	90
Solving:	Stress period:	3	Time step:	91
Solving:	Stress period:	3	Time step:	92
Solving:	Stress period:	3	Time step:	93
Solving:	Stress period:	3	Time step:	94
Solving:	Stress period:	3	Time step:	95
Solving:	Stress period:	3	Time step:	96
Solving:	Stress period:	3	Time step:	97
Solving:	Stress period:	3	Time step:	98
Solving:	Stress period:	3	Time step:	99
Solving:	Stress period:	3	Time step:	100
Solving:	Stress period:	3	Time step:	101
Solving:	Stress period:	3	Time step:	102
Solving:	Stress period:	3	Time step:	103
Solving:	Stress period:	3	Time step:	104
Solving:	Stress period:	3	Time step:	105
Solving:	Stress period:	3	Time step:	106
Solving:	Stress period:	3	Time step:	107
Solving:	Stress period:	3	Time step:	108
Solving:	Stress period:	3	Time step:	109
Solving:	Stress period:	3	Time step:	110
Solving:	Stress period:	3	Time step:	111
Solving:	Stress period:	3	Time step:	112
Solving:	Stress period:	3	Time step:	113
Solving:	Stress period:	3	Time step:	114
Solving:	Stress period:	3	Time step:	115
Solving:	Stress period:	3	Time step:	116
Solving:	Stress period:	3	Time step:	117
Solving:	Stress period:	3	Time step:	118
Solving:	Stress period:	3	Time step:	119
Solving:	Stress period:	3	Time step:	120
Solving:	Stress period:	4	Time step:	1
Solving:	Stress period:	4	Time step:	2
Solving:	Stress period:	4	Time step:	3
Solving:	Stress period:	4	Time step:	4
Solving:	Stress period:	4	Time step:	5
Solving:	Stress period:	4	Time step:	6
Solving:	Stress period:	4	Time step:	7
Solving:	Stress period:	4	Time step:	8
Solving:	Stress period:	4	Time step:	9
Solving:	Stress period:	4	Time step:	10
Solving:	Stress period:	4	Time step:	11
Solving:	Stress period:	4	Time step:	12
Solving:	Stress period:	4	Time step:	13
Solving:	Stress period:	4	Time step:	14

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	15
Solving:	Stress period:	4	Time step:	16
Solving:	Stress period:	4	Time step:	17
Solving:	Stress period:	4	Time step:	18
Solving:	Stress period:	4	Time step:	19
Solving:	Stress period:	4	Time step:	20
Solving:	Stress period:	4	Time step:	21
Solving:	Stress period:	4	Time step:	22
Solving:	Stress period:	4	Time step:	23
Solving:	Stress period:	4	Time step:	24
Solving:	Stress period:	4	Time step:	25
Solving:	Stress period:	4	Time step:	26
Solving:	Stress period:	4	Time step:	27
Solving:	Stress period:	4	Time step:	28
Solving:	Stress period:	4	Time step:	29
Solving:	Stress period:	4	Time step:	30
Solving:	Stress period:	4	Time step:	31
Solving:	Stress period:	4	Time step:	32
Solving:	Stress period:	4	Time step:	33
Solving:	Stress period:	4	Time step:	34
Solving:	Stress period:	4	Time step:	35
Solving:	Stress period:	4	Time step:	36
Solving:	Stress period:	4	Time step:	37
Solving:	Stress period:	4	Time step:	38
Solving:	Stress period:	4	Time step:	39
Solving:	Stress period:	4	Time step:	40
Solving:	Stress period:	4	Time step:	41
Solving:	Stress period:	4	Time step:	42
Solving:	Stress period:	4	Time step:	43
Solving:	Stress period:	4	Time step:	44
Solving:	Stress period:	4	Time step:	45
Solving:	Stress period:	4	Time step:	46
Solving:	Stress period:	4	Time step:	47
Solving:	Stress period:	4	Time step:	48
Solving:	Stress period:	4	Time step:	49
Solving:	Stress period:	4	Time step:	50
Solving:	Stress period:	4	Time step:	51
Solving:	Stress period:	4	Time step:	52
Solving:	Stress period:	4	Time step:	53
Solving:	Stress period:	4	Time step:	54
Solving:	Stress period:	4	Time step:	55
Solving:	Stress period:	4	Time step:	56
Solving:	Stress period:	4	Time step:	57
Solving:	Stress period:	4	Time step:	58
Solving:	Stress period:	4	Time step:	59
Solving:	Stress period:	4	Time step:	60
Solving:	Stress period:	4	Time step:	61
Solving:	Stress period:	4	Time step:	62
Solving:	Stress period:	4	Time step:	63
Solving:	Stress period:	4	Time step:	64
Solving:	Stress period:	4	Time step:	65
Solving:	Stress period:	4	Time step:	66

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	67
Solving:	Stress period:	4	Time step:	68
Solving:	Stress period:	4	Time step:	69
Solving:	Stress period:	4	Time step:	70
Solving:	Stress period:	4	Time step:	71
Solving:	Stress period:	4	Time step:	72
Solving:	Stress period:	4	Time step:	73
Solving:	Stress period:	4	Time step:	74
Solving:	Stress period:	4	Time step:	75
Solving:	Stress period:	4	Time step:	76
Solving:	Stress period:	4	Time step:	77
Solving:	Stress period:	4	Time step:	78
Solving:	Stress period:	4	Time step:	79
Solving:	Stress period:	4	Time step:	80
Solving:	Stress period:	4	Time step:	81
Solving:	Stress period:	4	Time step:	82
Solving:	Stress period:	4	Time step:	83
Solving:	Stress period:	4	Time step:	84
Solving:	Stress period:	4	Time step:	85
Solving:	Stress period:	4	Time step:	86
Solving:	Stress period:	4	Time step:	87
Solving:	Stress period:	4	Time step:	88
Solving:	Stress period:	4	Time step:	89
Solving:	Stress period:	4	Time step:	90
Solving:	Stress period:	4	Time step:	91
Solving:	Stress period:	4	Time step:	92
Solving:	Stress period:	4	Time step:	93
Solving:	Stress period:	4	Time step:	94
Solving:	Stress period:	4	Time step:	95
Solving:	Stress period:	4	Time step:	96
Solving:	Stress period:	4	Time step:	97
Solving:	Stress period:	4	Time step:	98
Solving:	Stress period:	4	Time step:	99
Solving:	Stress period:	4	Time step:	100
Solving:	Stress period:	4	Time step:	101
Solving:	Stress period:	4	Time step:	102
Solving:	Stress period:	4	Time step:	103
Solving:	Stress period:	4	Time step:	104
Solving:	Stress period:	4	Time step:	105
Solving:	Stress period:	4	Time step:	106
Solving:	Stress period:	4	Time step:	107
Solving:	Stress period:	4	Time step:	108
Solving:	Stress period:	4	Time step:	109
Solving:	Stress period:	4	Time step:	110
Solving:	Stress period:	4	Time step:	111
Solving:	Stress period:	4	Time step:	112
Solving:	Stress period:	4	Time step:	113
Solving:	Stress period:	4	Time step:	114
Solving:	Stress period:	4	Time step:	115
Solving:	Stress period:	4	Time step:	116
Solving:	Stress period:	4	Time step:	117
Solving:	Stress period:	4	Time step:	118

(continues on next page)

(continued from previous page)

```
Solving: Stress period: 4 Time step: 119
Solving: Stress period: 4 Time step: 120
```

```
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:36
Elapsed run time: 0.602 Seconds
```

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
 2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.
- Normal termination of simulation.

Method 3: Pass timeseries a dictionary of anything that could be passed to `ghb.ts.initialize`

```
[8]: # build ghb stress period data
ghb_spd_ts = {}
ghb_period = []
for layer, cond in zip(range(1, 3), [15.0, 1500.0]):
    for row in range(0, 15):
        ghb_period.append((layer, row, 9), "tides", cond, "Estuary-L2"))
ghb_spd_ts[0] = ghb_period

# build ts data
ts_data = []
for n in range(0, 365):
    ts_data.append((float(n / 11.73), float(n / 60.0)))
ts_dict = {
    "timeseries": ts_data,
    "time_series_namerecord": "tides",
    "interpolation_methodrecord": "linear",
    "filename": "method3.ts",
}

# build obs data
ghb_obs = {
    ("ghb_obs.csv", "binary"): [
        ("ghb-2-6-10", "GHB", (1, 5, 9)),
        ("ghb-3-6-10", "GHB", (2, 5, 9)),
    ],
    "ghb_flows.csv": [
        ("Estuary2", "GHB", "Estuary-L2"),
        ("Estuary3", "GHB", "Estuary-L3"),
    ],
}

# build ghb package
ghb = flopy.mf6.modflow.mfgwfghb.ModflowGwfghb(
    model,
```

(continues on next page)

(continued from previous page)

```

print_input=True,
print_flows=True,
save_flows=True,
boundnames=True,
pname="ghb",
timeseries=ts_dict,
maxbound=30,
stress_period_data=ghb_spd_ts,
)

sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

# clean up for next example
model.remove_package("ghb")

```

```

writing simulation...
writing simulation name file...
writing simulation tdis package...
writing solution package ims_-1...
writing model child_pkgs...
writing model name file...
writing package dis...
writing package ic...
writing package npf...
writing package oc...
writing package sto...
writing package ghb...
writing package ts_0...

```

```

MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

```

```

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water

(continues on next page)

(continued from previous page)

Resources Software User Rights Notice for complete use, copyright,
and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:36

Writing simulation list file: mfsim.lst

Using Simulation name file: mfsim.nam

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5
Solving:	Stress period:	2	Time step:	6
Solving:	Stress period:	2	Time step:	7
Solving:	Stress period:	2	Time step:	8
Solving:	Stress period:	2	Time step:	9
Solving:	Stress period:	2	Time step:	10
Solving:	Stress period:	2	Time step:	11
Solving:	Stress period:	2	Time step:	12
Solving:	Stress period:	2	Time step:	13
Solving:	Stress period:	2	Time step:	14
Solving:	Stress period:	2	Time step:	15
Solving:	Stress period:	2	Time step:	16
Solving:	Stress period:	2	Time step:	17
Solving:	Stress period:	2	Time step:	18
Solving:	Stress period:	2	Time step:	19
Solving:	Stress period:	2	Time step:	20
Solving:	Stress period:	2	Time step:	21
Solving:	Stress period:	2	Time step:	22
Solving:	Stress period:	2	Time step:	23
Solving:	Stress period:	2	Time step:	24
Solving:	Stress period:	2	Time step:	25
Solving:	Stress period:	2	Time step:	26
Solving:	Stress period:	2	Time step:	27
Solving:	Stress period:	2	Time step:	28
Solving:	Stress period:	2	Time step:	29
Solving:	Stress period:	2	Time step:	30
Solving:	Stress period:	2	Time step:	31
Solving:	Stress period:	2	Time step:	32
Solving:	Stress period:	2	Time step:	33
Solving:	Stress period:	2	Time step:	34
Solving:	Stress period:	2	Time step:	35
Solving:	Stress period:	2	Time step:	36
Solving:	Stress period:	2	Time step:	37
Solving:	Stress period:	2	Time step:	38
Solving:	Stress period:	2	Time step:	39
Solving:	Stress period:	2	Time step:	40
Solving:	Stress period:	2	Time step:	41
Solving:	Stress period:	2	Time step:	42

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	43
Solving:	Stress period:	2	Time step:	44
Solving:	Stress period:	2	Time step:	45
Solving:	Stress period:	2	Time step:	46
Solving:	Stress period:	2	Time step:	47
Solving:	Stress period:	2	Time step:	48
Solving:	Stress period:	2	Time step:	49
Solving:	Stress period:	2	Time step:	50
Solving:	Stress period:	2	Time step:	51
Solving:	Stress period:	2	Time step:	52
Solving:	Stress period:	2	Time step:	53
Solving:	Stress period:	2	Time step:	54
Solving:	Stress period:	2	Time step:	55
Solving:	Stress period:	2	Time step:	56
Solving:	Stress period:	2	Time step:	57
Solving:	Stress period:	2	Time step:	58
Solving:	Stress period:	2	Time step:	59
Solving:	Stress period:	2	Time step:	60
Solving:	Stress period:	2	Time step:	61
Solving:	Stress period:	2	Time step:	62
Solving:	Stress period:	2	Time step:	63
Solving:	Stress period:	2	Time step:	64
Solving:	Stress period:	2	Time step:	65
Solving:	Stress period:	2	Time step:	66
Solving:	Stress period:	2	Time step:	67
Solving:	Stress period:	2	Time step:	68
Solving:	Stress period:	2	Time step:	69
Solving:	Stress period:	2	Time step:	70
Solving:	Stress period:	2	Time step:	71
Solving:	Stress period:	2	Time step:	72
Solving:	Stress period:	2	Time step:	73
Solving:	Stress period:	2	Time step:	74
Solving:	Stress period:	2	Time step:	75
Solving:	Stress period:	2	Time step:	76
Solving:	Stress period:	2	Time step:	77
Solving:	Stress period:	2	Time step:	78
Solving:	Stress period:	2	Time step:	79
Solving:	Stress period:	2	Time step:	80
Solving:	Stress period:	2	Time step:	81
Solving:	Stress period:	2	Time step:	82
Solving:	Stress period:	2	Time step:	83
Solving:	Stress period:	2	Time step:	84
Solving:	Stress period:	2	Time step:	85
Solving:	Stress period:	2	Time step:	86
Solving:	Stress period:	2	Time step:	87
Solving:	Stress period:	2	Time step:	88
Solving:	Stress period:	2	Time step:	89
Solving:	Stress period:	2	Time step:	90
Solving:	Stress period:	2	Time step:	91
Solving:	Stress period:	2	Time step:	92
Solving:	Stress period:	2	Time step:	93
Solving:	Stress period:	2	Time step:	94

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	95
Solving:	Stress period:	2	Time step:	96
Solving:	Stress period:	2	Time step:	97
Solving:	Stress period:	2	Time step:	98
Solving:	Stress period:	2	Time step:	99
Solving:	Stress period:	2	Time step:	100
Solving:	Stress period:	2	Time step:	101
Solving:	Stress period:	2	Time step:	102
Solving:	Stress period:	2	Time step:	103
Solving:	Stress period:	2	Time step:	104
Solving:	Stress period:	2	Time step:	105
Solving:	Stress period:	2	Time step:	106
Solving:	Stress period:	2	Time step:	107
Solving:	Stress period:	2	Time step:	108
Solving:	Stress period:	2	Time step:	109
Solving:	Stress period:	2	Time step:	110
Solving:	Stress period:	2	Time step:	111
Solving:	Stress period:	2	Time step:	112
Solving:	Stress period:	2	Time step:	113
Solving:	Stress period:	2	Time step:	114
Solving:	Stress period:	2	Time step:	115
Solving:	Stress period:	2	Time step:	116
Solving:	Stress period:	2	Time step:	117
Solving:	Stress period:	2	Time step:	118
Solving:	Stress period:	2	Time step:	119
Solving:	Stress period:	2	Time step:	120
Solving:	Stress period:	3	Time step:	1
Solving:	Stress period:	3	Time step:	2
Solving:	Stress period:	3	Time step:	3
Solving:	Stress period:	3	Time step:	4
Solving:	Stress period:	3	Time step:	5
Solving:	Stress period:	3	Time step:	6
Solving:	Stress period:	3	Time step:	7
Solving:	Stress period:	3	Time step:	8
Solving:	Stress period:	3	Time step:	9
Solving:	Stress period:	3	Time step:	10
Solving:	Stress period:	3	Time step:	11
Solving:	Stress period:	3	Time step:	12
Solving:	Stress period:	3	Time step:	13
Solving:	Stress period:	3	Time step:	14
Solving:	Stress period:	3	Time step:	15
Solving:	Stress period:	3	Time step:	16
Solving:	Stress period:	3	Time step:	17
Solving:	Stress period:	3	Time step:	18
Solving:	Stress period:	3	Time step:	19
Solving:	Stress period:	3	Time step:	20
Solving:	Stress period:	3	Time step:	21
Solving:	Stress period:	3	Time step:	22
Solving:	Stress period:	3	Time step:	23
Solving:	Stress period:	3	Time step:	24
Solving:	Stress period:	3	Time step:	25
Solving:	Stress period:	3	Time step:	26

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	27
Solving:	Stress period:	3	Time step:	28
Solving:	Stress period:	3	Time step:	29
Solving:	Stress period:	3	Time step:	30
Solving:	Stress period:	3	Time step:	31
Solving:	Stress period:	3	Time step:	32
Solving:	Stress period:	3	Time step:	33
Solving:	Stress period:	3	Time step:	34
Solving:	Stress period:	3	Time step:	35
Solving:	Stress period:	3	Time step:	36
Solving:	Stress period:	3	Time step:	37
Solving:	Stress period:	3	Time step:	38
Solving:	Stress period:	3	Time step:	39
Solving:	Stress period:	3	Time step:	40
Solving:	Stress period:	3	Time step:	41
Solving:	Stress period:	3	Time step:	42
Solving:	Stress period:	3	Time step:	43
Solving:	Stress period:	3	Time step:	44
Solving:	Stress period:	3	Time step:	45
Solving:	Stress period:	3	Time step:	46
Solving:	Stress period:	3	Time step:	47
Solving:	Stress period:	3	Time step:	48
Solving:	Stress period:	3	Time step:	49
Solving:	Stress period:	3	Time step:	50
Solving:	Stress period:	3	Time step:	51
Solving:	Stress period:	3	Time step:	52
Solving:	Stress period:	3	Time step:	53
Solving:	Stress period:	3	Time step:	54
Solving:	Stress period:	3	Time step:	55
Solving:	Stress period:	3	Time step:	56
Solving:	Stress period:	3	Time step:	57
Solving:	Stress period:	3	Time step:	58
Solving:	Stress period:	3	Time step:	59
Solving:	Stress period:	3	Time step:	60
Solving:	Stress period:	3	Time step:	61
Solving:	Stress period:	3	Time step:	62
Solving:	Stress period:	3	Time step:	63
Solving:	Stress period:	3	Time step:	64
Solving:	Stress period:	3	Time step:	65
Solving:	Stress period:	3	Time step:	66
Solving:	Stress period:	3	Time step:	67
Solving:	Stress period:	3	Time step:	68
Solving:	Stress period:	3	Time step:	69
Solving:	Stress period:	3	Time step:	70
Solving:	Stress period:	3	Time step:	71
Solving:	Stress period:	3	Time step:	72
Solving:	Stress period:	3	Time step:	73
Solving:	Stress period:	3	Time step:	74
Solving:	Stress period:	3	Time step:	75
Solving:	Stress period:	3	Time step:	76
Solving:	Stress period:	3	Time step:	77
Solving:	Stress period:	3	Time step:	78

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	79
Solving:	Stress period:	3	Time step:	80
Solving:	Stress period:	3	Time step:	81
Solving:	Stress period:	3	Time step:	82
Solving:	Stress period:	3	Time step:	83
Solving:	Stress period:	3	Time step:	84
Solving:	Stress period:	3	Time step:	85
Solving:	Stress period:	3	Time step:	86
Solving:	Stress period:	3	Time step:	87
Solving:	Stress period:	3	Time step:	88
Solving:	Stress period:	3	Time step:	89
Solving:	Stress period:	3	Time step:	90
Solving:	Stress period:	3	Time step:	91
Solving:	Stress period:	3	Time step:	92
Solving:	Stress period:	3	Time step:	93
Solving:	Stress period:	3	Time step:	94
Solving:	Stress period:	3	Time step:	95
Solving:	Stress period:	3	Time step:	96
Solving:	Stress period:	3	Time step:	97
Solving:	Stress period:	3	Time step:	98
Solving:	Stress period:	3	Time step:	99
Solving:	Stress period:	3	Time step:	100
Solving:	Stress period:	3	Time step:	101
Solving:	Stress period:	3	Time step:	102
Solving:	Stress period:	3	Time step:	103
Solving:	Stress period:	3	Time step:	104
Solving:	Stress period:	3	Time step:	105
Solving:	Stress period:	3	Time step:	106
Solving:	Stress period:	3	Time step:	107
Solving:	Stress period:	3	Time step:	108
Solving:	Stress period:	3	Time step:	109
Solving:	Stress period:	3	Time step:	110
Solving:	Stress period:	3	Time step:	111
Solving:	Stress period:	3	Time step:	112
Solving:	Stress period:	3	Time step:	113
Solving:	Stress period:	3	Time step:	114
Solving:	Stress period:	3	Time step:	115
Solving:	Stress period:	3	Time step:	116
Solving:	Stress period:	3	Time step:	117
Solving:	Stress period:	3	Time step:	118
Solving:	Stress period:	3	Time step:	119
Solving:	Stress period:	3	Time step:	120
Solving:	Stress period:	4	Time step:	1
Solving:	Stress period:	4	Time step:	2
Solving:	Stress period:	4	Time step:	3
Solving:	Stress period:	4	Time step:	4
Solving:	Stress period:	4	Time step:	5
Solving:	Stress period:	4	Time step:	6
Solving:	Stress period:	4	Time step:	7
Solving:	Stress period:	4	Time step:	8
Solving:	Stress period:	4	Time step:	9
Solving:	Stress period:	4	Time step:	10

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	11
Solving:	Stress period:	4	Time step:	12
Solving:	Stress period:	4	Time step:	13
Solving:	Stress period:	4	Time step:	14
Solving:	Stress period:	4	Time step:	15
Solving:	Stress period:	4	Time step:	16
Solving:	Stress period:	4	Time step:	17
Solving:	Stress period:	4	Time step:	18
Solving:	Stress period:	4	Time step:	19
Solving:	Stress period:	4	Time step:	20
Solving:	Stress period:	4	Time step:	21
Solving:	Stress period:	4	Time step:	22
Solving:	Stress period:	4	Time step:	23
Solving:	Stress period:	4	Time step:	24
Solving:	Stress period:	4	Time step:	25
Solving:	Stress period:	4	Time step:	26
Solving:	Stress period:	4	Time step:	27
Solving:	Stress period:	4	Time step:	28
Solving:	Stress period:	4	Time step:	29
Solving:	Stress period:	4	Time step:	30
Solving:	Stress period:	4	Time step:	31
Solving:	Stress period:	4	Time step:	32
Solving:	Stress period:	4	Time step:	33
Solving:	Stress period:	4	Time step:	34
Solving:	Stress period:	4	Time step:	35
Solving:	Stress period:	4	Time step:	36
Solving:	Stress period:	4	Time step:	37
Solving:	Stress period:	4	Time step:	38
Solving:	Stress period:	4	Time step:	39
Solving:	Stress period:	4	Time step:	40
Solving:	Stress period:	4	Time step:	41
Solving:	Stress period:	4	Time step:	42
Solving:	Stress period:	4	Time step:	43
Solving:	Stress period:	4	Time step:	44
Solving:	Stress period:	4	Time step:	45
Solving:	Stress period:	4	Time step:	46
Solving:	Stress period:	4	Time step:	47
Solving:	Stress period:	4	Time step:	48
Solving:	Stress period:	4	Time step:	49
Solving:	Stress period:	4	Time step:	50
Solving:	Stress period:	4	Time step:	51
Solving:	Stress period:	4	Time step:	52
Solving:	Stress period:	4	Time step:	53
Solving:	Stress period:	4	Time step:	54
Solving:	Stress period:	4	Time step:	55
Solving:	Stress period:	4	Time step:	56
Solving:	Stress period:	4	Time step:	57
Solving:	Stress period:	4	Time step:	58
Solving:	Stress period:	4	Time step:	59
Solving:	Stress period:	4	Time step:	60
Solving:	Stress period:	4	Time step:	61
Solving:	Stress period:	4	Time step:	62

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	63
Solving:	Stress period:	4	Time step:	64
Solving:	Stress period:	4	Time step:	65
Solving:	Stress period:	4	Time step:	66
Solving:	Stress period:	4	Time step:	67
Solving:	Stress period:	4	Time step:	68
Solving:	Stress period:	4	Time step:	69
Solving:	Stress period:	4	Time step:	70
Solving:	Stress period:	4	Time step:	71
Solving:	Stress period:	4	Time step:	72
Solving:	Stress period:	4	Time step:	73
Solving:	Stress period:	4	Time step:	74
Solving:	Stress period:	4	Time step:	75
Solving:	Stress period:	4	Time step:	76
Solving:	Stress period:	4	Time step:	77
Solving:	Stress period:	4	Time step:	78
Solving:	Stress period:	4	Time step:	79
Solving:	Stress period:	4	Time step:	80
Solving:	Stress period:	4	Time step:	81
Solving:	Stress period:	4	Time step:	82
Solving:	Stress period:	4	Time step:	83
Solving:	Stress period:	4	Time step:	84
Solving:	Stress period:	4	Time step:	85
Solving:	Stress period:	4	Time step:	86
Solving:	Stress period:	4	Time step:	87
Solving:	Stress period:	4	Time step:	88
Solving:	Stress period:	4	Time step:	89
Solving:	Stress period:	4	Time step:	90
Solving:	Stress period:	4	Time step:	91
Solving:	Stress period:	4	Time step:	92
Solving:	Stress period:	4	Time step:	93
Solving:	Stress period:	4	Time step:	94
Solving:	Stress period:	4	Time step:	95
Solving:	Stress period:	4	Time step:	96
Solving:	Stress period:	4	Time step:	97
Solving:	Stress period:	4	Time step:	98
Solving:	Stress period:	4	Time step:	99
Solving:	Stress period:	4	Time step:	100
Solving:	Stress period:	4	Time step:	101
Solving:	Stress period:	4	Time step:	102
Solving:	Stress period:	4	Time step:	103
Solving:	Stress period:	4	Time step:	104
Solving:	Stress period:	4	Time step:	105
Solving:	Stress period:	4	Time step:	106
Solving:	Stress period:	4	Time step:	107
Solving:	Stress period:	4	Time step:	108
Solving:	Stress period:	4	Time step:	109
Solving:	Stress period:	4	Time step:	110
Solving:	Stress period:	4	Time step:	111
Solving:	Stress period:	4	Time step:	112
Solving:	Stress period:	4	Time step:	113
Solving:	Stress period:	4	Time step:	114

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 4 Time step: 115
Solving: Stress period: 4 Time step: 116
Solving: Stress period: 4 Time step: 117
Solving: Stress period: 4 Time step: 118
Solving: Stress period: 4 Time step: 119
Solving: Stress period: 4 Time step: 120

```

```

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:37
Elapsed run time: 0.496 Seconds

```

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
 2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.
- Normal termination of simulation.

Multiple time series packages

```

[9]: # build ghb stress period data
ghb_spd_ts = {}
ghb_period = []
for layer, cond in zip(range(1, 3), [15.0, 1500.0]):
    for row in range(0, 15):
        if row < 10:
            ghb_period.append(((layer, row, 9), "tides", cond, "Estuary-L2"))
        else:
            ghb_period.append(((layer, row, 9), "w1", cond, "Estuary-L2"))
ghb_spd_ts[0] = ghb_period

# build ts data
ts_data = []
for n in range(0, 365):
    ts_data.append((float(n / 11.73), float(n / 60.0)))
ts_data2 = []
for n in range(0, 365):
    ts_data2.append((float(0.0 + (n / 11.73)), 2 * float(n / 60.0)))
ts_data3 = []
for n in range(0, 365):
    ts_data3.append((float(0.0 + (n / 11.73)), 1.5 * float(n / 60.0)))

# build obs data
ghb_obs = {
    ("ghb_obs.csv", "binary"): [
        ("ghb-2-6-10", "GHB", (1, 5, 9)),
        ("ghb-3-6-10", "GHB", (2, 5, 9)),
    ],
}

```

(continues on next page)

(continued from previous page)

```

    "ghb_flows.csv": [
        ("Estuary2", "GHB", "Estuary-L2"),
        ("Estuary3", "GHB", "Estuary-L3"),
    ],
}
# build ghb package
ghb = flopy.mf6.modflow.mfgwfghb.ModflowGwfghb(
    model,
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    pname="ghb",
    maxbound=30,
    stress_period_data=ghb_spd_ts,
)

# initialize time series
ghb.ts.initialize(
    filename="tides.ts",
    timeseries=ts_data,
    time_series_namerecord="tides",
    interpolation_methodrecord="linearend",
    sfacrecord=1.1,
)

# append additional time series
ghb.ts.append_package(
    filename="wls.ts",
    timeseries=ts_data2,
    time_series_namerecord="w1",
    interpolation_methodrecord="stepwise",
    sfacrecord=1.2,
)

# append additional time series
ghb.ts.append_package(
    filename="wls2.ts",
    timeseries=ts_data3,
    time_series_namerecord="w12",
    interpolation_methodrecord="stepwise",
    sfacrecord=1.3,
)

# retrieve information from each time series
print(
    "{} is using {} interpolation".format(
        ghb.ts[0].filename,
        ghb.ts[0].interpolation_methodrecord.get_data()[0][0],
    )
)

print(
    "{} is using {} interpolation".format(
        ghb.ts[1].filename,

```

(continues on next page)

(continued from previous page)

```

        ghb.ts[1].interpolation_methodrecord.get_data()[0][0],
    )
)
print(
    "{} is using {} interpolation".format(
        ghb.ts[2].filename,
        ghb.ts[2].interpolation_methodrecord.get_data()[0][0],
    )
)

sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
tides.ts is using linearend interpolation
wls.ts is using stepwise interpolation
wls2.ts is using stepwise interpolation
writing simulation...
    writing simulation name file...
    writing simulation tdis package...
    writing solution package ims_1...
    writing model child_pkgs...
        writing model name file...
        writing package dis...
        writing package ic...
        writing package npf...
        writing package oc...
        writing package sto...
        writing package ghb...
        writing package ts_0...
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
    writing package ts_1...
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
    writing package ts_2...

```

(continues on next page)

(continued from previous page)

WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
 ↳ 'sfacval') based on shape "time_series_name".

MODFLOW 6
 U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
 VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
 Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
 Build 20220726_000000

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:37

Writing simulation list file: mfsim.lst
 Using Simulation name file: mfsim.nam

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5
Solving:	Stress period:	2	Time step:	6
Solving:	Stress period:	2	Time step:	7
Solving:	Stress period:	2	Time step:	8
Solving:	Stress period:	2	Time step:	9
Solving:	Stress period:	2	Time step:	10
Solving:	Stress period:	2	Time step:	11
Solving:	Stress period:	2	Time step:	12
Solving:	Stress period:	2	Time step:	13
Solving:	Stress period:	2	Time step:	14
Solving:	Stress period:	2	Time step:	15
Solving:	Stress period:	2	Time step:	16
Solving:	Stress period:	2	Time step:	17
Solving:	Stress period:	2	Time step:	18
Solving:	Stress period:	2	Time step:	19
Solving:	Stress period:	2	Time step:	20
Solving:	Stress period:	2	Time step:	21
Solving:	Stress period:	2	Time step:	22

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	23
Solving:	Stress period:	2	Time step:	24
Solving:	Stress period:	2	Time step:	25
Solving:	Stress period:	2	Time step:	26
Solving:	Stress period:	2	Time step:	27
Solving:	Stress period:	2	Time step:	28
Solving:	Stress period:	2	Time step:	29
Solving:	Stress period:	2	Time step:	30
Solving:	Stress period:	2	Time step:	31
Solving:	Stress period:	2	Time step:	32
Solving:	Stress period:	2	Time step:	33
Solving:	Stress period:	2	Time step:	34
Solving:	Stress period:	2	Time step:	35
Solving:	Stress period:	2	Time step:	36
Solving:	Stress period:	2	Time step:	37
Solving:	Stress period:	2	Time step:	38
Solving:	Stress period:	2	Time step:	39
Solving:	Stress period:	2	Time step:	40
Solving:	Stress period:	2	Time step:	41
Solving:	Stress period:	2	Time step:	42
Solving:	Stress period:	2	Time step:	43
Solving:	Stress period:	2	Time step:	44
Solving:	Stress period:	2	Time step:	45
Solving:	Stress period:	2	Time step:	46
Solving:	Stress period:	2	Time step:	47
Solving:	Stress period:	2	Time step:	48
Solving:	Stress period:	2	Time step:	49
Solving:	Stress period:	2	Time step:	50
Solving:	Stress period:	2	Time step:	51
Solving:	Stress period:	2	Time step:	52
Solving:	Stress period:	2	Time step:	53
Solving:	Stress period:	2	Time step:	54
Solving:	Stress period:	2	Time step:	55
Solving:	Stress period:	2	Time step:	56
Solving:	Stress period:	2	Time step:	57
Solving:	Stress period:	2	Time step:	58
Solving:	Stress period:	2	Time step:	59
Solving:	Stress period:	2	Time step:	60
Solving:	Stress period:	2	Time step:	61
Solving:	Stress period:	2	Time step:	62
Solving:	Stress period:	2	Time step:	63
Solving:	Stress period:	2	Time step:	64
Solving:	Stress period:	2	Time step:	65
Solving:	Stress period:	2	Time step:	66
Solving:	Stress period:	2	Time step:	67
Solving:	Stress period:	2	Time step:	68
Solving:	Stress period:	2	Time step:	69
Solving:	Stress period:	2	Time step:	70
Solving:	Stress period:	2	Time step:	71
Solving:	Stress period:	2	Time step:	72
Solving:	Stress period:	2	Time step:	73
Solving:	Stress period:	2	Time step:	74

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	75
Solving:	Stress period:	2	Time step:	76
Solving:	Stress period:	2	Time step:	77
Solving:	Stress period:	2	Time step:	78
Solving:	Stress period:	2	Time step:	79
Solving:	Stress period:	2	Time step:	80
Solving:	Stress period:	2	Time step:	81
Solving:	Stress period:	2	Time step:	82
Solving:	Stress period:	2	Time step:	83
Solving:	Stress period:	2	Time step:	84
Solving:	Stress period:	2	Time step:	85
Solving:	Stress period:	2	Time step:	86
Solving:	Stress period:	2	Time step:	87
Solving:	Stress period:	2	Time step:	88
Solving:	Stress period:	2	Time step:	89
Solving:	Stress period:	2	Time step:	90
Solving:	Stress period:	2	Time step:	91
Solving:	Stress period:	2	Time step:	92
Solving:	Stress period:	2	Time step:	93
Solving:	Stress period:	2	Time step:	94
Solving:	Stress period:	2	Time step:	95
Solving:	Stress period:	2	Time step:	96
Solving:	Stress period:	2	Time step:	97
Solving:	Stress period:	2	Time step:	98
Solving:	Stress period:	2	Time step:	99
Solving:	Stress period:	2	Time step:	100
Solving:	Stress period:	2	Time step:	101
Solving:	Stress period:	2	Time step:	102
Solving:	Stress period:	2	Time step:	103
Solving:	Stress period:	2	Time step:	104
Solving:	Stress period:	2	Time step:	105
Solving:	Stress period:	2	Time step:	106
Solving:	Stress period:	2	Time step:	107
Solving:	Stress period:	2	Time step:	108
Solving:	Stress period:	2	Time step:	109
Solving:	Stress period:	2	Time step:	110
Solving:	Stress period:	2	Time step:	111
Solving:	Stress period:	2	Time step:	112
Solving:	Stress period:	2	Time step:	113
Solving:	Stress period:	2	Time step:	114
Solving:	Stress period:	2	Time step:	115
Solving:	Stress period:	2	Time step:	116
Solving:	Stress period:	2	Time step:	117
Solving:	Stress period:	2	Time step:	118
Solving:	Stress period:	2	Time step:	119
Solving:	Stress period:	2	Time step:	120
Solving:	Stress period:	3	Time step:	1
Solving:	Stress period:	3	Time step:	2
Solving:	Stress period:	3	Time step:	3
Solving:	Stress period:	3	Time step:	4
Solving:	Stress period:	3	Time step:	5
Solving:	Stress period:	3	Time step:	6

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	7
Solving:	Stress period:	3	Time step:	8
Solving:	Stress period:	3	Time step:	9
Solving:	Stress period:	3	Time step:	10
Solving:	Stress period:	3	Time step:	11
Solving:	Stress period:	3	Time step:	12
Solving:	Stress period:	3	Time step:	13
Solving:	Stress period:	3	Time step:	14
Solving:	Stress period:	3	Time step:	15
Solving:	Stress period:	3	Time step:	16
Solving:	Stress period:	3	Time step:	17
Solving:	Stress period:	3	Time step:	18
Solving:	Stress period:	3	Time step:	19
Solving:	Stress period:	3	Time step:	20
Solving:	Stress period:	3	Time step:	21
Solving:	Stress period:	3	Time step:	22
Solving:	Stress period:	3	Time step:	23
Solving:	Stress period:	3	Time step:	24
Solving:	Stress period:	3	Time step:	25
Solving:	Stress period:	3	Time step:	26
Solving:	Stress period:	3	Time step:	27
Solving:	Stress period:	3	Time step:	28
Solving:	Stress period:	3	Time step:	29
Solving:	Stress period:	3	Time step:	30
Solving:	Stress period:	3	Time step:	31
Solving:	Stress period:	3	Time step:	32
Solving:	Stress period:	3	Time step:	33
Solving:	Stress period:	3	Time step:	34
Solving:	Stress period:	3	Time step:	35
Solving:	Stress period:	3	Time step:	36
Solving:	Stress period:	3	Time step:	37
Solving:	Stress period:	3	Time step:	38
Solving:	Stress period:	3	Time step:	39
Solving:	Stress period:	3	Time step:	40
Solving:	Stress period:	3	Time step:	41
Solving:	Stress period:	3	Time step:	42
Solving:	Stress period:	3	Time step:	43
Solving:	Stress period:	3	Time step:	44
Solving:	Stress period:	3	Time step:	45
Solving:	Stress period:	3	Time step:	46
Solving:	Stress period:	3	Time step:	47
Solving:	Stress period:	3	Time step:	48
Solving:	Stress period:	3	Time step:	49
Solving:	Stress period:	3	Time step:	50
Solving:	Stress period:	3	Time step:	51
Solving:	Stress period:	3	Time step:	52
Solving:	Stress period:	3	Time step:	53
Solving:	Stress period:	3	Time step:	54
Solving:	Stress period:	3	Time step:	55
Solving:	Stress period:	3	Time step:	56
Solving:	Stress period:	3	Time step:	57
Solving:	Stress period:	3	Time step:	58

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	59
Solving:	Stress period:	3	Time step:	60
Solving:	Stress period:	3	Time step:	61
Solving:	Stress period:	3	Time step:	62
Solving:	Stress period:	3	Time step:	63
Solving:	Stress period:	3	Time step:	64
Solving:	Stress period:	3	Time step:	65
Solving:	Stress period:	3	Time step:	66
Solving:	Stress period:	3	Time step:	67
Solving:	Stress period:	3	Time step:	68
Solving:	Stress period:	3	Time step:	69
Solving:	Stress period:	3	Time step:	70
Solving:	Stress period:	3	Time step:	71
Solving:	Stress period:	3	Time step:	72
Solving:	Stress period:	3	Time step:	73
Solving:	Stress period:	3	Time step:	74
Solving:	Stress period:	3	Time step:	75
Solving:	Stress period:	3	Time step:	76
Solving:	Stress period:	3	Time step:	77
Solving:	Stress period:	3	Time step:	78
Solving:	Stress period:	3	Time step:	79
Solving:	Stress period:	3	Time step:	80
Solving:	Stress period:	3	Time step:	81
Solving:	Stress period:	3	Time step:	82
Solving:	Stress period:	3	Time step:	83
Solving:	Stress period:	3	Time step:	84
Solving:	Stress period:	3	Time step:	85
Solving:	Stress period:	3	Time step:	86
Solving:	Stress period:	3	Time step:	87
Solving:	Stress period:	3	Time step:	88
Solving:	Stress period:	3	Time step:	89
Solving:	Stress period:	3	Time step:	90
Solving:	Stress period:	3	Time step:	91
Solving:	Stress period:	3	Time step:	92
Solving:	Stress period:	3	Time step:	93
Solving:	Stress period:	3	Time step:	94
Solving:	Stress period:	3	Time step:	95
Solving:	Stress period:	3	Time step:	96
Solving:	Stress period:	3	Time step:	97
Solving:	Stress period:	3	Time step:	98
Solving:	Stress period:	3	Time step:	99
Solving:	Stress period:	3	Time step:	100
Solving:	Stress period:	3	Time step:	101
Solving:	Stress period:	3	Time step:	102
Solving:	Stress period:	3	Time step:	103
Solving:	Stress period:	3	Time step:	104
Solving:	Stress period:	3	Time step:	105
Solving:	Stress period:	3	Time step:	106
Solving:	Stress period:	3	Time step:	107
Solving:	Stress period:	3	Time step:	108
Solving:	Stress period:	3	Time step:	109
Solving:	Stress period:	3	Time step:	110

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	111
Solving:	Stress period:	3	Time step:	112
Solving:	Stress period:	3	Time step:	113
Solving:	Stress period:	3	Time step:	114
Solving:	Stress period:	3	Time step:	115
Solving:	Stress period:	3	Time step:	116
Solving:	Stress period:	3	Time step:	117
Solving:	Stress period:	3	Time step:	118
Solving:	Stress period:	3	Time step:	119
Solving:	Stress period:	3	Time step:	120
Solving:	Stress period:	4	Time step:	1
Solving:	Stress period:	4	Time step:	2
Solving:	Stress period:	4	Time step:	3
Solving:	Stress period:	4	Time step:	4
Solving:	Stress period:	4	Time step:	5
Solving:	Stress period:	4	Time step:	6
Solving:	Stress period:	4	Time step:	7
Solving:	Stress period:	4	Time step:	8
Solving:	Stress period:	4	Time step:	9
Solving:	Stress period:	4	Time step:	10
Solving:	Stress period:	4	Time step:	11
Solving:	Stress period:	4	Time step:	12
Solving:	Stress period:	4	Time step:	13
Solving:	Stress period:	4	Time step:	14
Solving:	Stress period:	4	Time step:	15
Solving:	Stress period:	4	Time step:	16
Solving:	Stress period:	4	Time step:	17
Solving:	Stress period:	4	Time step:	18
Solving:	Stress period:	4	Time step:	19
Solving:	Stress period:	4	Time step:	20
Solving:	Stress period:	4	Time step:	21
Solving:	Stress period:	4	Time step:	22
Solving:	Stress period:	4	Time step:	23
Solving:	Stress period:	4	Time step:	24
Solving:	Stress period:	4	Time step:	25
Solving:	Stress period:	4	Time step:	26
Solving:	Stress period:	4	Time step:	27
Solving:	Stress period:	4	Time step:	28
Solving:	Stress period:	4	Time step:	29
Solving:	Stress period:	4	Time step:	30
Solving:	Stress period:	4	Time step:	31
Solving:	Stress period:	4	Time step:	32
Solving:	Stress period:	4	Time step:	33
Solving:	Stress period:	4	Time step:	34
Solving:	Stress period:	4	Time step:	35
Solving:	Stress period:	4	Time step:	36
Solving:	Stress period:	4	Time step:	37
Solving:	Stress period:	4	Time step:	38
Solving:	Stress period:	4	Time step:	39
Solving:	Stress period:	4	Time step:	40
Solving:	Stress period:	4	Time step:	41
Solving:	Stress period:	4	Time step:	42

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	43
Solving:	Stress period:	4	Time step:	44
Solving:	Stress period:	4	Time step:	45
Solving:	Stress period:	4	Time step:	46
Solving:	Stress period:	4	Time step:	47
Solving:	Stress period:	4	Time step:	48
Solving:	Stress period:	4	Time step:	49
Solving:	Stress period:	4	Time step:	50
Solving:	Stress period:	4	Time step:	51
Solving:	Stress period:	4	Time step:	52
Solving:	Stress period:	4	Time step:	53
Solving:	Stress period:	4	Time step:	54
Solving:	Stress period:	4	Time step:	55
Solving:	Stress period:	4	Time step:	56
Solving:	Stress period:	4	Time step:	57
Solving:	Stress period:	4	Time step:	58
Solving:	Stress period:	4	Time step:	59
Solving:	Stress period:	4	Time step:	60
Solving:	Stress period:	4	Time step:	61
Solving:	Stress period:	4	Time step:	62
Solving:	Stress period:	4	Time step:	63
Solving:	Stress period:	4	Time step:	64
Solving:	Stress period:	4	Time step:	65
Solving:	Stress period:	4	Time step:	66
Solving:	Stress period:	4	Time step:	67
Solving:	Stress period:	4	Time step:	68
Solving:	Stress period:	4	Time step:	69
Solving:	Stress period:	4	Time step:	70
Solving:	Stress period:	4	Time step:	71
Solving:	Stress period:	4	Time step:	72
Solving:	Stress period:	4	Time step:	73
Solving:	Stress period:	4	Time step:	74
Solving:	Stress period:	4	Time step:	75
Solving:	Stress period:	4	Time step:	76
Solving:	Stress period:	4	Time step:	77
Solving:	Stress period:	4	Time step:	78
Solving:	Stress period:	4	Time step:	79
Solving:	Stress period:	4	Time step:	80
Solving:	Stress period:	4	Time step:	81
Solving:	Stress period:	4	Time step:	82
Solving:	Stress period:	4	Time step:	83
Solving:	Stress period:	4	Time step:	84
Solving:	Stress period:	4	Time step:	85
Solving:	Stress period:	4	Time step:	86
Solving:	Stress period:	4	Time step:	87
Solving:	Stress period:	4	Time step:	88
Solving:	Stress period:	4	Time step:	89
Solving:	Stress period:	4	Time step:	90
Solving:	Stress period:	4	Time step:	91
Solving:	Stress period:	4	Time step:	92
Solving:	Stress period:	4	Time step:	93
Solving:	Stress period:	4	Time step:	94

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 4 Time step: 95
Solving: Stress period: 4 Time step: 96
Solving: Stress period: 4 Time step: 97
Solving: Stress period: 4 Time step: 98
Solving: Stress period: 4 Time step: 99
Solving: Stress period: 4 Time step: 100
Solving: Stress period: 4 Time step: 101
Solving: Stress period: 4 Time step: 102
Solving: Stress period: 4 Time step: 103
Solving: Stress period: 4 Time step: 104
Solving: Stress period: 4 Time step: 105
Solving: Stress period: 4 Time step: 106
Solving: Stress period: 4 Time step: 107
Solving: Stress period: 4 Time step: 108
Solving: Stress period: 4 Time step: 109
Solving: Stress period: 4 Time step: 110
Solving: Stress period: 4 Time step: 111
Solving: Stress period: 4 Time step: 112
Solving: Stress period: 4 Time step: 113
Solving: Stress period: 4 Time step: 114
Solving: Stress period: 4 Time step: 115
Solving: Stress period: 4 Time step: 116
Solving: Stress period: 4 Time step: 117
Solving: Stress period: 4 Time step: 118
Solving: Stress period: 4 Time step: 119
Solving: Stress period: 4 Time step: 120

```

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:38

Elapsed run time: 0.579 Seconds

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
 2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.
- Normal termination of simulation.

Time Array Series

Time array series can be set for any package through the package.tas object, and each package.tas object has several attributes that can be set:

package.tas.filename : str Name of time series file to create. The default is packagename + '.tas', e.g. mymodel.rcha.tas.

package.tas.tas_array : {double:[double]} Array containing the time array series information for specific times.
tas_array = {0.0: 0.0001, 200.0: [0.01, 0.02...]}

package.tas.time_series_namerecord : str Name by which a package references a particular time-array series. The name must be unique among all time-array series used in a package

package.tas.interpolation_methodrecord : list (of strings) List of interpolation methods to use for time array series. Method must be either 'stepwise' or 'linear'.

package.tas.sfacrecord_single : float Scale factor to multiply the time array series data column. Can only be used if there is one time series data column.

Method 1: Pass time array series to package constructor

```
[10]: tas = {0.0: 0.000002, 200.0: 0.0000001}
# create recharge package with time array series data
# flopy will generate a warning that there is not yet a time series name
# record for rchararray_1
rcha = flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha(
    model, timearrayseries=tas, recharge="TIMEARRAYSERIES rchararray_1"
)
# finish defining the time array series properties
rcha.tas.time_series_namerecord = "rchararray_1"
rcha.tas.interpolation_methodrecord = "LINEAR"

sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

# clean up for next example
model.remove_package("rcha")

WARNING: Time array series name rchararray_1 not found in any time series file
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims_1...
  writing model child_pkgs...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
    writing package oc...
    writing package sto...
    writing package ghb...
    writing package ts_0...
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
  writing package ts_1...
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
  writing package ts_2...
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
  writing package rcha_0...
```

(continues on next page)

(continued from previous page)

writing package tas_0...

MODFLOW 6
 U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
 VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
 Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
 Build 20220726_000000

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:38

Writing simulation list file: mfsim.lst

Using Simulation name file: mfsim.nam

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5
Solving:	Stress period:	2	Time step:	6
Solving:	Stress period:	2	Time step:	7
Solving:	Stress period:	2	Time step:	8
Solving:	Stress period:	2	Time step:	9
Solving:	Stress period:	2	Time step:	10
Solving:	Stress period:	2	Time step:	11
Solving:	Stress period:	2	Time step:	12
Solving:	Stress period:	2	Time step:	13
Solving:	Stress period:	2	Time step:	14
Solving:	Stress period:	2	Time step:	15
Solving:	Stress period:	2	Time step:	16
Solving:	Stress period:	2	Time step:	17
Solving:	Stress period:	2	Time step:	18
Solving:	Stress period:	2	Time step:	19
Solving:	Stress period:	2	Time step:	20
Solving:	Stress period:	2	Time step:	21
Solving:	Stress period:	2	Time step:	22
Solving:	Stress period:	2	Time step:	23

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	24
Solving:	Stress period:	2	Time step:	25
Solving:	Stress period:	2	Time step:	26
Solving:	Stress period:	2	Time step:	27
Solving:	Stress period:	2	Time step:	28
Solving:	Stress period:	2	Time step:	29
Solving:	Stress period:	2	Time step:	30
Solving:	Stress period:	2	Time step:	31
Solving:	Stress period:	2	Time step:	32
Solving:	Stress period:	2	Time step:	33
Solving:	Stress period:	2	Time step:	34
Solving:	Stress period:	2	Time step:	35
Solving:	Stress period:	2	Time step:	36
Solving:	Stress period:	2	Time step:	37
Solving:	Stress period:	2	Time step:	38
Solving:	Stress period:	2	Time step:	39
Solving:	Stress period:	2	Time step:	40
Solving:	Stress period:	2	Time step:	41
Solving:	Stress period:	2	Time step:	42
Solving:	Stress period:	2	Time step:	43
Solving:	Stress period:	2	Time step:	44
Solving:	Stress period:	2	Time step:	45
Solving:	Stress period:	2	Time step:	46
Solving:	Stress period:	2	Time step:	47
Solving:	Stress period:	2	Time step:	48
Solving:	Stress period:	2	Time step:	49
Solving:	Stress period:	2	Time step:	50
Solving:	Stress period:	2	Time step:	51
Solving:	Stress period:	2	Time step:	52
Solving:	Stress period:	2	Time step:	53
Solving:	Stress period:	2	Time step:	54
Solving:	Stress period:	2	Time step:	55
Solving:	Stress period:	2	Time step:	56
Solving:	Stress period:	2	Time step:	57
Solving:	Stress period:	2	Time step:	58
Solving:	Stress period:	2	Time step:	59
Solving:	Stress period:	2	Time step:	60
Solving:	Stress period:	2	Time step:	61
Solving:	Stress period:	2	Time step:	62
Solving:	Stress period:	2	Time step:	63
Solving:	Stress period:	2	Time step:	64
Solving:	Stress period:	2	Time step:	65
Solving:	Stress period:	2	Time step:	66
Solving:	Stress period:	2	Time step:	67
Solving:	Stress period:	2	Time step:	68
Solving:	Stress period:	2	Time step:	69
Solving:	Stress period:	2	Time step:	70
Solving:	Stress period:	2	Time step:	71
Solving:	Stress period:	2	Time step:	72
Solving:	Stress period:	2	Time step:	73
Solving:	Stress period:	2	Time step:	74
Solving:	Stress period:	2	Time step:	75

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	76
Solving:	Stress period:	2	Time step:	77
Solving:	Stress period:	2	Time step:	78
Solving:	Stress period:	2	Time step:	79
Solving:	Stress period:	2	Time step:	80
Solving:	Stress period:	2	Time step:	81
Solving:	Stress period:	2	Time step:	82
Solving:	Stress period:	2	Time step:	83
Solving:	Stress period:	2	Time step:	84
Solving:	Stress period:	2	Time step:	85
Solving:	Stress period:	2	Time step:	86
Solving:	Stress period:	2	Time step:	87
Solving:	Stress period:	2	Time step:	88
Solving:	Stress period:	2	Time step:	89
Solving:	Stress period:	2	Time step:	90
Solving:	Stress period:	2	Time step:	91
Solving:	Stress period:	2	Time step:	92
Solving:	Stress period:	2	Time step:	93
Solving:	Stress period:	2	Time step:	94
Solving:	Stress period:	2	Time step:	95
Solving:	Stress period:	2	Time step:	96
Solving:	Stress period:	2	Time step:	97
Solving:	Stress period:	2	Time step:	98
Solving:	Stress period:	2	Time step:	99
Solving:	Stress period:	2	Time step:	100
Solving:	Stress period:	2	Time step:	101
Solving:	Stress period:	2	Time step:	102
Solving:	Stress period:	2	Time step:	103
Solving:	Stress period:	2	Time step:	104
Solving:	Stress period:	2	Time step:	105
Solving:	Stress period:	2	Time step:	106
Solving:	Stress period:	2	Time step:	107
Solving:	Stress period:	2	Time step:	108
Solving:	Stress period:	2	Time step:	109
Solving:	Stress period:	2	Time step:	110
Solving:	Stress period:	2	Time step:	111
Solving:	Stress period:	2	Time step:	112
Solving:	Stress period:	2	Time step:	113
Solving:	Stress period:	2	Time step:	114
Solving:	Stress period:	2	Time step:	115
Solving:	Stress period:	2	Time step:	116
Solving:	Stress period:	2	Time step:	117
Solving:	Stress period:	2	Time step:	118
Solving:	Stress period:	2	Time step:	119
Solving:	Stress period:	2	Time step:	120
Solving:	Stress period:	3	Time step:	1
Solving:	Stress period:	3	Time step:	2
Solving:	Stress period:	3	Time step:	3
Solving:	Stress period:	3	Time step:	4
Solving:	Stress period:	3	Time step:	5
Solving:	Stress period:	3	Time step:	6
Solving:	Stress period:	3	Time step:	7

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	8
Solving:	Stress period:	3	Time step:	9
Solving:	Stress period:	3	Time step:	10
Solving:	Stress period:	3	Time step:	11
Solving:	Stress period:	3	Time step:	12
Solving:	Stress period:	3	Time step:	13
Solving:	Stress period:	3	Time step:	14
Solving:	Stress period:	3	Time step:	15
Solving:	Stress period:	3	Time step:	16
Solving:	Stress period:	3	Time step:	17
Solving:	Stress period:	3	Time step:	18
Solving:	Stress period:	3	Time step:	19
Solving:	Stress period:	3	Time step:	20
Solving:	Stress period:	3	Time step:	21
Solving:	Stress period:	3	Time step:	22
Solving:	Stress period:	3	Time step:	23
Solving:	Stress period:	3	Time step:	24
Solving:	Stress period:	3	Time step:	25
Solving:	Stress period:	3	Time step:	26
Solving:	Stress period:	3	Time step:	27
Solving:	Stress period:	3	Time step:	28
Solving:	Stress period:	3	Time step:	29
Solving:	Stress period:	3	Time step:	30
Solving:	Stress period:	3	Time step:	31
Solving:	Stress period:	3	Time step:	32
Solving:	Stress period:	3	Time step:	33
Solving:	Stress period:	3	Time step:	34
Solving:	Stress period:	3	Time step:	35
Solving:	Stress period:	3	Time step:	36
Solving:	Stress period:	3	Time step:	37
Solving:	Stress period:	3	Time step:	38
Solving:	Stress period:	3	Time step:	39
Solving:	Stress period:	3	Time step:	40
Solving:	Stress period:	3	Time step:	41
Solving:	Stress period:	3	Time step:	42
Solving:	Stress period:	3	Time step:	43
Solving:	Stress period:	3	Time step:	44
Solving:	Stress period:	3	Time step:	45
Solving:	Stress period:	3	Time step:	46
Solving:	Stress period:	3	Time step:	47
Solving:	Stress period:	3	Time step:	48
Solving:	Stress period:	3	Time step:	49
Solving:	Stress period:	3	Time step:	50
Solving:	Stress period:	3	Time step:	51
Solving:	Stress period:	3	Time step:	52
Solving:	Stress period:	3	Time step:	53
Solving:	Stress period:	3	Time step:	54
Solving:	Stress period:	3	Time step:	55
Solving:	Stress period:	3	Time step:	56
Solving:	Stress period:	3	Time step:	57
Solving:	Stress period:	3	Time step:	58
Solving:	Stress period:	3	Time step:	59

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	60
Solving:	Stress period:	3	Time step:	61
Solving:	Stress period:	3	Time step:	62
Solving:	Stress period:	3	Time step:	63
Solving:	Stress period:	3	Time step:	64
Solving:	Stress period:	3	Time step:	65
Solving:	Stress period:	3	Time step:	66
Solving:	Stress period:	3	Time step:	67
Solving:	Stress period:	3	Time step:	68
Solving:	Stress period:	3	Time step:	69
Solving:	Stress period:	3	Time step:	70
Solving:	Stress period:	3	Time step:	71
Solving:	Stress period:	3	Time step:	72
Solving:	Stress period:	3	Time step:	73
Solving:	Stress period:	3	Time step:	74
Solving:	Stress period:	3	Time step:	75
Solving:	Stress period:	3	Time step:	76
Solving:	Stress period:	3	Time step:	77
Solving:	Stress period:	3	Time step:	78
Solving:	Stress period:	3	Time step:	79
Solving:	Stress period:	3	Time step:	80
Solving:	Stress period:	3	Time step:	81
Solving:	Stress period:	3	Time step:	82
Solving:	Stress period:	3	Time step:	83
Solving:	Stress period:	3	Time step:	84
Solving:	Stress period:	3	Time step:	85
Solving:	Stress period:	3	Time step:	86
Solving:	Stress period:	3	Time step:	87
Solving:	Stress period:	3	Time step:	88
Solving:	Stress period:	3	Time step:	89
Solving:	Stress period:	3	Time step:	90
Solving:	Stress period:	3	Time step:	91
Solving:	Stress period:	3	Time step:	92
Solving:	Stress period:	3	Time step:	93
Solving:	Stress period:	3	Time step:	94
Solving:	Stress period:	3	Time step:	95
Solving:	Stress period:	3	Time step:	96
Solving:	Stress period:	3	Time step:	97
Solving:	Stress period:	3	Time step:	98
Solving:	Stress period:	3	Time step:	99
Solving:	Stress period:	3	Time step:	100
Solving:	Stress period:	3	Time step:	101
Solving:	Stress period:	3	Time step:	102
Solving:	Stress period:	3	Time step:	103
Solving:	Stress period:	3	Time step:	104
Solving:	Stress period:	3	Time step:	105
Solving:	Stress period:	3	Time step:	106
Solving:	Stress period:	3	Time step:	107
Solving:	Stress period:	3	Time step:	108
Solving:	Stress period:	3	Time step:	109
Solving:	Stress period:	3	Time step:	110
Solving:	Stress period:	3	Time step:	111

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	112
Solving:	Stress period:	3	Time step:	113
Solving:	Stress period:	3	Time step:	114
Solving:	Stress period:	3	Time step:	115
Solving:	Stress period:	3	Time step:	116
Solving:	Stress period:	3	Time step:	117
Solving:	Stress period:	3	Time step:	118
Solving:	Stress period:	3	Time step:	119
Solving:	Stress period:	3	Time step:	120
Solving:	Stress period:	4	Time step:	1
Solving:	Stress period:	4	Time step:	2
Solving:	Stress period:	4	Time step:	3
Solving:	Stress period:	4	Time step:	4
Solving:	Stress period:	4	Time step:	5
Solving:	Stress period:	4	Time step:	6
Solving:	Stress period:	4	Time step:	7
Solving:	Stress period:	4	Time step:	8
Solving:	Stress period:	4	Time step:	9
Solving:	Stress period:	4	Time step:	10
Solving:	Stress period:	4	Time step:	11
Solving:	Stress period:	4	Time step:	12
Solving:	Stress period:	4	Time step:	13
Solving:	Stress period:	4	Time step:	14
Solving:	Stress period:	4	Time step:	15
Solving:	Stress period:	4	Time step:	16
Solving:	Stress period:	4	Time step:	17
Solving:	Stress period:	4	Time step:	18
Solving:	Stress period:	4	Time step:	19
Solving:	Stress period:	4	Time step:	20
Solving:	Stress period:	4	Time step:	21
Solving:	Stress period:	4	Time step:	22
Solving:	Stress period:	4	Time step:	23
Solving:	Stress period:	4	Time step:	24
Solving:	Stress period:	4	Time step:	25
Solving:	Stress period:	4	Time step:	26
Solving:	Stress period:	4	Time step:	27
Solving:	Stress period:	4	Time step:	28
Solving:	Stress period:	4	Time step:	29
Solving:	Stress period:	4	Time step:	30
Solving:	Stress period:	4	Time step:	31
Solving:	Stress period:	4	Time step:	32
Solving:	Stress period:	4	Time step:	33
Solving:	Stress period:	4	Time step:	34
Solving:	Stress period:	4	Time step:	35
Solving:	Stress period:	4	Time step:	36
Solving:	Stress period:	4	Time step:	37
Solving:	Stress period:	4	Time step:	38
Solving:	Stress period:	4	Time step:	39
Solving:	Stress period:	4	Time step:	40
Solving:	Stress period:	4	Time step:	41
Solving:	Stress period:	4	Time step:	42
Solving:	Stress period:	4	Time step:	43

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	44
Solving:	Stress period:	4	Time step:	45
Solving:	Stress period:	4	Time step:	46
Solving:	Stress period:	4	Time step:	47
Solving:	Stress period:	4	Time step:	48
Solving:	Stress period:	4	Time step:	49
Solving:	Stress period:	4	Time step:	50
Solving:	Stress period:	4	Time step:	51
Solving:	Stress period:	4	Time step:	52
Solving:	Stress period:	4	Time step:	53
Solving:	Stress period:	4	Time step:	54
Solving:	Stress period:	4	Time step:	55
Solving:	Stress period:	4	Time step:	56
Solving:	Stress period:	4	Time step:	57
Solving:	Stress period:	4	Time step:	58
Solving:	Stress period:	4	Time step:	59
Solving:	Stress period:	4	Time step:	60
Solving:	Stress period:	4	Time step:	61
Solving:	Stress period:	4	Time step:	62
Solving:	Stress period:	4	Time step:	63
Solving:	Stress period:	4	Time step:	64
Solving:	Stress period:	4	Time step:	65
Solving:	Stress period:	4	Time step:	66
Solving:	Stress period:	4	Time step:	67
Solving:	Stress period:	4	Time step:	68
Solving:	Stress period:	4	Time step:	69
Solving:	Stress period:	4	Time step:	70
Solving:	Stress period:	4	Time step:	71
Solving:	Stress period:	4	Time step:	72
Solving:	Stress period:	4	Time step:	73
Solving:	Stress period:	4	Time step:	74
Solving:	Stress period:	4	Time step:	75
Solving:	Stress period:	4	Time step:	76
Solving:	Stress period:	4	Time step:	77
Solving:	Stress period:	4	Time step:	78
Solving:	Stress period:	4	Time step:	79
Solving:	Stress period:	4	Time step:	80
Solving:	Stress period:	4	Time step:	81
Solving:	Stress period:	4	Time step:	82
Solving:	Stress period:	4	Time step:	83
Solving:	Stress period:	4	Time step:	84
Solving:	Stress period:	4	Time step:	85
Solving:	Stress period:	4	Time step:	86
Solving:	Stress period:	4	Time step:	87
Solving:	Stress period:	4	Time step:	88
Solving:	Stress period:	4	Time step:	89
Solving:	Stress period:	4	Time step:	90
Solving:	Stress period:	4	Time step:	91
Solving:	Stress period:	4	Time step:	92
Solving:	Stress period:	4	Time step:	93
Solving:	Stress period:	4	Time step:	94
Solving:	Stress period:	4	Time step:	95

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 4 Time step: 96
Solving: Stress period: 4 Time step: 97
Solving: Stress period: 4 Time step: 98
Solving: Stress period: 4 Time step: 99
Solving: Stress period: 4 Time step: 100
Solving: Stress period: 4 Time step: 101
Solving: Stress period: 4 Time step: 102
Solving: Stress period: 4 Time step: 103
Solving: Stress period: 4 Time step: 104
Solving: Stress period: 4 Time step: 105
Solving: Stress period: 4 Time step: 106
Solving: Stress period: 4 Time step: 107
Solving: Stress period: 4 Time step: 108
Solving: Stress period: 4 Time step: 109
Solving: Stress period: 4 Time step: 110
Solving: Stress period: 4 Time step: 111
Solving: Stress period: 4 Time step: 112
Solving: Stress period: 4 Time step: 113
Solving: Stress period: 4 Time step: 114
Solving: Stress period: 4 Time step: 115
Solving: Stress period: 4 Time step: 116
Solving: Stress period: 4 Time step: 117
Solving: Stress period: 4 Time step: 118
Solving: Stress period: 4 Time step: 119
Solving: Stress period: 4 Time step: 120

```

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:38

Elapsed run time: 0.596 Seconds

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
 2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.
- Normal termination of simulation.

Method 2: Initialize time array series through rcha.tas.initialize

```

[11]: # create recharge package with recharge pointing to a time array series
# not yet defined. flopy will generate a warning that there is not yet a
# time series name record for rechararray_1
rcha = flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha(
    model, recharge="TIMEARRAYSERIES rchararray_1"
)
rch_array = 0.000002 * np.ones((15, 10))
rch_array[0, 0] = 0.0001
tas = {0.0: rch_array, 200.0: 0.0000001}

```

(continues on next page)

(continued from previous page)

```

# initialize the time array series
rcha.tas.initialize(
    filename="method2.tas",
    tas_array=tas,
    time_series_namerecord="rcharray_1",
    interpolation_methodrecord="LINEAR",
)

sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

# clean up for next example
model.remove_package("rcha")

```

```

WARNING: Time array series name rcharray_1 not found in any time series file
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims_-1...
  writing model child_pkgs...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
    writing package oc...
    writing package sto...
    writing package ghb...
    writing package ts_0...
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
  writing package ts_1...
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
  writing package ts_2...
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
  writing package rcha_0...
  writing package tas_0...

```

```

MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

```

```

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

```

This software has been approved for release by the U.S. Geological

(continues on next page)

(continued from previous page)

Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:38

Writing simulation list file: mfsim.lst

Using Simulation name file: mfsim.nam

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5
Solving:	Stress period:	2	Time step:	6
Solving:	Stress period:	2	Time step:	7
Solving:	Stress period:	2	Time step:	8
Solving:	Stress period:	2	Time step:	9
Solving:	Stress period:	2	Time step:	10
Solving:	Stress period:	2	Time step:	11
Solving:	Stress period:	2	Time step:	12
Solving:	Stress period:	2	Time step:	13
Solving:	Stress period:	2	Time step:	14
Solving:	Stress period:	2	Time step:	15
Solving:	Stress period:	2	Time step:	16
Solving:	Stress period:	2	Time step:	17
Solving:	Stress period:	2	Time step:	18
Solving:	Stress period:	2	Time step:	19
Solving:	Stress period:	2	Time step:	20
Solving:	Stress period:	2	Time step:	21
Solving:	Stress period:	2	Time step:	22
Solving:	Stress period:	2	Time step:	23
Solving:	Stress period:	2	Time step:	24
Solving:	Stress period:	2	Time step:	25
Solving:	Stress period:	2	Time step:	26
Solving:	Stress period:	2	Time step:	27
Solving:	Stress period:	2	Time step:	28
Solving:	Stress period:	2	Time step:	29
Solving:	Stress period:	2	Time step:	30
Solving:	Stress period:	2	Time step:	31
Solving:	Stress period:	2	Time step:	32
Solving:	Stress period:	2	Time step:	33

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	34
Solving:	Stress period:	2	Time step:	35
Solving:	Stress period:	2	Time step:	36
Solving:	Stress period:	2	Time step:	37
Solving:	Stress period:	2	Time step:	38
Solving:	Stress period:	2	Time step:	39
Solving:	Stress period:	2	Time step:	40
Solving:	Stress period:	2	Time step:	41
Solving:	Stress period:	2	Time step:	42
Solving:	Stress period:	2	Time step:	43
Solving:	Stress period:	2	Time step:	44
Solving:	Stress period:	2	Time step:	45
Solving:	Stress period:	2	Time step:	46
Solving:	Stress period:	2	Time step:	47
Solving:	Stress period:	2	Time step:	48
Solving:	Stress period:	2	Time step:	49
Solving:	Stress period:	2	Time step:	50
Solving:	Stress period:	2	Time step:	51
Solving:	Stress period:	2	Time step:	52
Solving:	Stress period:	2	Time step:	53
Solving:	Stress period:	2	Time step:	54
Solving:	Stress period:	2	Time step:	55
Solving:	Stress period:	2	Time step:	56
Solving:	Stress period:	2	Time step:	57
Solving:	Stress period:	2	Time step:	58
Solving:	Stress period:	2	Time step:	59
Solving:	Stress period:	2	Time step:	60
Solving:	Stress period:	2	Time step:	61
Solving:	Stress period:	2	Time step:	62
Solving:	Stress period:	2	Time step:	63
Solving:	Stress period:	2	Time step:	64
Solving:	Stress period:	2	Time step:	65
Solving:	Stress period:	2	Time step:	66
Solving:	Stress period:	2	Time step:	67
Solving:	Stress period:	2	Time step:	68
Solving:	Stress period:	2	Time step:	69
Solving:	Stress period:	2	Time step:	70
Solving:	Stress period:	2	Time step:	71
Solving:	Stress period:	2	Time step:	72
Solving:	Stress period:	2	Time step:	73
Solving:	Stress period:	2	Time step:	74
Solving:	Stress period:	2	Time step:	75
Solving:	Stress period:	2	Time step:	76
Solving:	Stress period:	2	Time step:	77
Solving:	Stress period:	2	Time step:	78
Solving:	Stress period:	2	Time step:	79
Solving:	Stress period:	2	Time step:	80
Solving:	Stress period:	2	Time step:	81
Solving:	Stress period:	2	Time step:	82
Solving:	Stress period:	2	Time step:	83
Solving:	Stress period:	2	Time step:	84
Solving:	Stress period:	2	Time step:	85

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	86
Solving:	Stress period:	2	Time step:	87
Solving:	Stress period:	2	Time step:	88
Solving:	Stress period:	2	Time step:	89
Solving:	Stress period:	2	Time step:	90
Solving:	Stress period:	2	Time step:	91
Solving:	Stress period:	2	Time step:	92
Solving:	Stress period:	2	Time step:	93
Solving:	Stress period:	2	Time step:	94
Solving:	Stress period:	2	Time step:	95
Solving:	Stress period:	2	Time step:	96
Solving:	Stress period:	2	Time step:	97
Solving:	Stress period:	2	Time step:	98
Solving:	Stress period:	2	Time step:	99
Solving:	Stress period:	2	Time step:	100
Solving:	Stress period:	2	Time step:	101
Solving:	Stress period:	2	Time step:	102
Solving:	Stress period:	2	Time step:	103
Solving:	Stress period:	2	Time step:	104
Solving:	Stress period:	2	Time step:	105
Solving:	Stress period:	2	Time step:	106
Solving:	Stress period:	2	Time step:	107
Solving:	Stress period:	2	Time step:	108
Solving:	Stress period:	2	Time step:	109
Solving:	Stress period:	2	Time step:	110
Solving:	Stress period:	2	Time step:	111
Solving:	Stress period:	2	Time step:	112
Solving:	Stress period:	2	Time step:	113
Solving:	Stress period:	2	Time step:	114
Solving:	Stress period:	2	Time step:	115
Solving:	Stress period:	2	Time step:	116
Solving:	Stress period:	2	Time step:	117
Solving:	Stress period:	2	Time step:	118
Solving:	Stress period:	2	Time step:	119
Solving:	Stress period:	2	Time step:	120
Solving:	Stress period:	3	Time step:	1
Solving:	Stress period:	3	Time step:	2
Solving:	Stress period:	3	Time step:	3
Solving:	Stress period:	3	Time step:	4
Solving:	Stress period:	3	Time step:	5
Solving:	Stress period:	3	Time step:	6
Solving:	Stress period:	3	Time step:	7
Solving:	Stress period:	3	Time step:	8
Solving:	Stress period:	3	Time step:	9
Solving:	Stress period:	3	Time step:	10
Solving:	Stress period:	3	Time step:	11
Solving:	Stress period:	3	Time step:	12
Solving:	Stress period:	3	Time step:	13
Solving:	Stress period:	3	Time step:	14
Solving:	Stress period:	3	Time step:	15
Solving:	Stress period:	3	Time step:	16
Solving:	Stress period:	3	Time step:	17

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	18
Solving:	Stress period:	3	Time step:	19
Solving:	Stress period:	3	Time step:	20
Solving:	Stress period:	3	Time step:	21
Solving:	Stress period:	3	Time step:	22
Solving:	Stress period:	3	Time step:	23
Solving:	Stress period:	3	Time step:	24
Solving:	Stress period:	3	Time step:	25
Solving:	Stress period:	3	Time step:	26
Solving:	Stress period:	3	Time step:	27
Solving:	Stress period:	3	Time step:	28
Solving:	Stress period:	3	Time step:	29
Solving:	Stress period:	3	Time step:	30
Solving:	Stress period:	3	Time step:	31
Solving:	Stress period:	3	Time step:	32
Solving:	Stress period:	3	Time step:	33
Solving:	Stress period:	3	Time step:	34
Solving:	Stress period:	3	Time step:	35
Solving:	Stress period:	3	Time step:	36
Solving:	Stress period:	3	Time step:	37
Solving:	Stress period:	3	Time step:	38
Solving:	Stress period:	3	Time step:	39
Solving:	Stress period:	3	Time step:	40
Solving:	Stress period:	3	Time step:	41
Solving:	Stress period:	3	Time step:	42
Solving:	Stress period:	3	Time step:	43
Solving:	Stress period:	3	Time step:	44
Solving:	Stress period:	3	Time step:	45
Solving:	Stress period:	3	Time step:	46
Solving:	Stress period:	3	Time step:	47
Solving:	Stress period:	3	Time step:	48
Solving:	Stress period:	3	Time step:	49
Solving:	Stress period:	3	Time step:	50
Solving:	Stress period:	3	Time step:	51
Solving:	Stress period:	3	Time step:	52
Solving:	Stress period:	3	Time step:	53
Solving:	Stress period:	3	Time step:	54
Solving:	Stress period:	3	Time step:	55
Solving:	Stress period:	3	Time step:	56
Solving:	Stress period:	3	Time step:	57
Solving:	Stress period:	3	Time step:	58
Solving:	Stress period:	3	Time step:	59
Solving:	Stress period:	3	Time step:	60
Solving:	Stress period:	3	Time step:	61
Solving:	Stress period:	3	Time step:	62
Solving:	Stress period:	3	Time step:	63
Solving:	Stress period:	3	Time step:	64
Solving:	Stress period:	3	Time step:	65
Solving:	Stress period:	3	Time step:	66
Solving:	Stress period:	3	Time step:	67
Solving:	Stress period:	3	Time step:	68
Solving:	Stress period:	3	Time step:	69

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	70
Solving:	Stress period:	3	Time step:	71
Solving:	Stress period:	3	Time step:	72
Solving:	Stress period:	3	Time step:	73
Solving:	Stress period:	3	Time step:	74
Solving:	Stress period:	3	Time step:	75
Solving:	Stress period:	3	Time step:	76
Solving:	Stress period:	3	Time step:	77
Solving:	Stress period:	3	Time step:	78
Solving:	Stress period:	3	Time step:	79
Solving:	Stress period:	3	Time step:	80
Solving:	Stress period:	3	Time step:	81
Solving:	Stress period:	3	Time step:	82
Solving:	Stress period:	3	Time step:	83
Solving:	Stress period:	3	Time step:	84
Solving:	Stress period:	3	Time step:	85
Solving:	Stress period:	3	Time step:	86
Solving:	Stress period:	3	Time step:	87
Solving:	Stress period:	3	Time step:	88
Solving:	Stress period:	3	Time step:	89
Solving:	Stress period:	3	Time step:	90
Solving:	Stress period:	3	Time step:	91
Solving:	Stress period:	3	Time step:	92
Solving:	Stress period:	3	Time step:	93
Solving:	Stress period:	3	Time step:	94
Solving:	Stress period:	3	Time step:	95
Solving:	Stress period:	3	Time step:	96
Solving:	Stress period:	3	Time step:	97
Solving:	Stress period:	3	Time step:	98
Solving:	Stress period:	3	Time step:	99
Solving:	Stress period:	3	Time step:	100
Solving:	Stress period:	3	Time step:	101
Solving:	Stress period:	3	Time step:	102
Solving:	Stress period:	3	Time step:	103
Solving:	Stress period:	3	Time step:	104
Solving:	Stress period:	3	Time step:	105
Solving:	Stress period:	3	Time step:	106
Solving:	Stress period:	3	Time step:	107
Solving:	Stress period:	3	Time step:	108
Solving:	Stress period:	3	Time step:	109
Solving:	Stress period:	3	Time step:	110
Solving:	Stress period:	3	Time step:	111
Solving:	Stress period:	3	Time step:	112
Solving:	Stress period:	3	Time step:	113
Solving:	Stress period:	3	Time step:	114
Solving:	Stress period:	3	Time step:	115
Solving:	Stress period:	3	Time step:	116
Solving:	Stress period:	3	Time step:	117
Solving:	Stress period:	3	Time step:	118
Solving:	Stress period:	3	Time step:	119
Solving:	Stress period:	3	Time step:	120
Solving:	Stress period:	4	Time step:	1

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	2
Solving:	Stress period:	4	Time step:	3
Solving:	Stress period:	4	Time step:	4
Solving:	Stress period:	4	Time step:	5
Solving:	Stress period:	4	Time step:	6
Solving:	Stress period:	4	Time step:	7
Solving:	Stress period:	4	Time step:	8
Solving:	Stress period:	4	Time step:	9
Solving:	Stress period:	4	Time step:	10
Solving:	Stress period:	4	Time step:	11
Solving:	Stress period:	4	Time step:	12
Solving:	Stress period:	4	Time step:	13
Solving:	Stress period:	4	Time step:	14
Solving:	Stress period:	4	Time step:	15
Solving:	Stress period:	4	Time step:	16
Solving:	Stress period:	4	Time step:	17
Solving:	Stress period:	4	Time step:	18
Solving:	Stress period:	4	Time step:	19
Solving:	Stress period:	4	Time step:	20
Solving:	Stress period:	4	Time step:	21
Solving:	Stress period:	4	Time step:	22
Solving:	Stress period:	4	Time step:	23
Solving:	Stress period:	4	Time step:	24
Solving:	Stress period:	4	Time step:	25
Solving:	Stress period:	4	Time step:	26
Solving:	Stress period:	4	Time step:	27
Solving:	Stress period:	4	Time step:	28
Solving:	Stress period:	4	Time step:	29
Solving:	Stress period:	4	Time step:	30
Solving:	Stress period:	4	Time step:	31
Solving:	Stress period:	4	Time step:	32
Solving:	Stress period:	4	Time step:	33
Solving:	Stress period:	4	Time step:	34
Solving:	Stress period:	4	Time step:	35
Solving:	Stress period:	4	Time step:	36
Solving:	Stress period:	4	Time step:	37
Solving:	Stress period:	4	Time step:	38
Solving:	Stress period:	4	Time step:	39
Solving:	Stress period:	4	Time step:	40
Solving:	Stress period:	4	Time step:	41
Solving:	Stress period:	4	Time step:	42
Solving:	Stress period:	4	Time step:	43
Solving:	Stress period:	4	Time step:	44
Solving:	Stress period:	4	Time step:	45
Solving:	Stress period:	4	Time step:	46
Solving:	Stress period:	4	Time step:	47
Solving:	Stress period:	4	Time step:	48
Solving:	Stress period:	4	Time step:	49
Solving:	Stress period:	4	Time step:	50
Solving:	Stress period:	4	Time step:	51
Solving:	Stress period:	4	Time step:	52
Solving:	Stress period:	4	Time step:	53

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	54
Solving:	Stress period:	4	Time step:	55
Solving:	Stress period:	4	Time step:	56
Solving:	Stress period:	4	Time step:	57
Solving:	Stress period:	4	Time step:	58
Solving:	Stress period:	4	Time step:	59
Solving:	Stress period:	4	Time step:	60
Solving:	Stress period:	4	Time step:	61
Solving:	Stress period:	4	Time step:	62
Solving:	Stress period:	4	Time step:	63
Solving:	Stress period:	4	Time step:	64
Solving:	Stress period:	4	Time step:	65
Solving:	Stress period:	4	Time step:	66
Solving:	Stress period:	4	Time step:	67
Solving:	Stress period:	4	Time step:	68
Solving:	Stress period:	4	Time step:	69
Solving:	Stress period:	4	Time step:	70
Solving:	Stress period:	4	Time step:	71
Solving:	Stress period:	4	Time step:	72
Solving:	Stress period:	4	Time step:	73
Solving:	Stress period:	4	Time step:	74
Solving:	Stress period:	4	Time step:	75
Solving:	Stress period:	4	Time step:	76
Solving:	Stress period:	4	Time step:	77
Solving:	Stress period:	4	Time step:	78
Solving:	Stress period:	4	Time step:	79
Solving:	Stress period:	4	Time step:	80
Solving:	Stress period:	4	Time step:	81
Solving:	Stress period:	4	Time step:	82
Solving:	Stress period:	4	Time step:	83
Solving:	Stress period:	4	Time step:	84
Solving:	Stress period:	4	Time step:	85
Solving:	Stress period:	4	Time step:	86
Solving:	Stress period:	4	Time step:	87
Solving:	Stress period:	4	Time step:	88
Solving:	Stress period:	4	Time step:	89
Solving:	Stress period:	4	Time step:	90
Solving:	Stress period:	4	Time step:	91
Solving:	Stress period:	4	Time step:	92
Solving:	Stress period:	4	Time step:	93
Solving:	Stress period:	4	Time step:	94
Solving:	Stress period:	4	Time step:	95
Solving:	Stress period:	4	Time step:	96
Solving:	Stress period:	4	Time step:	97
Solving:	Stress period:	4	Time step:	98
Solving:	Stress period:	4	Time step:	99
Solving:	Stress period:	4	Time step:	100
Solving:	Stress period:	4	Time step:	101
Solving:	Stress period:	4	Time step:	102
Solving:	Stress period:	4	Time step:	103
Solving:	Stress period:	4	Time step:	104
Solving:	Stress period:	4	Time step:	105

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 4 Time step: 106
Solving: Stress period: 4 Time step: 107
Solving: Stress period: 4 Time step: 108
Solving: Stress period: 4 Time step: 109
Solving: Stress period: 4 Time step: 110
Solving: Stress period: 4 Time step: 111
Solving: Stress period: 4 Time step: 112
Solving: Stress period: 4 Time step: 113
Solving: Stress period: 4 Time step: 114
Solving: Stress period: 4 Time step: 115
Solving: Stress period: 4 Time step: 116
Solving: Stress period: 4 Time step: 117
Solving: Stress period: 4 Time step: 118
Solving: Stress period: 4 Time step: 119
Solving: Stress period: 4 Time step: 120

```

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:39

Elapsed run time: 0.599 Seconds

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
 2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.
- Normal termination of simulation.

Method 3: Pass timearrayseries a dictionary of anything that could be passed to rcha.tas.initialize

```

[12]: rch_array = 0.0000001 * np.ones((15, 10))
rch_array[0, 0] = 0.0001
tas = {
    0.0: 0.000002,
    200.0: rch_array,
    "filename": "method3.tas",
    "time_series_namerecord": "rcharray_1",
    "interpolation_methodrecord": "LINEAR",
}
rcha = flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha(
    model, timearrayseries=tas, recharge="TIMEARRAYSERIES rcharray_1"
)

sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:

```

(continues on next page)

(continued from previous page)

```

raise ValueError("Failed to run.")

# clean up for next example
model.remove_package("rcha")

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims_-1...
  writing model child_pkgs...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
    writing package oc...
    writing package sto...
    writing package ghb...
    writing package ts_0...
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
    writing package ts_1...
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
    writing package ts_2...
WARNING: Unable to resolve dimension of ('ts', 'attributes', 'sfacrecord_single',
↳ 'sfacval') based on shape "time_series_name".
    writing package rcha_0...
    writing package tas_0...

MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

This software has been approved for release by the U.S. Geological
Survey (USGS). Although the software has been subjected to rigorous
review, the USGS reserves the right to update the software as needed
pursuant to further analysis and review. No warranty, expressed or
implied, is made by the USGS or the U.S. Government as to the
functionality of the software and related material nor shall the
fact of release constitute any such warranty. Furthermore, the
software is released on condition that neither the USGS nor the U.S.
Government shall be held liable for any damages resulting from its
authorized or unauthorized use. Also refer to the USGS Water
Resources Software User Rights Notice for complete use, copyright,
and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:39

```

(continues on next page)

(continued from previous page)

Writing simulation list file: mfsim.lst

Using Simulation name file: mfsim.nam

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5
Solving:	Stress period:	2	Time step:	6
Solving:	Stress period:	2	Time step:	7
Solving:	Stress period:	2	Time step:	8
Solving:	Stress period:	2	Time step:	9
Solving:	Stress period:	2	Time step:	10
Solving:	Stress period:	2	Time step:	11
Solving:	Stress period:	2	Time step:	12
Solving:	Stress period:	2	Time step:	13
Solving:	Stress period:	2	Time step:	14
Solving:	Stress period:	2	Time step:	15
Solving:	Stress period:	2	Time step:	16
Solving:	Stress period:	2	Time step:	17
Solving:	Stress period:	2	Time step:	18
Solving:	Stress period:	2	Time step:	19
Solving:	Stress period:	2	Time step:	20
Solving:	Stress period:	2	Time step:	21
Solving:	Stress period:	2	Time step:	22
Solving:	Stress period:	2	Time step:	23
Solving:	Stress period:	2	Time step:	24
Solving:	Stress period:	2	Time step:	25
Solving:	Stress period:	2	Time step:	26
Solving:	Stress period:	2	Time step:	27
Solving:	Stress period:	2	Time step:	28
Solving:	Stress period:	2	Time step:	29
Solving:	Stress period:	2	Time step:	30
Solving:	Stress period:	2	Time step:	31
Solving:	Stress period:	2	Time step:	32
Solving:	Stress period:	2	Time step:	33
Solving:	Stress period:	2	Time step:	34
Solving:	Stress period:	2	Time step:	35
Solving:	Stress period:	2	Time step:	36
Solving:	Stress period:	2	Time step:	37
Solving:	Stress period:	2	Time step:	38
Solving:	Stress period:	2	Time step:	39
Solving:	Stress period:	2	Time step:	40
Solving:	Stress period:	2	Time step:	41
Solving:	Stress period:	2	Time step:	42
Solving:	Stress period:	2	Time step:	43
Solving:	Stress period:	2	Time step:	44
Solving:	Stress period:	2	Time step:	45
Solving:	Stress period:	2	Time step:	46
Solving:	Stress period:	2	Time step:	47
Solving:	Stress period:	2	Time step:	48

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	49
Solving:	Stress period:	2	Time step:	50
Solving:	Stress period:	2	Time step:	51
Solving:	Stress period:	2	Time step:	52
Solving:	Stress period:	2	Time step:	53
Solving:	Stress period:	2	Time step:	54
Solving:	Stress period:	2	Time step:	55
Solving:	Stress period:	2	Time step:	56
Solving:	Stress period:	2	Time step:	57
Solving:	Stress period:	2	Time step:	58
Solving:	Stress period:	2	Time step:	59
Solving:	Stress period:	2	Time step:	60
Solving:	Stress period:	2	Time step:	61
Solving:	Stress period:	2	Time step:	62
Solving:	Stress period:	2	Time step:	63
Solving:	Stress period:	2	Time step:	64
Solving:	Stress period:	2	Time step:	65
Solving:	Stress period:	2	Time step:	66
Solving:	Stress period:	2	Time step:	67
Solving:	Stress period:	2	Time step:	68
Solving:	Stress period:	2	Time step:	69
Solving:	Stress period:	2	Time step:	70
Solving:	Stress period:	2	Time step:	71
Solving:	Stress period:	2	Time step:	72
Solving:	Stress period:	2	Time step:	73
Solving:	Stress period:	2	Time step:	74
Solving:	Stress period:	2	Time step:	75
Solving:	Stress period:	2	Time step:	76
Solving:	Stress period:	2	Time step:	77
Solving:	Stress period:	2	Time step:	78
Solving:	Stress period:	2	Time step:	79
Solving:	Stress period:	2	Time step:	80
Solving:	Stress period:	2	Time step:	81
Solving:	Stress period:	2	Time step:	82
Solving:	Stress period:	2	Time step:	83
Solving:	Stress period:	2	Time step:	84
Solving:	Stress period:	2	Time step:	85
Solving:	Stress period:	2	Time step:	86
Solving:	Stress period:	2	Time step:	87
Solving:	Stress period:	2	Time step:	88
Solving:	Stress period:	2	Time step:	89
Solving:	Stress period:	2	Time step:	90
Solving:	Stress period:	2	Time step:	91
Solving:	Stress period:	2	Time step:	92
Solving:	Stress period:	2	Time step:	93
Solving:	Stress period:	2	Time step:	94
Solving:	Stress period:	2	Time step:	95
Solving:	Stress period:	2	Time step:	96
Solving:	Stress period:	2	Time step:	97
Solving:	Stress period:	2	Time step:	98
Solving:	Stress period:	2	Time step:	99
Solving:	Stress period:	2	Time step:	100

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	101
Solving:	Stress period:	2	Time step:	102
Solving:	Stress period:	2	Time step:	103
Solving:	Stress period:	2	Time step:	104
Solving:	Stress period:	2	Time step:	105
Solving:	Stress period:	2	Time step:	106
Solving:	Stress period:	2	Time step:	107
Solving:	Stress period:	2	Time step:	108
Solving:	Stress period:	2	Time step:	109
Solving:	Stress period:	2	Time step:	110
Solving:	Stress period:	2	Time step:	111
Solving:	Stress period:	2	Time step:	112
Solving:	Stress period:	2	Time step:	113
Solving:	Stress period:	2	Time step:	114
Solving:	Stress period:	2	Time step:	115
Solving:	Stress period:	2	Time step:	116
Solving:	Stress period:	2	Time step:	117
Solving:	Stress period:	2	Time step:	118
Solving:	Stress period:	2	Time step:	119
Solving:	Stress period:	2	Time step:	120
Solving:	Stress period:	3	Time step:	1
Solving:	Stress period:	3	Time step:	2
Solving:	Stress period:	3	Time step:	3
Solving:	Stress period:	3	Time step:	4
Solving:	Stress period:	3	Time step:	5
Solving:	Stress period:	3	Time step:	6
Solving:	Stress period:	3	Time step:	7
Solving:	Stress period:	3	Time step:	8
Solving:	Stress period:	3	Time step:	9
Solving:	Stress period:	3	Time step:	10
Solving:	Stress period:	3	Time step:	11
Solving:	Stress period:	3	Time step:	12
Solving:	Stress period:	3	Time step:	13
Solving:	Stress period:	3	Time step:	14
Solving:	Stress period:	3	Time step:	15
Solving:	Stress period:	3	Time step:	16
Solving:	Stress period:	3	Time step:	17
Solving:	Stress period:	3	Time step:	18
Solving:	Stress period:	3	Time step:	19
Solving:	Stress period:	3	Time step:	20
Solving:	Stress period:	3	Time step:	21
Solving:	Stress period:	3	Time step:	22
Solving:	Stress period:	3	Time step:	23
Solving:	Stress period:	3	Time step:	24
Solving:	Stress period:	3	Time step:	25
Solving:	Stress period:	3	Time step:	26
Solving:	Stress period:	3	Time step:	27
Solving:	Stress period:	3	Time step:	28
Solving:	Stress period:	3	Time step:	29
Solving:	Stress period:	3	Time step:	30
Solving:	Stress period:	3	Time step:	31
Solving:	Stress period:	3	Time step:	32

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	33
Solving:	Stress period:	3	Time step:	34
Solving:	Stress period:	3	Time step:	35
Solving:	Stress period:	3	Time step:	36
Solving:	Stress period:	3	Time step:	37
Solving:	Stress period:	3	Time step:	38
Solving:	Stress period:	3	Time step:	39
Solving:	Stress period:	3	Time step:	40
Solving:	Stress period:	3	Time step:	41
Solving:	Stress period:	3	Time step:	42
Solving:	Stress period:	3	Time step:	43
Solving:	Stress period:	3	Time step:	44
Solving:	Stress period:	3	Time step:	45
Solving:	Stress period:	3	Time step:	46
Solving:	Stress period:	3	Time step:	47
Solving:	Stress period:	3	Time step:	48
Solving:	Stress period:	3	Time step:	49
Solving:	Stress period:	3	Time step:	50
Solving:	Stress period:	3	Time step:	51
Solving:	Stress period:	3	Time step:	52
Solving:	Stress period:	3	Time step:	53
Solving:	Stress period:	3	Time step:	54
Solving:	Stress period:	3	Time step:	55
Solving:	Stress period:	3	Time step:	56
Solving:	Stress period:	3	Time step:	57
Solving:	Stress period:	3	Time step:	58
Solving:	Stress period:	3	Time step:	59
Solving:	Stress period:	3	Time step:	60
Solving:	Stress period:	3	Time step:	61
Solving:	Stress period:	3	Time step:	62
Solving:	Stress period:	3	Time step:	63
Solving:	Stress period:	3	Time step:	64
Solving:	Stress period:	3	Time step:	65
Solving:	Stress period:	3	Time step:	66
Solving:	Stress period:	3	Time step:	67
Solving:	Stress period:	3	Time step:	68
Solving:	Stress period:	3	Time step:	69
Solving:	Stress period:	3	Time step:	70
Solving:	Stress period:	3	Time step:	71
Solving:	Stress period:	3	Time step:	72
Solving:	Stress period:	3	Time step:	73
Solving:	Stress period:	3	Time step:	74
Solving:	Stress period:	3	Time step:	75
Solving:	Stress period:	3	Time step:	76
Solving:	Stress period:	3	Time step:	77
Solving:	Stress period:	3	Time step:	78
Solving:	Stress period:	3	Time step:	79
Solving:	Stress period:	3	Time step:	80
Solving:	Stress period:	3	Time step:	81
Solving:	Stress period:	3	Time step:	82
Solving:	Stress period:	3	Time step:	83
Solving:	Stress period:	3	Time step:	84

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	85
Solving:	Stress period:	3	Time step:	86
Solving:	Stress period:	3	Time step:	87
Solving:	Stress period:	3	Time step:	88
Solving:	Stress period:	3	Time step:	89
Solving:	Stress period:	3	Time step:	90
Solving:	Stress period:	3	Time step:	91
Solving:	Stress period:	3	Time step:	92
Solving:	Stress period:	3	Time step:	93
Solving:	Stress period:	3	Time step:	94
Solving:	Stress period:	3	Time step:	95
Solving:	Stress period:	3	Time step:	96
Solving:	Stress period:	3	Time step:	97
Solving:	Stress period:	3	Time step:	98
Solving:	Stress period:	3	Time step:	99
Solving:	Stress period:	3	Time step:	100
Solving:	Stress period:	3	Time step:	101
Solving:	Stress period:	3	Time step:	102
Solving:	Stress period:	3	Time step:	103
Solving:	Stress period:	3	Time step:	104
Solving:	Stress period:	3	Time step:	105
Solving:	Stress period:	3	Time step:	106
Solving:	Stress period:	3	Time step:	107
Solving:	Stress period:	3	Time step:	108
Solving:	Stress period:	3	Time step:	109
Solving:	Stress period:	3	Time step:	110
Solving:	Stress period:	3	Time step:	111
Solving:	Stress period:	3	Time step:	112
Solving:	Stress period:	3	Time step:	113
Solving:	Stress period:	3	Time step:	114
Solving:	Stress period:	3	Time step:	115
Solving:	Stress period:	3	Time step:	116
Solving:	Stress period:	3	Time step:	117
Solving:	Stress period:	3	Time step:	118
Solving:	Stress period:	3	Time step:	119
Solving:	Stress period:	3	Time step:	120
Solving:	Stress period:	4	Time step:	1
Solving:	Stress period:	4	Time step:	2
Solving:	Stress period:	4	Time step:	3
Solving:	Stress period:	4	Time step:	4
Solving:	Stress period:	4	Time step:	5
Solving:	Stress period:	4	Time step:	6
Solving:	Stress period:	4	Time step:	7
Solving:	Stress period:	4	Time step:	8
Solving:	Stress period:	4	Time step:	9
Solving:	Stress period:	4	Time step:	10
Solving:	Stress period:	4	Time step:	11
Solving:	Stress period:	4	Time step:	12
Solving:	Stress period:	4	Time step:	13
Solving:	Stress period:	4	Time step:	14
Solving:	Stress period:	4	Time step:	15
Solving:	Stress period:	4	Time step:	16

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	17
Solving:	Stress period:	4	Time step:	18
Solving:	Stress period:	4	Time step:	19
Solving:	Stress period:	4	Time step:	20
Solving:	Stress period:	4	Time step:	21
Solving:	Stress period:	4	Time step:	22
Solving:	Stress period:	4	Time step:	23
Solving:	Stress period:	4	Time step:	24
Solving:	Stress period:	4	Time step:	25
Solving:	Stress period:	4	Time step:	26
Solving:	Stress period:	4	Time step:	27
Solving:	Stress period:	4	Time step:	28
Solving:	Stress period:	4	Time step:	29
Solving:	Stress period:	4	Time step:	30
Solving:	Stress period:	4	Time step:	31
Solving:	Stress period:	4	Time step:	32
Solving:	Stress period:	4	Time step:	33
Solving:	Stress period:	4	Time step:	34
Solving:	Stress period:	4	Time step:	35
Solving:	Stress period:	4	Time step:	36
Solving:	Stress period:	4	Time step:	37
Solving:	Stress period:	4	Time step:	38
Solving:	Stress period:	4	Time step:	39
Solving:	Stress period:	4	Time step:	40
Solving:	Stress period:	4	Time step:	41
Solving:	Stress period:	4	Time step:	42
Solving:	Stress period:	4	Time step:	43
Solving:	Stress period:	4	Time step:	44
Solving:	Stress period:	4	Time step:	45
Solving:	Stress period:	4	Time step:	46
Solving:	Stress period:	4	Time step:	47
Solving:	Stress period:	4	Time step:	48
Solving:	Stress period:	4	Time step:	49
Solving:	Stress period:	4	Time step:	50
Solving:	Stress period:	4	Time step:	51
Solving:	Stress period:	4	Time step:	52
Solving:	Stress period:	4	Time step:	53
Solving:	Stress period:	4	Time step:	54
Solving:	Stress period:	4	Time step:	55
Solving:	Stress period:	4	Time step:	56
Solving:	Stress period:	4	Time step:	57
Solving:	Stress period:	4	Time step:	58
Solving:	Stress period:	4	Time step:	59
Solving:	Stress period:	4	Time step:	60
Solving:	Stress period:	4	Time step:	61
Solving:	Stress period:	4	Time step:	62
Solving:	Stress period:	4	Time step:	63
Solving:	Stress period:	4	Time step:	64
Solving:	Stress period:	4	Time step:	65
Solving:	Stress period:	4	Time step:	66
Solving:	Stress period:	4	Time step:	67
Solving:	Stress period:	4	Time step:	68

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	69
Solving:	Stress period:	4	Time step:	70
Solving:	Stress period:	4	Time step:	71
Solving:	Stress period:	4	Time step:	72
Solving:	Stress period:	4	Time step:	73
Solving:	Stress period:	4	Time step:	74
Solving:	Stress period:	4	Time step:	75
Solving:	Stress period:	4	Time step:	76
Solving:	Stress period:	4	Time step:	77
Solving:	Stress period:	4	Time step:	78
Solving:	Stress period:	4	Time step:	79
Solving:	Stress period:	4	Time step:	80
Solving:	Stress period:	4	Time step:	81
Solving:	Stress period:	4	Time step:	82
Solving:	Stress period:	4	Time step:	83
Solving:	Stress period:	4	Time step:	84
Solving:	Stress period:	4	Time step:	85
Solving:	Stress period:	4	Time step:	86
Solving:	Stress period:	4	Time step:	87
Solving:	Stress period:	4	Time step:	88
Solving:	Stress period:	4	Time step:	89
Solving:	Stress period:	4	Time step:	90
Solving:	Stress period:	4	Time step:	91
Solving:	Stress period:	4	Time step:	92
Solving:	Stress period:	4	Time step:	93
Solving:	Stress period:	4	Time step:	94
Solving:	Stress period:	4	Time step:	95
Solving:	Stress period:	4	Time step:	96
Solving:	Stress period:	4	Time step:	97
Solving:	Stress period:	4	Time step:	98
Solving:	Stress period:	4	Time step:	99
Solving:	Stress period:	4	Time step:	100
Solving:	Stress period:	4	Time step:	101
Solving:	Stress period:	4	Time step:	102
Solving:	Stress period:	4	Time step:	103
Solving:	Stress period:	4	Time step:	104
Solving:	Stress period:	4	Time step:	105
Solving:	Stress period:	4	Time step:	106
Solving:	Stress period:	4	Time step:	107
Solving:	Stress period:	4	Time step:	108
Solving:	Stress period:	4	Time step:	109
Solving:	Stress period:	4	Time step:	110
Solving:	Stress period:	4	Time step:	111
Solving:	Stress period:	4	Time step:	112
Solving:	Stress period:	4	Time step:	113
Solving:	Stress period:	4	Time step:	114
Solving:	Stress period:	4	Time step:	115
Solving:	Stress period:	4	Time step:	116
Solving:	Stress period:	4	Time step:	117
Solving:	Stress period:	4	Time step:	118
Solving:	Stress period:	4	Time step:	119
Solving:	Stress period:	4	Time step:	120

(continues on next page)

(continued from previous page)

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:40
 Elapsed run time: 0.604 Seconds

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
 2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'child_pkgs_test.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.
- Normal termination of simulation.

5.2.11 Working with the Multi-node Well (MNW2) Package

```
[1]: import os

[2]: import sys
from tempfile import TemporaryDirectory

import numpy as np

try:
    import pandas as pd
except:
    pass

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
try:
    print("pandas version: {}".format(pd.__version__))
except:
    pass
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
pandas version: 2.0.3
flopy version: 3.4.3
```


Make an MNW2 package from scratch

```
[3]: # temporary directory
temp_dir = TemporaryDirectory()
model_ws = temp_dir.name

m = flopy.modflow.Modflow("mnw2example", model_ws=model_ws)
dis = flopy.modflow.ModflowDis(
    nrow=5, ncol=5, nlay=3, nper=3, top=10, botm=0, model=m
)
```

MNW2 information by node

(this could be prepared externally from well records and read in from a csv or excel file) * this table has two multi-node wells, the first (well1) consisting of two nodes that are manually specified (where the variable **rw** is specified by node)
 * node that some variables that are constant for the whole well are also included (losstype, zpump, etc.)

```
[4]: node_data = pd.DataFrame(
    [
        [1, 1, 9.5, 7.1, "well1", "skin", -1, 0, 0, 0, 1.0, 2.0, 5.0, 6.2],
        [1, 1, 7.1, 5.1, "well1", "skin", -1, 0, 0, 0, 0.5, 2.0, 5.0, 6.2],
        [3, 3, 9.1, 3.7, "well2", "skin", -1, 0, 0, 0, 1.0, 2.0, 5.0, 4.1],
    ],
    columns=[
        "i",
        "j",
        "ztop",
        "zbotm",
        "wellid",
        "losstype",
        "pumploc",
        "qlimit",
        "ppflag",
        "pumpcap",
        "rw",
        "rskin",
        "kskin",
        "zpump",
    ],
)
node_data
```

```
[4]:
```

	i	j	ztop	zbotm	wellid	losstype	pumploc	qlimit	ppflag	pumpcap	rw	\
0	1	1	9.5	7.1	well1	skin	-1	0	0	0	1.0	
1	1	1	7.1	5.1	well1	skin	-1	0	0	0	0.5	
2	3	3	9.1	3.7	well2	skin	-1	0	0	0	1.0	
	rskin		kskin	zpump								
0	2.0		5.0	6.2								
1	2.0		5.0	6.2								
2	2.0		5.0	4.1								

convert the DataFrame to a recarray for compatibility with flopy

```
[5]: node_data = node_data.to_records()
node_data

[5]: rec.array([(0, 1, 1, 9.5, 7.1, 'well1', 'skin', -1, 0, 0, 0, 1., 2., 5., 6.2),
               (1, 1, 1, 7.1, 5.1, 'well1', 'skin', -1, 0, 0, 0, 0.5, 2., 5., 6.2),
               (2, 3, 3, 9.1, 3.7, 'well2', 'skin', -1, 0, 0, 0, 1., 2., 5., 4.1)],
              dtype=[('index', '<i8'), ('i', '<i8'), ('j', '<i8'), ('ztop', '<f8'), ('zbotm',
→ '<f8'), ('wellid', 'O'), ('losstype', 'O'), ('pumploc', '<i8'), ('qlimit', '<i8'), (
→ 'ppflag', '<i8'), ('pumpcap', '<i8'), ('rw', '<f8'), ('rskin', '<f8'), ('kskin', '<f8
→ '), ('zpump', '<f8')])
```

Stress period information

(could also be developed externally)

```
[6]: stress_period_data = pd.DataFrame(
    [
        [0, "well1", 0],
        [1, "well1", 100.0],
        [0, "well2", 0],
        [1, "well2", 1000.0],
    ],
    columns=["per", "wellid", "qdes"],
)
stress_period_data
```

```
[6]:   per wellid  qdes
0    0  well1    0.0
1    1  well1  100.0
2    0  well2    0.0
3    1  well2 1000.0
```

```
[7]: pers = stress_period_data.groupby("per")
stress_period_data = {i: pers.get_group(i).to_records() for i in [0, 1]}
stress_period_data
```

```
[7]: {0: rec.array([(0, 0, 'well1', 0.), (2, 0, 'well2', 0.)],
                  dtype=[('index', '<i8'), ('per', '<i8'), ('wellid', 'O'), ('qdes', '<f8')]),
      1: rec.array([(1, 1, 'well1', 100.), (3, 1, 'well2', 1000.)],
                  dtype=[('index', '<i8'), ('per', '<i8'), ('wellid', 'O'), ('qdes', '<f8')])}
```

Make ModflowMnw2 package object

- note that extraneous columns in `node_data` and `stress_period_data` are ignored
- if `itmp` is positive, it must equal the number of active wells being specified in `stress_period_data`, otherwise the package class will raise an error.

```
[8]: mnw2 = flopy.modflow.ModflowMnw2(
    model=m,
    mnwmax=2,
    node_data=node_data,
    stress_period_data=stress_period_data,
    itmp=[2, 2, -1], # reuse second per pumping for last stress period
)
```

```
[9]: # "nodtot" is computed automatically
mnw2.nodtot
```

```
[9]: 3
```

```
[10]: pd.DataFrame(mnw2.node_data)
```

```
[10]:
```

	k	i	j	ztop	zbotm	wellid	losstype	pumploc	qlimit	ppflag	...	hlim	\
0	0	1	1	9.5	7.1	well1	skin	-1	0	0	...	0.0	
1	0	1	1	7.1	5.1	well1	skin	-1	0	0	...	0.0	
2	0	3	3	9.1	3.7	well2	skin	-1	0	0	...	0.0	

	qcut	qfrcmn	qfrcmx	hlift	liftq0	liftqmax	hwtol	liftn	qn
0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

[3 rows x 33 columns]

```
[11]: pd.DataFrame(mnw2.stress_period_data[0])
```

```
[11]:
```

	k	i	j	wellid	qdes	capmult	cprime	hlim	qcut	qfrcmn	qfrcmx
0	0	1	1	well1	0.0	0	0.0	0.0	0	0.0	0.0
1	0	3	3	well2	0.0	0	0.0	0.0	0	0.0	0.0

```
[12]: pd.DataFrame(mnw2.stress_period_data[1])
```

```
[12]:
```

	k	i	j	wellid	qdes	capmult	cprime	hlim	qcut	qfrcmn	qfrcmx
0	0	1	1	well1	100.0	0	0.0	0.0	0	0.0	0.0
1	0	3	3	well2	1000.0	0	0.0	0.0	0	0.0	0.0

```
[13]: tmp = flopy.modflow.ModflowMnw2(
    model=m,
    itmp=[1, 1, -1], # reuse second per pumping for last stress period
)
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/mbase.py:652:
↳ UserWarning: Unit 34 of package MNW2 already in use.
    warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/mbase.py:661:
```

(continues on next page)

(continued from previous page)

```
↳UserWarning: Two packages of the same type, Replacing existing 'MNW2' package.
warn(
```

empty node_data and stress_period_data tables can also be generated by the package class, and then filled

```
[14]: node_data = tmp.get_empty_node_data(3)
node_data

[14]: rec.array([(0, 0, 0, 0., 0., 0, 0, 0, 0, 0, 0, 0, 0., 0., 0., 0., 0., 0., 0., 0., 0, 0, 0, 0.,
↳0., 0., 0, 0., 0., 0., 0., 0., 0., 0.),
           (0, 0, 0, 0., 0., 0, 0, 0, 0, 0, 0, 0, 0., 0., 0., 0., 0., 0., 0., 0., 0, 0, 0, 0.,
↳0., 0., 0, 0., 0., 0., 0., 0., 0., 0.),
           (0, 0, 0, 0., 0., 0, 0, 0, 0, 0, 0, 0, 0., 0., 0., 0., 0., 0., 0., 0., 0, 0, 0, 0.,
↳0., 0., 0, 0., 0., 0., 0., 0., 0., 0.)],
          dtype=[('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('ztop', '<f4'), ('zbotm', '
↳<f4'), ('wellid', 'O'), ('losstype', 'O'), ('pumploc', '<i8'), ('qlimit', '<i8'), (
↳'ppflag', '<i8'), ('pumpcap', '<i8'), ('rw', '<f4'), ('rskin', '<f4'), ('kskin', '<f4
↳'), ('B', '<f4'), ('C', '<f4'), ('P', '<f4'), ('cwc', '<f4'), ('pp', '<f4'), ('pumplay
↳', '<i8'), ('pumprow', '<i8'), ('pumpcol', '<i8'), ('zpump', '<f4'), ('hlim', '<f4'), (
↳'qcut', '<i8'), ('qfrcmn', '<f4'), ('qfrcmx', '<f4'), ('hlift', '<f4'), ('liftq0', '<f4
↳'), ('liftqmax', '<f4'), ('hwtol', '<f4'), ('liftn', '<f4'), ('qn', '<f4')])]
```

Mnw objects

at the base of the flopy.mnw2 module is the **Mnw** object class, which describes a single multi-node well. A list or dict of **Mnw** objects can be used to build a package (using the example above):

```
flopy.modflow.ModflowMnw2(model=m, mnwmax=2,
                          mnw=<dict or list of Mnw objects>,
                          itmp=[1, 1, -1], # reuse second per pumping for last stress period
                          )
```

or if node_data and stress_period_data are supplied, the **Mnw** objects are created on initialization of the ModflowMnw2 class instance, and assigned to the .mnw attribute, as items in a dictionary keyed by wellid.

```
[15]: mnw2.mnw

[15]: {'well1': <flopy.modflow.mfmnw2.Mnw at 0x7f375144abb0>,
      'well2': <flopy.modflow.mfmnw2.Mnw at 0x7f375144ad30>}

[16]: mnw2.mnw["well1"].rw

[16]: [1.0, 0.5]
```

Note that Mnw object attributes for variables that vary by node are lists (e.g. rw above)

Each Mnw object has its own node_data and stress_period_data

```
[17]: pd.DataFrame(mnw2.mnw["well1"].node_data)
```

```
[17]:      k  i  j  ztop  zbotm wellid losstype  pumploc  qlimit  ppflag  ...  hlim  \
0  0  1  1   9.5   7.1  well1    skin      -1         0         0  ...   0.0
1  0  1  1   7.1   5.1  well1    skin      -1         0         0  ...   0.0

      qcut  qfrcmn  qfrcmx  hlift  liftq0  liftqmax  hwtol  liftn  qn
0         0         0.0     0.0     0.0     0.0     0.0     0.0     0.0  0.0
1         0         0.0     0.0     0.0     0.0     0.0     0.0     0.0  0.0

[2 rows x 33 columns]
```

Instead of a dict keyed by stress period, `Mnw.stress_period_data` is a recarray with pumping data listed by stress period for that well

- note that data for period 2, where `itmp < 1`, is shown (was copied from s.p. 1 during construction of the **Mnw** object)

```
[18]: pd.DataFrame(mnw2.mnw["well2"].stress_period_data)
```

```
[18]:      k  i  j  per    qdes  capmult  cprime  hlim  qcut  qfrcmn  qfrcmx
0  0  3  3    0     0.0         0     0.0  0.0    0     0.0    0.0
1  0  3  3    1  1000.0         0     0.0  0.0    0     0.0    0.0
2  0  3  3    1  1000.0         0     0.0  0.0    0     0.0    0.0
```

Build the same package using only the Mnw objects

```
[19]: mnw2fromobj = flopy.modflow.ModflowMnw2(
    model=m,
    mnwmax=2,
    mnw=mnw2.mnw,
    itmp=[2, 2, -1], # reuse second per pumping for last stress period
)
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/mbase.py:652:
↳UserWarning: Unit 34 of package MNW2 already in use.
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/mbase.py:661:
↳UserWarning: Two packages of the same type, Replacing existing 'MNW2' package.
warn(
```

```
[20]: pd.DataFrame(mnw2fromobj.node_data)
```

```
[20]:      k  i  j  ztop  zbotm wellid losstype  pumploc  qlimit  ppflag  ...  hlim  \
0  0  1  1   9.5   7.1  well1    skin      -1         0         0  ...   0.0
1  0  1  1   7.1   5.1  well1    skin      -1         0         0  ...   0.0
2  0  3  3   9.1   3.7  well2    skin      -1         0         0  ...   0.0

      qcut  qfrcmn  qfrcmx  hlift  liftq0  liftqmax  hwtol  liftn  qn
0         0         0.0     0.0     0.0     0.0     0.0     0.0     0.0  0.0
1         0         0.0     0.0     0.0     0.0     0.0     0.0     0.0  0.0
2         0         0.0     0.0     0.0     0.0     0.0     0.0     0.0  0.0
```

(continues on next page)

(continued from previous page)

```
0    0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1    0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
2    0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
```

[3 rows x 33 columns]

```
[21]: pd.DataFrame(mnw2fromobj.stress_period_data[0])
```

```
[21]:   k  i  j wellid  qdes  capmult  cprime  hlim  qcut  qfrcmn  qfrcmx
0  0  1  1  well1   0.0         0    0.0   0.0    0    0.0    0.0
1  0  3  3  well2   0.0         0    0.0   0.0    0    0.0    0.0
```

```
[22]: pd.DataFrame(mnw2fromobj.stress_period_data[1])
```

```
[22]:   k  i  j wellid  qdes  capmult  cprime  hlim  qcut  qfrcmn  qfrcmx
0  0  1  1  well1  100.0         0    0.0   0.0    0    0.0    0.0
1  0  3  3  well2 1000.0         0    0.0   0.0    0    0.0    0.0
```

By default, the `node_data` and `stress_period_data` tables attached to the `ModflowMnw2` package class are definitive

- on writing of the package output (`mnw2.write_file()`), the **Mnw** objects are regenerated from the tables. This setting is controlled by the default argument `use_tables=True`. To write the package file using the **Mnw** objects (ignoring the tables), use `mnw2.write_file(use_tables=False)`.

```
[23]: perl = flopy.modflow.ModflowMnw2.get_empty_stress_period_data(itmp=2)
perl
```

```
[23]: rec.array([(0, 0, 0, 0, 0., 0, 0., 0., 0, 0., 0.),
                (0, 0, 0, 0, 0., 0, 0., 0., 0, 0., 0.)],
              dtype=[('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('wellid', '0'), ('qdes', '<f4'),
                    ('capmult', '<i8'), ('cprime', '<f4'), ('hlim', '<f4'), ('qcut', '<i8'), ('qfrcmn',
                    '<f4'), ('qfrcmx', '<f4')])
```

Write an MNW2 package file and inspect the results

```
[24]: mnw2.write_file(os.path.join(model_ws, "test.mnw2"))
```

```
[25]: junk = [
    print(l.strip("\n"))
    for l in open(os.path.join(model_ws, "test.mnw2")).readlines()
]
```

```
# MNW2 package for MODFLOW-2005 generated by Flopy 3.4.3
2 0 0
well1 -2
      skin -1 0 0 0
          -1.0000000E+00    2.0000000E+00    5.0000000E+00
          9.5000000E+00    7.0999999E+00 2 2    1.0000000E+00
          7.0999999E+00    5.0999999E+00 2 2    5.0000000E-01
```

(continues on next page)

(continued from previous page)

```

        6.1999998E+00
well2 -1
        skin -1 0 0 0
            1.0000000E+00    2.0000000E+00    5.0000000E+00
            9.1000004E+00    3.7000000E+00 4 4
            4.0999999E+00
2 Stress Period 1
well1    0.0000000E+00
well2    0.0000000E+00
2 Stress Period 2
well1    1.0000000E+02
well2    1.0000000E+03
-1 Stress Period 3

```

Load some example MNW2 packages

```
[26]: path = os.path.join(".", "..", "examples", "data", "mnw2_examples")
      m = flopy.modflow.Modflow("MNW2-Fig28", model_ws=model_ws)
      dis = flopy.modflow.ModflowDis.load(os.path.join(path, "MNW2-Fig28.dis"), m)
```

```
[27]: m.get_package_list()
```

```
[27]: ['DIS']
```

```
[28]: mnw2pth = os.path.join(path, "MNW2-Fig28.mnw2")
      mnw2 = flopy.modflow.ModflowMnw2.load(mnw2pth, m)
```

```
[29]: pd.DataFrame(mnw2.node_data)
```

```
[29]:
```

	k	i	j	ztop	zbotm	wellid	losstype	pumploc	qlimit	ppflag	...	\
0	0	29	40	-5.0	-65.0	well-a	skin	0	1	0

	hlim	qcut	qfrcmn	qfrcmx	hlift	liftq0	liftqmax	hwtol	liftn	qn
0	-7.5	-1	0.1	0.15	0.0	0.0	0.0	0.0	0.0	0.0

[1 rows x 33 columns]

```
[30]: pd.DataFrame(mnw2.stress_period_data[0])
```

```
[30]:
```

	k	i	j	wellid	qdes	capmult	cprime	hlim	qcut	qfrcmn	qfrcmx
0	0	29	40	well-a	0.0	0	0.0	0.0	0	0.0	0.0

```
[31]: mnw2.mnw
```

```
[31]: {'well-a': <flopy.modflow.mfmnw2.Mnw at 0x7f3751395d00>}
```

```
[32]: pd.DataFrame(mnw2.mnw["well-a"].stress_period_data)
```

```
[32]:
```

	k	i	j	per	qdes	capmult	cprime	hlim	qcut	qfrcmn	qfrcmx
0	0	29	40	0	0.0	0	0.0	0.0	0	0.0	0.0
1	0	29	40	1	-10000.0	0	0.0	0.0	0	0.0	0.0
2	0	29	40	2	-10000.0	0	0.0	0.0	0	0.0	0.0

```
[33]: path = os.path.join(".", ".", "examples", "data", "mnw2_examples")
m = flopy.modflow.Modflow("br", model_ws=model_ws)
mnw2 = flopy.modflow.ModflowMnw2.load(path + "/BadRiver_cal.mnw2", m)
```

```
[34]: df = pd.DataFrame(mnw2.node_data)
df.loc[:, df.sum(axis=0) != 0]
```

```
[34]:
```

	i	j	ztop	zbotm	wellid	losstype	pumploc	\
0	294	503	181.630005	161.630005	br_birch1	skin	-1	
1	295	503	179.119995	159.119995	br_birch2	skin	-1	
2	175	342	400.220001	312.220001	br_diaperville1	skin	-1	
3	174	342	399.119995	312.119995	br_diaperville2	skin	-1	
4	248	454	565.200012	555.200012	br_franks1	skin	-1	
5	249	453	564.419983	554.419983	br_franks2	skin	-1	
6	180	396	453.959991	447.959991	br_odanah1	skin	-1	
7	181	395	450.559998	444.559998	br_odanah2	skin	-1	
8	181	395	380.489990	371.489990	br_odanah3	skin	-1	
9	180	396	450.739990	444.739990	br_odanah4	skin	-1	
10	170	350	475.410004	472.410004	br_oldschool	skin	-1	
11	172	312	377.410004	348.410004	br_recycle	skin	-1	
12	216	412	562.200012	542.200012	br_unspec	skin	-1	
13	516	424	1079.910034	1072.910034	cfsp_ncp1	skin	-1	
14	515	424	1077.900024	1074.900024	cfsp_ncp2	skin	-1	
15	539	415	1093.010010	1003.010010	cfsp_of	skin	-1	
16	360	2	706.330017	699.330017	hayward_bait_n	skin	-1	
17	360	6	705.289978	698.289978	hayward_bait_ne	skin	-1	
18	362	4	696.979980	689.979980	hayward_bait_of	skin	-1	
19	456	699	1352.050049	1342.050049	ironbelt2	skin	-1	
20	456	699	1350.439941	1343.439941	ironbelt3	skin	-1	
21	599	414	1198.839966	1188.839966	mellen2	skin	-1	
22	576	408	1151.599976	1131.599976	mellen3	skin	-1	
23	253	149	622.179993	602.179993	milestone	skin	-1	
24	208	148	520.119995	500.119995	nsp	skin	-1	
25	80	162	535.299988	30.299999	washburn1	skin	-1	
26	91	143	541.280029	-67.720001	washburn2	skin	-1	

	rw	rskin	kskin	zpump
0	1.0	2.0	10.0	162.630005
1	1.0	2.0	10.0	160.119995
2	1.0	2.0	10.0	313.220001
3	1.0	2.0	10.0	313.119995
4	1.0	2.0	10.0	556.200012
5	1.0	2.0	10.0	555.419983
6	1.0	2.0	10.0	448.959991
7	1.0	2.0	10.0	445.559998
8	1.0	2.0	10.0	372.489990
9	1.0	2.0	10.0	445.739990
10	1.0	2.0	10.0	473.410004
11	1.0	2.0	10.0	349.410004
12	1.0	2.0	10.0	543.200012
13	1.0	2.0	10.0	1073.910034
14	1.0	2.0	10.0	1075.900024
15	1.0	2.0	10.0	1004.010010

(continues on next page)

(continued from previous page)

16	1.0	2.0	10.0	700.330017
17	1.0	2.0	10.0	699.289978
18	1.0	2.0	10.0	690.979980
19	1.0	2.0	10.0	1343.050049
20	1.0	2.0	10.0	1344.439941
21	1.0	2.0	10.0	1189.839966
22	1.0	2.0	10.0	1132.599976
23	1.0	2.0	10.0	603.179993
24	1.0	2.0	10.0	501.119995
25	1.0	2.0	10.0	31.299999
26	1.0	2.0	10.0	-66.720001

```
[35]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass
```

5.2.12 Accessing MODFLOW 6 Output

This tutorial shows how to access output from MODFLOW 6 models and packages by using the built in `.output` attribute on any MODFLOW 6 model or package object

```
[1]: import os
      from pathlib import Path
      from tempfile import TemporaryDirectory
```

```
[2]: import numpy as np
```

```
[3]: # ## Package import
      import flopy
```

Load a simple demonstration model

```
[4]: exe_name = "mf6"
      project_root_path = Path.cwd().parent.parent
      ws = os.path.abspath(os.path.dirname(""))
      sim_ws = str(
          project_root_path / "examples" / "data" / "mf6" / "test001e_UZF_3lay"
      )
```

```
[5]: # load the model
      sim = flopy.mf6.MFSimulation.load(
          sim_ws=sim_ws,
          exe_name=exe_name,
          verbosity_level=0,
      )
      # change the simulation path, rewrite the files, and run the model
```

(continues on next page)

(continued from previous page)

```
temp_dir = TemporaryDirectory()
sim_ws = temp_dir.name
sim.set_sim_path(sim_ws)
sim.write_simulation(silent=True)
sim.run_simulation(silent=True)
```

```
[5]: (True, [])
```

Get output using the .output attribute

The output attribute dynamically generates methods for each package based on the available output options within that package. A list of all available outputs are:

head()	Method to get the HeadFile object for the model. Accessed from the model object or the OC package object
budget()	Method to get the CellBudgetFile object for the model. Accessed from the model object or the OC package object
budgetcsv()	Method to get the MODFLOW-6 budget csv as a CsvFile object. Valid for model, oc, and advanced packages such as MAW, UZF, LAK
zonebudget()	Method to get the ZoneBudget6 object for the model. Accessed from the model object or the OC package object
obs()	Method to get observation file data in the form of a MF6Obs object. Accessed from any package that allows observations.
csv()	Method to get csv output data in the form of a CsvFile object. Example files are inner and outer iteration files from IMS
package_convergence	Method to get csv based package convergence information from packages such as SFR, LAK, UZF, and MAW. Returns a CsvFile object
stage()	Method to get binary stage file output from the SFR and LAK packages
concentration()	Method to get the binary concentration file output from a groundwater transport model. Accessed from the model object or the OC package object
cim()	Method to get immobile concentration output from the CIM package
density()	Method to get density file output from the BUY package

Get head file and cell budget file outputs

The head file output and cell budget file output can be loaded from either the model object or the OC package object.

```
[6]: ml = sim.get_model("gwf_1")
```

```
[7]: bud = ml.output.budget()
bud.get_data(idx=0, full3D=True)
```

```
[7]: [array([[ 0.,  0.,  0.,  0., -0.,  0.,  0.,  0., -0.,  0.,  0.,  0.,
          -0.,  0.,  0.,  0., -0.,  0.,  0.,  0., -0.,  0.,  0.,  0.,
          -0.,  0.,  0.,  0., -0.,  0.,  0.,  0., -0.,  0.,  0.,  0.,
          -0.,  0.,  0., -0.,  0.,  0.,  0., -0., -0.,  0.,  0.,  0.,
          -0., -0.,  0.,  0.,  0., -0., -0.,  0.,  0.,  0., -0., -0.,
           0.,  0.,  0., -0., -0.,  0.,  0.,  0., -0., -0.,  0.,  0.,
           0., -0., -0.,  0.,  0.,  0., -0., -0.,  0.,  0.,  0., -0.,
          -0.,  0.,  0., -0.,  0.,  0., -0., -0.,  0.,  0., -0., -0.,
```

(continues on next page)

[illegible]

Chapter 5. Tutorials

(continued from previous page)

```

→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (9.60e+02, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.00e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.04e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.08e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.12e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.16e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.20e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.24e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.28e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.32e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.36e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.40e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.44e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.48e+03, 0.00000000e+00, 0.      , 0., 0.      , 0., 0., 0.      ,
→ 0.      , 0.      , 0., 0., 0., 0.      , 0.      , 0.00000000e+00),
    (1.52e+03, 0.00000000e+00, 0.      , 0., 128.0217841 , 0., 0., 0.02722978,
→ 108.61508013, 19.37931792, 0., 0., 0., 128.0217841 , 128.02162783, 1.22061576e-04),
    (1.56e+03, 0.00000000e+00, 0.      , 0., 138.99381474, 0., 0., 0.02616677,
→ 103.79225125, 35.17524461, 0., 0., 0., 138.99381474, 138.99366263, 1.09439391e-04),
    (1.60e+03, 0.00000000e+00, 0.      , 0., 136.4972012 , 0., 0., 0.0225728 ,
→ 89.06987566, 47.4046209 , 0., 0., 0., 136.4972012 , 136.49706937, 9.65814295e-05),
    (1.64e+03, 0.00000000e+00, 0.      , 0., 134.35958704, 0., 0., 0.01957412,
→ 76.88688611, 57.45301299, 0., 0., 0., 134.35958704, 134.35947321, 8.47152128e-05),
    (1.68e+03, 0.00000000e+00, 0.      , 0., 131.24481479, 0., 0., 0.01672315,
→ 65.43186869, 65.79613475, 0., 0., 0., 131.24481479, 131.24472659, 6.72047136e-05),
    (1.72e+03, 0.00000000e+00, 0.      , 0., 123.13434058, 0., 0., 0.01312794,
→ 51.19484895, 71.92589496, 0., 0., 0., 123.13434058, 123.13387185, 3.80661214e-04),
    (1.76e+03, 0.00000000e+00, 0.      , 0., 114.00353442, 0., 0., 0.0097074 ,
→ 37.75854966, 76.23496189, 0., 0., 0., 114.00353442, 114.00321895, 2.76719717e-04),
    (1.80e+03, 0.00000000e+00, 0.      , 0., 110.11487407, 0., 0., 0.0078339 ,
→ 30.41322044, 79.693573 , 0., 0., 0., 110.11487407, 110.11462734, 2.24068325e-04),
    (1.84e+03, 0.00000000e+00, 0.      , 0., 105.40602911, 0., 0., 0.00594997,
→ 23.06535254, 82.33456504, 0., 0., 0., 105.40602911, 105.40586755, 1.53273477e-04),
    (1.88e+03, 0.00000000e+00, 0.      , 0., 95.08914871, 0., 0., 0.00307764,
→ 11.90628617, 83.17970019, 0., 0., 0., 95.08914871, 95.089064 , 8.90844871e-05),
    (1.92e+03, 0.00000000e+00, 0.      , 0., 94.69475504, 0., 0., 0.00271043,
→ 10.48542653, 84.20654512, 0., 0., 0., 94.69475504, 94.69468209, 7.70440756e-05),
    (1.96e+03, 0.00000000e+00, 0.      , 0., 89.77312957, 0., 0., 0.00134165,

```

(continues on next page)

(continued from previous page)

```

→ 5.18451079, 84.58703172, 0., 0., 0., 89.77312957, 89.77288416, 2.73360282e-04),
    (2.00e+03, 3.28550762e-05, 0.12927097, 0., 85.77949492, 0., 0., 0.00037754,
→ 1.45488492, 84.45346195, 0., 0., 0., 85.90879875, 85.90872441, 8.65285791e-05),
    (2.04e+03, 1.11673220e-03, 4.33428447, 0., 79.06662935, 0., 0., 0.,
→ 0., 83.40218928, 0., 0., 0., 83.40203055, 83.40218928, -1.90324298e-04),
    (2.08e+03, 9.50436281e-04, 3.6847171, 0., 79.01606345, 0., 0., 0.,
→ 0., 82.70187216, 0., 0., 0., 82.70173099, 82.70187216, -1.70697486e-04),
    (2.12e+03, 8.09420799e-04, 3.13589527, 0., 79.07017251, 0., 0., 0.,
→ 0., 82.20700075, 0., 0., 0., 82.2068772, 82.20700075, -1.50282359e-04),
    (2.16e+03, 1.74634716e-03, 6.76514808, 0., 74.44436606, 0., 0., 0.,
→ 0., 81.21149171, 0., 0., 0., 81.21126049, 81.21149171, -2.84711568e-04),
    (2.20e+03, 2.91726961e-03, 11.31295605, 0., 67.97128273, 0., 0., 0.,
→ 0., 79.28753204, 0., 0., 0., 79.28715604, 79.28753204, -4.74219376e-04),
    (2.24e+03, 3.18846934e-03, 12.36643397, 0., 65.05084871, 0., 0., 0.,
→ 0., 77.42085884, 0., 0., 0., 77.42047115, 77.42085884, -5.00756338e-04),
    (2.28e+03, 3.08245610e-03, 11.96212155, 0., 63.67724795, 0., 0., 0.,
→ 0., 75.64282768, 0., 0., 0., 75.64245196, 75.64282768, -4.96712518e-04),
    (2.32e+03, 2.66394411e-03, 10.34112384, 0., 63.8844137, 0., 0., 0.,
→ 0., 74.22852942, 0., 0., 0., 74.22820148, 74.22852942, -4.41794671e-04),
    (2.36e+03, 2.31518433e-03, 8.99067397, 0., 64.05458652, 0., 0., 0.,
→ 0., 73.04786211, 0., 0., 0., 73.04757567, 73.04786211, -3.92124078e-04),
    (2.40e+03, 2.17404830e-03, 8.44571365, 0., 63.52276663, 0., 0., 0.,
→ 0., 71.97090718, 0., 0., 0., 71.97065433, 71.97090718, -3.51328184e-04),
    (2.44e+03, 2.81993508e-03, 10.96537535, 0., 59.39318153, 0., 0., 0.,
→ 0., 70.36169271, 0., 0., 0., 70.36137682, 70.36169271, -4.48957261e-04),
    (2.48e+03, 2.99544059e-03, 11.65308129, 0., 57.07292771, 0., 0., 0.,
→ 0., 68.72930656, 0., 0., 0., 68.72900444, 68.72930656, -4.39583720e-04),
    (2.52e+03, 4.00001313e-03, 15.57599202, 0., 50.80019371, 0., 0., 0.,
→ 0., 66.38055059, 0., 0., 0., 66.38018574, 66.38055059, -5.49633887e-04),
    (2.56e+03, 3.81480656e-03, 14.86546946, 0., 49.32928447, 0., 0., 0.,
→ 0., 64.19891797, 0., 0., 0., 64.19856873, 64.19891797, -5.43997899e-04)],
    dtype=[('totim', '<f8'), ('STO-SS(STORAGE)_IN', '<f8'), ('STO-SY(STORAGE)_IN',
→ '<f8'), ('CHD(CHD-1)_IN', '<f8'), ('UZF-GWRCH(UZF-1)_IN', '<f8'), ('UZF-GWD(UZF-1)_IN',
→ '<f8'), ('UZF-GWET(UZF-1)_IN', '<f8'), ('STO-SS(STORAGE)_OUT', '<f8'), ('STO-
→ SY(STORAGE)_OUT', '<f8'), ('CHD(CHD-1)_OUT', '<f8'), ('UZF-GWRCH(UZF-1)_OUT', '<f8'), (
→ 'UZF-GWD(UZF-1)_OUT', '<f8'), ('UZF-GWET(UZF-1)_OUT', '<f8'), ('TOTAL_IN', '<f8'), (
→ 'TOTAL_OUT', '<f8'), ('PERCENT_DIFFERENCE', '<f8')])

```

```

[9]: hds = ml.output.head()
     hds.get_data()

```

```

[9]: array([[[-25., -24.93660487, -24.88816332, -24.85548043,
            -24.83902108, -24.83902108, -24.85548043, -24.88816332,
            -24.93660487, -25.          ]],

          [[[-25., -24.93660487, -24.88816332, -24.85548043,
            -24.83902108, -24.83902108, -24.85548043, -24.88816332,
            -24.93660487, -25.          ]],

          [[[-25., -24.93660487, -24.88816332, -24.85548043,
            -24.83902108, -24.83902108, -24.85548043, -24.88816332,
            -24.93660487, -25.          ]]]

```

```
[10]: bud = ml.oc.output.budget()
      bud.get_data(idx=0, full3D=True)

[10]: [array([[ 0.,  0.,  0.,  0., -0.,  0.,  0.,  0., -0.,  0.,  0.,  0.,
          -0.,  0.,  0.,  0., -0.,  0.,  0., -0.,  0.,  0.,  0.,
          -0.,  0.,  0.,  0., -0.,  0.,  0.,  0., -0.,  0.,  0.,  0.,
          -0.,  0.,  0., -0.,  0.,  0.,  0., -0., -0.,  0.,  0.,  0.,
          -0., -0.,  0.,  0.,  0., -0., -0.,  0.,  0.,  0., -0., -0.,
           0.,  0.,  0., -0., -0.,  0.,  0.,  0., -0., -0.,  0.,  0.,
           0., -0., -0.,  0.,  0.,  0., -0., -0.,  0.,  0.,  0., -0.,
          -0.,  0.,  0., -0.,  0.,  0., -0., -0.,  0.,  0., -0., -0.,
           0.,  0., -0., -0.,  0.,  0., -0., -0.,  0.,  0., -0., -0.,
           0.,  0., -0., -0.,  0.,  0., -0., -0.,  0.,  0., -0., -0.,
           0.,  0., -0., -0.]])]
```

```
[11]: hds = ml.oc.output.head()
      hds.get_data()

[11]: array([[[-25.          , -24.93660487, -24.88816332, -24.85548043,
          -24.83902108, -24.83902108, -24.85548043, -24.88816332,
          -24.93660487, -25.          ]],

          [[-25.          , -24.93660487, -24.88816332, -24.85548043,
          -24.83902108, -24.83902108, -24.85548043, -24.88816332,
          -24.93660487, -25.          ]],

          [[-25.          , -24.93660487, -24.88816332, -24.85548043,
          -24.83902108, -24.83902108, -24.85548043, -24.88816332,
          -24.93660487, -25.          ]]])]
```

Get output associated with a specific package

The `.output` attribute is tied to the package object and allows the user to get the output types specified in the MODFLOW 6 package. Here is an example with a UZF package that has UZF budget file output, budgetcsv file output, package convergence output, and observation data.

```
[12]: uzf = ml.uzf
      uzf_bud = uzf.output.budget()
      uzf_bud.get_data(idx=0)

[12]: [rec.array([( 1,  9,  0., 100000.), ( 9,  1, -0., 100000.),
          ( 2, 10,  0., 100000.), (10,  2, -0., 100000.),
          ( 3, 11,  0., 100000.), (11,  3, -0., 100000.),
          ( 4, 12,  0., 100000.), (12,  4, -0., 100000.),
          ( 5, 13,  0., 100000.), (13,  5, -0., 100000.),
          ( 6, 14,  0., 100000.), (14,  6, -0., 100000.),
          ( 7, 15,  0., 100000.), (15,  7, -0., 100000.),
          ( 8, 16,  0., 100000.), (16,  8, -0., 100000.),
          ( 9, 17,  0., 100000.), (17,  9, -0., 100000.),
          (10, 18,  0., 100000.), (18, 10, -0., 100000.),
          (11, 19,  0., 100000.), (19, 11, -0., 100000.),
          (12, 20,  0., 100000.), (20, 12, -0., 100000.),
          (13, 21,  0., 100000.), (21, 13, -0., 100000.)])]
```

(continues on next page)

(continued from previous page)

```

        (14, 22, 0., 100000.), (22, 14, -0., 100000.),
        (15, 23, 0., 100000.), (23, 15, -0., 100000.),
        (16, 24, 0., 100000.), (24, 16, -0., 100000.)],
        dtype=[('node', '<i4'), ('node2', '<i4'), ('q', '<f8'), ('FLOW-AREA', '<f8
→')]

```

```

[13]: uzf_budcsv = uzf.output.budgetcsv()
      uzf_budcsv.data

```

```

[13]: rec.array([(2.00e+00, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→01124365, 7999.98875635, 8000., 8000., 3.41060513e-14),
        (4.00e+00, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→02248729, 7999.97751271, 8000., 8000., -2.27373675e-14),
        (6.00e+00, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→03373091, 7999.96626909, 8000., 8000., 6.82121026e-14),
        (8.00e+00, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→04497452, 7999.95502458, 8000., 7999.9999991, 1.12417183e-08),
        (1.00e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→10119265, 7999.89880735, 8000., 8000., 1.13686838e-13),
        (1.20e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→15741054, 7999.84258946, 8000., 8000., -1.13686838e-14),
        (1.40e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→21362819, 7999.78637181, 8000., 8000., 6.82121026e-14),
        (1.60e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→26984561, 7999.73015439, 8000., 8000., -5.68434189e-14),
        (1.80e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→32606279, 7999.67393721, 8000., 8000., -2.38742359e-13),
        (2.00e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→38227973, 7999.61772027, 8000., 8000., 6.82121026e-14),
        (2.20e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→43849643, 7999.56150357, 8000., 8000., 2.04636308e-13),
        (2.40e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→13492205, 7999.86507795, 8000., 8000., -9.09494702e-14),
        (2.60e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→19113979, 7999.80886021, 8000., 8000., -3.63797881e-13),
        (2.80e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.2473573
→, 7999.7526427, 8000., 8000., 4.66116035e-13),
        (3.00e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.3035747
→, 7999.6964253, 8000., 8000., -9.09494702e-14),
        (3.20e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→35979212, 7999.64020788, 8000., 8000., -7.95807864e-14),
        (3.40e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.4160093
→, 7999.5839907, 8000., 8000., -9.09494702e-14),
        (3.60e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→47222624, 7999.52777376, 8000., 8000., -7.95807864e-14),
        (3.80e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→52844295, 7999.47155705, 8000., 8000., -9.09494702e-14),
        (4.00e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→22486856, 7999.77513144, 8000., 8000., 7.27595761e-13),
        (4.20e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→28108631, 7999.71891369, 8000., 8000., 2.04636308e-13),
        (4.40e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.

```

(continues on next page)

(continued from previous page)

```

→ 33475407, 7999.66524593, 8000.,      , 8000.,      , -1.02318154e-13),
    (4.60e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→ 38480503, 7999.61519497, 8000.,      , 8000.,      , -5.11590770e-13),
    (4.80e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→ 43407536, 7999.56558735, 8000.,      , 7999.99966271, 4.21615857e-06),
    (5.00e+01, 0., 8000., 0., 0., 1.18939114e-01, 0., 0., 0., 0.
→ 48208139, 7999.63685772, 8000.11893911, 8000.11893911, 4.54740590e-13),
    (5.20e+01, 0., 8000., 0., 0., 2.40000000e-01, 0., 0., 0., 0.
→ 62375028, 7999.61624972, 8000.24      , 8000.24      , 4.54733709e-14),
    (5.40e+01, 0., 8000., 0., 0., 2.40000000e-01, 0., 0., 0., 0.
→ 64056589, 7999.59943411, 8000.24      , 8000.24      , -1.13683427e-13),
    (5.60e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→ 32744504, 7999.67255496, 8000.,      , 8000.,      , -2.16004992e-13),
    (5.80e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→ 36195984, 7999.63804016, 8000.,      , 8000.,      , 1.93267624e-13),
    (6.00e+01, 0., 8000., 0., 0., 0.00000000e+00, 0., 0., 0., 0.
→ 39268212, 7999.60731788, 8000.,      , 8000.,      , 4.66116035e-13),
    (1.10e+02, 0., 0., 0., 0., 3.78724873e+03, 0., 0., 0., 0.
→ 41882422, 3786.82970478, 3787.24873018, 3787.24852899, 5.31225330e-06),
    (1.60e+02, 0., 0., 0., 0., 6.28239222e+02, 0., 0., 0., 0.
→ 51502761, 627.69903841, 628.23922183, 628.21406602, 4.00425715e-03),
    (2.10e+02, 0., 0., 0., 0., 3.79980452e+02, 0., 0., 0., 0.
→ 56814228, 379.36208201, 379.98045151, 379.9302243 , 1.32192414e-02),
    (2.60e+02, 0., 0., 0., 0., 1.22681840e+02, 0., 0., 0., 0.
→ 60336166, 122.00319618, 122.68183971, 122.60655784, 6.13823347e-02),
    (3.10e+02, 0., 0., 0., 0., 1.22634110e+02, 0., 0., 0., 0.
→ 58736429, 121.9464052 , 122.63410992, 122.53376948, 8.18544694e-02),
    (3.60e+02, 0., 0., 0., 0., 1.22586380e+02, 0., 0., 0., 0.
→ 58904253, 121.8719518 , 122.58638012, 122.46099433, 1.02335961e-01),
    (4.10e+02, 0., 0., 0., 0., 1.22538650e+02, 0., 0., 0., 0.
→ 57596753, 121.81226742, 122.53865033, 122.38823496, 1.22824712e-01),
    (4.60e+02, 0., 0., 0., 0., 1.22490921e+02, 0., 0., 0., 0.
→ 56435326, 121.75113567, 122.49092053, 122.31548892, 1.43322724e-01),
    (5.10e+02, 0., 0., 0., 0., 1.22443191e+02, 0., 0., 0., 0.
→ 55419916, 121.68855904, 122.44319073, 122.2427582 , 1.63828398e-01),
    (5.60e+02, 0., 0., 0., 0., 1.22395461e+02, 0., 0., 0., 0.
→ 56040475, 121.60963737, 122.39546094, 122.17004212, 1.84342282e-01),
    (6.00e+02, 0., 0., 0., 0., 1.74018709e+01, 0., 0., 0., 0.
→ 55999969, 16.82414951, 17.4018709 , 17.3841492 , 1.01889790e-01),
    (6.40e+02, 0., 0., 0., 0., 1.51944083e+02, 0., 0., 0., 0.
→ 55144142, 151.39263928, 151.94408282, 151.9440807 , 1.39256758e-06),
    (6.80e+02, 0., 0., 0., 0., 2.72064073e+02, 0., 0., 0., 0.
→ 54288289, 271.52105105, 272.06407257, 272.06393394, 5.09554104e-05),
    (7.20e+02, 0., 0., 0., 0., 2.43435052e+02, 0., 0., 0., 0.
→ 53432434, 242.90043095, 243.43505167, 243.43475529, 1.21747747e-04),
    (7.60e+02, 0., 0., 0., 0., 2.24436380e+02, 0., 0., 0., 0.
→ 52576579, 223.91019007, 224.43638039, 224.43595586, 1.89152075e-04),
    (8.00e+02, 0., 0., 0., 0., 1.92144820e+02, 0., 0., 0., 0.
→ 51720724, 191.62706411, 192.14481959, 192.14427135, 2.85328186e-04),
    (8.40e+02, 0., 0., 0., 0., 1.89295138e+02, 0., 0., 0., 0.
→ 50864868, 188.7858185 , 189.29513782, 189.29446718, 3.54282046e-04),
    (8.80e+02, 0., 0., 0., 0., 1.59711534e+02, 0., 0., 0., 0.

```

(continues on next page)

(continued from previous page)

```

→50009013, 159.2106566 , 159.71153439, 159.71074673, 4.93175545e-04),
    (9.20e+02, 0., 0., 0., 1.54912199e+02, 0., 0., 0., 0.
→49153158, 154.41976469, 154.91219927, 154.91129626, 5.82916597e-04),
    (9.60e+02, 0., 0., 0., 1.53348712e+02, 0., 0., 0., 0.
→48297303, 152.8647234 , 153.34871173, 153.34769643, 6.62088144e-04),
    (1.00e+03, 0., 0., 0., 1.22470337e+02, 0., 0., 0., 0.
→47441447, 121.99480078, 122.47033704, 122.46921525, 9.15973800e-04),
    (1.04e+03, 0., 0., 0., 1.22428683e+02, 0., 0., 0., 0.
→46585592, 121.9616002 , 122.42868329, 122.42745612, 1.00236285e-03),
    (1.08e+03, 0., 0., 0., 1.22381264e+02, 0., 0., 0., 0.
→45729737, 121.92263707, 122.38126375, 122.37993444, 1.08621048e-03),
    (1.12e+03, 0., 0., 0., 1.22333850e+02, 0., 0., 0., 0.
→44873881, 121.88368331, 122.33385014, 122.33242212, 1.16731720e-03),
    (1.16e+03, 0., 0., 0., 1.22274162e+02, 0., 0., 0., 0.
→44018026, 121.83245857, 122.27416212, 122.27263883, 1.24580486e-03),
    (1.20e+03, 0., 0., 0., 1.22222909e+02, 0., 0., 0., 0.
→43162171, 121.78967189, 122.22290873, 122.2212936 , 1.32147141e-03),
    (1.24e+03, 0., 0., 0., 1.22175517e+02, 0., 0., 0., 0.
→42306316, 121.75075005, 122.17551675, 122.17381321, 1.39435034e-03),
    (1.28e+03, 0., 0., 0., 1.22128131e+02, 0., 0., 0., 0.4145046
→, 121.71183756, 122.12813069, 122.12634217, 1.46447817e-03),
    (1.32e+03, 0., 0., 0., 1.22080751e+02, 0., 0., 0., 0.
→40594605, 121.67293442, 122.08075056, 122.07888047, 1.53185243e-03),
    (1.36e+03, 0., 0., 0., 1.22022484e+02, 0., 0., 0., 0.3973875
→, 121.62314831, 122.02248402, 122.0205358 , 1.59661237e-03),
    (1.40e+03, 0., 0., 0., 1.21761590e+02, 0., 0., 0., 0.
→38882895, 121.3707379 , 121.76158975, 121.75956685, 1.66137130e-03),
    (1.44e+03, 0., 0., 0., 1.15127548e+02, 0., 0., 0., 0.
→38027039, 114.74518428, 115.1275483 , 115.12545468, 1.81853759e-03),
    (1.48e+03, 0., 0., 0., 1.06834995e+02, 0., 0., 0., 0.
→37171184, 106.46112218, 106.83499475, 106.83283402, 2.02250892e-03),
    (1.52e+03, 0., 0., 0., 1.28387162e+02, 128.0217841 , 0., 0., 0.
→36315329, 0. , 128.3871623 , 128.38493739, 1.73299045e-03),
    (1.56e+03, 0., 0., 0., 1.39350694e+02, 138.99381474, 0., 0., 0.
→35459473, 0. , 139.35069441, 139.34840947, 1.63971258e-03),
    (1.60e+03, 0., 0., 0., 1.36845580e+02, 136.4972012 , 0., 0., 0.
→34603618, 0. , 136.84557981, 136.84323738, 1.71174694e-03),
    (1.64e+03, 0., 0., 0., 1.34699461e+02, 134.35958704, 0., 0., 0.
→33747763, 0. , 134.69946073, 134.69706466, 1.77884019e-03),
    (1.68e+03, 0., 0., 0., 1.31576180e+02, 131.24481479, 0., 0., 0.
→32891908, 0. , 131.57617996, 131.57373387, 1.85908572e-03),
    (1.72e+03, 0., 0., 0., 1.23457194e+02, 123.13434058, 0., 0., 0.
→32036052, 0. , 123.45719445, 123.4547011 , 2.01962857e-03),
    (1.76e+03, 0., 0., 0., 1.14317874e+02, 114.00353442, 0., 0., 0.
→31180197, 0. , 114.31787365, 114.31533639, 2.21950275e-03),
    (1.80e+03, 0., 0., 0., 1.10420694e+02, 110.11487407, 0., 0., 0.
→30324342, 0. , 110.42069432, 110.41811749, 2.33367700e-03),
    (1.84e+03, 0., 0., 0., 1.05703327e+02, 105.40602911, 0., 0., 0.
→29468487, 0. , 105.7033274 , 105.70071398, 2.47244563e-03),
    (1.88e+03, 0., 0., 0., 9.53779221e+01, 95.08914871, 0., 0., 0.
→28612631, 0. , 95.37792207, 95.37527503, 2.77535835e-03),
    (1.92e+03, 0., 0., 0., 9.49750001e+01, 94.69475504, 0., 0., 0.

```

(continues on next page)

(continued from previous page)

```

→27756776, 0., 94.97500009, 94.9723228, 2.81897970e-03),
    (1.96e+03, 0., 0., 0., 0., 9.00448429e+01, 89.77312957, 0., 0., 0.
→26900921, 0., 90.0448429, 90.04213878, 3.00312769e-03),
    (2.00e+03, 0., 0., 0., 0., 8.60426727e+01, 85.77949492, 0., 0., 0.
→26045065, 0., 86.04267268, 86.03994557, 3.16952968e-03),
    (2.04e+03, 0., 0., 0., 0., 7.93212676e+01, 79.06662935, 0., 0., 0.2518921
→, 0., 79.3212676, 79.31852145, 3.46212104e-03),
    (2.08e+03, 0., 0., 0., 0., 7.92621596e+01, 79.01606345, 0., 0., 0.
→24333355, 0., 79.26215964, 79.259397, 3.48550200e-03),
    (2.12e+03, 0., 0., 0., 0., 7.93077233e+01, 79.07017251, 0., 0., 0.234775
→, 0., 79.30772334, 79.30494751, 3.50012862e-03),
    (2.16e+03, 0., 0., 0., 0., 7.46733681e+01, 74.44436606, 0., 0., 0.
→22621644, 0., 74.67336811, 74.67058251, 3.73045242e-03),
    (2.20e+03, 0., 0., 0., 0., 6.81917326e+01, 67.97128273, 0., 0., 0.
→21765789, 0., 68.19173259, 68.18894062, 4.09438250e-03),
    (2.24e+03, 0., 0., 0., 0., 6.52627430e+01, 65.05084871, 0., 0., 0.
→20909934, 0., 65.26274298, 65.25994805, 4.28268329e-03),
    (2.28e+03, 0., 0., 0., 0., 6.38805832e+01, 63.67724795, 0., 0., 0.
→20054079, 0., 63.88058323, 63.87778874, 4.37465570e-03),
    (2.32e+03, 0., 0., 0., 0., 6.40791861e+01, 63.8844137, 0., 0., 0.
→19198223, 0., 64.07918605, 64.07639593, 4.35427472e-03),
    (2.36e+03, 0., 0., 0., 0., 6.42407922e+01, 64.05458652, 0., 0., 0.
→18342368, 0., 64.2407922, 64.2380102, 4.33067676e-03),
    (2.40e+03, 0., 0., 0., 0., 6.37004030e+01, 63.52276663, 0., 0., 0.
→17486513, 0., 63.70040299, 63.69763176, 4.35050707e-03),
    (2.44e+03, 0., 0., 0., 0., 5.95622453e+01, 59.39318153, 0., 0., 0.
→16630658, 0., 59.56224528, 59.55948811, 4.62916709e-03),
    (2.48e+03, 0., 0., 0., 0., 5.72334154e+01, 57.07292771, 0., 0., 0.
→15774802, 0., 57.23341545, 57.23067573, 4.78702424e-03),
    (2.52e+03, 0., 0., 0., 0., 5.09521020e+01, 50.80019371, 0., 0., 0.
→14918947, 0., 50.95210203, 50.94938318, 5.33623414e-03),
    (2.56e+03, 0., 0., 0., 0., 4.94726100e+01, 49.32928447, 0., 0., 0.
→14063092, 0., 49.47260998, 49.46991539, 5.44677831e-03)],
    dtype=[('totim', '<f8'), ('GWF_IN', '<f8'), ('INFILTRATION_IN', '<f8'), ('REJ-
→INF_IN', '<f8'), ('UZET_IN', '<f8'), ('STORAGE_IN', '<f8'), ('GWF_OUT', '<f8'), (
→'INFILTRATION_OUT', '<f8'), ('REJ-INF_OUT', '<f8'), ('UZET_OUT', '<f8'), ('STORAGE_OUT
→', '<f8'), ('TOTAL_IN', '<f8'), ('TOTAL_OUT', '<f8'), ('PERCENT_DIFFERENCE', '<f8')])

```

```

[14]: uzf_conv = uzf.output.package_convergence()
      if uzf_conv is not None:
          uzf_conv.data[0:10]

```

```

[15]: uzf_obs = uzf.output.obs()
      uzf_obs.data[0:10]

```

```

[15]: rec.array([( 2., 0.05, 0.05, 0.), ( 4., 0.05, 0.05, 0.),
    ( 6., 0.05, 0.05, 0.), ( 8., 0.05, 0.05, 0.),
    (10., 0.05, 0.05, 0.), (12., 0.05, 0.05, 0.),
    (14., 0.05, 0.05, 0.), (16., 0.05, 0.05, 0.),
    (18., 0.05, 0.05, 0.), (20., 0.05, 0.05, 0.)],
      dtype=[('totim', '<f8'), ('ID3_DPTH=8.0', '<f8'), ('ID3_DPTH=24.0', '<f8'), (
→'ID3_RCH', '<f8')])

```

Check which output types are available in a package

The `.output` attribute also has a `methods()` function that returns a list of available output functions for a given package. Here are a couple of examples

```
[16]: print("UZF package: ", uzf.output.methods())
      print("Model object: ", ml.output.methods())
      print("OC package: ", ml.oc.output.methods())
      print("DIS package: ", ml.dis.output.methods())

UZF package:  ['zonebudget()', 'budget()', 'budgetcsv()', 'package_convergence()', 'obs()'
→ '']
Model object:  ['list()', 'zonebudget()', 'budget()', 'budgetcsv()', 'head()']
OC package:  ['list()', 'zonebudget()', 'budget()', 'budgetcsv()', 'head()']
DIS package:  None
```

Managing multiple observation and csv file outputs in the same package

For many packages, multiple observation output files can be used. The `obs()` and `csv()` functions allow the user to specify a observation file or csv file name. If no name is specified, the `obs()` and `csv()` methods will return the first file that is listed in the package.

```
[17]: output = ml.obs[0].output
      obs_names = output.obs_names
      output.obs(f=obs_names[0]).data[0:10]

[17]: rec.array([( 2., 0.05, 0.05, 0.), ( 4., 0.05, 0.05, 0.),
                ( 6., 0.05, 0.05, 0.), ( 8., 0.05, 0.05, 0.),
                (10., 0.05, 0.05, 0.), (12., 0.05, 0.05, 0.),
                (14., 0.05, 0.05, 0.), (16., 0.05, 0.05, 0.),
                (18., 0.05, 0.05, 0.), (20., 0.05, 0.05, 0.)],
              dtype=[('totim', '<f8'), ('ID3_DPTH=8.0', '<f8'), ('ID3_DPTH=24.0', '<f8'), (
→ 'ID3_RCH', '<f8')])
```

Creating and running ZoneBudget for MF6

For the model and many packages, zonebudget can be run on the cell budget file. The `.output` method allows the user to easily build a ZoneBudget6 instance, then run the model, and view output. First we'll build a layered zone array, then build and run zonebudget

```
[18]: zarr = np.ones(ml.modelgrid.shape, dtype=int)
      for i in range(1, 4):
          zarr[i - 1] *= i
```

```
[19]: zonbud = ml.output.zonebudget(zarr)
      zonbud.change_model_ws(sim_ws)
      zonbud.write_input()
      zonbud.run_model()
```

```
FloPy is using the following executable to run the model: ../../home/runner/.local/bin/
→ modflow/zbud6
```

ZONEBUDGET Version 6

(continues on next page)

(continued from previous page)

U.S. GEOLOGICAL SURVEY
VERSION 6.4.2 06/28/2023

...

Normal Termination

[19]: (True, [])

```
[20]: df = zonbud.get_dataframes(net=True)
df = df.reset_index()
df
```

```
[20]:
```

	totim	name	ZONE_1	ZONE_2	ZONE_3
0	2.0	ZONE_0	0.0	0.000000e+00	0.000000e+00
1	2.0	ZONE_1	0.0	0.000000e+00	0.000000e+00
2	2.0	ZONE_2	0.0	0.000000e+00	0.000000e+00
3	2.0	ZONE_3	0.0	0.000000e+00	0.000000e+00
4	4.0	ZONE_0	0.0	0.000000e+00	0.000000e+00
..
355	2520.0	ZONE_3	0.0	0.000000e+00	0.000000e+00
356	2560.0	ZONE_0	0.0	0.000000e+00	0.000000e+00
357	2560.0	ZONE_1	0.0	0.000000e+00	0.000000e+00
358	2560.0	ZONE_2	0.0	0.000000e+00	-5.684342e-10
359	2560.0	ZONE_3	0.0	5.684342e-10	0.000000e+00

[360 rows x 5 columns]

```
[21]: try:
temp_dir.cleanup()
except:
# prevent windows permission error
pass
```

5.2.13 SFR2 package loading and querying

```
[1]: import os
```

```
[2]: import sys

# run installed version of flopy or add local path
try:
import flopy
except:
fpth = os.path.abspath(os.path.join("..", ".."))
sys.path.append(fpth)
import flopy

print(sys.version)
print("flopy version: {}".format(flopy.__version__))
```

```
3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
flopy version: 3.4.3
```

Create a model instance

```
[3]: m = flopy.modflow.Modflow()
```

Read the SFR2 file

```
[4]: f = os.path.join(
    "..", "..", "examples", "data", "mf2005_test", "testsfr2_tab_ICALC2.sfr"
)
stuff = open(f).readlines()
stuff

[4]: ['# SFR2 Package input file for hypothetical test simulation\n',
      '# Test of SFR2 Package; revised October 1, 2011\n',
      '# Example for using keyword options\n',
      '# Stress period inflow rates are now specified in testsfr2_tab.tab TRANROUTE\n',
      'REACHINPUT \n',
      'TABFILES 1 50\n',
      ' 100 1 0 0 1.0 0.00001 -1 0 5 20 5 20 0 1 0.7 0.0001      Item 1: NSTRM NSS
↳NSFRPAR NPARSEG CONST DLEAK ISTCB1 ISTCB2 {ISFROPT} {NSTRAIL} {ISUZN} {NSFRSETS}\n',
      '      1      4      1      1      1      200.0 \n',
      '      1      4      2      1      2      200.0 \n',
      '      1      4      3      1      3      200.0 \n',
      '      1      4      4      1      4      200.0 \n',
      '      1      4      5      1      5      200.0 \n',
      '      1      4      6      1      6      200.0 \n',
      '      1      4      7      1      7      200.0 \n',
      '      1      4      8      1      8      200.0 \n',
      '      1      4      9      1      9      200.0 \n',
      '      1      4     10      1     10      200.0 \n',
      '      1      4     11      1     11      200.0 \n',
      '      1      4     12      1     12      200.0 \n',
      '      1      4     13      1     13      200.0 \n',
      '      1      4     14      1     14      200.0 \n',
      '      1      4     15      1     15      200.0 \n',
      '      1      4     16      1     16      200.0 \n',
      '      1      4     17      1     17      200.0 \n',
      '      1      4     18      1     18      200.0 \n',
      '      1      4     19      1     19      200.0 \n',
      '      1      4     20      1     20      200.0 \n',
      '      1      4     21      1     21      200.0 \n',
      '      1      4     22      1     22      200.0 \n',
      '      1      4     23      1     23      200.0 \n',
      '      1      4     24      1     24      200.0 \n',
      '      1      4     25      1     25      200.0 \n',
      '      1      4     26      1     26      200.0 \n',
      '      1      4     27      1     27      200.0 \n',
      '      1      4     28      1     28      200.0 \n',
      '      1      4     29      1     29      200.0 \n',
      '      1      4     30      1     30      200.0 \n',
```

(continues on next page)

(continued from previous page)

'	1	4	31	1	31	200.0	\n',
'	1	4	32	1	32	200.0	\n',
'	1	4	33	1	33	200.0	\n',
'	1	4	34	1	34	200.0	\n',
'	1	4	35	1	35	200.0	\n',
'	1	4	36	1	36	200.0	\n',
'	1	4	37	1	37	200.0	\n',
'	1	4	38	1	38	200.0	\n',
'	1	4	39	1	39	200.0	\n',
'	1	4	40	1	40	200.0	\n',
'	1	4	41	1	41	200.0	\n',
'	1	4	42	1	42	200.0	\n',
'	1	4	43	1	43	200.0	\n',
'	1	4	44	1	44	200.0	\n',
'	1	4	45	1	45	200.0	\n',
'	1	4	46	1	46	200.0	\n',
'	1	4	47	1	47	200.0	\n',
'	1	4	48	1	48	200.0	\n',
'	1	4	49	1	49	200.0	\n',
'	1	4	50	1	50	200.0	\n',
'	1	4	51	1	51	200.0	\n',
'	1	4	52	1	52	200.0	\n',
'	1	4	53	1	53	200.0	\n',
'	1	4	54	1	54	200.0	\n',
'	1	4	55	1	55	200.0	\n',
'	1	4	56	1	56	200.0	\n',
'	1	4	57	1	57	200.0	\n',
'	1	4	58	1	58	200.0	\n',
'	1	4	59	1	59	200.0	\n',
'	1	4	60	1	60	200.0	\n',
'	1	4	61	1	61	200.0	\n',
'	1	4	62	1	62	200.0	\n',
'	1	4	63	1	63	200.0	\n',
'	1	4	64	1	64	200.0	\n',
'	1	4	65	1	65	200.0	\n',
'	1	4	66	1	66	200.0	\n',
'	1	4	67	1	67	200.0	\n',
'	1	4	68	1	68	200.0	\n',
'	1	4	69	1	69	200.0	\n',
'	1	4	70	1	70	200.0	\n',
'	1	4	71	1	71	200.0	\n',
'	1	4	72	1	72	200.0	\n',
'	1	4	73	1	73	200.0	\n',
'	1	4	74	1	74	200.0	\n',
'	1	4	75	1	75	200.0	\n',
'	1	4	76	1	76	200.0	\n',
'	1	4	77	1	77	200.0	\n',
'	1	4	78	1	78	200.0	\n',
'	1	4	79	1	79	200.0	\n',
'	1	4	80	1	80	200.0	\n',
'	1	4	81	1	81	200.0	\n',
'	1	4	82	1	82	200.0	\n',

(continues on next page)

(continued from previous page)

[illegible]

(continues on next page)

[illegible]

(continued from previous page)

[illegible]

(continues on next page)

[illegible]

(continues on next page)

(continued from previous page)

'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'\n',
'0\n',
'\n',
'\n',
'\n',
'1\n',
'\n',
'\n',
'\n',
'2\n',
'\n',
'\n',
'\n',
'3\n',
'\n',
'\n',
'\n',
'4\n',
'\n',
'\n',
'\n',
'5\n',
'\n',
'\n',
'\n',
'6\n',
'\n',
'\n',
'\n',
'7\n',
'\n',
'\n',
'\n',
'8\n',
'\n',

(continues on next page)

(continued from previous page)

```
'\n',
'\n',
'9\n',
'\n',
'\n',
'\n',
'0\n',
'\n',
'\n',
'\n',
'1\n',
'\n',
'\n',
'\n',
'2\n',
'\n',
'\n',
'\n',
'3\n',
'\n',
'\n',
'\n',
'4\n',
'\n',
'\n',
'\n',
'5\n',
'\n',
'\n',
'\n',
'6\n',
'\n',
'\n',
'\n',
'7\n',
'\n',
'\n',
'\n',
'8\n',
'\n',
'\n',
'\n',
'9\n',
'\n',
'\n',
'\n',
'0\n',
'\n',
'\n',
'\n',
'1\n',
'\n',
```

(continues on next page)

(continued from previous page)

```
'\n',
'\n',
'2\n',
'\n',
'\n',
'\n',
'3\n',
'\n',
'\n',
'\n',
'4\n',
'\n',
'\n',
' \n',
'5\n',
'\n',
'\n',
'\n',
'6\n',
'\n',
'\n',
'\n',
'7\n',
'\n',
'\n',
'\n',
'8\n',
'\n',
'\n',
'\n',
'9\n',
'\n',
'\n',
'\n',
'0\n',
'\n',
'\n',
' \n',
'1\n',
'\n',
'\n',
' \n',
'2\n',
'\n',
'\n',
'\n',
'3\n',
'\n',
'\n',
'\n',
'4\n',
'\n',
```

(continues on next page)

(continued from previous page)

```
'\n',
'\n',
'5\n',
'\n',
'\n',
'\n',
'6\n',
'\n',
'\n',
'\n',
'7\n',
'\n',
'\n',
'\n',
'8\n',
'\n',
'\n',
'\n',
'9\n',
'\n',
'\n',
'\n',
'0\n',
'\n',
'\n',
'\n',
']
```

Load the SFR2 file

```
[5]: sfr = flopy.modflow.ModflowSfr2.load(f, m, nper=50)
```

```
[6]: sfr.segment_data.keys()
```

```
[6]: dict_keys([0])
```

Query the reach data in the SFR2 file

```
[7]: sfr.reach_data
```

```
[7]: rec.array([(0, 0, 3, 0, 1, 1, 200., 0., 0., 0., 0., 0., 0., 0., 1, 0),
(0, 0, 3, 1, 1, 2, 200., 0., 0., 0., 0., 0., 0., 0., 2, 0),
(0, 0, 3, 2, 1, 3, 200., 0., 0., 0., 0., 0., 0., 0., 3, 0),
(0, 0, 3, 3, 1, 4, 200., 0., 0., 0., 0., 0., 0., 0., 4, 0),
(0, 0, 3, 4, 1, 5, 200., 0., 0., 0., 0., 0., 0., 0., 5, 0),
(0, 0, 3, 5, 1, 6, 200., 0., 0., 0., 0., 0., 0., 0., 6, 0),
(0, 0, 3, 6, 1, 7, 200., 0., 0., 0., 0., 0., 0., 0., 7, 0),
(0, 0, 3, 7, 1, 8, 200., 0., 0., 0., 0., 0., 0., 0., 8, 0),
(0, 0, 3, 8, 1, 9, 200., 0., 0., 0., 0., 0., 0., 0., 9, 0),
(0, 0, 3, 9, 1, 10, 200., 0., 0., 0., 0., 0., 0., 0., 10, 0),
(0, 0, 3, 10, 1, 11, 200., 0., 0., 0., 0., 0., 0., 0., 11, 0),
(0, 0, 3, 11, 1, 12, 200., 0., 0., 0., 0., 0., 0., 0., 12, 0),
(0, 0, 3, 12, 1, 13, 200., 0., 0., 0., 0., 0., 0., 0., 13, 0),
```

(continues on next page)

(continued from previous page)

```

(0, 0, 3, 13, 1, 14, 200., 0., 0., 0., 0., 0., 0., 0., 0., 14, 0),
(0, 0, 3, 14, 1, 15, 200., 0., 0., 0., 0., 0., 0., 0., 0., 15, 0),
(0, 0, 3, 15, 1, 16, 200., 0., 0., 0., 0., 0., 0., 0., 0., 16, 0),
(0, 0, 3, 16, 1, 17, 200., 0., 0., 0., 0., 0., 0., 0., 0., 17, 0),
(0, 0, 3, 17, 1, 18, 200., 0., 0., 0., 0., 0., 0., 0., 0., 18, 0),
(0, 0, 3, 18, 1, 19, 200., 0., 0., 0., 0., 0., 0., 0., 0., 19, 0),
(0, 0, 3, 19, 1, 20, 200., 0., 0., 0., 0., 0., 0., 0., 0., 20, 0),
(0, 0, 3, 20, 1, 21, 200., 0., 0., 0., 0., 0., 0., 0., 0., 21, 0),
(0, 0, 3, 21, 1, 22, 200., 0., 0., 0., 0., 0., 0., 0., 0., 22, 0),
(0, 0, 3, 22, 1, 23, 200., 0., 0., 0., 0., 0., 0., 0., 0., 23, 0),
(0, 0, 3, 23, 1, 24, 200., 0., 0., 0., 0., 0., 0., 0., 0., 24, 0),
(0, 0, 3, 24, 1, 25, 200., 0., 0., 0., 0., 0., 0., 0., 0., 25, 0),
(0, 0, 3, 25, 1, 26, 200., 0., 0., 0., 0., 0., 0., 0., 0., 26, 0),
(0, 0, 3, 26, 1, 27, 200., 0., 0., 0., 0., 0., 0., 0., 0., 27, 0),
(0, 0, 3, 27, 1, 28, 200., 0., 0., 0., 0., 0., 0., 0., 0., 28, 0),
(0, 0, 3, 28, 1, 29, 200., 0., 0., 0., 0., 0., 0., 0., 0., 29, 0),
(0, 0, 3, 29, 1, 30, 200., 0., 0., 0., 0., 0., 0., 0., 0., 30, 0),
(0, 0, 3, 30, 1, 31, 200., 0., 0., 0., 0., 0., 0., 0., 0., 31, 0),
(0, 0, 3, 31, 1, 32, 200., 0., 0., 0., 0., 0., 0., 0., 0., 32, 0),
(0, 0, 3, 32, 1, 33, 200., 0., 0., 0., 0., 0., 0., 0., 0., 33, 0),
(0, 0, 3, 33, 1, 34, 200., 0., 0., 0., 0., 0., 0., 0., 0., 34, 0),
(0, 0, 3, 34, 1, 35, 200., 0., 0., 0., 0., 0., 0., 0., 0., 35, 0),
(0, 0, 3, 35, 1, 36, 200., 0., 0., 0., 0., 0., 0., 0., 0., 36, 0),
(0, 0, 3, 36, 1, 37, 200., 0., 0., 0., 0., 0., 0., 0., 0., 37, 0),
(0, 0, 3, 37, 1, 38, 200., 0., 0., 0., 0., 0., 0., 0., 0., 38, 0),
(0, 0, 3, 38, 1, 39, 200., 0., 0., 0., 0., 0., 0., 0., 0., 39, 0),
(0, 0, 3, 39, 1, 40, 200., 0., 0., 0., 0., 0., 0., 0., 0., 40, 0),
(0, 0, 3, 40, 1, 41, 200., 0., 0., 0., 0., 0., 0., 0., 0., 41, 0),
(0, 0, 3, 41, 1, 42, 200., 0., 0., 0., 0., 0., 0., 0., 0., 42, 0),
(0, 0, 3, 42, 1, 43, 200., 0., 0., 0., 0., 0., 0., 0., 0., 43, 0),
(0, 0, 3, 43, 1, 44, 200., 0., 0., 0., 0., 0., 0., 0., 0., 44, 0),
(0, 0, 3, 44, 1, 45, 200., 0., 0., 0., 0., 0., 0., 0., 0., 45, 0),
(0, 0, 3, 45, 1, 46, 200., 0., 0., 0., 0., 0., 0., 0., 0., 46, 0),
(0, 0, 3, 46, 1, 47, 200., 0., 0., 0., 0., 0., 0., 0., 0., 47, 0),
(0, 0, 3, 47, 1, 48, 200., 0., 0., 0., 0., 0., 0., 0., 0., 48, 0),
(0, 0, 3, 48, 1, 49, 200., 0., 0., 0., 0., 0., 0., 0., 0., 49, 0),
(0, 0, 3, 49, 1, 50, 200., 0., 0., 0., 0., 0., 0., 0., 0., 50, 0),
(0, 0, 3, 50, 1, 51, 200., 0., 0., 0., 0., 0., 0., 0., 0., 51, 0),
(0, 0, 3, 51, 1, 52, 200., 0., 0., 0., 0., 0., 0., 0., 0., 52, 0),
(0, 0, 3, 52, 1, 53, 200., 0., 0., 0., 0., 0., 0., 0., 0., 53, 0),
(0, 0, 3, 53, 1, 54, 200., 0., 0., 0., 0., 0., 0., 0., 0., 54, 0),
(0, 0, 3, 54, 1, 55, 200., 0., 0., 0., 0., 0., 0., 0., 0., 55, 0),
(0, 0, 3, 55, 1, 56, 200., 0., 0., 0., 0., 0., 0., 0., 0., 56, 0),
(0, 0, 3, 56, 1, 57, 200., 0., 0., 0., 0., 0., 0., 0., 0., 57, 0),
(0, 0, 3, 57, 1, 58, 200., 0., 0., 0., 0., 0., 0., 0., 0., 58, 0),
(0, 0, 3, 58, 1, 59, 200., 0., 0., 0., 0., 0., 0., 0., 0., 59, 0),
(0, 0, 3, 59, 1, 60, 200., 0., 0., 0., 0., 0., 0., 0., 0., 60, 0),
(0, 0, 3, 60, 1, 61, 200., 0., 0., 0., 0., 0., 0., 0., 0., 61, 0),
(0, 0, 3, 61, 1, 62, 200., 0., 0., 0., 0., 0., 0., 0., 0., 62, 0),
(0, 0, 3, 62, 1, 63, 200., 0., 0., 0., 0., 0., 0., 0., 0., 63, 0),
(0, 0, 3, 63, 1, 64, 200., 0., 0., 0., 0., 0., 0., 0., 0., 64, 0),
(0, 0, 3, 64, 1, 65, 200., 0., 0., 0., 0., 0., 0., 0., 0., 65, 0),

```

(continues on next page)

(continued from previous page)

```

(0, 0, 3, 65, 1, 66, 200., 0., 0., 0., 0., 0., 0., 0., 66, 0),
(0, 0, 3, 66, 1, 67, 200., 0., 0., 0., 0., 0., 0., 0., 67, 0),
(0, 0, 3, 67, 1, 68, 200., 0., 0., 0., 0., 0., 0., 0., 68, 0),
(0, 0, 3, 68, 1, 69, 200., 0., 0., 0., 0., 0., 0., 0., 69, 0),
(0, 0, 3, 69, 1, 70, 200., 0., 0., 0., 0., 0., 0., 0., 70, 0),
(0, 0, 3, 70, 1, 71, 200., 0., 0., 0., 0., 0., 0., 0., 71, 0),
(0, 0, 3, 71, 1, 72, 200., 0., 0., 0., 0., 0., 0., 0., 72, 0),
(0, 0, 3, 72, 1, 73, 200., 0., 0., 0., 0., 0., 0., 0., 73, 0),
(0, 0, 3, 73, 1, 74, 200., 0., 0., 0., 0., 0., 0., 0., 74, 0),
(0, 0, 3, 74, 1, 75, 200., 0., 0., 0., 0., 0., 0., 0., 75, 0),
(0, 0, 3, 75, 1, 76, 200., 0., 0., 0., 0., 0., 0., 0., 76, 0),
(0, 0, 3, 76, 1, 77, 200., 0., 0., 0., 0., 0., 0., 0., 77, 0),
(0, 0, 3, 77, 1, 78, 200., 0., 0., 0., 0., 0., 0., 0., 78, 0),
(0, 0, 3, 78, 1, 79, 200., 0., 0., 0., 0., 0., 0., 0., 79, 0),
(0, 0, 3, 79, 1, 80, 200., 0., 0., 0., 0., 0., 0., 0., 80, 0),
(0, 0, 3, 80, 1, 81, 200., 0., 0., 0., 0., 0., 0., 0., 81, 0),
(0, 0, 3, 81, 1, 82, 200., 0., 0., 0., 0., 0., 0., 0., 82, 0),
(0, 0, 3, 82, 1, 83, 200., 0., 0., 0., 0., 0., 0., 0., 83, 0),
(0, 0, 3, 83, 1, 84, 200., 0., 0., 0., 0., 0., 0., 0., 84, 0),
(0, 0, 3, 84, 1, 85, 200., 0., 0., 0., 0., 0., 0., 0., 85, 0),
(0, 0, 3, 85, 1, 86, 200., 0., 0., 0., 0., 0., 0., 0., 86, 0),
(0, 0, 3, 86, 1, 87, 200., 0., 0., 0., 0., 0., 0., 0., 87, 0),
(0, 0, 3, 87, 1, 88, 200., 0., 0., 0., 0., 0., 0., 0., 88, 0),
(0, 0, 3, 88, 1, 89, 200., 0., 0., 0., 0., 0., 0., 0., 89, 0),
(0, 0, 3, 89, 1, 90, 200., 0., 0., 0., 0., 0., 0., 0., 90, 0),
(0, 0, 3, 90, 1, 91, 200., 0., 0., 0., 0., 0., 0., 0., 91, 0),
(0, 0, 3, 91, 1, 92, 200., 0., 0., 0., 0., 0., 0., 0., 92, 0),
(0, 0, 3, 92, 1, 93, 200., 0., 0., 0., 0., 0., 0., 0., 93, 0),
(0, 0, 3, 93, 1, 94, 200., 0., 0., 0., 0., 0., 0., 0., 94, 0),
(0, 0, 3, 94, 1, 95, 200., 0., 0., 0., 0., 0., 0., 0., 95, 0),
(0, 0, 3, 95, 1, 96, 200., 0., 0., 0., 0., 0., 0., 0., 96, 0),
(0, 0, 3, 96, 1, 97, 200., 0., 0., 0., 0., 0., 0., 0., 97, 0),
(0, 0, 3, 97, 1, 98, 200., 0., 0., 0., 0., 0., 0., 0., 98, 0),
(0, 0, 3, 98, 1, 99, 200., 0., 0., 0., 0., 0., 0., 0., 99, 0),
(0, 0, 3, 99, 1, 100, 200., 0., 0., 0., 0., 0., 0., 0., 100, 0)],
dtype=[('node', '<i8'), ('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('iseg', '<i8'
→'), ('ireach', '<i8'), ('rchlen', '<f4'), ('strtop', '<f4'), ('slope', '<f4'), (
→'strthick', '<f4'), ('strhcl', '<f4'), ('thts', '<f4'), ('thti', '<f4'), ('eps', '<f4'
→'), ('uhc', '<f4'), ('reachID', '<i8'), ('outreach', '<i8')]]

```

Query the channel flow data in the SFR2 file

```
[8]: sfr.channel_flow_data
```

```
[8]: {}
```

Query the channel geometry data in the SFR2 file

```
[9]: sfr.channel_geometry_data
```

```
[9]: {0: {1: [[0.0, 2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0],
[6.0, 4.5, 3.5, 0.0, 0.3, 3.5, 4.5, 6.0]]}}
```

Query dataset 5 data in the SFR2 file

```
[10]: sfr.dataset_5
```

```
[10]: {0: [1, 0, 0, 0],
      1: [-1, 0, 0, 0],
      2: [-1, 0, 0, 0],
      3: [-1, 0, 0, 0],
      4: [-1, 0, 0, 0],
      5: [-1, 0, 0, 0],
      6: [-1, 0, 0, 0],
      7: [-1, 0, 0, 0],
      8: [-1, 0, 0, 0],
      9: [-1, 0, 0, 0],
      10: [-1, 0, 0, 0],
      11: [-1, 0, 0, 0],
      12: [-1, 0, 0, 0],
      13: [-1, 0, 0, 0],
      14: [-1, 0, 0, 0],
      15: [-1, 0, 0, 0],
      16: [-1, 0, 0, 0],
      17: [-1, 0, 0, 0],
      18: [-1, 0, 0, 0],
      19: [-1, 0, 0, 0],
      20: [-1, 0, 0, 0],
      21: [-1, 0, 0, 0],
      22: [-1, 0, 0, 0],
      23: [-1, 0, 0, 0],
      24: [-1, 0, 0, 0],
      25: [-1, 0, 0, 0],
      26: [-1, 0, 0, 0],
      27: [-1, 0, 0, 0],
      28: [-1, 0, 0, 0],
      29: [-1, 0, 0, 0],
      30: [-1, 0, 0, 0],
      31: [-1, 0, 0, 0],
      32: [-1, 0, 0, 0],
      33: [-1, 0, 0, 0],
      34: [-1, 0, 0, 0],
      35: [-1, 0, 0, 0],
      36: [-1, 0, 0, 0],
      37: [-1, 0, 0, 0],
      38: [-1, 0, 0, 0],
      39: [-1, 0, 0, 0],
      40: [-1, 0, 0, 0],
      41: [-1, 0, 0, 0],
      42: [-1, 0, 0, 0],
      43: [-1, 0, 0, 0],
      44: [-1, 0, 0, 0],
      45: [-1, 0, 0, 0],
      46: [-1, 0, 0, 0],
      47: [-1, 0, 0, 0],
      48: [-1, 0, 0, 0],
      49: [-1, 0, 0, 0]}
```

Query the TABFILES dictionary in the SFR2 object to determine the TABFILES data in the SFR2 file

```
[11]: sfr.tabfiles_dict
[11]: {1: {'numval': 50, 'inuit': 55}}

[12]: sfr.tabfiles
[12]: True
```

5.2.14 Unconfined steady-state flow model

This tutorial demonstrates a simple MODFLOW 6 model with FloPy.

Getting Started

```
[1]: from pathlib import Path
     from tempfile import TemporaryDirectory

[2]: import matplotlib.pyplot as plt
     import numpy as np

[3]: import flopy
```

We are creating a square model with a specified head equal to $h1$ along the edge of the model in layer 1. A well is located in the center of the upper-left quadrant in layer 10. First, set the name of the model and the parameters of the model: the number of layers N_{lay} , the number of rows and columns N , lengths of the sides of the model L , aquifer thickness H , hydraulic conductivity k , and the well pumping rate q .

```
[4]: h1 = 100
     Nlay = 10
     N = 101
     L = 400.0
     H = 50.0
     k = 1.0
     q = -1000.0
```

Create the Flopy Model Objects

One big difference between MODFLOW 6 and previous MODFLOW versions is that MODFLOW 6 is based on the concept of a simulation. A simulation consists of the following:

- Temporal discretization (TDIS)
- One or more models
- Zero or more exchanges (instructions for how models are coupled)
- Solutions

For this simple example, the simulation consists of the temporal discretization (TDIS) package, a groundwater flow (GWF) model, and an iterative model solution (IMS), which controls how the GWF model is solved.

Create a temporary workspace, then the FloPy simulation object

```
[5]: temp_dir = TemporaryDirectory()
workspace = Path(temp_dir.name)
name = "tutorial01_mf6"
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws=workspace
)
```

Create the Flopy TDIS object

```
[6]: tdis = flopy.mf6.ModflowTdis(
    sim, pname="tdis", time_units="DAYS", nper=1, perioddata=[(1.0, 1, 1.0)]
)
```

Create the Flopy IMS Package object

```
[7]: ims = flopy.mf6.ModflowIms(
    sim,
    pname="ims",
    complexity="SIMPLE",
    linear_acceleration="BICGSTAB",
)
```

Create the Flopy groundwater flow (gwf) model object

```
[8]: model_nam_file = f"{name}.nam"
gwf = flopy.mf6.ModflowGwf(
    sim,
    modelname=name,
    model_nam_file=model_nam_file,
    save_flows=True,
    newtonoptions="NEWTON UNDER_RELAXATION",
)
```

Now that the overall simulation is set up, we can focus on building the groundwater flow model. The groundwater flow model will be built by adding packages to it that describe the model characteristics.

Create the discretization (DIS) Package

Define the discretization of the model. All layers are given equal thickness. The `bot` array is build from `H` and the `Nlay` values to indicate top and bottom of each layer, and `delrow` and `delcol` are computed from model size `L` and number of cells `N`. Once these are all computed, the Discretization file is built.

```
[9]: bot = np.linspace(-H / Nlay, -H, Nlay)
delrow = delcol = L / (N - 1)
dis = flopy.mf6.ModflowGwfdis(
    gwf,
```

(continues on next page)

(continued from previous page)

```

    nlay=Nlay,
    nrow=N,
    ncol=N,
    delr=delrow,
    delc=delcol,
    top=0.0,
    botm=bot,
)

```

Create the initial conditions (IC) Package

```

[10]: start = h1 * np.ones((Nlay, N, N))
      ic = flopy.mf6.ModflowGwfic(gwf, pname="ic", strt=start)

```

Create the node property flow (NPF) Package

```

[11]: npf = flopy.mf6.ModflowGwfnpf(
      gwf,
      icelltype=1,
      k=k,
)

```

Create the constant head (CHD) Package

List information is created a bit differently for MODFLOW 6 than for other MODFLOW versions. The cellid (layer, row, column, for a regular grid) can be entered as a tuple as the first entry. Remember that these must be zero-based indices!

```

[12]: chd_rec = []
      layer = 0
      for row_col in range(0, N):
          chd_rec.append(((layer, row_col, 0), h1))
          chd_rec.append(((layer, row_col, N - 1), h1))
          if row_col != 0 and row_col != N - 1:
              chd_rec.append(((layer, 0, row_col), h1))
              chd_rec.append(((layer, N - 1, row_col), h1))
      chd = flopy.mf6.ModflowGwfchd(
          gwf,
          stress_period_data=chd_rec,
      )

```

The CHD Package stored the constant heads in a structured array, also called a `numpy.recarray`. We can get a pointer to the recarray for the first stress period (`iper = 0`) as follows.

```

[13]: iper = 0
      ra = chd.stress_period_data.get_data(key=iper)
      ra

```

```
[13]: rec.array([(0, 0, 0), 100.), ((0, 0, 100), 100.), ((0, 1, 0), 100.),
              ((0, 1, 100), 100.), ((0, 0, 1), 100.), ((0, 100, 1), 100.),
              ((0, 2, 0), 100.), ((0, 2, 100), 100.), ((0, 0, 2), 100.),
              ((0, 100, 2), 100.), ((0, 3, 0), 100.), ((0, 3, 100), 100.),
              ((0, 0, 3), 100.), ((0, 100, 3), 100.), ((0, 4, 0), 100.),
              ((0, 4, 100), 100.), ((0, 0, 4), 100.), ((0, 100, 4), 100.),
              ((0, 5, 0), 100.), ((0, 5, 100), 100.), ((0, 0, 5), 100.),
              ((0, 100, 5), 100.), ((0, 6, 0), 100.), ((0, 6, 100), 100.),
              ((0, 0, 6), 100.), ((0, 100, 6), 100.), ((0, 7, 0), 100.),
              ((0, 7, 100), 100.), ((0, 0, 7), 100.), ((0, 100, 7), 100.),
              ((0, 8, 0), 100.), ((0, 8, 100), 100.), ((0, 0, 8), 100.),
              ((0, 100, 8), 100.), ((0, 9, 0), 100.), ((0, 9, 100), 100.),
              ((0, 0, 9), 100.), ((0, 100, 9), 100.), ((0, 10, 0), 100.),
              ((0, 10, 100), 100.), ((0, 0, 10), 100.), ((0, 100, 10), 100.),
              ((0, 11, 0), 100.), ((0, 11, 100), 100.), ((0, 0, 11), 100.),
              ((0, 100, 11), 100.), ((0, 12, 0), 100.), ((0, 12, 100), 100.),
              ((0, 0, 12), 100.), ((0, 100, 12), 100.), ((0, 13, 0), 100.),
              ((0, 13, 100), 100.), ((0, 0, 13), 100.), ((0, 100, 13), 100.),
              ((0, 14, 0), 100.), ((0, 14, 100), 100.), ((0, 0, 14), 100.),
              ((0, 100, 14), 100.), ((0, 15, 0), 100.), ((0, 15, 100), 100.),
              ((0, 0, 15), 100.), ((0, 100, 15), 100.), ((0, 16, 0), 100.),
              ((0, 16, 100), 100.), ((0, 0, 16), 100.), ((0, 100, 16), 100.),
              ((0, 17, 0), 100.), ((0, 17, 100), 100.), ((0, 0, 17), 100.),
              ((0, 100, 17), 100.), ((0, 18, 0), 100.), ((0, 18, 100), 100.),
              ((0, 0, 18), 100.), ((0, 100, 18), 100.), ((0, 19, 0), 100.),
              ((0, 19, 100), 100.), ((0, 0, 19), 100.), ((0, 100, 19), 100.),
              ((0, 20, 0), 100.), ((0, 20, 100), 100.), ((0, 0, 20), 100.),
              ((0, 100, 20), 100.), ((0, 21, 0), 100.), ((0, 21, 100), 100.),
              ((0, 0, 21), 100.), ((0, 100, 21), 100.), ((0, 22, 0), 100.),
              ((0, 22, 100), 100.), ((0, 0, 22), 100.), ((0, 100, 22), 100.),
              ((0, 23, 0), 100.), ((0, 23, 100), 100.), ((0, 0, 23), 100.),
              ((0, 100, 23), 100.), ((0, 24, 0), 100.), ((0, 24, 100), 100.),
              ((0, 0, 24), 100.), ((0, 100, 24), 100.), ((0, 25, 0), 100.),
              ((0, 25, 100), 100.), ((0, 0, 25), 100.), ((0, 100, 25), 100.),
              ((0, 26, 0), 100.), ((0, 26, 100), 100.), ((0, 0, 26), 100.),
              ((0, 100, 26), 100.), ((0, 27, 0), 100.), ((0, 27, 100), 100.),
              ((0, 0, 27), 100.), ((0, 100, 27), 100.), ((0, 28, 0), 100.),
              ((0, 28, 100), 100.), ((0, 0, 28), 100.), ((0, 100, 28), 100.),
              ((0, 29, 0), 100.), ((0, 29, 100), 100.), ((0, 0, 29), 100.),
              ((0, 100, 29), 100.), ((0, 30, 0), 100.), ((0, 30, 100), 100.),
              ((0, 0, 30), 100.), ((0, 100, 30), 100.), ((0, 31, 0), 100.),
              ((0, 31, 100), 100.), ((0, 0, 31), 100.), ((0, 100, 31), 100.),
              ((0, 32, 0), 100.), ((0, 32, 100), 100.), ((0, 0, 32), 100.),
              ((0, 100, 32), 100.), ((0, 33, 0), 100.), ((0, 33, 100), 100.),
              ((0, 0, 33), 100.), ((0, 100, 33), 100.), ((0, 34, 0), 100.),
              ((0, 34, 100), 100.), ((0, 0, 34), 100.), ((0, 100, 34), 100.),
              ((0, 35, 0), 100.), ((0, 35, 100), 100.), ((0, 0, 35), 100.),
              ((0, 100, 35), 100.), ((0, 36, 0), 100.), ((0, 36, 100), 100.),
              ((0, 0, 36), 100.), ((0, 100, 36), 100.), ((0, 37, 0), 100.),
              ((0, 37, 100), 100.), ((0, 0, 37), 100.), ((0, 100, 37), 100.),
              ((0, 38, 0), 100.), ((0, 38, 100), 100.), ((0, 0, 38), 100.),
              ((0, 100, 38), 100.), ((0, 39, 0), 100.), ((0, 39, 100), 100.),
              ((0, 0, 39), 100.), ((0, 100, 39), 100.), ((0, 40, 0), 100.),
```

(continues on next page)

(continued from previous page)

```

((0, 40, 100), 100.), ((0, 0, 40), 100.), ((0, 100, 40), 100.),
((0, 41, 0), 100.), ((0, 41, 100), 100.), ((0, 0, 41), 100.),
((0, 100, 41), 100.), ((0, 42, 0), 100.), ((0, 42, 100), 100.),
((0, 0, 42), 100.), ((0, 100, 42), 100.), ((0, 43, 0), 100.),
((0, 43, 100), 100.), ((0, 0, 43), 100.), ((0, 100, 43), 100.),
((0, 44, 0), 100.), ((0, 44, 100), 100.), ((0, 0, 44), 100.),
((0, 100, 44), 100.), ((0, 45, 0), 100.), ((0, 45, 100), 100.),
((0, 0, 45), 100.), ((0, 100, 45), 100.), ((0, 46, 0), 100.),
((0, 46, 100), 100.), ((0, 0, 46), 100.), ((0, 100, 46), 100.),
((0, 47, 0), 100.), ((0, 47, 100), 100.), ((0, 0, 47), 100.),
((0, 100, 47), 100.), ((0, 48, 0), 100.), ((0, 48, 100), 100.),
((0, 0, 48), 100.), ((0, 100, 48), 100.), ((0, 49, 0), 100.),
((0, 49, 100), 100.), ((0, 0, 49), 100.), ((0, 100, 49), 100.),
((0, 50, 0), 100.), ((0, 50, 100), 100.), ((0, 0, 50), 100.),
((0, 100, 50), 100.), ((0, 51, 0), 100.), ((0, 51, 100), 100.),
((0, 0, 51), 100.), ((0, 100, 51), 100.), ((0, 52, 0), 100.),
((0, 52, 100), 100.), ((0, 0, 52), 100.), ((0, 100, 52), 100.),
((0, 53, 0), 100.), ((0, 53, 100), 100.), ((0, 0, 53), 100.),
((0, 100, 53), 100.), ((0, 54, 0), 100.), ((0, 54, 100), 100.),
((0, 0, 54), 100.), ((0, 100, 54), 100.), ((0, 55, 0), 100.),
((0, 55, 100), 100.), ((0, 0, 55), 100.), ((0, 100, 55), 100.),
((0, 56, 0), 100.), ((0, 56, 100), 100.), ((0, 0, 56), 100.),
((0, 100, 56), 100.), ((0, 57, 0), 100.), ((0, 57, 100), 100.),
((0, 0, 57), 100.), ((0, 100, 57), 100.), ((0, 58, 0), 100.),
((0, 58, 100), 100.), ((0, 0, 58), 100.), ((0, 100, 58), 100.),
((0, 59, 0), 100.), ((0, 59, 100), 100.), ((0, 0, 59), 100.),
((0, 100, 59), 100.), ((0, 60, 0), 100.), ((0, 60, 100), 100.),
((0, 0, 60), 100.), ((0, 100, 60), 100.), ((0, 61, 0), 100.),
((0, 61, 100), 100.), ((0, 0, 61), 100.), ((0, 100, 61), 100.),
((0, 62, 0), 100.), ((0, 62, 100), 100.), ((0, 0, 62), 100.),
((0, 100, 62), 100.), ((0, 63, 0), 100.), ((0, 63, 100), 100.),
((0, 0, 63), 100.), ((0, 100, 63), 100.), ((0, 64, 0), 100.),
((0, 64, 100), 100.), ((0, 0, 64), 100.), ((0, 100, 64), 100.),
((0, 65, 0), 100.), ((0, 65, 100), 100.), ((0, 0, 65), 100.),
((0, 100, 65), 100.), ((0, 66, 0), 100.), ((0, 66, 100), 100.),
((0, 0, 66), 100.), ((0, 100, 66), 100.), ((0, 67, 0), 100.),
((0, 67, 100), 100.), ((0, 0, 67), 100.), ((0, 100, 67), 100.),
((0, 68, 0), 100.), ((0, 68, 100), 100.), ((0, 0, 68), 100.),
((0, 100, 68), 100.), ((0, 69, 0), 100.), ((0, 69, 100), 100.),
((0, 0, 69), 100.), ((0, 100, 69), 100.), ((0, 70, 0), 100.),
((0, 70, 100), 100.), ((0, 0, 70), 100.), ((0, 100, 70), 100.),
((0, 71, 0), 100.), ((0, 71, 100), 100.), ((0, 0, 71), 100.),
((0, 100, 71), 100.), ((0, 72, 0), 100.), ((0, 72, 100), 100.),
((0, 0, 72), 100.), ((0, 100, 72), 100.), ((0, 73, 0), 100.),
((0, 73, 100), 100.), ((0, 0, 73), 100.), ((0, 100, 73), 100.),
((0, 74, 0), 100.), ((0, 74, 100), 100.), ((0, 0, 74), 100.),
((0, 100, 74), 100.), ((0, 75, 0), 100.), ((0, 75, 100), 100.),
((0, 0, 75), 100.), ((0, 100, 75), 100.), ((0, 76, 0), 100.),
((0, 76, 100), 100.), ((0, 0, 76), 100.), ((0, 100, 76), 100.),
((0, 77, 0), 100.), ((0, 77, 100), 100.), ((0, 0, 77), 100.),
((0, 100, 77), 100.), ((0, 78, 0), 100.), ((0, 78, 100), 100.),
((0, 0, 78), 100.), ((0, 100, 78), 100.), ((0, 79, 0), 100.),

```

(continues on next page)

(continued from previous page)

```
((0, 79, 100), 100.), ((0, 0, 79), 100.), ((0, 100, 79), 100.),
((0, 80, 0), 100.), ((0, 80, 100), 100.), ((0, 0, 80), 100.),
((0, 100, 80), 100.), ((0, 81, 0), 100.), ((0, 81, 100), 100.),
((0, 0, 81), 100.), ((0, 100, 81), 100.), ((0, 82, 0), 100.),
((0, 82, 100), 100.), ((0, 0, 82), 100.), ((0, 100, 82), 100.),
((0, 83, 0), 100.), ((0, 83, 100), 100.), ((0, 0, 83), 100.),
((0, 100, 83), 100.), ((0, 84, 0), 100.), ((0, 84, 100), 100.),
((0, 0, 84), 100.), ((0, 100, 84), 100.), ((0, 85, 0), 100.),
((0, 85, 100), 100.), ((0, 0, 85), 100.), ((0, 100, 85), 100.),
((0, 86, 0), 100.), ((0, 86, 100), 100.), ((0, 0, 86), 100.),
((0, 100, 86), 100.), ((0, 87, 0), 100.), ((0, 87, 100), 100.),
((0, 0, 87), 100.), ((0, 100, 87), 100.), ((0, 88, 0), 100.),
((0, 88, 100), 100.), ((0, 0, 88), 100.), ((0, 100, 88), 100.),
((0, 89, 0), 100.), ((0, 89, 100), 100.), ((0, 0, 89), 100.),
((0, 100, 89), 100.), ((0, 90, 0), 100.), ((0, 90, 100), 100.),
((0, 0, 90), 100.), ((0, 100, 90), 100.), ((0, 91, 0), 100.),
((0, 91, 100), 100.), ((0, 0, 91), 100.), ((0, 100, 91), 100.),
((0, 92, 0), 100.), ((0, 92, 100), 100.), ((0, 0, 92), 100.),
((0, 100, 92), 100.), ((0, 93, 0), 100.), ((0, 93, 100), 100.),
((0, 0, 93), 100.), ((0, 100, 93), 100.), ((0, 94, 0), 100.),
((0, 94, 100), 100.), ((0, 0, 94), 100.), ((0, 100, 94), 100.),
((0, 95, 0), 100.), ((0, 95, 100), 100.), ((0, 0, 95), 100.),
((0, 100, 95), 100.), ((0, 96, 0), 100.), ((0, 96, 100), 100.),
((0, 0, 96), 100.), ((0, 100, 96), 100.), ((0, 97, 0), 100.),
((0, 97, 100), 100.), ((0, 0, 97), 100.), ((0, 100, 97), 100.),
((0, 98, 0), 100.), ((0, 98, 100), 100.), ((0, 0, 98), 100.),
((0, 100, 98), 100.), ((0, 99, 0), 100.), ((0, 99, 100), 100.),
((0, 0, 99), 100.), ((0, 100, 99), 100.), ((0, 100, 0), 100.),
((0, 100, 100), 100.)],
dtype=[('cellid', 'O'), ('head', '<f8')])
```

Create the well (WEL) Package

Add a well in model layer 10.

```
[14]: wel_rec = [(Nlay - 1, int(N / 4), int(N / 4), q)]
wel = flopy.mf6.ModflowGwfwel(
    gwf,
    stress_period_data=wel_rec,
)
```


Create the output control (OC) Package

Save heads and budget output to binary files and print heads to the model listing file at the end of the stress period.

```
[15]: headfile = f"{name}.hds"
      head_filerecord = [headfile]
      budgetfile = f"{name}.cbb"
      budget_filerecord = [budgetfile]
      saverecord = [("HEAD", "ALL"), ("BUDGET", "ALL")]
      printrecord = [("HEAD", "LAST")]
      oc = flopy.mf6.ModflowGwfoc(
          gwf,
          saverecord=saverecord,
          head_filerecord=head_filerecord,
          budget_filerecord=budget_filerecord,
          printrecord=printrecord,
      )
```

Create the MODFLOW 6 Input Files and Run the Model

Once all the flopy objects are created, it is very easy to create all of the input files and run the model.

Write the datasets

```
[16]: sim.write_simulation()

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims...
  writing model tutorial01_mf6...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
    writing package chd_0...
  INFORMATION: maxbound in ('gwf6', 'chd', 'dimensions') changed to 400 based on size of
  ↳ stress_period_data
    writing package wel_0...
  INFORMATION: maxbound in ('gwf6', 'wel', 'dimensions') changed to 1 based on size of
  ↳ stress_period_data
    writing package oc...
```

Run the Simulation

We can also run the simulation from python, but only if the MODFLOW 6 executable is available. The executable can be made available by putting the executable in a folder that is listed in the system path variable. Another option is to just put a copy of the executable in the simulation folder, though this should generally be avoided. A final option is to provide a full path to the executable when the simulation is constructed. This would be done by specifying `exe_name` with the full path.

```
[17]: success, buff = sim.run_simulation()
      assert success, "MODFLOW 6 did not terminate normally."
```

```
FloPy is using the following executable to run the model: ../../home/runner/.local/bin/
↳ modflow/mf6
```

```

                                MODFLOW 6
      U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
                                VERSION 6.4.2 06/28/2023
```

```
MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_0000000
```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

```
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:56
```

```
Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam
```

```
      Solving:  Stress period:      1      Time step:      1
```

```
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:38:59
Elapsed run time:  2.593 Seconds
```

```
Normal termination of simulation.
```

Post-Process Head Results

First, a link to the heads file is created with using the `.output.head()` method on the groundwater flow model. The link and the `get_data` method are used with the step number and period number for which we want to retrieve data. A three-dimensional array is returned of size `nlay`, `nrow`, `ncol`. Matplotlib contouring functions are used to make contours of the layers or a cross-section.

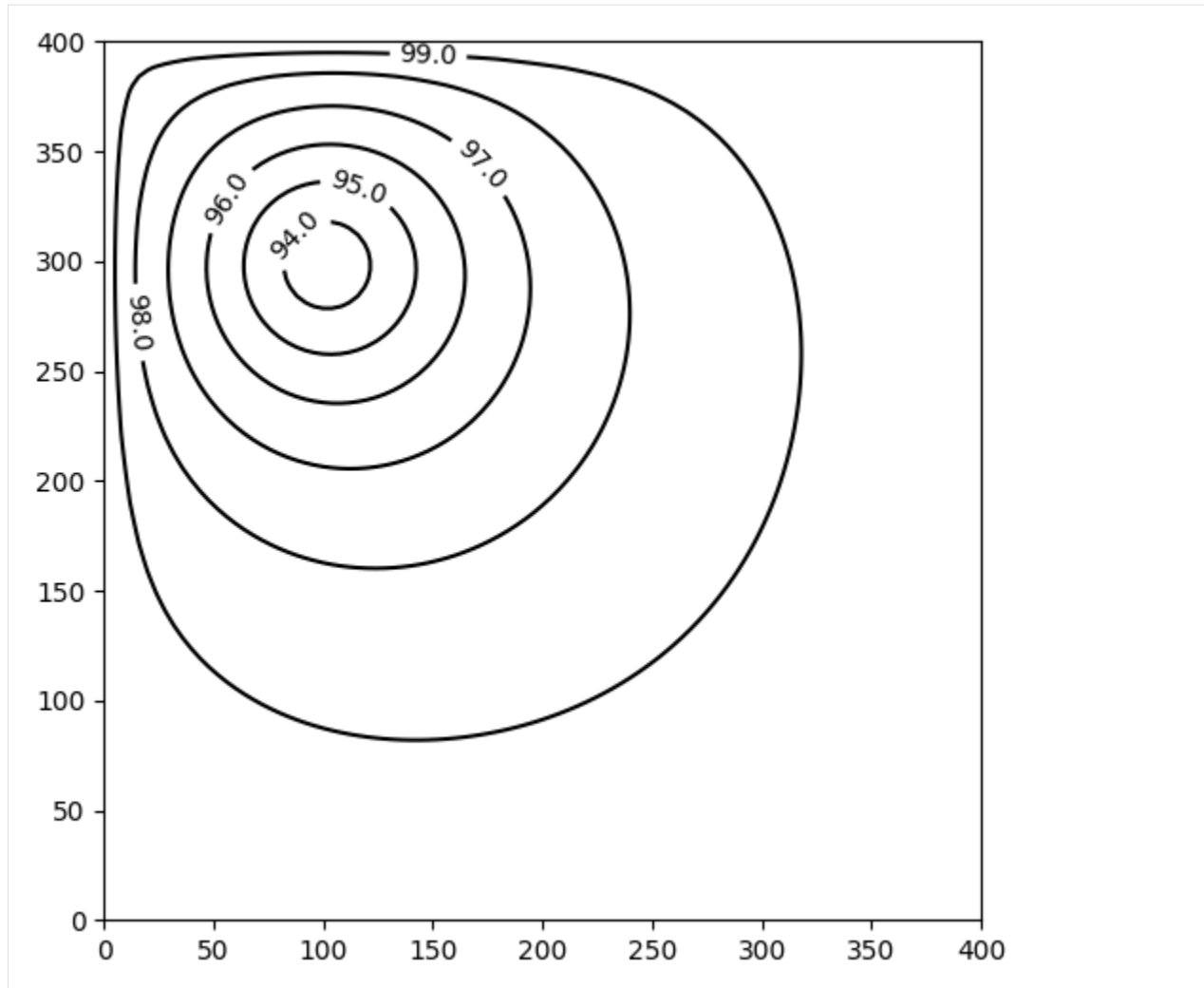
Read the binary head file and plot the results. We can use the Flopy `.output.head()` method on the groundwater flow model object (`gwf`). Also set a few variables that will be used for plotting.

```
[18]: h = gwf.output.head().get_data(kstpkper=(0, 0))
      x = y = np.linspace(0, L, N)
      y = y[::-1]
      vmin, vmax = 90.0, 100.0
      contour_intervals = np.arange(90, 100.1, 1.0)
```

Plot a Map of Layer 1

```
[19]: fig = plt.figure(figsize=(6, 6))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      c = ax.contour(x, y, h[0], contour_intervals, colors="black")
      plt.clabel(c, fmt="%2.1f")

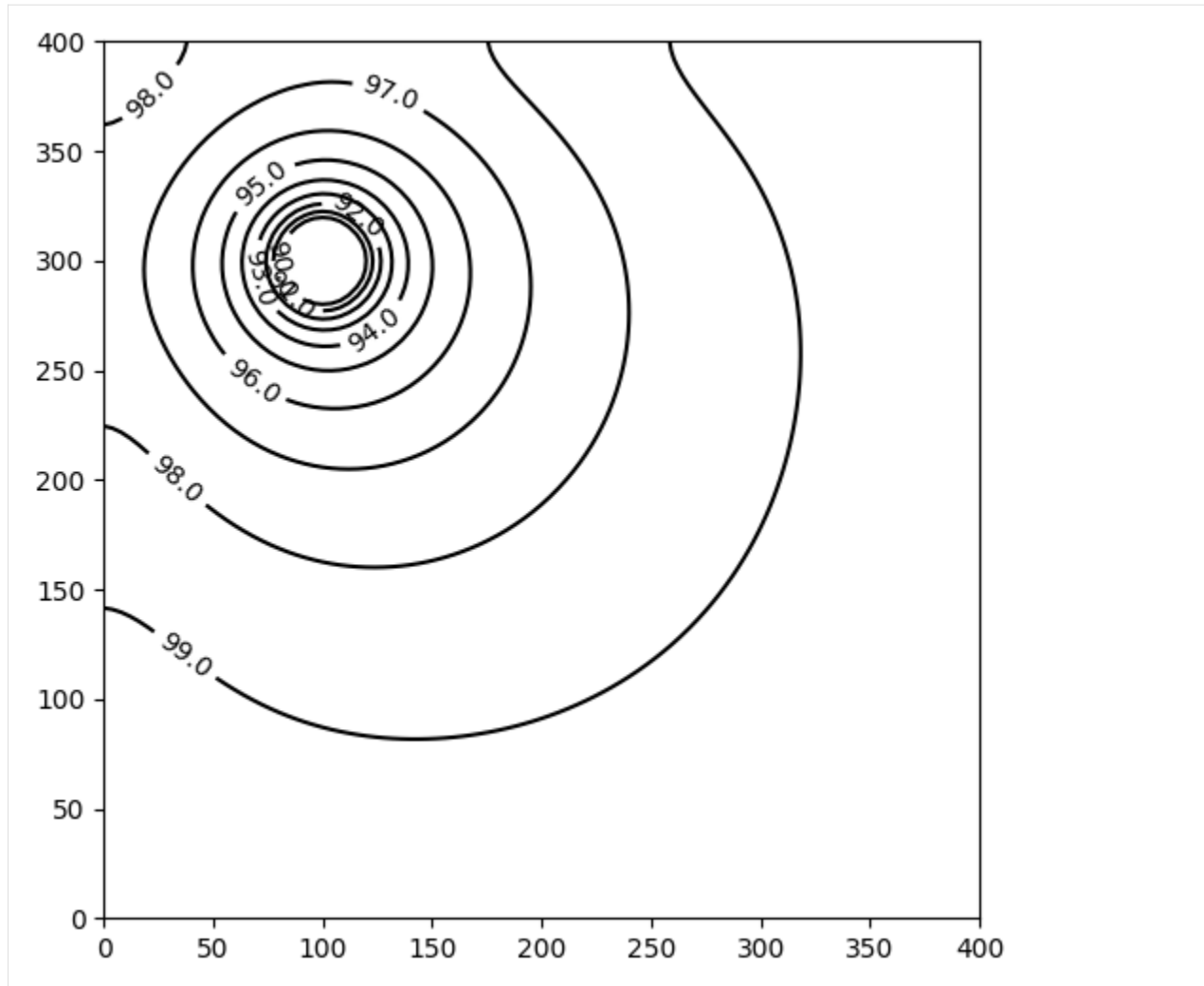
[19]: <a list of 6 text.Text objects>
```



Plot a Map of Layer 10

```
[20]: x = y = np.linspace(0, L, N)
      y = y[::-1]
      fig = plt.figure(figsize=(6, 6))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      c = ax.contour(x, y, h[-1], contour_intervals, colors="black")
      plt.clabel(c, fmt="%1.1f")
```

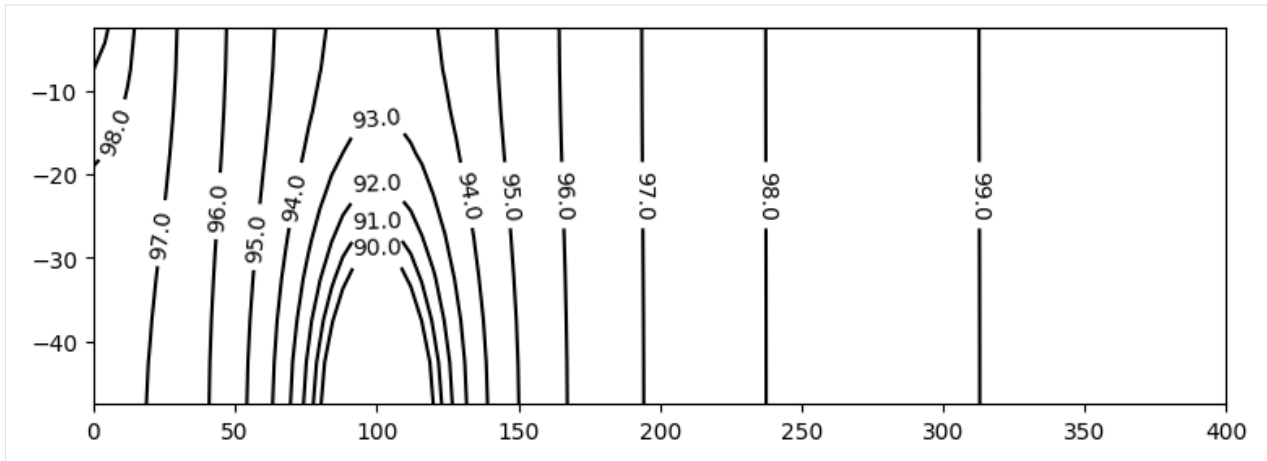
```
[20]: <a list of 11 text.Text objects>
```



Plot a Cross-section along row 25

```
[21]: z = np.linspace(-H / Nlay / 2, -H + H / Nlay / 2, Nlay)
fig = plt.figure(figsize=(9, 3))
ax = fig.add_subplot(1, 1, 1, aspect="auto")
c = ax.contour(x, z, h[:, int(N / 4), :], contour_intervals, colors="black")
plt.clabel(c, fmt="%1.1f")
```

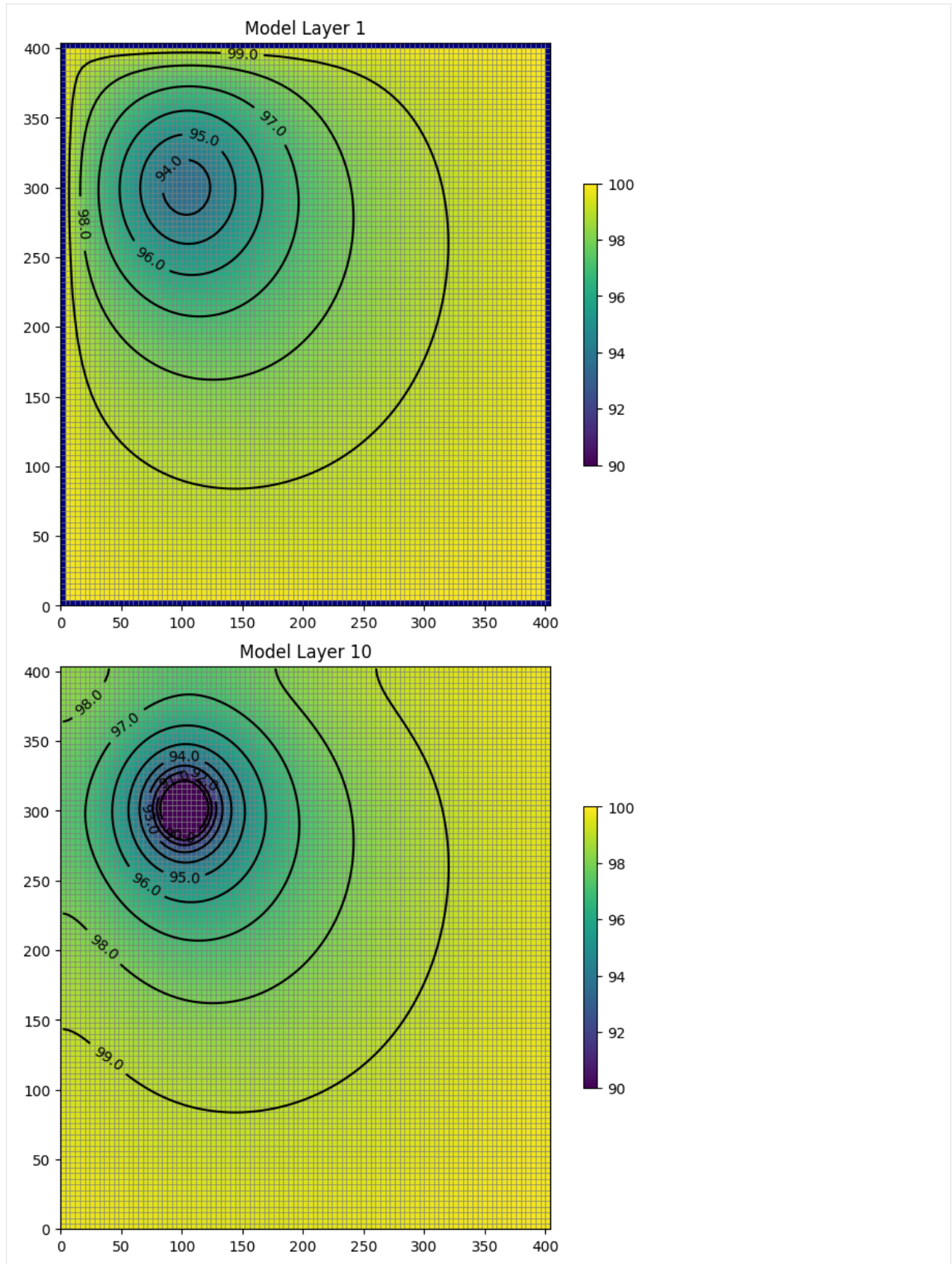
```
[21]: <a list of 15 text.Text objects>
```



Use the FloPy PlotMapView() capabilities for MODFLOW 6

Plot a Map of heads in Layers 1 and 10

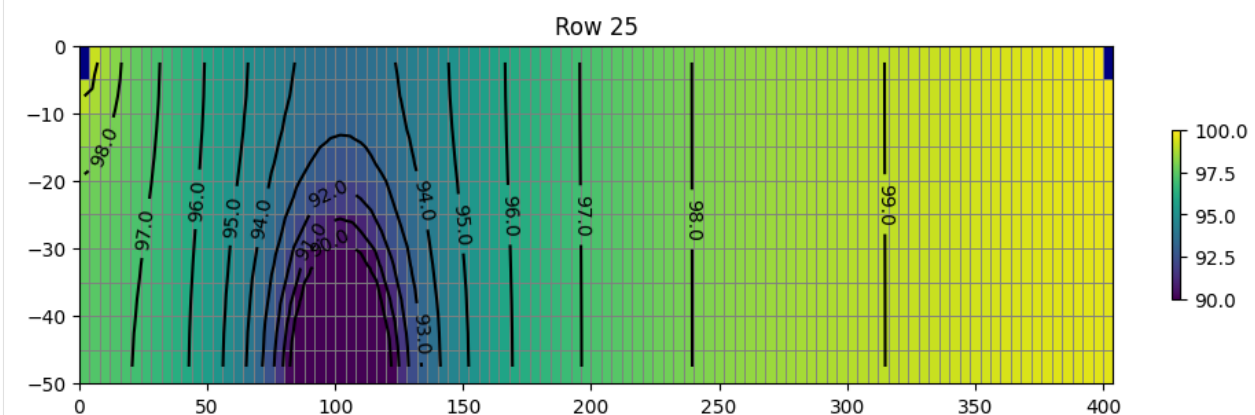
```
[22]: fig, axes = plt.subplots(2, 1, figsize=(6, 12), constrained_layout=True)
# first subplot
ax = axes[0]
ax.set_title("Model Layer 1")
modelmap = flopy.plot.PlotMapView(model=gwf, ax=ax)
pa = modelmap.plot_array(h, vmin=vmin, vmax=vmax)
quadmesh = modelmap.plot_bc("CHD")
linecollection = modelmap.plot_grid(lw=0.5, color="0.5")
contours = modelmap.contour_array(
    h,
    levels=contour_intervals,
    colors="black",
)
ax.clabel(contours, fmt="%2.1f")
cb = plt.colorbar(pa, shrink=0.5, ax=ax)
# second subplot
ax = axes[1]
ax.set_title(f"Model Layer {Nlay}")
modelmap = flopy.plot.PlotMapView(model=gwf, ax=ax, layer=Nlay - 1)
linecollection = modelmap.plot_grid(lw=0.5, color="0.5")
pa = modelmap.plot_array(h, vmin=vmin, vmax=vmax)
quadmesh = modelmap.plot_bc("CHD")
contours = modelmap.contour_array(
    h,
    levels=contour_intervals,
    colors="black",
)
ax.clabel(contours, fmt="%2.1f")
cb = plt.colorbar(pa, shrink=0.5, ax=ax)
```



Use the FloPy PlotCrossSection() capabilities for MODFLOW 6

Plot a cross-section of heads along row 25

```
[23]: fig, ax = plt.subplots(1, 1, figsize=(9, 3), constrained_layout=True)
# first subplot
ax.set_title("Row 25")
modelmap = flopy.plot.PlotCrossSection(
    model=gwf,
    ax=ax,
    line={"row": int(N / 4)},
)
pa = modelmap.plot_array(h, vmin=vmin, vmax=vmax)
quadmesh = modelmap.plot_bc("CHD")
linecollection = modelmap.plot_grid(lw=0.5, color="0.5")
contours = modelmap.contour_array(
    h,
    levels=contour_intervals,
    colors="black",
)
ax.clabel(contours, fmt="%2.1f")
cb = plt.colorbar(pa, shrink=0.5, ax=ax)
```



Determine the Flow Residual

The FLOW-JA-FACE cell-by-cell budget data can be processed to determine the flow residual for each cell in a MODFLOW 6 model. The diagonal position for each row in the FLOW-JA-FACE cell-by-cell budget data contains the flow residual for each cell and can be extracted using the `flopy.mf6.utils.get_residuals()` function.

First extract the FLOW-JA-FACE array from the cell-by-cell budget file

```
[24]: flowja = gwf.oc.output.budget().get_data(text="FLOW-JA-FACE", kstpker=(0, 0))[
    0
]
```

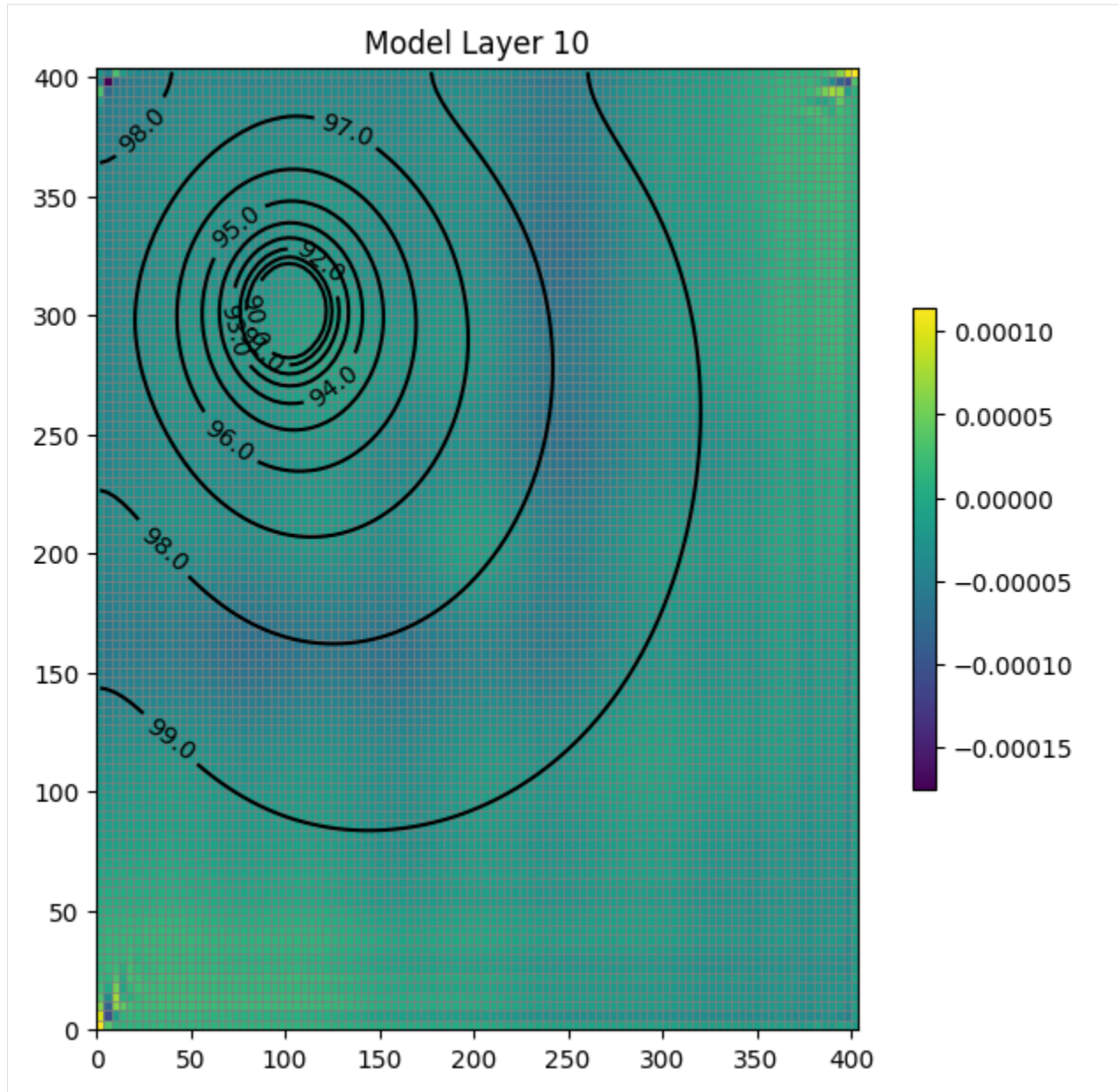
Next extract the flow residual. The MODFLOW 6 binary grid file is passed into the function because it contains the `ia` array that defines the location of the diagonal position in the FLOW-JA-FACE array.


```
[25]: grb_file = workspace / f"{name}.dis.grb"
      residual = flopy.mf6.utils.get_residuals(flowja, grb_file=grb_file)
```

Plot a Map of the flow error in Layer 10

```
[26]: fig, ax = plt.subplots(1, 1, figsize=(6, 6), constrained_layout=True)
      ax.set_title("Model Layer 10")
      modelmap = flopy.plot.PlotMapView(model=gwf, ax=ax, layer=Nlay - 1)
      pa = modelmap.plot_array(residual)
      quadmesh = modelmap.plot_bc("CHD")
      linecollection = modelmap.plot_grid(lw=0.5, color="0.5")
      contours = modelmap.contour_array(
          h,
          levels=contour_intervals,
          colors="black",
      )
      ax.clabel(contours, fmt="%2.1f")
      plt.colorbar(pa, shrink=0.5)
```

```
[26]: <matplotlib.colorbar.Colorbar at 0x7f4d757e38b0>
```



Clean up the temporary directory

```
[27]: try:
      temp_dir.cleanup()
except PermissionError:
    # can occur on windows: https://docs.python.org/3/library/tempfile.html#tempfile.
    TemporaryDirectory
    pass
```

5.3 MODFLOW-2005

5.3.1 MODFLOW-2005 Basic stress packages

FloPy has a new way to enter boundary conditions for some MODFLOW packages. These changes are substantial. Boundary conditions can now be entered as a list of boundaries, as a numpy recarray, or as a dictionary. These different styles are described in this notebook.

FloPy also now requires zero-based input. This means that **all boundaries are entered in zero-based layer, row, and column indices**. This means that older FloPy scripts will need to be modified to account for this change. If you are familiar with Python, this should be natural, but if not, then it may take some time to get used to zero-based numbering. FloPy users submit all information in zero-based form, and FloPy converts this to the one-based form required by MODFLOW.

The following MODFLOW-2005 packages are affected by this change:

- Well
- Drain
- River
- General-Head Boundary
- Time-Variant Constant Head

This notebook explains the different ways to enter these types of boundary conditions.

```
[1]: # begin by importing flopy
import os
import sys
from tempfile import TemporaryDirectory

import numpy as np

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

# temporary directory
temp_dir = TemporaryDirectory()
workspace = os.path.join(temp_dir.name)

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
flopy version: 3.4.3
```

List of Boundaries

Boundary condition information is passed to a package constructor as `stress_period_data`. In its simplest form, `stress_period_data` can be a list of individual boundaries, which themselves are lists. The following shows a simple example for a MODFLOW River Package boundary:

```
[2]: stress_period_data = [
    [
        2,
        3,
        4,
        10.7,
        5000.0,
        -5.7,
    ], # layer, row, column, stage, conductance, river bottom
    [
        2,
        3,
        5,
        10.7,
        5000.0,
        -5.7,
    ], # layer, row, column, stage, conductance, river bottom
    [
        2,
        3,
        6,
        10.7,
        5000.0,
        -5.7,
    ], # layer, row, column, stage, conductance, river bottom
]
m = flopy.modflow.Modflow(modelname="test", model_ws=workspace)
riv = flopy.modflow.ModflowRiv(m, stress_period_data=stress_period_data)
m.write_input()
```

If we look at the River Package created here, you see that the layer, row, and column numbers have been increased by one.

```
[3]: !head -n 10 '../examples/data/test.riv'
head: cannot open '../examples/data/test.riv' for reading: No such file or directory
```

If this model had more than one stress period, then FloPy will assume that this boundary condition information applies until the end of the simulation

```
[4]: m = flopy.modflow.Modflow(modelname="test", model_ws=workspace)
dis = flopy.modflow.ModflowDis(m, nper=3)
riv = flopy.modflow.ModflowRiv(m, stress_period_data=stress_period_data)
m.write_input()
!head -n 10 '../examples/data/test.riv'
head: cannot open '../examples/data/test.riv' for reading: No such file or directory
```

Recarray of Boundaries

Numpy allows the use of recarrays, which are numpy arrays in which each column of the array may be given a different type. Boundary conditions can be entered as recarrays. Information on the structure of the recarray for a boundary condition package can be obtained from that particular package. The structure of the recarray is contained in the dtype.

```
[5]: riv_dtype = flopy.modflow.ModflowRiv.get_default_dtype()
print(riv_dtype)

[('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('stage', '<f4'), ('cond', '<f4'), ('rbot', '
↪<f4')]
```

Now that we know the structure of the recarray that we want to create, we can create a new one as follows.

```
[6]: stress_period_data = np.zeros((3), dtype=riv_dtype)
stress_period_data = stress_period_data.view(np.recarray)
print("stress_period_data: ", stress_period_data)
print("type is: ", type(stress_period_data))

stress_period_data: [(0, 0, 0, 0., 0., 0.) (0, 0, 0, 0., 0., 0.) (0, 0, 0, 0., 0., 0.)]
type is: <class 'numpy.recarray'>
```

We can then fill the recarray with our boundary conditions.

```
[7]: stress_period_data[0] = (2, 3, 4, 10.7, 5000.0, -5.7)
stress_period_data[1] = (2, 3, 5, 10.7, 5000.0, -5.7)
stress_period_data[2] = (2, 3, 6, 10.7, 5000.0, -5.7)
print(stress_period_data)

[(2, 3, 4, 10.7, 5000., -5.7) (2, 3, 5, 10.7, 5000., -5.7)
 (2, 3, 6, 10.7, 5000., -5.7)]
```

```
[8]: m = flopy.modflow.Modflow(modelname="test", model_ws=workspace)
riv = flopy.modflow.ModflowRiv(m, stress_period_data=stress_period_data)
m.write_input()
!head -n 10 '../examples/data/test.riv'

head: cannot open '../examples/data/test.riv' for reading: No such file or directory
```

As before, if we have multiple stress periods, then this recarray will apply to all of them.

```
[9]: m = flopy.modflow.Modflow(modelname="test", model_ws=workspace)
dis = flopy.modflow.ModflowDis(m, nper=3)
riv = flopy.modflow.ModflowRiv(m, stress_period_data=stress_period_data)
m.write_input()
!head -n 10 '../examples/data/test.riv'

head: cannot open '../examples/data/test.riv' for reading: No such file or directory
```

Dictionary of Boundaries

The power of the new functionality in FloPy3 is the ability to specify a dictionary for `stress_period_data`. If specified as a dictionary, the key is the stress period number (**as a zero-based number**), and the value is either a nested list, an integer value of 0 or -1, or a recarray for that stress period.

Let's say that we want to use the following schedule for our rivers: 0. No rivers in stress period zero 1. Rivers specified by a list in stress period 1 2. No rivers 3. No rivers 4. No rivers 5. Rivers specified by a recarray 6. Same recarray rivers 7. Same recarray rivers 8. Same recarray rivers

```
[10]: sp1 = [
    [
        2,
        3,
        4,
        10.7,
        5000.0,
        -5.7,
    ], # layer, row, column, stage, conductance, river bottom
    [
        2,
        3,
        5,
        10.7,
        5000.0,
        -5.7,
    ], # layer, row, column, stage, conductance, river bottom
    [
        2,
        3,
        6,
        10.7,
        5000.0,
        -5.7,
    ], # layer, row, column, stage, conductance, river bottom
]
print(sp1)

[[2, 3, 4, 10.7, 5000.0, -5.7], [2, 3, 5, 10.7, 5000.0, -5.7], [2, 3, 6, 10.7, 5000.0, -
↪5.7]]
```

```
[11]: riv_dtype = flopy.modflow.ModflowRiv.get_default_dtype()
sp5 = np.zeros((3), dtype=riv_dtype)
sp5 = sp5.view(np.recarray)
sp5[0] = (2, 3, 4, 20.7, 5000.0, -5.7)
sp5[1] = (2, 3, 5, 20.7, 5000.0, -5.7)
sp5[2] = (2, 3, 6, 20.7, 5000.0, -5.7)
print(sp5)

[(2, 3, 4, 20.7, 5000., -5.7) (2, 3, 5, 20.7, 5000., -5.7)
 (2, 3, 6, 20.7, 5000., -5.7)]
```

```
[12]: sp_dict = {0: 0, 1: sp1, 2: 0, 5: sp5}
m = flopy.modflow.Modflow(modelname="test", model_ws=workspace)
```

(continues on next page)

(continued from previous page)

```
dis = flopy.modflow.ModflowDis(m, nper=8)
riv = flopy.modflow.ModflowRiv(m, stress_period_data=sp_dict)
m.write_input()
!head -n 10 '../examples/data/test.riv'

head: cannot open '../examples/data/test.riv' for reading: No such file or directory
```

MODFLOW Auxiliary Variables

FloPy works with MODFLOW auxiliary variables by allowing the recarray to contain additional columns of information. The auxiliary variables must be specified as package options as shown in the example below.

In this example, we also add a string in the last column of the list in order to name each boundary condition. In this case, however, we do not include boundname as an auxiliary variable as MODFLOW would try to read it as a floating point number.

```
[13]: # create an empty array with an iface auxiliary variable at the end
riva_dtype = [
    ("k", "<i8"),
    ("i", "<i8"),
    ("j", "<i8"),
    ("stage", "<f4"),
    ("cond", "<f4"),
    ("rbot", "<f4"),
    ("iface", "<i4"),
    ("boundname", object),
]
riva_dtype = np.dtype(riva_dtype)
stress_period_data = np.zeros((3), dtype=riva_dtype)
stress_period_data = stress_period_data.view(np.recarray)
print("stress_period_data: ", stress_period_data)
print("type is: ", type(stress_period_data))

stress_period_data: [(0, 0, 0, 0., 0., 0., 0, 0) (0, 0, 0, 0., 0., 0., 0, 0)
 (0, 0, 0, 0., 0., 0., 0, 0)]
type is: <class 'numpy.recarray'>
```

```
[14]: stress_period_data[0] = (2, 3, 4, 10.7, 5000.0, -5.7, 1, "riv1")
stress_period_data[1] = (2, 3, 5, 10.7, 5000.0, -5.7, 2, "riv2")
stress_period_data[2] = (2, 3, 6, 10.7, 5000.0, -5.7, 3, "riv3")
print(stress_period_data)

[(2, 3, 4, 10.7, 5000., -5.7, 1, 'riv1')
 (2, 3, 5, 10.7, 5000., -5.7, 2, 'riv2')
 (2, 3, 6, 10.7, 5000., -5.7, 3, 'riv3')]
```

```
[15]: m = flopy.modflow.Modflow(modelname="test", model_ws=workspace)
riv = flopy.modflow.ModflowRiv(
    m,
    stress_period_data=stress_period_data,
    dtype=riva_dtype,
    options=["aux iface"],
```

(continues on next page)

(continued from previous page)

```
)
m.write_input()
!head -n 10 '../examples/data/test.riv'

head: cannot open '../examples/data/test.riv' for reading: No such file or directory
```

Working with Unstructured Grids

FloPy can create an unstructured grid boundary condition package for MODFLOW-USG. This can be done by specifying a custom dtype for the recarray. The following shows an example of how that can be done.

```
[16]: # create an empty array based on nodenumber instead of layer, row, and column
rivu_dtype = [
    ("nodenumber", "<i8"),
    ("stage", "<f4"),
    ("cond", "<f4"),
    ("rbot", "<f4"),
]
rivu_dtype = np.dtype(rivu_dtype)
stress_period_data = np.zeros((3), dtype=rivu_dtype)
stress_period_data = stress_period_data.view(np.recarray)
print("stress_period_data: ", stress_period_data)
print("type is: ", type(stress_period_data))

stress_period_data: [(0, 0., 0., 0.) (0, 0., 0., 0.) (0, 0., 0., 0.)]
type is: <class 'numpy.recarray'>
```

```
[17]: stress_period_data[0] = (77, 10.7, 5000.0, -5.7)
stress_period_data[1] = (245, 10.7, 5000.0, -5.7)
stress_period_data[2] = (450034, 10.7, 5000.0, -5.7)
print(stress_period_data)

[( 77, 10.7, 5000., -5.7) ( 245, 10.7, 5000., -5.7)
 (450034, 10.7, 5000., -5.7)]
```

```
[18]: m = flopy.modflow.Modflow(modelname="test", model_ws=workspace)
riv = flopy.modflow.ModflowRiv(
    m, stress_period_data=stress_period_data, dtype=rivu_dtype
)
m.write_input()
!head -n 10 '../examples/data/test.riv'

head: cannot open '../examples/data/test.riv' for reading: No such file or directory
```


Combining two boundary condition packages

```
[19]: ml = flopy.modflow.Modflow(modelname="test", model_ws=workspace)
dis = flopy.modflow.ModflowDis(ml, 10, 10, 10, 10)
sp_data1 = {3: [1, 1, 1, 1.0], 5: [1, 2, 4, 4.0]}
wel1 = flopy.modflow.ModflowWel(ml, stress_period_data=sp_data1)
ml.write_input()
!head -n 10 '../examples/data/test.wel'

head: cannot open '../examples/data/test.wel' for reading: No such file or directory
```

```
[20]: sp_data2 = {0: [1, 1, 3, 3.0], 8: [9, 2, 4, 4.0]}
wel2 = flopy.modflow.ModflowWel(ml, stress_period_data=sp_data2)
ml.write_input()
!head -n 10 '../examples/data/test.wel'

head: cannot open '../examples/data/test.wel' for reading: No such file or directory

/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/mbase.py:652:
↳ UserWarning: Unit 20 of package WEL already in use.
    warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/mbase.py:661:
↳ UserWarning: Two packages of the same type, Replacing existing 'WEL' package.
    warn(
```

Now we create a third wel package, using the `MfList.append()` method:

```
[21]: wel3 = flopy.modflow.ModflowWel(
        ml,
        stress_period_data=wel2.stress_period_data.append(wel1.stress_period_data),
    )
ml.write_input()
!head -n 10 '../examples/data/test.wel'

head: cannot open '../examples/data/test.wel' for reading: No such file or directory
```

5.3.2 Error checking for MODFLOW-2005 models

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("flopy version: {}".format(flopy.__version__))
```

```
3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
flopy version: 3.4.3
```

Set the working directory

```
[2]: path = os.path.join("../", "../", "examples", "data", "mf2005_test")
```

Load example dataset and change the model work space

```
[3]: m = flopy.modflow.Modflow.load("test1ss.nam", model_ws=path)
temp_dir = TemporaryDirectory()
workspace = temp_dir.name
m.change_model_ws(workspace)
```

By default, the checker performs a model-level check when a set of model files are loaded, unless load is called with `check=False`. The load check only produces screen output if load is called with `verbose=True`. Checks are also performed at the package level when an individual package is loaded

The check() method

Each model and each package has a `check()` method. The check method has three arguments:

```
[4]: help(m.check)
```

Help on method check in module flopy.mbase:

```
check(f: Union[str, os.PathLike, NoneType] = None, verbose=True, level=1) method of
↳ flopy.modflow.mf.Modflow instance
```

Check model data for common errors.

Parameters

`f` : str or PathLike, optional, default None

String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If `f` is None, check method does not write results to a summary file. (default is None)

`verbose` : bool

Boolean flag used to determine if check method results are written to the screen

`level` : int

Check method analysis level. If `level=0`, summary checks are performed. If `level=1`, full checks are performed.

Returns

None

Examples

(continues on next page)

(continued from previous page)

```

-----

>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.check()

```

The check class

By default, check is called at the model level without a summary output file, but with `verbose=True` and `level=1`. The check methods return an instance of the **check** class, which is housed with the flopy utilities.

```
[5]: chk = m.check()
```

```

test1ss MODEL DATA VALIDATION SUMMARY:
  3 Warnings:
    OC package: action(s) defined in OC stress_period_data ignored as they are not part
    ↳ the stress periods defined by DIS
    RCH package: Mean R/T ratio < checker warning threshold of 2e-08 for 1 stress periods
    RCH package: Variable NRCHOP set to value other than 3

Checks that passed:
  Unit number conflicts
  Compatible solver package
  DIS package: zero or negative thickness
  DIS package: thin cells (less than checker threshold of 1.0)
  DIS package: nan values in top array
  DIS package: nan values in bottom array
  BAS6 package: isolated cells in ibound array
  BAS6 package: Not a number
  LPF package: zero or negative horizontal hydraulic conductivity values
  LPF package: negative horizontal anisotropy values
  LPF package: vertical hydraulic conductivity values below checker threshold of 1e-11
  LPF package: vertical hydraulic conductivity values above checker threshold of
    ↳ 1000000.0
  LPF package: horizontal hydraulic conductivity values below checker threshold of 1e-
    ↳ 11
  LPF package: horizontal hydraulic conductivity values above checker threshold of
    ↳ 1000000.0
  GHB package: BC indices valid
  GHB package: not a number (Nan) entries
  GHB package: BC in inactive cells
  GHB package: BC elevation below cell bottom

```

Summary array

Most of the attributes and methods in **check** are intended to be used by the `check()` methods. The central attribute of **check** is the summary array:

```
[6]: chk.summary_array
[6]: rec.array([('Warning', 'OC', 0, 0, 0, 0.00000000e+00, '\r    OC package: action(s)\n
↳ defined in OC stress_period_data ignored as they are not part the stress periods\n
↳ defined by DIS'),
              ('Warning', 'RCH', 0, 0, 0, 1.1785914e-09, '\r    RCH package: Mean R/T ratio
↳ < checker warning threshold of 2e-08 for 1 stress periods'),
              ('Warning', 'RCH', 0, 0, 0, 1.00000000e+00, '\r    RCH package: Variable\n
↳ NRCHOP set to value other than 3')],
              dtype=[('type', 'O'), ('package', 'O'), ('k', '<i8'), ('i', '<i8'), ('j', '<i8
↳ '), ('value', '<f8'), ('desc', 'O')])
```

This is a numpy record array that summarizes errors and warnings found by the checker. The package, layer-row-column location of the error, the offending value, and a description of the error are provided. In the checker, errors and warnings are loosely defined as follows: ##### Errors:

either input that would cause MODFLOW to crash, or inputs that almost certainly mis-represent the intended conceptual model.

Warnings:

inputs that are potentially problematic, but may be intentional.

each package-level check produces a **check** instance with a summary array. The model level checks combine the summary arrays from the packages into a master summary array. At the model and the package levels, the summary array is used to generate the screen output shown above. At either level, the summary array can be written to a csv file by supply a filename to the `f` argument. Specifying `level=2` prints the summary array to the screen.

```
[7]: m.check(level=2)

Errors and/or Warnings encountered.
Errors and/or Warnings encountered.

test1ss MODEL DATA VALIDATION SUMMARY:
  3 Warnings:
    OC package: action(s) defined in OC stress_period_data ignored as they are not part\
↳ the stress periods defined by DIS
    RCH package: Mean R/T ratio < checker warning threshold of 2e-08 for 1 stress periods
    RCH package: Variable NRCHOP set to value other than 3

Checks that passed:
  Unit number conflicts
  Compatible solver package
  DIS package: zero or negative thickness
  DIS package: thin cells (less than checker threshold of 1.0)
  DIS package: nan values in top array
  DIS package: nan values in bottom array
  BAS6 package: isolated cells in ibound array
  BAS6 package: Not a number
  LPF package: zero or negative horizontal hydraulic conductivity values
```

(continues on next page)

(continued from previous page)

```

    LPF package: negative horizontal anisotropy values
    LPF package: vertical hydraulic conductivity values below checker threshold of 1e-11
    LPF package: vertical hydraulic conductivity values above checker threshold of 1000000.0
    LPF package: horizontal hydraulic conductivity values below checker threshold of 1e-11
    LPF package: horizontal hydraulic conductivity values above checker threshold of 1000000.0
    GHB package: BC indices valid
    GHB package: not a number (Nan) entries
    GHB package: BC in inactive cells
    GHB package: BC elevation below cell bottom

DETAILED SUMMARY:
type    package k      i      j      value  desc
Warning OC      0      0      0      0.00e+00  OC package: action(s) defined in OC stress_period_data ignored as they are not part the stress periods defined by DIS
Warning RCH     0      0      0      1.18e-09  RCH package: Mean R/T ratio < checker warning threshold of 2e-08 for 1 stress periods
Warning RCH     0      0      0      1.00e+00  RCH package: Variable NRCHOP set to value other than 3

```

```
[7]: <flopy.utils.check.check at 0x7f96478c24c0>
```

example of package level check and summary file

```
[8]: m.rch.check()
```

```

RCH PACKAGE DATA VALIDATION:
  2 Warnings:
    Mean R/T ratio < checker warning threshold of 2e-08 for 1 stress periods
    Variable NRCHOP set to value other than 3

```

```
[8]: <flopy.utils.check.check at 0x7f96704e3700>
```

example of summary output file

```
[9]: m.check(f=os.path.join(workspace, "checksummary.csv"))
```

```

test1ss MODEL DATA VALIDATION SUMMARY:
  3 Warnings:
    OC package: action(s) defined in OC stress_period_data ignored as they are not part the stress periods defined by DIS
    RCH package: Mean R/T ratio < checker warning threshold of 2e-08 for 1 stress periods
    RCH package: Variable NRCHOP set to value other than 3
  see ../../../../tmp/tmp12evcy_r/checksummary.csv for details.

Checks that passed:

```

(continues on next page)

(continued from previous page)

```

Unit number conflicts
Compatible solver package
DIS package: zero or negative thickness
DIS package: thin cells (less than checker threshold of 1.0)
DIS package: nan values in top array
DIS package: nan values in bottom array
BAS6 package: isolated cells in ibound array
BAS6 package: Not a number
LPF package: zero or negative horizontal hydraulic conductivity values
LPF package: negative horizontal anisotropy values
LPF package: vertical hydraulic conductivity values below checker threshold of 1e-11
LPF package: vertical hydraulic conductivity values above checker threshold of ↵
↵1000000.0
LPF package: horizontal hydraulic conductivity values below checker threshold of 1e-
↵11
LPF package: horizontal hydraulic conductivity values above checker threshold of ↵
↵1000000.0
GHB package: BC indices valid
GHB package: not a number (Nan) entries
GHB package: BC in inactive cells
GHB package: BC elevation below cell bottom

```

```
[9]: <flopy.utils.check.check at 0x7f967055da00>
```

```
[10]: summary_pth = os.path.join(workspace, "checksummary.csv")
```

```

try:
    import pandas as pd

    df = pd.read_csv(summary_pth)
except:
    df = open(summary_pth).readlines()
df

```

```

[10]:
      type package  k  i  j  value  \
0  Warning      OC  0  0  0    0.0
1  Warning      RCH  0  0  0    0.0
2  Warning      RCH  0  0  0    1.0

                                desc
0  action(s) defined in OC stress_period_data ign...
1  Mean R/T ratio < checker warning threshold of ...
2      Variable NRCHOP set to value other than 3

```

checking on write_input()

checking is also performed by default when `write_input()` is called at the package or model level. Checking on write is performed with the same `verbose` setting as specified for the model. However, if errors or warnings are encountered and `level=1` (default) or higher, a screen message notifies the user of the errors.

By default, the checks performed on `load()` and `write_input()` save results to a summary file, which is named after the package or the model.

```
[11]: m.write_input()
```

5.3.3 Loading MODFLOW-2005 models

This tutorial demonstrates how to load models from disk.

First import flopy.

```
[1]: import os
import sys

try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
flopy version: 3.4.3
```

The load() method

To load a MODFLOW 2005 model, use the `Modflow.load()` method. The method's first argument is the path or name of the model namefile. Other parameters include:

- `model_ws`: the model workspace
- `verbose`: whether to write diagnostic information useful for troubleshooting
- `check`: whether to check for model configuration errors

```
[2]: model_ws = os.path.join("..", "..", "examples", "data", "mf2005_test")
ml = flopy.modflow.Modflow.load(
    "bcf2ss.nam",
    model_ws=model_ws,
    verbose=True,
    version="mf2005",
    check=False,
)
```

```
Creating new model with name: bcf2ss
```

```
-----
Parsing the namefile --> /home/runner/work/flopy/flopy/.docs/Notebooks/../../examples/
↳ data/mf2005_test/bcf2ss.nam
could not set filehandle to bcf2ss.lst
```

```
-----
External unit dictionary:
```

```
{7: filename:/home/runner/work/flopy/flopy/.docs/Notebooks/../../examples/data/mf2005_
↳ test/bcf2ss.lst, filetype:LIST, 8: filename:/home/runner/work/flopy/flopy/.docs/
↳ Notebooks/../../examples/data/mf2005_test/bcf2ss.ba6, filetype:BAS6, 11: filename:/
↳ home/runner/work/flopy/flopy/.docs/Notebooks/../../examples/data/mf2005_test/bcf2ss.
↳ bc6, filetype:BCF6, 12: filename:/home/runner/work/flopy/flopy/.docs/Notebooks/../../
↳ examples/data/mf2005_test/bcf2ss.wel, filetype:WEL, 14: filename:/home/runner/work/
↳ flopy/flopy/.docs/Notebooks/../../examples/data/mf2005_test/bcf2ss.riv, filetype:RIV,
↳ 18: filename:/home/runner/work/flopy/flopy/.docs/Notebooks/../../examples/data/mf2005_
↳ test/bcf2ss.rch, filetype:RCH, 19: filename:/home/runner/work/flopy/flopy/.docs/
↳ Notebooks/../../examples/data/mf2005_test/bcf2ss.pcg, filetype:PCG, 20: filename:/home/
↳ runner/work/flopy/flopy/.docs/Notebooks/../../examples/data/mf2005_test/bcf2ss.oc,
↳ filetype:OC, 10: filename:/home/runner/work/flopy/flopy/.docs/Notebooks/../../examples/
↳ data/mf2005_test/bcf2ss.dis, filetype:DIS}
```

```
-----
ModflowBas6 free format:True
```

```
loading dis package file...
```

```
    Loading dis package with:
```

```
        2 layers, 10 rows, 15 columns, and 2 stress periods
```

```
    loading laycbd...
```

```
    loading delr...
```

```
    loading delc...
```

```
    loading top...
```

```
    loading botm...
```

```
        for 2 layers and 1 confining beds
```

```
    loading stress period data...
```

```
        for 2 stress periods
```

```
adding Package: DIS
```

```
    DIS package load...success
```

```
    LIST package load...skipped
```

```
loading bas6 package file...
```

```
adding Package: BAS6
```

```
    BAS6 package load...success
```

```
loading bcf package file...
```

```
    loading ipakcb, HDRY, IWDFLG, WETFCT, IWETIT, IHDWET...
```

```
    loading LAYCON...
```

```
    loading TRPY...
```

```
    loading hy layer 1...
```

```
    loading vcont layer 1...
```

```
    loading sf2 layer 1...
```

```
    loading tran layer 2...
```

```
Adding bcf2ss.cbc (unit=30) to the output list.
```

(continues on next page)

(continued from previous page)

```

adding Package: BCF6
    BCF6 package load...success
loading wel package file...
    loading <class 'flopy.modflow.mfwel.ModflowWel'> for kper      1
    loading <class 'flopy.modflow.mfwel.ModflowWel'> for kper      2
adding Package: WEL
    WEL package load...success
loading riv package file...
    loading <class 'flopy.modflow.mfriv.ModflowRiv'> for kper      1
    loading <class 'flopy.modflow.mfriv.ModflowRiv'> for kper      2
adding Package: RIV
    RIV package load...success
loading rch package file...
    loading rech stress period    1...
adding Package: RCH
    RCH package load...success
loading pcg package file...
    explicit ihcofadd in file
adding Package: PCG
    PCG package load...success
loading oc package file...
head file name will be generated by flopy
drawdown file name will be generated by flopy
adding Package: OC
    OC package load...success

```

WARNING:

```
External file unit 30 does not exist in ext_unit_dict.
```

WARNING:

```
External file unit 50 does not exist in ext_unit_dict.
```

WARNING:

```
External file unit 0 does not exist in ext_unit_dict.
```

The following 8 packages were successfully loaded.

```

bcf2ss.dis
bcf2ss.ba6
bcf2ss.bc6
bcf2ss.wel
bcf2ss.riv
bcf2ss.rch
bcf2ss.pcg
bcf2ss.oc

```

The following 1 packages were not loaded.

```
bcf2ss.lst
```

Auxiliary variables

Below we load a model containing auxiliary variables, then access them.

```
[3]: model_ws = os.path.join("../", "..", "examples", "data", "mp6")
ml = flopy.modflow.Modflow.load(
    "EXAMPLE.nam",
    model_ws=model_ws,
    verbose=False,
    version="mf2005",
    check=False,
)
```

Auxiliary IFACE data are in the river package. Retrieve it from the model object.

```
[4]: riv = ml.riv.stress_period_data[0]
```

Confirm that the iface auxiliary data have been read by looking at the dtype.

```
[5]: riv.dtype
[5]: dtype((numpy.record, [('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('stage', '<f4'), ('cond
→', '<f4'), ('rbot', '<f4'), ('iface', '<f4')]))
```

The iface data is accessible from the recarray.

```
[6]: riv["iface"]
[6]: array([6., 6., 6., 6., 6., 6., 6., 6., 6., 6., 6., 6., 6., 6., 6., 6., 6.,
        6., 6., 6., 6., 6., 6., 6., 6.], dtype=float32)
```

5.3.4 MODFLOW Tutorial 1: Confined Steady-State Flow Model

This tutorial demonstrates use of FloPy to develop a simple MODFLOW-2005 model.

```
[1]: # ## Getting Started
#
# If FloPy has been properly installed, then it can be imported as follows:
from pathlib import Path
from tempfile import TemporaryDirectory
```

```
[2]: import numpy as np
```

```
[3]: import flopy
```

Now that we can import flopy, we begin creating our simple MODFLOW model. numpy is imported to create arrays of model data.

Creating the MODFLOW Model

One of the nice things about creating models in python is that it is very easy to change one or two things and completely change the grid resolution for your model. So in this example, we will design our python script so that the number of layers, columns, and rows can be easily changed.

We can create a very simple MODFLOW model that has a basic package (BAS), discretization input file (DIS), layer-property flow (LPF) package, output control (OC), and preconditioned conjugate gradient (PCG) solver. Each one of these has its own input file, which will be created automatically by flopy, provided that we pass flopy the correct information.

Discretization

We start by creating a temporary workspace, then a flopy model object.

```
[4]: temp_dir = TemporaryDirectory()
workspace = Path(temp_dir.name)
name = "tutorial01_mf"
mf = flopy.modflow.Modflow(name, exe_name="mf2005", model_ws=workspace)
```

Next, let's proceed by defining our model domain and creating a MODFLOW grid to span the domain.

```
[5]: Lx = 1000.0
Ly = 1000.0
ztop = 0.0
zbot = -50.0
nlay = 1
nrow = 10
ncol = 10
delr = Lx / ncol
delc = Ly / nrow
delv = (ztop - zbot) / nlay
botm = np.linspace(ztop, zbot, nlay + 1)
```

With this information, we can now create the flopy discretization object by entering the following:

```
[6]: dis = flopy.modflow.ModflowDis(
    mf, nlay, nrow, ncol, delr=delr, delc=delc, top=ztop, botm=botm[1:]
)
```

Basic Package

Next we can create a flopy object that represents the MODFLOW Basic Package. For this simple model, we will assign constant head values of 10. and 0. to the first and last model columns (in all layers), respectively. The python code for doing this is:

```
[7]: ibound = np.ones((nlay, nrow, ncol), dtype=np.int32)
ibound[:, :, 0] = -1
ibound[:, :, -1] = -1
strt = np.ones((nlay, nrow, ncol), dtype=np.float32)
strt[:, :, 0] = 10.0
```

(continues on next page)

(continued from previous page)

```
strt[:, :, -1] = 0.0
bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
```

Layer-Property Flow Package

Constant values of 10. are assigned for the horizontal and vertical hydraulic conductivity:

```
[8]: lpf = flopy.modflow.ModflowLpf(mf, hk=10.0, vka=10.0, ipakcb=53)
```

Because we did not specify a value for laytyp, Flopy will use the default value of 0, which means that this model will be confined.

Output Control

Here we can use the default OC settings by specifying the following:

```
[9]: spd = {(0, 0): ["print head", "print budget", "save head", "save budget"]}
oc = flopy.modflow.ModflowOc(mf, stress_period_data=spd, compact=True)
```

The stress period dictionary is used to set what output is saved for the corresponding stress period and time step. In this case, the tuple (0, 0) means that stress period 1 and time step 1 for MODFLOW will have output saved. Head and budgets will be printed and head and budget information will be saved.

Preconditioned Conjugate Gradient Package

The default settings used by flopy will be used by specifying the following commands:

```
[10]: pcg = flopy.modflow.ModflowPcg(mf)
```

Writing the MODFLOW Data Files

The MODFLOW input data files are written by simply issuing the following:

```
[11]: mf.write_input()
```

Running the Model

Flopy can also be used to run the model. The model object (mf in this example) has an attached method that will run the model. For this to work, the MODFLOW program must be located somewhere within the system path, or within the working directory. In this example, we have specified that the name of the executable program is 'mf2005'. Issue the following to run the model:

```
[12]: success, buff = mf.run_model()
assert success, "MODFLOW did not terminate normally."
```

```
FloPy is using the following executable to run the model: ../../home/runner/.local/bin/
↳modflow/mf2005
```

```

MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017
```

```
Using NAME file: tutorial01_mf.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:39:21
```

```
Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:39:21
Elapsed run time: 0.002 Seconds
```

```
Normal termination of simulation
```

Here we have used `run_model`, and we could also have specified values for the optional keywords `silent`, `pause`, and `report`.

Post-Processing the Results

Now that we have successfully built and run our MODFLOW model, we can look at the results. MODFLOW writes the simulated heads to a binary data output file. We cannot look at these heads with a text editor, but flopy has a binary utility that can be used to read the heads. The following statements will read the binary head file and create a plot of simulated heads for layer 1:

```
[13]: import matplotlib.pyplot as plt
```

```
[14]: import flopy.utils.binaryfile as bf
```

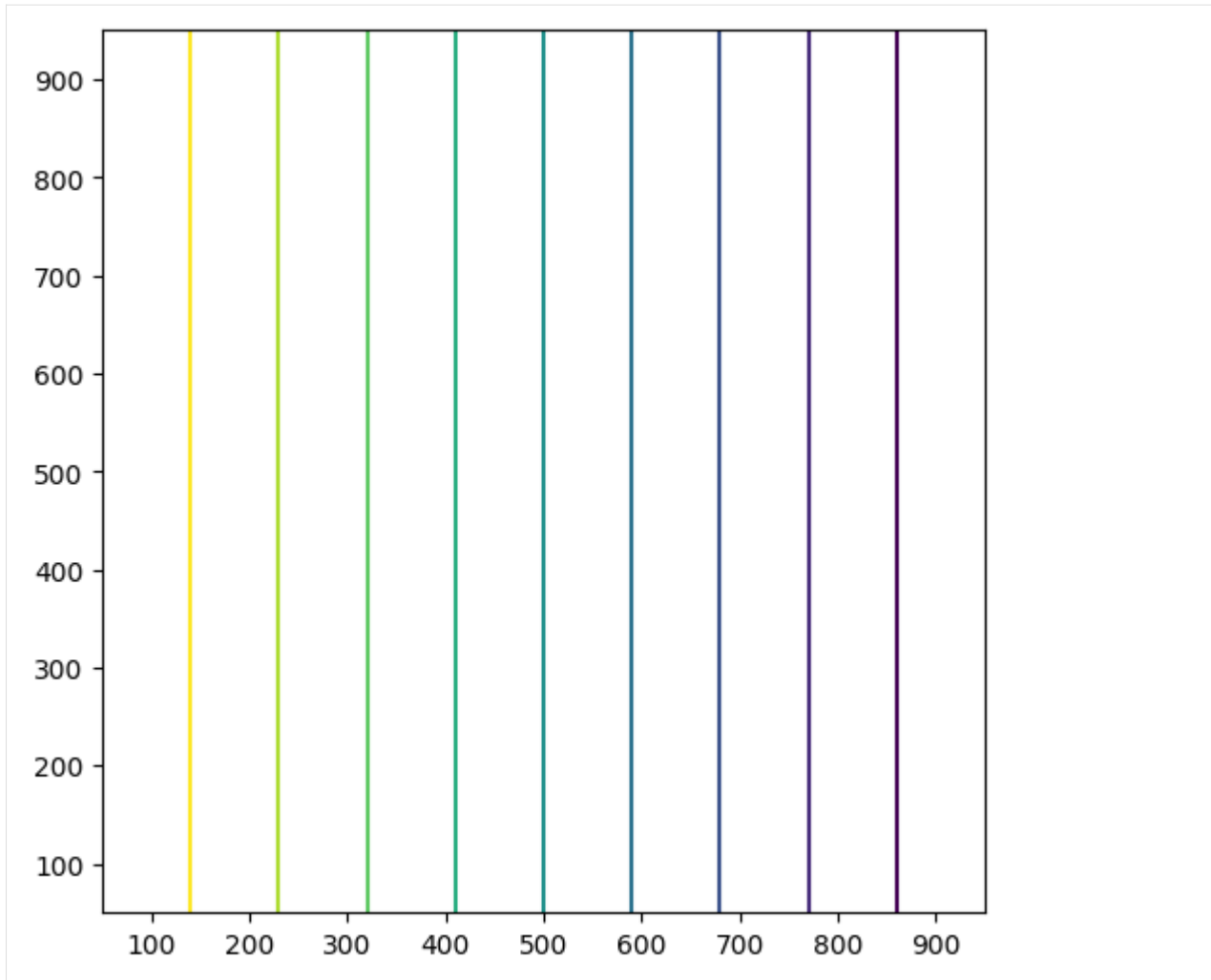
Extract the heads

```
[15]: hds = bf.HeadFile(Path(workspace) / f"{name}.hds")
head = hds.get_data(totim=1.0)
```

Contour the heads

```
[16]: extent = (delr / 2.0, Lx - delr / 2.0, Ly - delc / 2.0, delc / 2.0)
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.contour(head[0, :, :], levels=np.arange(1, 10, 1), extent=extent)
```

```
[16]: <matplotlib.contour.QuadContourSet at 0x7fee97437b50>
```



FloPy also has some pre-canned plotting capabilities that can be accessed using the `PlotMapView()` class. The following code shows how to use the `plotmapview` class to plot boundary conditions (IBOUND), plot the grid, plot head contours, and plot vectors:

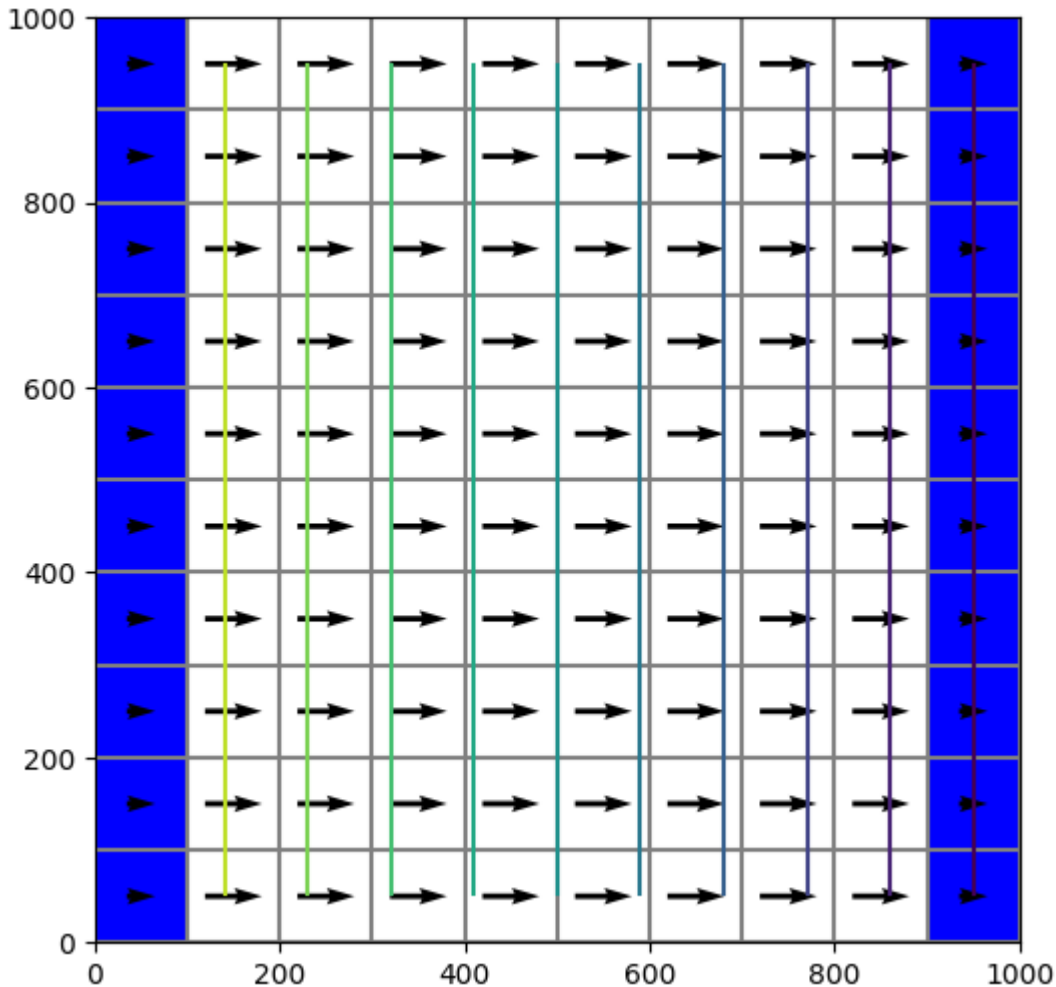
```
[17]: # Extract the heads
hds = bf.HeadFile(Path(workspace) / f"{name}.hds")
times = hds.get_times()
head = hds.get_data(totim=times[-1])
```

Extract the cell-by-cell flows

```
[18]: cbb = bf.CellBudgetFile(Path(workspace) / f"{name}.cbc")
kstpker_list = cbb.get_kstpker()
frf = cbb.get_data(text="FLOW RIGHT FACE", totim=times[-1])[0]
fff = cbb.get_data(text="FLOW FRONT FACE", totim=times[-1])[0]
qx, qy, qz = flopy.utils.postprocessing.get_specific_discharge(
    (frf, fff, None), mf, head
)
```

Create the figure

```
[19]: fig = plt.figure(figsize=(6, 6))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      modelmap = flopy.plot.PlotMapView(model=mf, layer=0, ax=ax)
      qm = modelmap.plot_ibound()
      lc = modelmap.plot_grid()
      cs = modelmap.contour_array(head, levels=np.linspace(0, 10, 11))
      quiver = modelmap.plot_vector(qx, qy)
```



Clean up the temporary workspace

```
[20]: try:
      temp_dir.cleanup()
      except:
          # prevent windows permission error
          pass
```

5.3.5 MODFLOW Tutorial 2: Unconfined Transient Flow Model

In this example, we will convert the tutorial 1 model into an unconfined, transient flow model with time varying boundaries. Instead of using constant heads for the left and right boundaries (by setting `ibound` to -1), we will use general head boundaries. We will have the model consider the following conditions:

- Initial conditions – head is 10.0 everywhere
- Period 1 (1 day) – steady state with left and right GHB stage = 10.
- Period 2 (100 days) – left GHB with stage = 10., right GHB with stage set to 0.
- Period 3 (100 days) – pumping well at model center with rate = -500., left and right GHB = 10., and 0.

We will start with selected model commands from the previous tutorial.

```
[1]: # ## Getting Started
#
# As shown in the previous MODFLOW tutorial, import flopy.
from pathlib import Path
from tempfile import TemporaryDirectory
```

```
[2]: import numpy as np
```

```
[3]: import flopy
```

Creating the MODFLOW Model

Define the Model Extent, Grid Resolution, and Characteristics

Assign the model information

```
[4]: Lx = 1000.0
Ly = 1000.0
ztop = 10.0
zbot = -50.0
nlay = 1
nrow = 10
ncol = 10
delr = Lx / ncol
delc = Ly / nrow
delv = (ztop - zbot) / nlay
botm = np.linspace(ztop, zbot, nlay + 1)
hk = 1.0
vka = 1.0
sy = 0.1
ss = 1.0e-4
laytyp = 1
```

Variables for the BAS package Note that changes from the MODFLOW tutorial 1

```
[5]: ibound = np.ones((nlay, nrow, ncol), dtype=np.int32)
strt = 10.0 * np.ones((nlay, nrow, ncol), dtype=np.float32)
```


Define the Stress Periods

To create a model with multiple stress periods, we need to define `nper`, `perlen`, `nstp`, and `steady`. This is done in the following block in a manner that allows us to pass these variable directly to the discretization object:

```
[6]: nper = 3
      perlen = [1, 100, 100]
      nstp = [1, 100, 100]
      steady = [True, False, False]
```

Create Time-Invariant Flopy Objects

With this information, we can now create the static flopy objects that do not change with time:

```
[7]: temp_dir = TemporaryDirectory()
      workspace = temp_dir.name
      name = "tutorial02_mf"
      mf = flopy.modflow.Modflow(name, exe_name="mf2005", model_ws=workspace)
      dis = flopy.modflow.ModflowDis(
          mf,
          nlay,
          nrow,
          ncol,
          delr=delr,
          delc=delc,
          top=ztop,
          botm=botm[1:],
          nper=nper,
          perlen=perlen,
          nstp=nstp,
          steady=steady,
      )
      bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
      lpf = flopy.modflow.ModflowLpf(
          mf, hk=hk, vka=vka, sy=sy, ss=ss, laytyp=laytyp, ipakcb=53
      )
      pcg = flopy.modflow.ModflowPcg(mf)
```

Transient General-Head Boundary Package

At this point, our model is ready to add our transient boundary packages. First, we will create the GHB object, which is of the following type: `flopy.modflow.ModflowGhb()`.

The key to creating Flopy transient boundary packages is recognizing that the boundary data is stored in a dictionary with key values equal to the zero-based stress period number and values equal to the boundary conditions for that stress period. For a GHB the values can be a two-dimensional nested list of `[layer, row, column, stage, conductance]`.

```
[8]: stageleft = 10.0
      stageright = 10.0
      bound_sp1 = []
```

(continues on next page)

(continued from previous page)

```

for il in range(nlay):
    condleft = hk * (stageleft - zbot) * delc
    condright = hk * (stageright - zbot) * delc
    for ir in range(nrow):
        bound_sp1.append([il, ir, 0, stageleft, condleft])
        bound_sp1.append([il, ir, ncol - 1, stageright, condright])
print("Adding ", len(bound_sp1), "GHBs for stress period 1.")

```

Adding 20 GHBs for stress period 1.

```

[9]: # Make list for stress period 2
stageleft = 10.0
stageright = 0.0
condleft = hk * (stageleft - zbot) * delc
condright = hk * (stageright - zbot) * delc
bound_sp2 = []
for il in range(nlay):
    for ir in range(nrow):
        bound_sp2.append([il, ir, 0, stageleft, condleft])
        bound_sp2.append([il, ir, ncol - 1, stageright, condright])
print("Adding ", len(bound_sp2), "GHBs for stress period 2.")

```

Adding 20 GHBs for stress period 2.

```

[10]: # We do not need to add a dictionary entry for stress period 3.
# Flopy will automatically take the list from stress period 2 and apply it
# to the end of the simulation
stress_period_data = {0: bound_sp1, 1: bound_sp2}

```

```

[11]: # Create the flopy ghb object
ghb = flopy.modflow.ModflowGhb(mf, stress_period_data=stress_period_data)

```

Transient Well Package

Now we can create the well package object, which is of the type, `flopy.modflow.ModflowWel()`.

```

[12]: # Create the well package
# Remember to use zero-based layer, row, column indices!
pumping_rate = -500.0
wel_sp1 = [[0, nrow / 2 - 1, ncol / 2 - 1, 0.0]]
wel_sp2 = [[0, nrow / 2 - 1, ncol / 2 - 1, 0.0]]
wel_sp3 = [[0, nrow / 2 - 1, ncol / 2 - 1, pumping_rate]]
stress_period_data = {0: wel_sp1, 1: wel_sp2, 2: wel_sp3}
wel = flopy.modflow.ModflowWel(mf, stress_period_data=stress_period_data)

```

Output Control

Here we create the output control package object, which is of the type `flopy.modflow.ModflowOc()`.

```
[13]: stress_period_data = {}
      for kper in range(nper):
          for kstp in range(nstp[kper]):
              stress_period_data[(kper, kstp)] = [
                  "save head",
                  "save drawdown",
                  "save budget",
                  "print head",
                  "print budget",
              ]
      oc = flopy.modflow.ModflowOc(
          mf, stress_period_data=stress_period_data, compact=True
      )
```

Running the Model

Run the model with the `run_model` method, which returns a success flag and the stream of output. With `run_model`, we have some finer control, that allows us to suppress the output.

```
[14]: # Write the model input files
      mf.write_input()
```

```
[15]: # Run the model
      success, mfoutput = mf.run_model(silent=True, pause=False)
      assert success, "MODFLOW did not terminate normally."
```

Post-Processing the Results

Once again, we can read heads from the MODFLOW binary output file, using the `flopy.utils.binaryfile()` module. Included with the `HeadFile` object are several methods that we will use here:

- `get_times()` will return a list of times contained in the binary head file
- `get_data()` will return a three-dimensional head array for the specified time
- `get_ts()` will return a time series array `[ntimes, headval]` for the specified cell

Using these methods, we can create head plots and hydrographs from the model results.

```
[16]: # Imports
      import matplotlib.pyplot as plt
```

```
[17]: import flopy.utils.binaryfile as bf
```

```
[18]: # Create the headfile and budget file objects
      headobj = bf.HeadFile(Path(workspace) / f"{name}.hds")
      times = headobj.get_times()
      cbb = bf.CellBudgetFile(Path(workspace) / f"{name}.cbc")
```

```
[19]: # Setup contour parameters
levels = np.linspace(0, 10, 11)
extent = (delr / 2.0, Lx - delr / 2.0, delc / 2.0, Ly - delc / 2.0)
print("Levels: ", levels)
print("Extent: ", extent)

Levels:  [ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
Extent:  (50.0, 950.0, 50.0, 950.0)
```

```
[20]: # Well point for plotting
wpt = (450.0, 550.0)
```

Create a figure with maps for three times

```
[21]: # Make the plots
fig = plt.figure(figsize=(5, 15))
mytimes = [1.0, 101.0, 201.0]
for iplot, time in enumerate(mytimes):
    print("*****Processing time: ", time)
    head = headobj.get_data(totim=time)
    # Print statistics
    print("Head statistics")
    print("  min: ", head.min())
    print("  max: ", head.max())
    print("  std: ", head.std())

    # Extract flow right face and flow front face
    frf = cbb.get_data(text="FLOW RIGHT FACE", totim=time)[0]
    fff = cbb.get_data(text="FLOW FRONT FACE", totim=time)[0]

    # Create a map for this time
    ax = fig.add_subplot(len(mytimes), 1, iplot + 1, aspect="equal")
    ax.set_title(f"stress period {iplot + 1}")

    pmv = flopy.plot.PlotMapView(model=mf, layer=0, ax=ax)
    qm = pmv.plot_ibound()
    lc = pmv.plot_grid()
    qm = pmv.plot_bc("GHB", alpha=0.5)
    if head.min() != head.max():
        cs = pmv.contour_array(head, levels=levels)
        plt.clabel(cs, inline=1, fontsize=10, fmt="%1.1f")
        quiver = pmv.plot_vector(frf, fff)

    mfc = "None"
    if (iplot + 1) == len(mytimes):
        mfc = "black"
    ax.plot(
        wpt[0],
        wpt[1],
        lw=0,
        marker="o",
        markersize=8,
        markeredgewidth=0.5,
```

(continues on next page)

(continued from previous page)

```
        markeredgecolor="black",
        markerfacecolor=mfc,
        zorder=9,
    )
    ax.text(wpt[0] + 25, wpt[1] - 25, "well", size=12, zorder=12)
```

```
*****Processing time: 1.0
```

```
Head statistics
```

```
min: 10.0
```

```
max: 10.0
```

```
std: 0.0
```

```
*****Processing time: 101.0
```

```
Head statistics
```

```
min: 0.025931068
```

```
max: 9.998436
```

```
std: 3.2574987
```

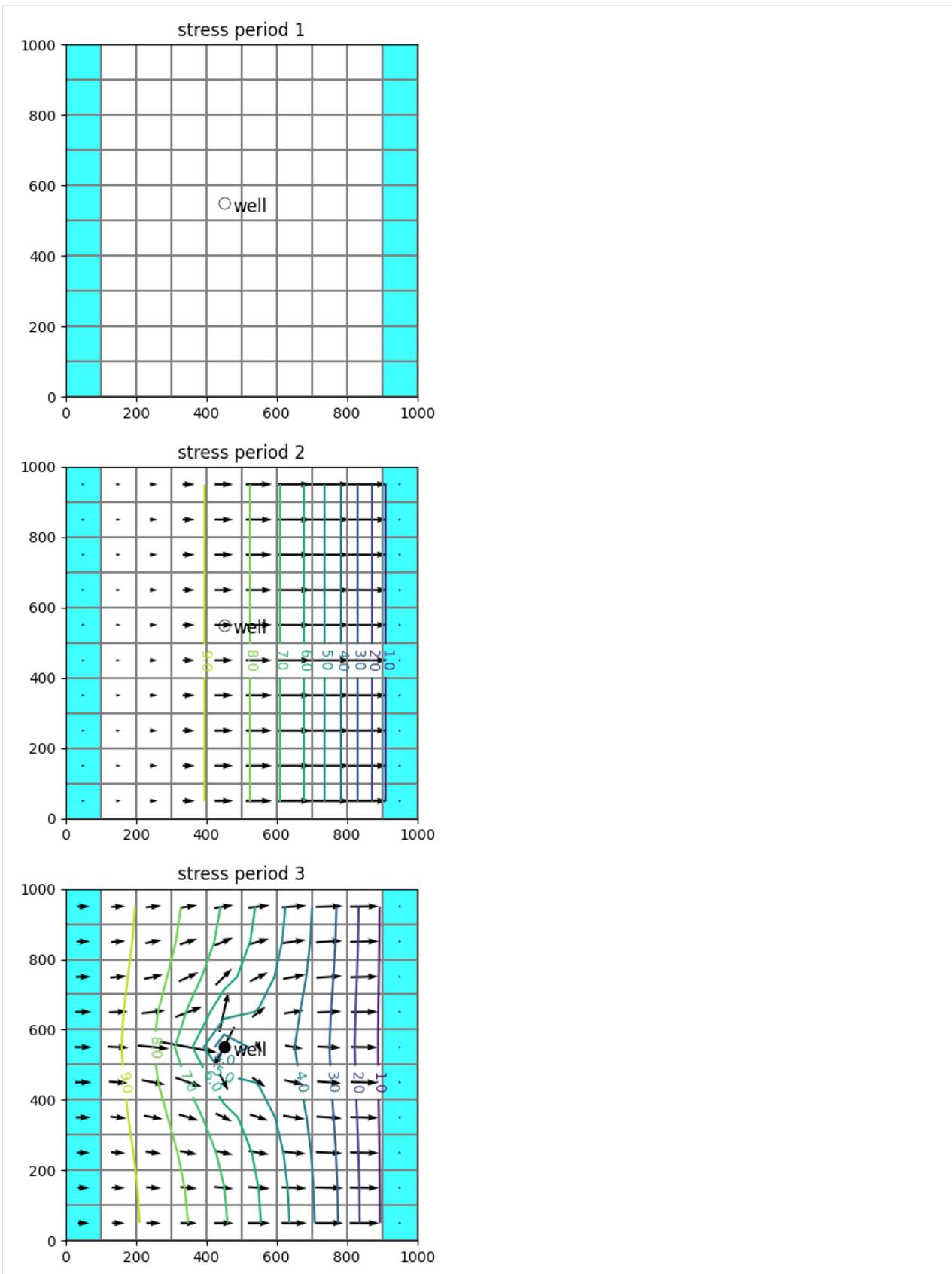
```
*****Processing time: 201.0
```

```
Head statistics
```

```
min: 0.016297927
```

```
max: 9.994038
```

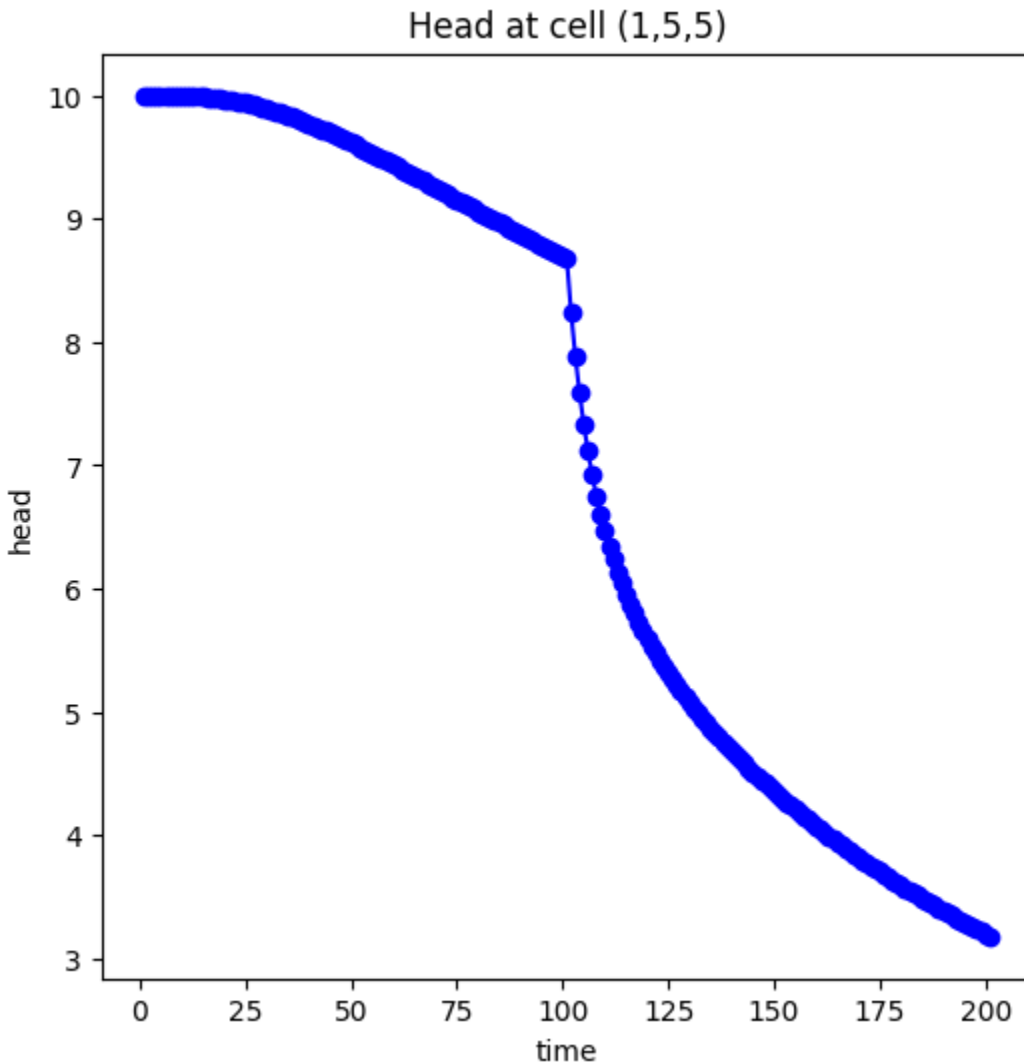
```
std: 3.1544707
```



Create a hydrograph

```
[22]: # Plot the head versus time
idx = (0, int(nrow / 2) - 1, int(ncol / 2) - 1)
ts = headobj.get_ts(idx)
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1)
ttl = f"Head at cell ({idx[0] + 1},{idx[1] + 1},{idx[2] + 1})"
ax.set_title(ttl)
ax.set_xlabel("time")
ax.set_ylabel("head")
ax.plot(ts[:, 0], ts[:, 1], "bo-")
```

```
[22]: [<matplotlib.lines.Line2D at 0x7fa904ddbbb0>]
```



```
[23]: try:
        temp_dir.cleanup()
    except:
        # prevent windows permission error
```

(continues on next page)

```
pass
```

5.4 MODFLOW-LGR

5.4.1 Local grid refinement (LGR) utility

FloPy has a utility for creating and managing a parent grid and a child grid. Both grids must regular MODFLOW grids consisting of layers, rows, and columns. The term Local Grid Refinement (LGR) is used to denote this configuration. With the current version, only one child grid is supported; however, the utility could be used multiple times to create multiple child models. Those child models could also be used with the utility to create grandchildren models, though it is unclear if there would be benefit to constructing models with this configuration.

The FloPy LGR utility is designed to work primarily with MODFLOW 6 as MODFLOW 6 can be used to run multiple groundwater models in the same simulation. These models can be tightly coupled at their interfaces. This LGR approach is based on the idea of creating multiple model instances, each with their own input and output. This approach is different from using an unstructured grid to represent a single mesh with local refinements in areas of interest.

First set the path and import the required packages. The flopy path doesn't have to be set if you install flopy from a binary installer. If you want to run this notebook, you have to set the path to your own flopy path.

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

import flopy
from flopy.utils.lgrutil import Lgr

# temporary directory
temp_dir = TemporaryDirectory()
workspace = os.path.join(temp_dir.name, "mf6lgr")
# make sure workspace directory exists
if not os.path.isdir(workspace):
    os.makedirs(workspace, exist_ok=True)

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```


LGR basics

The LGR Utility works by defining the parent grid. Then cells within the parent grid are marked as being replaced by a child grid with a specified level of refinement. The terminology and variable descriptions here are based on concepts developed for the MODFLOW-LGR program.

```
[2]: # set up the parent grid information
xoffp = 0.0
yoffp = 0.0
nlayp = 3
nrowp = 25
ncolp = 45
dx = 100.0
dy = 100.0
dz = 20.0
delrp = dx * np.ones(ncolp, dtype=float)
delcp = dy * np.ones(nrowp, dtype=float)
topp = dz * np.ones((nrowp, ncolp), dtype=float)
botmp = np.empty((nlayp, nrowp, ncolp), dtype=float)
for k in range(nlayp):
    botmp[k] = -(k + 1) * dz

[3]: # Define relation of child to parent using
# these parent indices to show where child is active
istart = int(nrowp / 3)
istop = int(nrowp * 2 / 3) + 1
jstart = int(ncolp / 3)
jstop = int(ncolp * 2 / 3) + 1
kstart = 0
kstop = 1 + 1

# idomainp has a 1 where parent is active and 0 where child is active
idomainp = np.ones((nlayp, nrowp, ncolp), dtype=int)
idomainp[kstart:kstop, istart:istop, jstart:jstop] = 0

# Set the number of child layers per parent layer (list of length parent nlay)
# The following will result in 4 child layers that span model layers
# 1 and 2 of the parent grid. Vertical dimension of child cells will be half
# of the parent vertical dimension.
ncpp1 = np.array(nlayp * [2], dtype=int)
ncpp1[kstop:] = 0
print(f"ncpp1: {ncpp1}")

# Set the number of child cells per parent cell
ncpp = 5

# Create the Lgr utility object, which contains methods for connecting a
# parent and child model in a MODFLOW 6 simulation
lgr = Lgr(
    nlayp,
    nrowp,
    ncolp,
    delrp,
```

(continues on next page)

(continued from previous page)

```

    delcp,
    topp,
    botmp,
    idomainp,
    ncpp=ncpp,
    ncpl=ncpl,
    xllp=xoffp,
    yllp=yoffp,
)

```

```
ncpl: [2 2 0]
```

```

[4]: # information about the parent and child models
      # can be accessed using simple grid objects returned
      # on the fly by lgr.parent and lgr.child
      print("Parent grid information")
      parent = lgr.parent
      print(f"parent nlay: {parent.nlay}")
      print(f"parent nrow: {parent.nrow}")
      print(f"parent ncol: {parent.ncol}")
      print(f"parent delr[0]: {parent.delr[0]}")
      print(f"parent delc[0]: {parent.delc[0]}")
      print(f"parent top[0, 0]: {parent.top[0, 0]}")
      print(f"parent botm[:, 0, 0]: {parent.botm[:, 0, 0]}")

      print("\nChild grid information")
      child = lgr.child
      print(f"child nlay: {child.nlay}")
      print(f"child nrow: {child.nrow}")
      print(f"child ncol: {child.ncol}")
      print(f"child delr[0]: {child.delr[0]}")
      print(f"child delc[0]: {child.delc[0]}")
      print(f"child top[0, 0]: {child.top[0, 0]}")
      print(f"child botm[:, 0, 0]: {child.botm[:, 0, 0]}")

```

```

Parent grid information
parent nlay: 3
parent nrow: 25
parent ncol: 45
parent delr[0]: 100.0
parent delc[0]: 100.0
parent top[0, 0]: 20.0
parent botm[:, 0, 0]: [-20. -40. -60.]

```

```

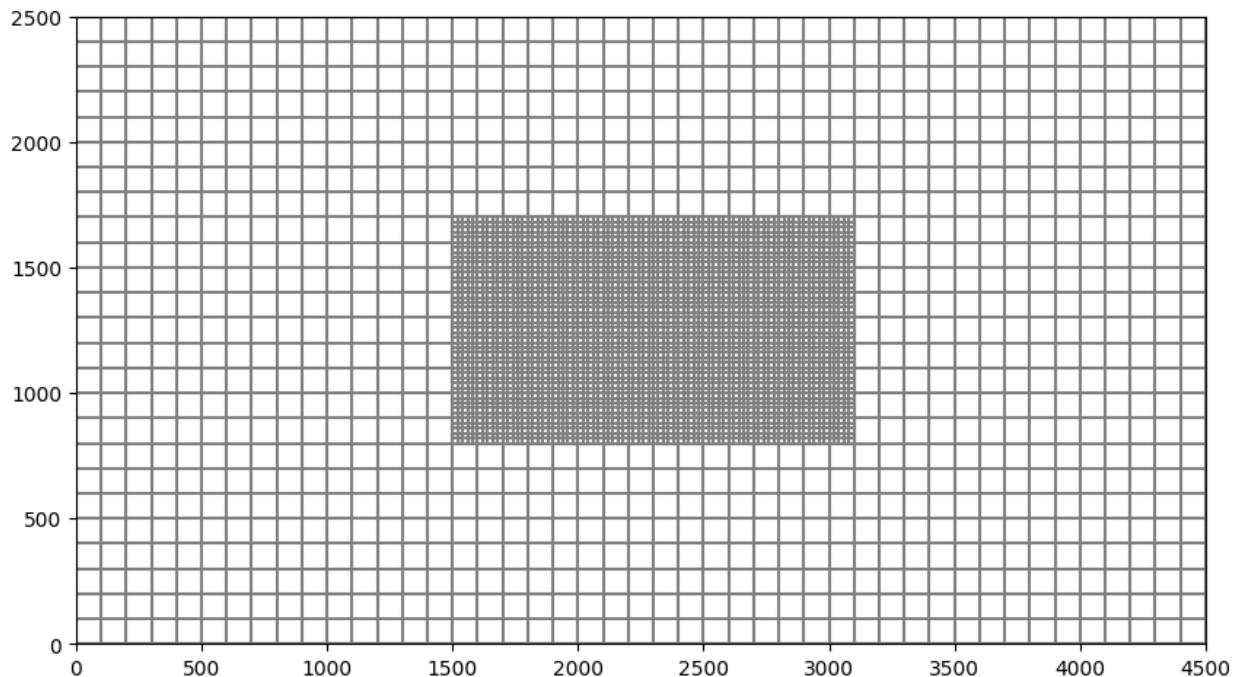
Child grid information
child nlay: 4
child nrow: 45
child ncol: 80
child delr[0]: 20.0
child delc[0]: 20.0
child top[0, 0]: 20.0
child botm[:, 0, 0]: [ 0. -20. -30. -40.]

```

```
[5]: # from the Lgr object, it is possible to ask for a flopy modelgrid object
# for the parent and child. These objects are created on the fly and
# can be used to plot the grids prior to creating the MODFLOW 6 simulation
mgs = lgr.parent.modelgrid
mgc = lgr.child.modelgrid

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mgc.plot(ax=ax)
mgs.plot(ax=ax)
```

```
[5]: <matplotlib.collections.LineCollection at 0x7f47009fc460>
```



Simple example model

For this example, we reproduce the simple model shown on the front page of the flopy GitHub repository. In this adaptation, we insert a finer, locally refined grid in the middle of the domain, but we tightly couple the parent and child models.

```
[6]: # set up the parent grid information
xoffp = 0.0
yoffp = 0.0
nlayp = 5
nrowp = 5
ncolp = 5
dx = 100.0
dy = 100.0
dz = 100.0
delrp = dx * np.ones(ncolp, dtype=float)
delcp = dy * np.ones(nrowp, dtype=float)
```

(continues on next page)

(continued from previous page)

```

topp = dz * np.ones((nrowp, ncolp), dtype=float)
botmp = np.empty((nlayp, nrowp, ncolp), dtype=float)
for k in range(nlayp):
    botmp[k] = -(k + 1) * dz

# Define relation of child to parent
# idomainp has a 1 where parent is active and 0 where child is active
idomainp = np.ones((nlayp, nrowp, ncolp), dtype=int)
idomainp[:, 1:4, 1:4] = 0

# Set the number of child cells per parent cell
ncpp = 5

# Set the number of child layers per parent layer (list of cell parent nlay)
ncppl = [1, 1, 1, 1, 1]

# Create the Lgr utility object, which contains methods for connecting a
# parent and child model in a MODFLOW 6 simulation
lgr = Lgr(
    nlayp,
    nrowp,
    ncolp,
    delrp,
    delcp,
    topp,
    botmp,
    idomainp,
    ncpp=ncpp,
    ncppl=ncppl,
    xllp=xoffp,
    yllp=yoffp,
)

# from the Lgr object, it is possible to ask for a flopy modelgrid object
# for the parent and child. These objects are created on the fly and
# can be used to plot the grids prior to creating the MODFLOW 6 simulation
mgp = lgr.parent.modelgrid
mgc = lgr.child.modelgrid

```

```

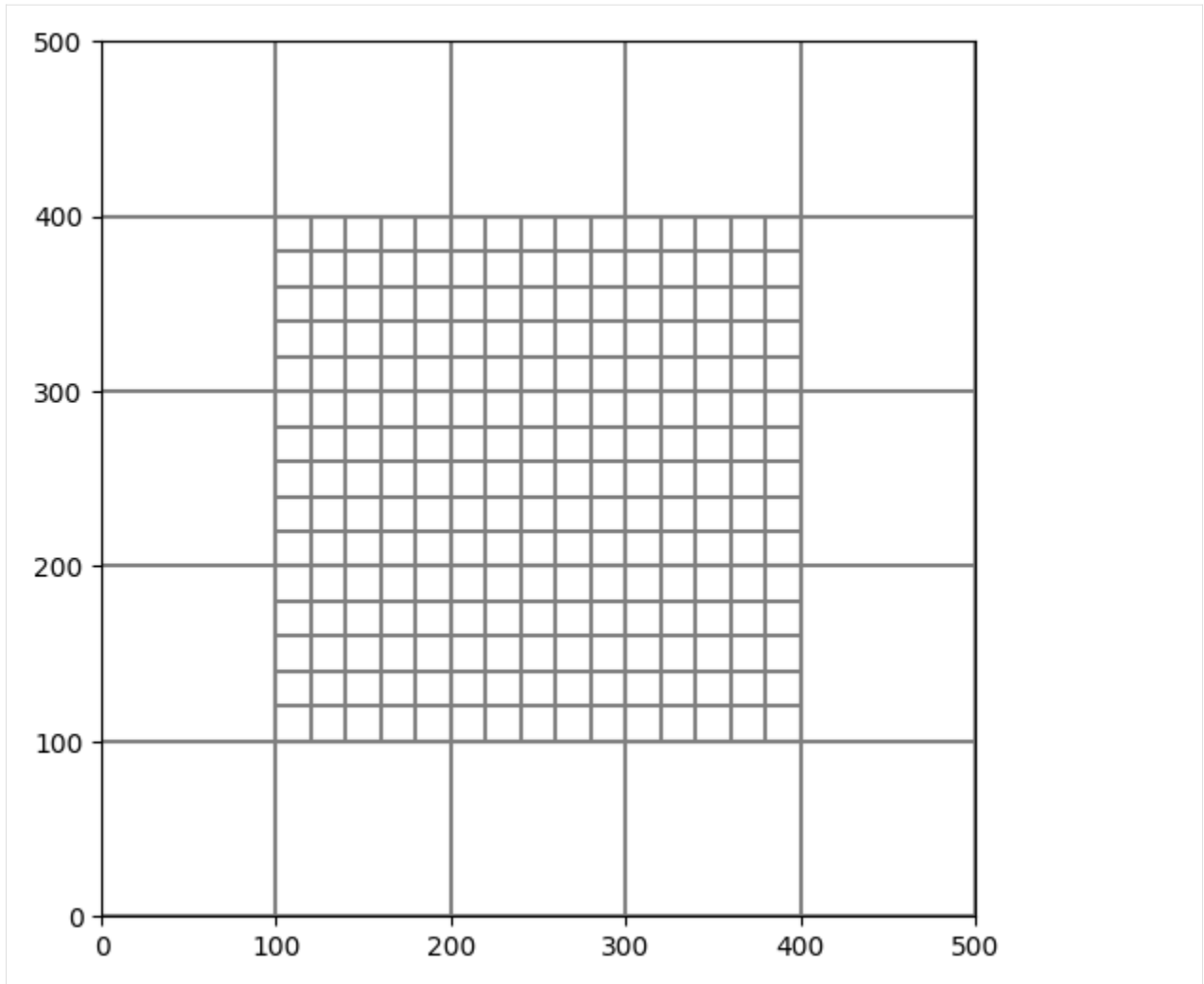
[7]: # plot the child and parent grids, which can be
# done easily using the flopy modelgrid objects
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mgc.plot(ax=ax)
mgp.plot(ax=ax)

```

```

[7]: <matplotlib.collections.LineCollection at 0x7f47009566a0>

```



Build the MODFLOW 6 Model

Now that we have the grid, we can setup and run the MODFLOW 6 model.

```
[8]: # set up the MODFLOW 6 simulation
ws = os.path.join(workspace, "ex1")
simname = "lgr"
pname = "parent"
cname = "child"
sim = flopy.mf6.MFSimulation(sim_name=simname, sim_ws=ws, exe_name="mf6")
tdis = flopy.mf6.ModflowTdis(sim)

# Because we will use the xt3d option for coupling the models
# we need use to the asymmetric BICGSTAB linear solver
ims = flopy.mf6.ModflowIms(sim, linear_acceleration="BICGSTAB")

# retrieve the exchange data from the lgr object
exchangedata = lgr.get_exchange_data(angldegx=True, cdist=True)
```

(continues on next page)

(continued from previous page)

```

nexg = len(exchangedata)

# When creating the exchange, which couples the child and parent
# models, use the xt3d option, which is an alternative to the
# ghost-node correction. This xt3d option was added as a new
# capability for the gwt-gwt and gwf-gwf exchanges in MODFLOW version 6.3.0.
exg = flopy.mf6.ModflowGwfgwf(
    sim,
    exgtype="GWF6-GWF6",
    xt3d=True,
    auxiliary=["angldegx", "cdist"],
    exgmnamea=pname,
    exgmnameb=cname,
    nexg=nexg,
    exchangedata=exchangedata,
)

# Set up the parent model and use the lgr.parent object to
# help provide the necessary information.
lgrp = lgr.parent
gwfp = flopy.mf6.ModflowGwf(sim, modelname=pname, save_flows=True)
dis = flopy.mf6.ModflowGwfdis(gwfp, **lgrp.get_gridprops_dis6())
ic = flopy.mf6.ModflowGwfic(gwfp)
npf = flopy.mf6.ModflowGwfnpf(gwfp, save_specific_discharge=True)
chdspd = [[(0, 0, 0), 1.0], [(0, lgrp.nrow - 1, lgrp.ncol - 1), 0.0]]
chd = flopy.mf6.ModflowGwfchd(gwfp, stress_period_data=chdspd)
oc = flopy.mf6.ModflowGwfoc(
    gwfp,
    budget_filerecord=pname + ".bud",
    head_filerecord=pname + ".hds",
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
)

# Set up the child model and use the lgr.child object to
# help provide the necessary information.
lgrc = lgr.child
gwfc = flopy.mf6.ModflowGwf(sim, modelname=cname, save_flows=True)
dis = flopy.mf6.ModflowGwfdis(gwfc, **lgrc.get_gridprops_dis6())
ic = flopy.mf6.ModflowGwfic(gwfc)
npf = flopy.mf6.ModflowGwfnpf(gwfc, save_specific_discharge=True)
oc = flopy.mf6.ModflowGwfoc(
    gwfc,
    budget_filerecord=cname + ".bud",
    head_filerecord=cname + ".hds",
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
)

sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)

```

(continues on next page)

(continued from previous page)

```

else:
    raise ValueError("Failed to run.")

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims_-1...
  writing package lgr.gwfgwf...
  writing model parent...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
    writing package chd_0...
INFORMATION: maxbound in ('gwf6', 'chd', 'dimensions') changed to 2 based on size of
↳ stress_period_data
  writing package oc...
writing model child...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package oc...

                          MODFLOW 6
                U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
                      VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
                      Build 20220726_0000000

This software has been approved for release by the U.S. Geological
Survey (USGS). Although the software has been subjected to rigorous
review, the USGS reserves the right to update the software as needed
pursuant to further analysis and review. No warranty, expressed or
implied, is made by the USGS or the U.S. Government as to the
functionality of the software and related material nor shall the
fact of release constitute any such warranty. Furthermore, the
software is released on condition that neither the USGS nor the U.S.
Government shall be held liable for any damages resulting from its
authorized or unauthorized use. Also refer to the USGS Water
Resources Software User Rights Notice for complete use, copyright,
and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:19

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

Solving: Stress period:      1      Time step:      1

```

(continues on next page)

(continued from previous page)

```
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:19
Elapsed run time: 0.147 Seconds
```

```
Normal termination of simulation.
```

```
[9]: # load and store the head arrays from the parent and child models
head = [gwfp.output.head().get_data(), gwfc.output.head().get_data()]

# load and store the specific discharge results for the parent and child models
bud = gwfp.output.budget()
spdis = bud.get_data(text="DATA-SPDIS")[0]
spdisp = flopy.utils.postprocessing.get_specific_discharge(spdis, gwfp)
bud = gwfc.output.budget()
spdis = bud.get_data(text="DATA-SPDIS")[0]
spdisc = flopy.utils.postprocessing.get_specific_discharge(spdis, gwfc)
```

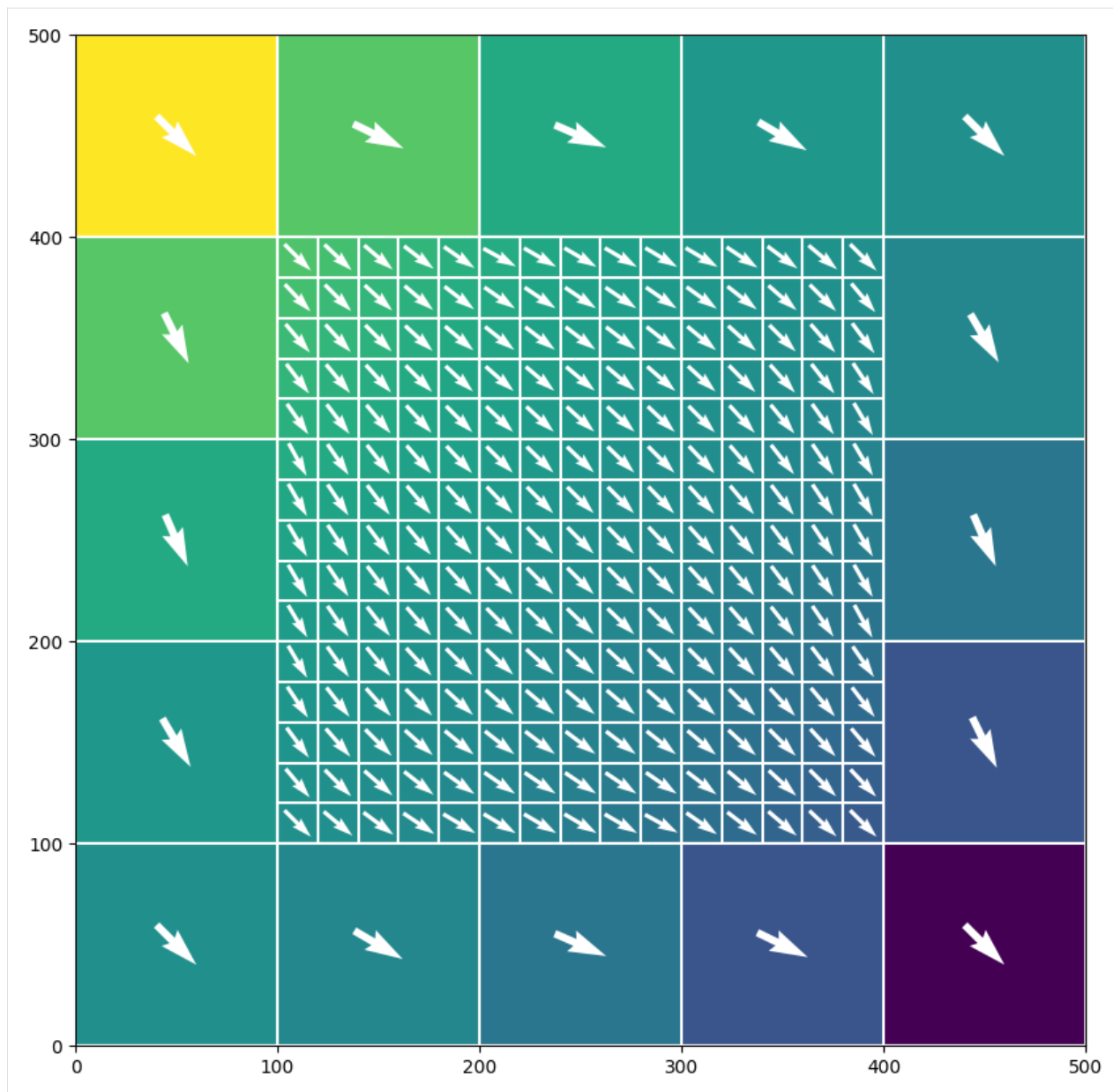
```
[10]: # plot the results from both the parent and child models
# on the same figure
f = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
pmvp = flopy.plot.PlotMapView(gwfp, ax=ax)
pmvc = flopy.plot.PlotMapView(gwfc, ax=ax, extent=pmvp.extent)

pmvp.plot_array(head[0], vmin=0.0, vmax=1.0)
pmvc.plot_array(head[1], vmin=0.0, vmax=1.0)

pmvp.plot_grid(colors="white")
pmvc.plot_grid(colors="white")

pmvp.plot_vector(spdisp[0], spdisp[1], normalize=True, color="white")
pmvc.plot_vector(spdisc[0], spdisc[1], normalize=True, color="white")
```

```
[10]: <matplotlib.quiver.Quiver at 0x7f46fe304100>
```

Coupled flow and transport model

For this example, we simulate coupled flow and transport for a LGR configuration.

```
[11]: # set up the parent grid information
xoffp = 0.0
yoffp = 0.0
nlayp = 3
nrowp = 25
ncolp = 45
dx = 10.0
dy = 10.0
```

(continues on next page)

(continued from previous page)

```

dz = 20.0
delrp = dx * np.ones(ncolp, dtype=float)
delcp = dy * np.ones(nrowp, dtype=float)
topp = dz * np.ones((nrowp, ncolp), dtype=float)
botmp = np.empty((nlayp, nrowp, ncolp), dtype=float)
for k in range(nlayp):
    botmp[k] = -(k + 1) * dz

# Define relation of child to parent using
# these parent indices to show where child is active
istart = int(nrowp / 3)
istop = int(nrowp * 2 / 3) + 1
jstart = int(ncolp / 3)
jstop = int(ncolp * 2 / 3) + 1
kstart = 0
kstop = 1 + 1

# idomainp has a 1 where parent is active and 0 where child is active
idomainp = np.ones((nlayp, nrowp, ncolp), dtype=int)
idomainp[kstart:kstop, istart:istop, jstart:jstop] = 0

# Set the number of child layers per parent layer
ncppl = np.array(nlayp * [1], dtype=int)
ncppl[kstop:] = 0
print(f"ncppl: {ncppl}")

# Set the number of child cells per parent cell
ncpp = 3

# Create the Lgr utility object, which contains methods for connecting a
# parent and child model in a MODFLOW 6 simulation
lgr = Lgr(
    nlayp,
    nrowp,
    ncolp,
    delrp,
    delcp,
    topp,
    botmp,
    idomainp,
    ncpp=ncpp,
    ncppl=ncppl,
    xllp=xoffp,
    yllp=yoffp,
)

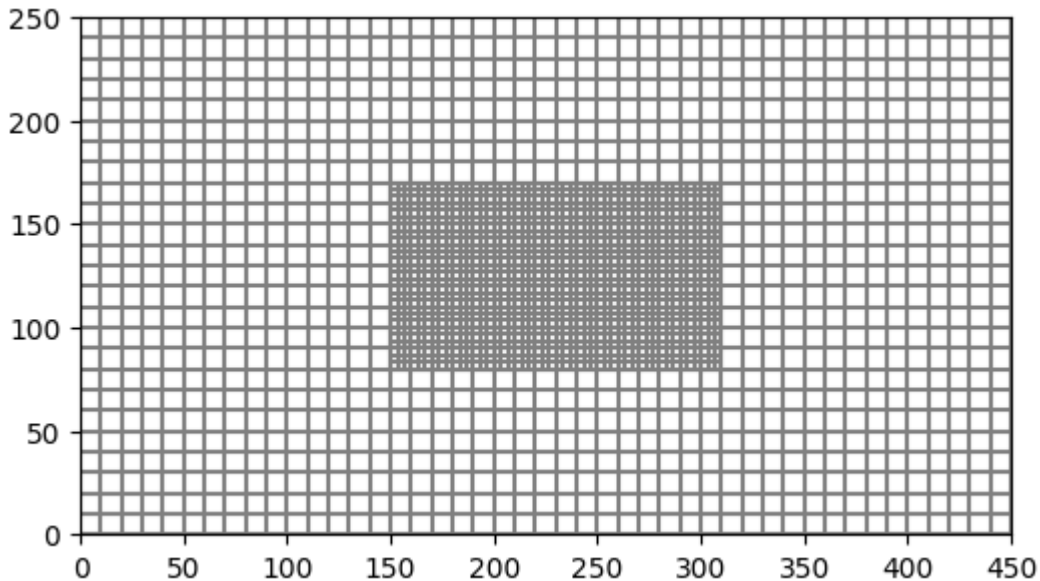
# from the Lgr object, it is possible to ask for a flopy modelgrid object
# for the parent and child. These objects are created on the fly and
# can be used to plot the grids prior to creating the MODFLOW 6 simulation
mgp = lgr.parent.modelgrid
mgc = lgr.child.modelgrid

```

```
ncpp1: [1 1 0]
```

```
[12]: # plot the child and parent grids, which can be
      # done easily using the flopy modelgrid objects
      fig = plt.figure(figsize=(6, 6))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      mgc.plot(ax=ax)
      mgp.plot(ax=ax)
```

```
[12]: <matplotlib.collections.LineCollection at 0x7f46fc1d3be0>
```



Build the MODFLOW 6 Model

Now that we have the grid, we can setup and run the MODFLOW 6 model.

```
[13]: # set up the MODFLOW 6 simulation
      ws = os.path.join(workspace, "ex2")
      simname = "lgr"
      sim = flopy.mf6.MFSimulation(sim_name=simname, sim_ws=ws, exe_name="mf6")
      tdis = flopy.mf6.ModflowTdis(sim, perioddata=[(50 * 365.0, 10, 1.0)])

      pname = "parent-flow"
      cname = "child-flow"

      # Because we will use the xt3d option for coupling the models
      # we need use to the asymmetric BICGSTAB linear solver
      ims_flow = flopy.mf6.ModflowIms(
          sim, linear_acceleration="BICGSTAB", filename="flow.ims"
      )

      # retrieve the exchange data from the lgr object
      exchangedata = lgr.get_exchange_data(angldegx=True, cdist=True)
```

(continues on next page)

(continued from previous page)

```

nexg = len(exchangedata)

# When creating the exchange, which couples the child and parent
# models, use the xt3d option, which is an alternative to the
# ghost-node correction. This xt3d option was added as a new
# capability for the gwt-gwt and gwf-gwf exchanges in MODFLOW version 6.3.0.
exg = flopy.mf6.ModflowGwfgwf(
    sim,
    exgtype="GWF6-GWF6",
    xt3d=True,
    auxiliary=["angldegx", "cdist"],
    exgmnamea=pname,
    exgmnameb=cname,
    nexg=nexg,
    exchangedata=exchangedata,
)

# Set up the parent model and use the lgr.parent object to
# help provide the necessary information.
lgrp = lgr.parent
gwfp = flopy.mf6.ModflowGwf(sim, modelname=pname, save_flows=True)
dis = flopy.mf6.ModflowGwfdis(gwfp, **lgrp.get_gridprops_dis6())
ic = flopy.mf6.ModflowGwfic(gwfp)
npf = flopy.mf6.ModflowGwfnpf(gwfp, save_specific_discharge=True)
chdspd = [[(0, 0, 0), 1.0], [(0, lgrp.nrow - 1, lgrp.ncol - 1), 0.0]]
chd = flopy.mf6.ModflowGwfchd(gwfp, stress_period_data=chdspd)
oc = flopy.mf6.ModflowGwfoc(
    gwfp,
    budget_filerecord=pname + ".bud",
    head_filerecord=pname + ".hds",
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
)

# Set up the child model and use the lgr.child object to
# help provide the necessary information.
lgrc = lgr.child
gwfc = flopy.mf6.ModflowGwf(sim, modelname=cname, save_flows=True)
dis = flopy.mf6.ModflowGwfdis(gwfc, **lgrc.get_gridprops_dis6())
ic = flopy.mf6.ModflowGwfic(gwfc)
npf = flopy.mf6.ModflowGwfnpf(gwfc, save_specific_discharge=True)
oc = flopy.mf6.ModflowGwfoc(
    gwfc,
    budget_filerecord=cname + ".bud",
    head_filerecord=cname + ".hds",
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
)

```

```

[14]: # now set up transport model
include_transport = True
if include_transport:
    pname = "parent-tran"
    cname = "child-tran"

```

(continues on next page)

(continued from previous page)

```

# retrieve the exchange data from the lgr object
exchangedata = lgr.get_exchange_data(angldegx=True, cdist=True)
nexg = len(exchangedata)

# When creating the exchange, which couples the child and parent
# models, use the xt3d option, which is an alternative to the
# ghost-node correction. This xt3d option was added as a new
# capability for the gwt-gwt and gwf-gwf exchanges in MODFLOW version 6.3.0.
exg = flopy.mf6.ModflowGwtgwt(
    sim,
    exgtype="GWT6-GWT6",
    gwfname1=gwfp.name,
    gwfname2=gwfc.name,
    # xt3d=True,
    auxiliary=["angldegx", "cdist"],
    exgmnamea=pname,
    exgmnameb=cname,
    nexg=nexg,
    exchangedata=exchangedata,
)

# Set up the parent model and use the lgr.parent object to
# help provide the necessary information.
lgrp = lgr.parent
gwtp = flopy.mf6.ModflowGwt(sim, modelname=pname, save_flows=True)
dis = flopy.mf6.ModflowGwtDis(gwtp, **lgrp.get_gridprops_dis6())
mst = flopy.mf6.ModflowGwtmst(gwtp, porosity=0.2)
ic = flopy.mf6.ModflowGwtic(gwtp)
adv = flopy.mf6.ModflowGwtadv(gwtp)
dsp = flopy.mf6.ModflowGwtdsp(gwtp, alh=1.0, ath1=0.1)
ssm = flopy.mf6.ModflowGwtssm(gwtp)
oc = flopy.mf6.ModflowGwtoc(
    gwtp,
    budget_filerecord=pname + ".bud",
    concentration_filerecord=pname + ".ucn",
    saverecord=[("CONCENTRATION", "ALL"), ("BUDGET", "ALL")],
)

# Set up the child model and use the lgr.child object to
# help provide the necessary information.
lgrc = lgr.child
gwtc = flopy.mf6.ModflowGwt(sim, modelname=cname, save_flows=True)
dis = flopy.mf6.ModflowGwtDis(gwtc, **lgrc.get_gridprops_dis6())
mst = flopy.mf6.ModflowGwtmst(gwtc, porosity=0.2)
ic = flopy.mf6.ModflowGwtic(gwtc)
adv = flopy.mf6.ModflowGwtadv(gwtc)
dsp = flopy.mf6.ModflowGwtdsp(gwtc, alh=1.0, ath1=0.1)
# ssm = flopy.mf6.ModflowGwtssm(gwtc)
cnccspd = {0: [(0, int(lgrc.nrow / 2), int(lgrc.ncol / 2)), 1000.0]}
cnc = flopy.mf6.ModflowGwtcnc(gwtc, stress_period_data=cnccspd)
oc = flopy.mf6.ModflowGwtoc(

```

(continues on next page)

(continued from previous page)

```

    gwtc,
    budget_filerecord=cname + ".bud",
    concentration_filerecord=cname + ".ucn",
    saverecord=[("CONCENTRATION", "ALL"), ("BUDGET", "ALL")],
    printrecord=[("CONCENTRATION", "LAST"), ("BUDGET", "ALL")],
)

ims_tran = flopy.mf6.ModflowIms(
    sim, linear_acceleration="BICGSTAB", filename="tran.ims"
)
sim.register_ims_package(ims_tran, [gwtp.name, gwtc.name])

# couple flow and transport models
gwfgwt_p = flopy.mf6.ModflowGwfgwt(
    sim,
    exgtype="GWF6-GWT6",
    exgmnamea=gwfp.name,
    exgmnameb=gwtp.name,
    filename="gwfp_gwtp.gwfgwt",
)
gwfgwt_c = flopy.mf6.ModflowGwfgwt(
    sim,
    exgtype="GWF6-GWT6",
    exgmnamea=gwfc.name,
    exgmnameb=gwtc.name,
    filename="gwfc_gwtc.gwfgwt",
)

```

```

[15]: # write and run simulation
sim.write_simulation(silent=True)
success, buff = sim.run_simulation(silent=True, report=True)
assert success, "Model did not run to completion."

```

```

[16]: # load and store the head arrays from the parent and child models
head = [gwfp.output.head().get_data(), gwfc.output.head().get_data()]
conc = [
    gwtp.output.concentration().get_data(),
    gwtc.output.concentration().get_data(),
]

# load and store the specific discharge results for the parent and child models
bud = gwfp.output.budget()
spdis = bud.get_data(text="DATA-SPDIS")[0]
spdisp = flopy.utils.postprocessing.get_specific_discharge(spdis, gwfp)
bud = gwfc.output.budget()
spdis = bud.get_data(text="DATA-SPDIS")[0]
spdisc = flopy.utils.postprocessing.get_specific_discharge(spdis, gwfc)

```

```

[17]: # plot the results from both the parent and child models
# on the same figure
f = plt.figure(figsize=(12, 10))

```

(continues on next page)

(continued from previous page)

```

ax = plt.subplot(1, 1, 1, aspect="equal")
pmvp = flopy.plot.PlotMapView(gwfp, ax=ax)
pmvc = flopy.plot.PlotMapView(gwfc, ax=ax, extent=pmvp.extent)

# color flood head
# pmvp.plot_array(head[0], vmin=0., vmax=1., masked_values=[1.e30])
# pmvc.plot_array(head[1], vmin=0., vmax=1.)

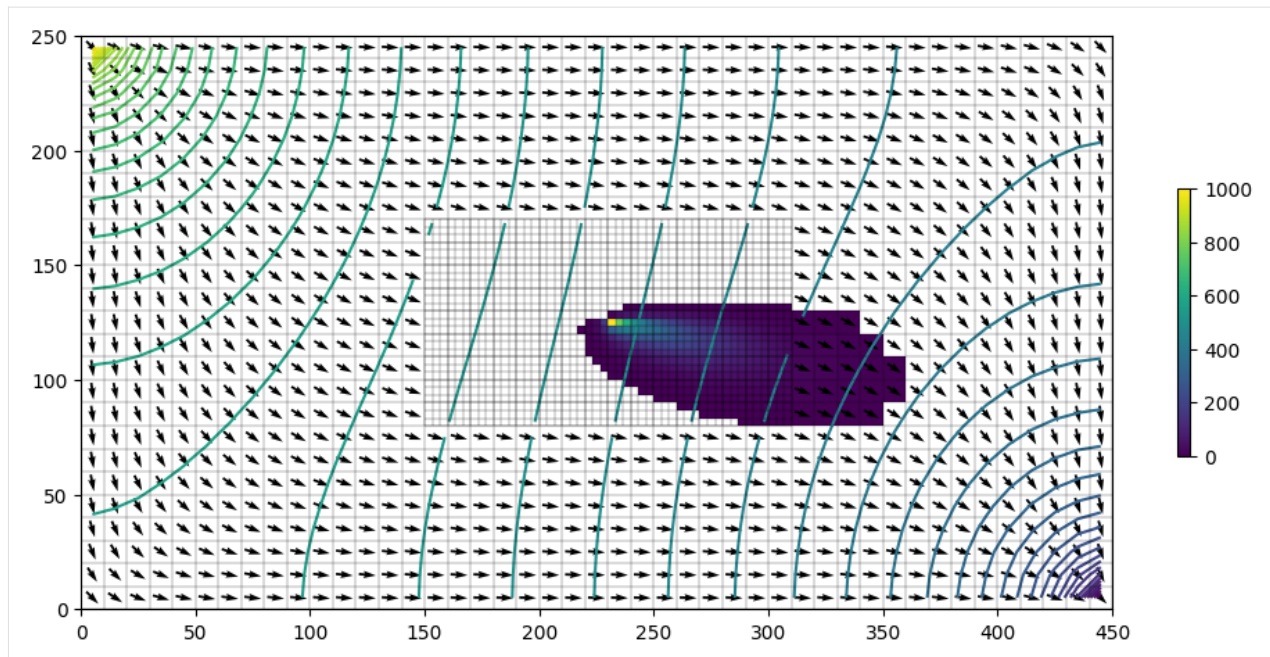
# contour head
cs = pmvp.contour_array(
    head[0], levels=np.linspace(0, 1), masked_values=[1.0e30]
)
cs = pmvc.contour_array(
    head[1], levels=np.linspace(0, 1), masked_values=[1.0e30]
)

# color flood concentrations
a1 = conc[0]
a2 = conc[1]
a1 = np.ma.masked_where(a1 < 1.0, a1)
a2 = np.ma.masked_where(a2 < 1.0, a2)
vmin = 0.0
vmax = 1000.0
pmvp.plot_array(a1, vmin=vmin, vmax=vmax, masked_values=[1.0e30])
pa = pmvc.plot_array(a2, vmin=vmin, vmax=vmax)
plt.colorbar(pa, shrink=0.25)

# draw parent and child grids
pmvp.plot_grid(colors="black", linewidths=0.2)
pmvc.plot_grid(colors="black", linewidths=0.2)

# plot vectors
qv = pmvp.plot_vector(spdisp[0], spdisp[1], normalize=True, color="black")
# qv = pmvc.plot_vector(spdisc[0], spdisc[1], normalize=True, color="white")

```



```
[18]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

5.5 MODFLOW-NWT

5.5.1 Working with MODFLOW-NWT v 1.1 option blocks

In MODFLOW-NWT an option block is present for the WEL file, UZF file, and SFR file. This block takes keyword arguments that are supplied in an option line in other versions of MODFLOW.

The OptionBlock class was created to provide compatibility with the MODFLOW-NWT option block and allow the user to easily edit values within the option block

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

from flopy.utils import OptionBlock
```

(continues on next page)

(continued from previous page)

```
print(sys.version)
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
flopy version: 3.4.3
```

```
[2]: load_ws = os.path.join("../", "../", "examples", "data", "options", "sagehen")

# temporary directory
temp_dir = TemporaryDirectory()
model_ws = os.path.join(temp_dir.name, "nwt_options", "output")
```

Loading a MODFLOW-NWT model that has option block options

It is critical to set the version flag in `flopy.modflow.Modflow.load()` to `version='mfnnwt'`

We are going to load a modified version of the Sagehen test problem from GSFLOW to illustrate compatibility

```
[3]: mfexe = "mfnnwt"

ml = flopy.modflow.Modflow.load(
    "sagehen.nam", model_ws=load_ws, exe_name=mfexe, version="mfnnwt"
)
ml.change_model_ws(new_pth=model_ws)
ml.write_input()

loading iuzfbnd array...
loading vks array...
loading eps array...
loading thts array...
stress period 1:
    loading finf array...
stress period 2:

creating model workspace...
../../../../../../../../tmp/tmpf10zd5qg/nwt_options/output
```

Let's look at the options attribute of the UZF object

The `uzf.options` attribute is an `OptionBlock` object. The representation of this object is the option block that will be written to output, which allows the user to easily check to make sure the block has the options they want.

```
[4]: uzf = ml.get_package("UZF")
uzf.options

[4]: OPTIONS
NOSURFLEAK
ETSQUARE 0.2
SAVEFINF
END
```

The `OptionBlock` object also has attributes which correspond to the option names listed in the online guide to modflow. The user can call and edit the options within the option block

```
[5]: print(uzf.options.nosurfleak)
      print(uzf.options.savefinf)
```

```
True
True
```

```
[6]: uzf.options.etsquare = False
      uzf.options
```

```
[6]: OPTIONS
      NOSURFLEAK
      SAVEFINF
      END
```

```
[7]: uzf.options.etsquare = True
      uzf.options
```

```
[7]: OPTIONS
      NOSURFLEAK
      ETSQUARE 0.2
      SAVEFINF
      END
```

The user can also see the single line representation of the options

```
[8]: uzf.options.single_line_options
```

```
[8]: 'NOSURFLEAK ETSQUARE 0.2 SAVEFINF'
```

And the user can easily change to single line options writing

```
[9]: uzf.options.block = False

      # write out only the uzf file
      uzf_name = "uzf_opt.uzf"
      uzf.write_file(os.path.join(model_ws, uzf_name))
```

Now let's examine the first few lines of the new UZF file

```
[10]: f = open(os.path.join(model_ws, uzf_name))
      for ix, line in enumerate(f):
          if ix == 3:
              break
          else:
              print(line)

      # UZF package for MODFLOW-NWT generated by Flopy 3.4.3
```

(continues on next page)

(continued from previous page)

```

NOSURFLEAK ETSQUARE 0.2 SAVEFINF
      3      1      0      0      0      0      15      100
↪4  1.0000000E+00 #NUZTOP IUZFOPT IRUNFLG IETFLG ipakcb IUZFCB2 NTRAIL NSETS NUZGAGES

```

And let's load the new UZF file

```

[11]: uzf2 = flopy.modflow.ModflowUzf1.load(
        os.path.join(model_ws, uzf_name), ml, check=False
    )

    loading iuzfbnd array...
    loading vks array...
    loading eps array...
    loading thts array...
stress period 1:
    loading finf array...
stress period 2:

/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/mbase.py:661:
↪UserWarning: Two packages of the same type, Replacing existing 'UZF' package.
    warn(

```

Now we can look at the options object, and check if it's block or line format

block=False indicates that options will be written as line format

```

[12]: print(uzf2.options)
      print(uzf2.options.block)

OPTIONS
NOSURFLEAK
ETSQUARE 0.2
SAVEFINF
END

False

```

Finally we can convert back to block format

```

[13]: uzf2.options.block = True
      uzf2.write_file(os.path.join(model_ws, uzf_name))
      ml.remove_package("UZF")

      uzf3 = flopy.modflow.ModflowUzf1.load(
          os.path.join(model_ws, uzf_name), ml, check=False
      )
      print("\n")
      print(uzf3.options)
      print(uzf3.options.block)

```

```
loading iuzfbnd array...
loading vks array...
loading eps array...
loading thts array...
stress period 1:
  loading finf array...
stress period 2:
```

```
OPTIONS
NOSURFLEAK
ETSQUARE 0.2
SAVEFINF
END
```

```
True
```

We can also look at the WEL object

```
[14]: wel = ml.get_package("WEL")
      wel.options
```

```
[14]: OPTIONS
      SPECIFY 0.1 90
      END
```

Let's write this out as a single line option block and examine the first few lines

```
[15]: wel_name = "wel_opt.wel"
      wel.options.block = False

      wel.write_file(os.path.join(model_ws, wel_name))

      f = open(os.path.join(model_ws, wel_name))
      for ix, line in enumerate(f):
          if ix == 4:
              break
          else:
              print(line)

# WEL package for MODFLOW-NWT generated by Flopy 3.4.3

          5          0 NOPRINT

SPECIFY 0.1 90

          5          0 # stress period 1
```

And we can load the new single line options WEL file and confirm that it is being read as an option line

```
[16]: ml.remove_package("WEL")
      wel2 = flopy.modflow.ModflowWel.load(
            os.path.join(model_ws, wel_name), ml, nper=ml.nper, check=False
        )

      wel2.options
      wel2.options.block

[16]: False
```

Building an OptionBlock from scratch

The user can also build an OptionBlock object from scratch to add to a ModflowSfr2, ModflowUzf1, or ModflowWel file.

The OptionBlock class has two required parameters and one optional parameter

option_line: a one line, string based representation of the options

package: a modflow package object

block: boolean flag for line based or block based options

```
[17]: opt_line = "specify 0.1 20"
      options = OptionBlock(opt_line, flopy.modflow.ModflowWel, block=True)
      options

[17]: OPTIONS
      SPECIFY 0.1 20
      END
```

from here we can set the noprint flag by using options.noprint

```
[18]: options.noprint = True
```

and the user can also add auxillary variables by using options.auxillary

```
[19]: options.auxillary = ["aux", "iface"]
```

Now we can create a new wel file using this OptionBlock

and write it to output

```
[20]: wel3 = flopy.modflow.ModflowWel(
        ml,
        stress_period_data=wel.stress_period_data,
        options=options,
        unitnumber=99,
    )

    wel3.write_file(os.path.join(model_ws, wel_name))

/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/mbase.py:661:
↳ UserWarning: Two packages of the same type, Replacing existing 'WEL' package.
    warn(
```

And now let's examine the first few lines of the file

```
[21]: f = open(os.path.join(model_ws, wel_name))
      for ix, line in enumerate(f):
          if ix == 8:
              break
          else:
              print(line)

# WEL package for MODFLOW-NWT generated by Flopy 3.4.3

OPTIONS

SPECIFY 0.1 20

END

          5          0 NOPRINT AUX IFACE

          5          0 # stress period 1

          1          35          12          20.0

          1          36          13          21.0
```

We can see that everything that the OptionBlock class writes out options in the correct location.

The user can also switch the options over to option line style and write out the output too!

```
[22]: wel3.options.block = False
      wel3.write_file(os.path.join(model_ws, wel_name))

f = open(os.path.join(model_ws, wel_name))
for ix, line in enumerate(f):
    if ix == 6:
        break
    else:
        print(line)

# WEL package for MODFLOW-NWT generated by Flopy 3.4.3

          5          0 NOPRINT AUX IFACE

SPECIFY 0.1 20

          5          0 # stress period 1

          1          35          12          20.0

          1          36          13          21.0
```

```
[23]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass
```

5.6 MT3DMS

5.6.1 MT3D-USGS: Transport with the SFR/LAK/UZF Packages (SFT/LKT/UZT), and chemical reactions (RCT)

A more comprehensive demonstration of setting up an MT3D-USGS model that uses all of the new packages included in the first release of MT3D-USGS. Also includes RCT.

Problem Description:

- 300 row x 300 col x 3 layer x 2 stress period model
- Flow model uses SFR, LAK, and UZF with connections between all three
- Transport model simulates streamflow transport (SFT), with connection to a single lake (LKT)
- Transport model simulates overland runoff and spring discharge (UZT) to surface water network

Start by importing some libraries:

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
```

(continues on next page)

(continued from previous page)

```
matplotlib version: 3.7.3
flopy version: 3.4.3
```

Create a MODFLOW model and store it, in this case in the variable ‘mf’. The modelname will be the name given to all MODFLOW files. The exe_name should be the name of the MODFLOW executable. In this case, we want to use version: ‘mfnwt’ for MODFLOW-NWT

```
[2]: # temporary directory
temp_dir = TemporaryDirectory()
model_ws = temp_dir.name

modelpth = os.path.join(model_ws, "no3")
modelname = "no3"
mfexe = "mfnwt"
mtexe = "mt3dusgs"

# Make sure modelpth directory exists
if not os.path.isdir(modelpth):
    os.makedirs(modelpth, exist_ok=True)

# Instantiate MODFLOW object in flopy
mf = flopy.modflow.Modflow(
    modelname=modelname, exe_name=mfexe, model_ws=modelpth, version="mfnwt"
)
```

Set up model discretization

```
[3]: Lx = 90000.0
Ly = 90000.0
nrow = 300
ncol = 300
nlay = 3

delr = Lx / ncol
delc = Ly / nrow

xmax = ncol * delr
ymax = nrow * delc

X, Y = np.meshgrid(
    np.linspace(delr / 2, xmax - delr / 2, ncol),
    np.linspace(ymax - delc / 2, 0 + delc / 2, nrow),
)
```


Instantiate output control (oc) package for MODFLOW-NWT

```
[4]: oc = flopy.modflow.ModflowOc(mf)
```

Instantiate solver package for MODFLOW-NWT

```
[5]: # Newton-Raphson Solver: Create a flopy nwt package object

headtol = 1.0e-4
flxutol = 5
maxiterout = 5000
thickfact = 1e-06
linmeth = 2
iprnwt = 1
ibotav = 1

nwt = flopy.modflow.ModflowNwt(
    mf,
    headtol=headtol,
    flxutol=flxutol,
    maxiterout=maxiterout,
    thickfact=thickfact,
    linmeth=linmeth,
    iprnwt=iprnwt,
    ibotav=ibotav,
    options="SIMPLE",
)
```

Instantiate discretization (DIS) package for MODFLOW-NWT

```
[6]: elv_pth = os.path.join(
    "..",
    "..",
    "examples",
    "data",
    "mt3d_example_sft_lkt_uzt",
    "dis_arrays",
    "grnd_elv.txt",
)

# Top of Layer 1 elevation determined using GW Vistas and stored locally
grndElv = np.loadtxt(elv_pth)

# Bottom of layer 1 elevation also determined from use of GUI and stored locally
bt1_pth = os.path.join(
    "..",
    "..",
    "examples",
    "data",
```

(continues on next page)

(continued from previous page)

```

    "mt3d_example_sft_lkt_uzt",
    "dis_arrays",
    "bot1.txt",
)
bot1Elv = np.loadtxt(bt1_pth)

bot2Elv = np.ones(bot1Elv.shape) * 100
bot3Elv = np.zeros(bot2Elv.shape)

botm = [bot1Elv, bot2Elv, bot3Elv]
botm = np.array(botm)
Steady = [False, False]
nstp = [1, 1]
tsmult = [1.0, 1.0]

# Stress periods
perlen = [9131.25, 9131.25]

# Create the discretization object
# itmuni = 4 (days); lenuni = 1 (feet)
dis = flopy.modflow.ModflowDis(
    mf,
    nlay,
    nrow,
    ncol,
    nper=2,
    delr=delr,
    delc=delc,
    top=grndElv,
    botm=botm,
    laycbd=0,
    itmuni=4,
    lenuni=1,
    steady=Steady,
    nstp=nstp,
    tsmult=tsmult,
    perlen=perlen,
)

```

Instantiate upstream weighting (UPW) flow package for MODFLOW-NWT

[7]: *# UPW must be instantiated after DIS. Otherwise, during the mf.write_input() procedures, # flopy will crash.*

```

# First line of UPW input is: IUPWCB HDRY NPUPW IPHDry
hdry = -1.00e30
iphdry = 0

# Next variables are: LAYTYP, LAYAVG, CHANI, LAYVKA, LAYWET
laytyp = [1, 3, 3] # >0: convertible

```

(continues on next page)

(continued from previous page)

```

layavg = 0 # 0: harmonic mean
chani = 1.0 # >0: CHANI is the horizontal anisotropy for the entire layer
layvka = 0 # =0: indicates VKA is vertical hydraulic conductivity
laywet = 0 # Always set equal to zero in UPW package
hk = 20
# hani = 1 # Not needed because CHANI > 1
vka = 0.5 # Is equal to vert. K b/c LAYVKA = 0
ss = 0.00001
sy = 0.20

upw = flopy.modflow.ModflowUpw(
    mf,
    laytyp=laytyp,
    layavg=layavg,
    chani=chani,
    layvka=layvka,
    laywet=laywet,
    ipakcb=53,
    hdry=hdry,
    iphdry=iphdry,
    hk=hk,
    vka=vka,
    ss=ss,
    sy=sy,
)

```

Instantiate basic (BAS or BA6) package for MODFLOW-NWT

```

[8]: ibnd1_pth = os.path.join(
    "..",
    "..",
    "examples",
    "data",
    "mt3d_example_sft_lkt_uzt",
    "bas_arrays",
    "ibnd_lay1.txt",
)
ibnd1 = np.loadtxt(ibnd1_pth)
ibnd2 = np.ones(ibnd1.shape)
ibnd3 = np.ones(ibnd2.shape)

ibnd = [ibnd1, ibnd2, ibnd3]
ibnd = np.array(ibnd)

StHd1_pth = os.path.join(
    "..",
    "..",
    "examples",
    "data",
    "mt3d_example_sft_lkt_uzt",
)

```

(continues on next page)

(continued from previous page)

```

    "bas_arrays",
    "strthd1.txt",
)
StHd1 = np.loadtxt(StHd1_pth)

StHd2_pth = os.path.join(
    "..",
    "..",
    "examples",
    "data",
    "mt3d_example_sft_lkt_uzt",
    "bas_arrays",
    "strthd2.txt",
)
StHd2 = np.loadtxt(StHd2_pth)

StHd3_pth = os.path.join(
    "..",
    "..",
    "examples",
    "data",
    "mt3d_example_sft_lkt_uzt",
    "bas_arrays",
    "strthd3.txt",
)
StHd3 = np.loadtxt(StHd3_pth)

strtElev = [StHd1, StHd2, StHd3]
strtElev = np.array(strtElev)

hdry = 999.0

bas = flopy.modflow.ModflowBas(mf, ibound=ibnd, hnoflo=hdry, strt=strtElev)

```

Instantiate general head boundary (GHB) package for MODFLOW-NWT

```

[9]: # GHB boundaries are located along the top (north) and bottom (south)
     # edges of the domain, all 3 layers.

     elev_stpt_row1 = 308.82281
     elev_stpt_row300 = 239.13811
     elev_slp = (308.82281 - 298.83649) / (ncol - 1)

     sp = []
     for k in [0, 1, 2]: # These indices need to be adjusted for 0-based moronicism
         for i in [
             0,
             299,
         ]: # These indices need to be adjusted for 0-based silliness
             for j in np.arange(

```

(continues on next page)

(continued from previous page)

```

0, 300, 1
): # These indices need to be adjusted for 0-based foolishness
   # Skipping cells not satisfying the conditions below
   if (i == 1 and (j < 27 or j > 31)) or (
       i == 299 and (j < 26 or j > 31)
   ):
       if i % 2 == 0:
           sp.append(
               [
                   k,
                   i,
                   j,
                   elev_stpt_row1 - (elev_slp * (j - 1)),
                   11.3636,
               ]
           )
       else:
           sp.append(
               [
                   k,
                   i,
                   j,
                   elev_stpt_row300 - (elev_slp * (j - 1)),
                   11.3636,
               ]
           )

for k in [0, 1, 2]:
    for j in np.arange(26, 32, 1):
        sp.append([k, 299, j, 238.20, 3409.0801])

ghb = flopy.modflow.ModflowGhb(mf, stress_period_data=sp)

```

Instantiate streamflow routing (SFR2) package for MODFLOW-NWT

```

[10]: # Read pre-prepared reach data into numpy recarrays using numpy.genfromtxt()
      # Remember that the cell indices stored in the pre-prepared NO3_ReachInput.csv file are
      ↪ based on 0-based indexing.
      # Flopy will convert to 1-based when it writes the files

      rpth = os.path.join(
          "..",
          "..",
          "examples",
          "data",
          "mt3d_example_sft_lkt_uzt",
          "sfr_data",
          "no3_reachinput.csv",
      )

```

(continues on next page)

(continued from previous page)

```

reach_data = np.genfromtxt(rpth, delimiter=",", names=True)
reach_data

# Read pre-prepared segment data into numpy recarrays using numpy.genfromtxt()

spth = os.path.join(
    "..",
    "..",
    "examples",
    "data",
    "mt3d_example_sft_lkt_uzt",
    "sfr_data",
    "no3_segmentdata.csv",
)
ss_segment_data = np.genfromtxt(spth, delimiter=",", names=True)
segment_data = {0: ss_segment_data, 1: ss_segment_data}
segment_data[0][0:1]["width1"]

nstrm = len(reach_data)
nss = len(segment_data[0])
nsfrpar = 0
const = 128390.4 # constant for manning's equation, units of cfs
dleak = 0.0001
ipakcb = 53 # flag for writing SFR output to cell-by-cell budget (on unit 53)
istcb2 = 37 # flag for writing SFR output to text file
isfropt = 1
dataset_5 = {
    0: [nss, 0, 0],
    1: [-1, 0, 0],
} # dataset 5 (see online guide) (ITMP, IRDFLG, IPTFLG)

# Input arguments generally follow the variable names defined in the Online Guide to
# MODFLOW
sfr = flopy.modflow.ModflowSfr2(
    mf,
    nstrm=nstrm,
    nss=nss,
    const=const,
    dleak=dleak,
    ipakcb=ipakcb,
    istcb2=istcb2,
    isfropt=isfropt,
    reachinput=True,
    reach_data=reach_data,
    segment_data=segment_data,
    dataset_5=dataset_5,
    unit_number=15,
)

```

Instantiate Lake (LAK) package for MODFLOW-NWT

```
[11]: # Read pre-prepared lake arrays
LakArr_pth = os.path.join(
    ".",
    ".",
    "examples",
    "data",
    "mt3d_example_sft_lkt_uzt",
    "lak_arrays",
    "lakarr1.txt",
)
LakArr_lyr1 = np.loadtxt(LakArr_pth)
LakArr_lyr2 = np.zeros(LakArr_lyr1.shape)
LakArr_lyr3 = np.zeros(LakArr_lyr2.shape)

LakArr = [LakArr_lyr1, LakArr_lyr2, LakArr_lyr3]
LakArr = np.array(LakArr)

nlakes = int(np.max(LakArr))
ipakcb = ipakcb # From above
theta = -1.0 # Implicit
nssitr = 10 # Maximum number of iterations for Newton's method
sscnr = 1.000e-03 # Convergence criterion for equilibrium lake stage solution
surfdep = 2.000e00 # Height of small topological variations in lake-bottom
stages = 268.00 # Initial stage of each lake at the beginning of the run

# ITMP > 0, read lake definition data
# ITMP1 0, read new recharge, evaporation, runoff, and withdrawal data for each lake
# LWRT > 0, suppresses printout from the lake package

bdlknc_lyr1 = LakArr_lyr1.copy()
bdlknc_lyr2 = LakArr_lyr1.copy()
bdlknc_lyr3 = np.zeros((LakArr_lyr1.shape))

# Need to expand bdlknc_lyr1 non-zero values by 1 in either direction
# (left/right and up/down)
for i in np.arange(0, LakArr_lyr1.shape[0]):
    for j in np.arange(0, LakArr_lyr1.shape[1]):
        im1 = i - 1
        ip1 = i + 1
        jm1 = j - 1
        jp1 = j + 1

        if im1 >= 0:
            if LakArr_lyr1[i, j] == 1 and LakArr_lyr1[im1, j] == 0:
                bdlknc_lyr1[im1, j] = 1

        if ip1 < LakArr_lyr1.shape[0]:
            if LakArr_lyr1[i, j] == 1 and LakArr_lyr1[ip1, j] == 0:
                bdlknc_lyr1[ip1, j] = 1
```

(continues on next page)

(continued from previous page)

```

    if jm1 >= 0:
        if LakArr_lyr1[i, j] == 1 and LakArr_lyr1[i, jm1] == 0:
            bdlknc_lyr1[i, jm1] = 1

    if jp1 < LakArr_lyr1.shape[1]:
        if LakArr_lyr1[i, j] == 1 and LakArr_lyr1[i, jp1] == 0:
            bdlknc_lyr1[i, jp1] = 1

bdlknc = [bdlknc_lyr1, bdlknc_lyr2, bdlknc_lyr3]
bdlknc = np.array(bdlknc)

flux_data = {0: [[0.0073, 0.0073, 0.0, 0.0]], 1: [[0.0073, 0.0073, 0.0, 0.0]]}

lak = flopy.modflow.ModflowLak(
    mf,
    nlakes=nlakes,
    ipakcb=ipakcb,
    theta=theta,
    nssitr=nssitr,
    sscncr=sscncr,
    surfdep=surfdep,
    stages=stages,
    lakarr=LakArr,
    bdlknc=bdlknc,
    flux_data=flux_data,
    unit_number=16,
)

```

Instantiate gage package for use with MODFLOW-NWT package

```

[12]: gages = [
    [1, 225, 90, 3],
    [2, 68, 91, 3],
    [3, 33, 92, 3],
    [4, 165, 93, 3],
    [5, 123, 94, 3],
    [6, 77, 95, 3],
    [7, 173, 96, 3],
    [8, 328, 97, 3],
    [9, 115, 98, 3],
    [-1, -101, 1],
]

# gages = [[1,38,61,1],[2,67,62,1], [3,176,63,1], [4,152,64,1], [5,186,65,1], [6,31,66,
↪ 1]]
files = [
    "no3.gag",
    "seg1_gag.out",
    "seg2_gag.out",

```

(continues on next page)

(continued from previous page)

```

    "seg3_gag.out",
    "seg4_gag.out",
    "seg5_gag.out",
    "seg6_gag.out",
    "seg7_gag.out",
    "seg8_gag.out",
    "seg9_gag.out",
    "lak1_gag.out",
]

numgage = len(gages)
gage = flopy.modflow.ModflowGage(
    mf, numgage=numgage, gage_data=gages, filenames=files
)

```

Instantiate Unsaturated-Zone Flow (UZF) package for MODFLOW-NWT

```

[13]: nuztop = 2
      iuzfopt = 2
      irunflg = 1
      ietflg = 0
      iuzfcb = 52
      iuzfcb2 = 0
      ntrail2 = 20
      nsets2 = 20
      nuzgag = 2
      surfdep = 2.0

      eps = 3.0
      thts = 0.30
      thti = 0.13079

      fname_uzbnd = os.path.join(
          "..",
          "..",
          "examples",
          "data",
          "mt3d_example_sft_lkt_uzt",
          "uzf_arrays",
          "iuzbnd.txt",
      )
      fname_runbnd = os.path.join(
          "..",
          "..",
          "examples",
          "data",
          "mt3d_example_sft_lkt_uzt",
          "uzf_arrays",
          "irunbnd.txt",
      )

```

(continues on next page)

(continued from previous page)

```

iuzfbnd = np.loadtxt(fname_uzbnd)
irunbnd = np.loadtxt(fname_runbnd)

uzgag = [[106, 160, 121, 3], [1, 1, -122, 1]]

finf = {0: 1.8250e-03, 1: 1.8250e-03}

uzf = flopy.modflow.ModflowUzf1(
    mf,
    nuztop=nuztop,
    iuzfopt=iuzfopt,
    irunflg=irunflg,
    ietflg=ietflg,
    ipakcb=iuzfcb,
    iuzfcb2=iuzfcb2,
    ntrail2=ntrail2,
    nsets=nsets2,
    surfdep=surfdep,
    uzgag=uzgag,
    iuzfbnd=1,
    irunbnd=0,
    vks=1.0e-6,
    eps=3.5,
    thts=0.35,
    thtr=0.15,
    thti=0.20,
)

```

Instantiate Drain (DRN) package for MODFLOW-NWT

```

[14]: fname_drnElv = os.path.join(
    "..",
    "..",
    "examples",
    "data",
    "mt3d_example_sft_lkt_uzt",
    "drn_arrays",
    "elv.txt",
)
fname_drnCond = os.path.join(
    "..",
    "..",
    "examples",
    "data",
    "mt3d_example_sft_lkt_uzt",
    "drn_arrays",
    "cond.txt",
)

```

(continues on next page)

(continued from previous page)

```

drnElv = np.loadtxt(fname_drnElv)
drnCond = np.loadtxt(fname_drnCond)

drnElv_lst = pd.DataFrame(
    {
        "lay": 1,
        "row": np.nonzero(drnElv)[0] + 1,
        "col": np.nonzero(drnElv)[1] + 1,
        "elv": drnElv[np.nonzero(drnElv)],
        "cond": drnCond[np.nonzero(drnCond)],
    },
    columns=["lay", "row", "col", "elv", "cond"],
)

# Convert the DataFrame into a list of lists for the drn constructor
stress_period_data = drnElv_lst.values.tolist()

# Create a dictionary, 1 entry for each of the two stress periods.
stress_period_data = {0: stress_period_data, 1: stress_period_data}

drn = flopy.modflow.ModflowDrn(
    mf, ipakcb=ipakcb, stress_period_data=stress_period_data
)

```

Instantiate linkage with mass transport routing (LMT) package for MODFLOW-NWT (generates linker file)

```

[15]: lmt = flopy.modflow.ModflowLmt(
    mf,
    output_file_name="NO3.ftl",
    output_file_header="extended",
    output_file_format="formatted",
    package_flows=["all"],
)

```

Now work on MT3D-USGS file creation

```

[16]: # Start by setting up MT3D-USGS class and pass in MODFLOW-NWT object for setting up a
      ↪ number of the BTN arrays

mt = flopy.mt3d.Mt3dms(
    modflowmodel=mf,
    modelname=modelname,
    model_ws=modelpth,
    version="mt3d-usgs",
    namefile_ext="mtnam",
    exe_name=mtexe,
    ftlfilename="no3.ftl",
)

```

(continues on next page)

(continued from previous page)

```
    ftlfree=True,  
)
```

Instantiate basic transport (BTN) package for MT3D-USGS

```
[17]: ncomp = 1  
      lunit = "FT"  
      sconc = 0.0  
      prsity = 0.3  
      cinact = -1.0  
      thkmin = 0.000001  
      nprs = -2  
      nprobs = 10  
      nprmas = 10  
      dt0 = 0.1  
      nstp = 1  
      mxstrn = 500  
      ttsmult = 1.2  
      ttsmax = 100  
  
      # These observations need to be entered with 0-based indexing  
      obs = [[0, 104, 158], [1, 104, 158], [2, 104, 158]]  
  
      btn = flopy.mt3d.Mt3dBtn(  
          mt,  
          MFStyleArr=True,  
          DRYCell=True,  
          lunit=lunit,  
          sconc=sconc,  
          ncomp=ncomp,  
          prsity=prsity,  
          cinact=cinact,  
          obs=obs,  
          thkmin=thkmin,  
          nprs=nprs,  
          nprobs=nprobs,  
          chkmas=True,  
          nprmas=nprmas,  
          dt0=dt0,  
          nstp=nstp,  
          mxstrn=mxstrn,  
          ttsmult=ttsmult,  
          ttsmax=ttsmax,  
      )
```

Instantiate advection (ADV) package for MT3D-USGS

```
[18]: mixelm = 0
      percel = 1.00000
      mxpart = 5000
      nadvfd = 1 # (1 = Upstream weighting)

      adv = flopy.mt3d.Mt3dAdv(
          mt, mixelm=mixelm, percel=percel, mxpart=mxpart, nadvfd=nadvfd
      )
```

Instantiate generalized conjugate gradient solver (GCG) package for MT3D-USGS

```
[19]: mxiter = 1
      iter1 = 50
      isolve = 3
      ncrs = 0
      accl = 1.0000000
      cclose = 1.000e-06
      iprgcg = 5

      gcg = flopy.mt3d.Mt3dGcg(
          mt,
          mxiter=mxiter,
          iter1=iter1,
          isolve=isolve,
          ncrs=ncrs,
          accl=accl,
          cclose=cclose,
          iprgcg=iprgcg,
      )
```

Instantiate dispersion (DSP) package for MT3D-USGS

```
[20]: al = 0.1 # longitudinal dispersivity
      trpt = 0.1 # ratio of the horizontal transverse dispersivity to 'AL'
      trpv = 0.1 # ratio of the vertical transverse dispersivity to 'AL'
      dmcoef = 1.00000e-10

      dsp = flopy.mt3d.Mt3dDsp(
          mt, al=al, trpt=trpt, trpv=trpv, dmcoef=dmcoef, multiDiff=True
      )
```

Instantiate source-sink mixing (SSM) package for MT3D-USGS

```
[21]: # no user-specified concentrations associated with boundary conditions

mxss = 11199

ssm = flopy.mt3d.Mt3dSsm(mt, mxss=mxss)
```

Instantiate reaction (RCT) package for MT3D-USGS

```
[22]: isothm = 0
       ireact = 1
       irctop = 2
       igetsc = 0
       ireaction = 0

rc1 = 6.3258e-04 # first-order reaction rate for the dissolved phase
rc2 = 0.0 # Decay on Soil Layer

rct = flopy.mt3d.Mt3dRct(
    mt, isothm=isothm, ireact=ireact, igetsc=igetsc, rc1=rc1, rc2=rc2
)
```

Instantiate streamflow transport (SFT) package for MT3D-USGS

```
[23]: nsfinit = len(reach_data)
       mxsfbc = len(reach_data)
       icbcsf = 0
       ioutobs = 92
       isfsolv = 1
       wimp = 0.5
       wups = 1.0
       cclosesf = 1.0e-6
       mxitersf = 10
       crntsf = 1.0
       iprtxmd = 0
       coldsf = 0
       dispsf = 0
       obs_sf = [225, 293, 326, 491, 614, 691, 864, 1192, 1307]
       sf_stress_period_data = {0: [0, 0, 0], 1: [0, 0, 0], 2: [0, 0, 0]}

       gage_output = [None, None, "no3.sftobs"]

       sft = flopy.mt3d.Mt3dSft(
           mt,
           nsfinit=nsfinit,
           mxsfbc=mxsfbc,
           icbcsf=icbcsf,
           ioutobs=ioutobs,
```

(continues on next page)

(continued from previous page)

```

isfsolv=isfsolv,
wimp=wimp,
wups=wups,
cclosesf=cclosesf,
mxitersf=mxitersf,
crntsf=crntsf,
iprtxmd=iprtxmd,
coldsf=coldsf,
dispsf=dispsf,
nobssf=len(obs_sf),
obs_sf=obs_sf,
sf_stress_period_data=sf_stress_period_data,
filenames=gage_output,
)

```

Instantiate unsaturated-zone transport (Uzt) package for MT3D-USGS

```

[24]: mxuzcon = np.count_nonzero(irunbnd)
icbcuz = 45
iet = 0
wc = np.ones((nlay, nrow, ncol)) * 0.29
sdh = np.ones((nlay, nrow, ncol))

uzt = flopy.mt3d.Mt3dUzt(
    mt,
    mxuzcon=mxuzcon,
    icbcuz=icbcuz,
    iet=iet,
    iuzfbnd=iuzfbnd,
    sdh=sdh,
    cuzinf=1.4158e-03,
    filenames="no3",
)

```

Instantiate lake transport (Lkt) package for MT3D-USGS

```

[25]: nlkinit = 1
mxlkbc = 720
icbclk = 81
ietlak = 1
coldlak = 1

lkt_flux_data = {0: [[0, 1, 0.01667]], 1: [[0, 1, 0.02667]]}

lkt = flopy.mt3d.Mt3dLkt(
    mt,
    nlkinit=nlkinit,
    mxlkbc=mxlkbc,

```

(continues on next page)

(continued from previous page)

```
icbclk=icbclk,  
ietlak=ietlak,  
coldlak=coldlak,  
lk_stress_period_data=lkt_flux_data,  
)
```

Write the MT3D-USGS input files for inspecting and running

```
[26]: mf.write_input()  
      mt.write_input()  
  
      # mf.run_model()  
      # mt.run_model()
```

5.6.2 Using FloPy to simplify the use of the MT3DMS SSM package

A multi-component transport demonstration

```
[1]: import os  
      import sys  
      from tempfile import TemporaryDirectory  
  
      import numpy as np  
  
      # run installed version of flopy or add local path  
      try:  
          import flopy  
      except:  
          fpth = os.path.abspath(os.path.join("..", ".."))  
          sys.path.append(fpth)  
          import flopy  
  
      print(sys.version)  
      print("numpy version: {}".format(np.__version__))  
      print("flopy version: {}".format(flopy.__version__))  
  
      3.8.18 (default, Aug 28 2023, 08:27:22)  
      [GCC 11.4.0]  
      numpy version: 1.24.4  
      flopy version: 3.4.3
```

First, we will create a simple model structure

```
[2]: nlay, nrow, ncol = 10, 10, 10  
      perlen = np.zeros((10), dtype=float) + 10  
      nper = len(perlen)  
  
      ibound = np.ones((nlay, nrow, ncol), dtype=int)
```

(continues on next page)

(continued from previous page)

```

botm = np.arange(-1, -11, -1)
top = 0.0

```

Create the MODFLOW packages

```

[3]: # temporary directory
temp_dir = TemporaryDirectory()
model_ws = temp_dir.name

modelname = "ssmex"
mf = flopy.modflow.Modflow(modelname, model_ws=model_ws)
dis = flopy.modflow.ModflowDis(
    mf,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    perlen=perlen,
    nper=nper,
    botm=botm,
    top=top,
    steady=False,
)
bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=top)
lpf = flopy.modflow.ModflowLpf(mf, hk=100, vka=100, ss=0.00001, sy=0.1)
oc = flopy.modflow.ModflowOc(mf)
pcg = flopy.modflow.ModflowPcg(mf)
rch = flopy.modflow.ModflowRch(mf)

```

We'll track the cell locations for the SSM data using the MODFLOW boundary conditions.

Get a dictionary (dict) that has the SSM itype for each of the boundary types.

```

[4]: itype = flopy.mt3d.Mt3dSsm.itype_dict()
print(itype)
print(flopy.mt3d.Mt3dSsm.get_default_dtype())
ssm_data = {}

{'CHD': 1, 'BAS6': 1, 'PBC': 1, 'WEL': 2, 'DRN': 3, 'RIV': 4, 'GHB': 5, 'MAS': 15, 'CC': 1,
'→ -1}
[('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('css', '<f4'), ('itype', '<i8')]

```

Add a general head boundary (ghb). The general head boundary head (bhead) is 0.1 for the first 5 stress periods with a component 1 (comp_1) concentration of 1.0 and a component 2 (comp_2) concentration of 100.0. Then bhead is increased to 0.25 and comp_1 concentration is reduced to 0.5 and comp_2 concentration is increased to 200.0

```

[5]: ghb_data = {}
print(flopy.modflow.ModflowGhb.get_default_dtype())
ghb_data[0] = [(4, 4, 4, 0.1, 1.5)]
ssm_data[0] = [(4, 4, 4, 1.0, itype["GHB"], 1.0, 100.0)]
ghb_data[5] = [(4, 4, 4, 0.25, 1.5)]
ssm_data[5] = [(4, 4, 4, 0.5, itype["GHB"], 0.5, 200.0)]

```

(continues on next page)

(continued from previous page)

```
for k in range(nlay):
    for i in range(nrow):
        ghb_data[0].append((k, i, 0, 0.0, 100.0))
        ssm_data[0].append((k, i, 0, 0.0, itype["GHB"], 0.0, 0.0))

ghb_data[5] = [(4, 4, 4, 0.25, 1.5)]
ssm_data[5] = [(4, 4, 4, 0.5, itype["GHB"], 0.5, 200.0)]
for k in range(nlay):
    for i in range(nrow):
        ghb_data[5].append((k, i, 0, -0.5, 100.0))
        ssm_data[5].append((k, i, 0, 0.0, itype["GHB"], 0.0, 0.0))

[('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('bhead', '<f4'), ('cond', '<f4')]
```

Add an injection well. The injection rate (flux) is 10.0 with a comp_1 concentration of 10.0 and a comp_2 concentration of 0.0 for all stress periods. WARNING: since we changed the SSM data in stress period 6, we need to add the well to the ssm_data for stress period 6.

```
[6]: wel_data = {}
print(flopy.modflow.ModflowWel.get_default_dtype())
wel_data[0] = [(0, 4, 8, 10.0)]
ssm_data[0].append((0, 4, 8, 10.0, itype["WEL"], 10.0, 0.0))
ssm_data[5].append((0, 4, 8, 10.0, itype["WEL"], 10.0, 0.0))

[('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('flux', '<f4')]
```

Add the GHB and WEL packages to the mf MODFLOW object instance.

```
[7]: ghb = flopy.modflow.ModflowGhb(mf, stress_period_data=ghb_data)
wel = flopy.modflow.ModflowWel(mf, stress_period_data=wel_data)
```

Create the MT3DMS packages

```
[8]: mt = flopy.mt3d.Mt3dms(modflowmodel=mf, modelname=modelname, model_ws=model_ws)
btn = flopy.mt3d.Mt3dBtn(mt, sconc=0, ncomp=2, sconc2=50.0)
adv = flopy.mt3d.Mt3dAdv(mt)
ssm = flopy.mt3d.Mt3dSsm(mt, stress_period_data=ssm_data)
gcg = flopy.mt3d.Mt3dGcg(mt)

found 'rch' in modflow model, resetting crch to 0.0
SSM: setting crch for component 2 to zero. kwarg name crch2
```

Let's verify that stress_period_data has the right dtype

```
[9]: print(ssm.stress_period_data.dtype)

[('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('css', '<f4'), ('itype', '<i8'), ('cssm(01)',
→ '<f4'), ('cssm(02)', '<f4')]
```

Create the SEAWAT packages

```
[10]: swt = flopy.seawat.Seawat(
        modflowmodel=mf,
        mt3dmodel=mt,
        modelname=modelname,
        namefile_ext="nam_swt",
        model_ws=model_ws,
    )
    vdf = flopy.seawat.SeawatVdf(swt, mtdnconc=0, iwttable=0, indense=-1)
```

```
[11]: mf.write_input()
        mt.write_input()
        swt.write_input()
```

And finally, modify the vdf package to fix indense.

```
[12]: fname = modelname + ".vdf"
        f = open(os.path.join(model_ws, fname), "r")
        lines = f.readlines()
        f.close()
        f = open(os.path.join(model_ws, fname), "w")
        for line in lines:
            f.write(line)
        for kper in range(nper):
            f.write("-1\n")
        f.close()
```

```
[13]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass
```

5.6.3 SEAWAT Tutorial 1: Henry Saltwater Intrusion Problem

In this tutorial, we will use FloPy to create, run, and post process the Henry saltwater intrusion problem using SEAWAT Version 4.

```
[1]: # ## Getting Started
        from pathlib import Path
        from tempfile import TemporaryDirectory
```

```
[2]: import numpy as np
```

```
[3]: import flopy
```

Input variables for the Henry Problem

```
[4]: Lx = 2.0
     Lz = 1.0
     nlay = 50
     nrow = 1
     ncol = 100
     delr = Lx / ncol
     delc = 1.0
     delv = Lz / nlay
     henry_top = 1.0
     henry_botm = np.linspace(henry_top - delv, 0.0, nlay)
     qinflow = 5.702 # m3/day
     dmcoef = 0.57024 # m2/day Could also try 1.62925 as another case of the Henry problem
     hk = 864.0 # m/day
```

Create the basic MODFLOW model structure

```
[5]: temp_dir = TemporaryDirectory()
     workspace = temp_dir.name
     name = "seawat_henry"
     swt = flopy.seawat.Seawat(name, exe_name="swtv4", model_ws=workspace)
     print(swt.namefile)

seawat_henry.nam
```

save cell fluxes to unit 53

```
[6]: ipakcb = 53
```

Add DIS package to the MODFLOW model

```
[7]: dis = flopy.modflow.ModflowDis(
     swt,
     nlay,
     nrow,
     ncol,
     nper=1,
     delr=delr,
     delc=delc,
     laycbd=0,
     top=henry_top,
     botm=henry_botm,
     perlen=1.5,
     nstp=15,
)
```

```
[8]: # Variables for the BAS package
     ibound = np.ones((nlay, nrow, ncol), dtype=np.int32)
     ibound[:, :, -1] = -1
```

Add BAS package to the MODFLOW model

```
[9]: bas = flopy.modflow.ModflowBas(swt, ibound, 0)
```

Add LPF package to the MODFLOW model

```
[10]: lpf = flopy.modflow.ModflowLpf(swt, hk=hk, vka=hk, ipakcb=ipakcb)
```

Add PCG Package to the MODFLOW model

```
[11]: pcg = flopy.modflow.ModflowPcg(swt, hclose=1.0e-8)
```

Add OC package to the MODFLOW model

```
[12]: oc = flopy.modflow.ModflowOc(
    swt,
    stress_period_data={(0, 0): ["save head", "save budget"]},
    compact=True,
)
```

Create WEL and SSM data

```
[13]: itype = flopy.mt3d.Mt3dSsm.itype_dict()
wel_data = {}
ssm_data = {}
wel_sp1 = []
ssm_sp1 = []
for k in range(nlay):
    wel_sp1.append([k, 0, 0, qinflow / nlay])
    ssm_sp1.append([k, 0, 0, 0.0, itype["WEL"]])
    ssm_sp1.append([k, 0, ncol - 1, 35.0, itype["BAS6"]])
wel_data[0] = wel_sp1
ssm_data[0] = ssm_sp1
wel = flopy.modflow.ModflowWel(swt, stress_period_data=wel_data, ipakcb=ipakcb)
```

Create the basic MT3DMS model structure

```
[14]: btn = flopy.mt3d.Mt3dBtn(
    swt,
    nprs=-5,
    prsity=0.35,
    sconc=35.0,
    ifmtcn=0,
    chkmas=False,
    nprobs=10,
    nprmas=10,
    dt0=0.001,
)
adv = flopy.mt3d.Mt3dAdv(swt, mixelm=0)
dsp = flopy.mt3d.Mt3dDsp(swt, al=0.0, trpt=1.0, trpv=1.0, dmcoef=dmcoef)
gcg = flopy.mt3d.Mt3dGcg(swt, iter1=500, mxiter=1, isolve=1, cclose=1e-7)
ssm = flopy.mt3d.Mt3dSsm(swt, stress_period_data=ssm_data)
```

Create the SEAWAT model structure

```
[15]: vdf = flopy.seawat.SeawatVdf(
        swt,
        iwtable=0,
        densemin=0,
        densemax=0,
        denseref=1000.0,
        denseslp=0.7143,
        firstdt=1e-3,
    )
```

Write the input files

```
[16]: swt.write_input()
```

Run the model

```
[17]: success, buff = swt.run_model(silent=True, report=True)
       assert success, "SEAWAT did not terminate normally."
```

Post-process the results

```
[18]: import numpy as np
```

```
[19]: import flopy.utils.binaryfile as bf
```

Load the concentration data

```
[20]: ucncobj = bf.UcnFile(Path(workspace) / "MT3D001.UCN", model=swt)
       times = ucncobj.get_times()
       concentration = ucncobj.get_data(totim=times[-1])
```

Load the cell-by-cell flow data

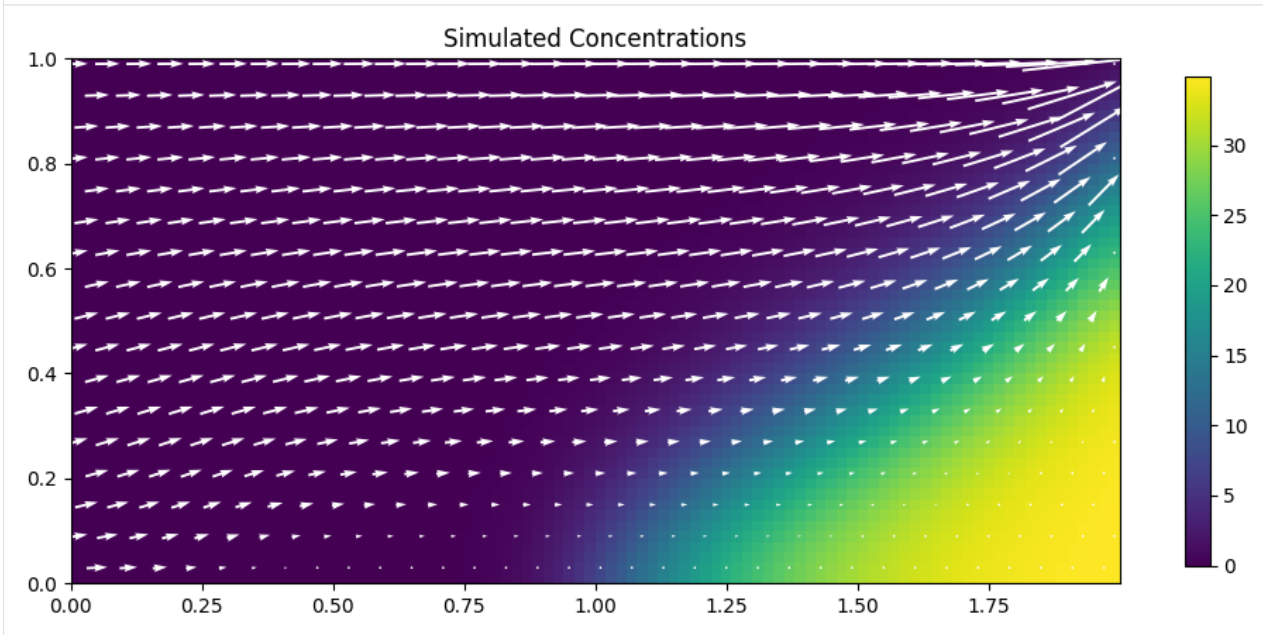
```
[21]: cbbobj = bf.CellBudgetFile(Path(workspace) / f"{name}.cbc")
       times = cbbobj.get_times()
       qx = cbbobj.get_data(text="flow right face", totim=times[-1])[0]
       qy = np.zeros((nlay, nrow, ncol), dtype=float)
       qz = cbbobj.get_data(text="flow lower face", totim=times[-1])[0]
```

Create a plot with concentrations and flow vectors

```
[22]: import matplotlib.pyplot as plt
```

```
[23]: fig = plt.figure(figsize=(12, 9))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
pmv = flopy.plot.PlotCrossSection(model=swt, ax=ax, line={"row": 0})
arr = pmv.plot_array(concentration)
pmv.plot_vector(qx, qy, -qz, color="white", kstep=3, hstep=3)
plt.colorbar(arr, shrink=0.5, ax=ax)
ax.set_title("Simulated Concentrations")
```

```
[23]: Text(0.5, 1.0, 'Simulated Concentrations')
```



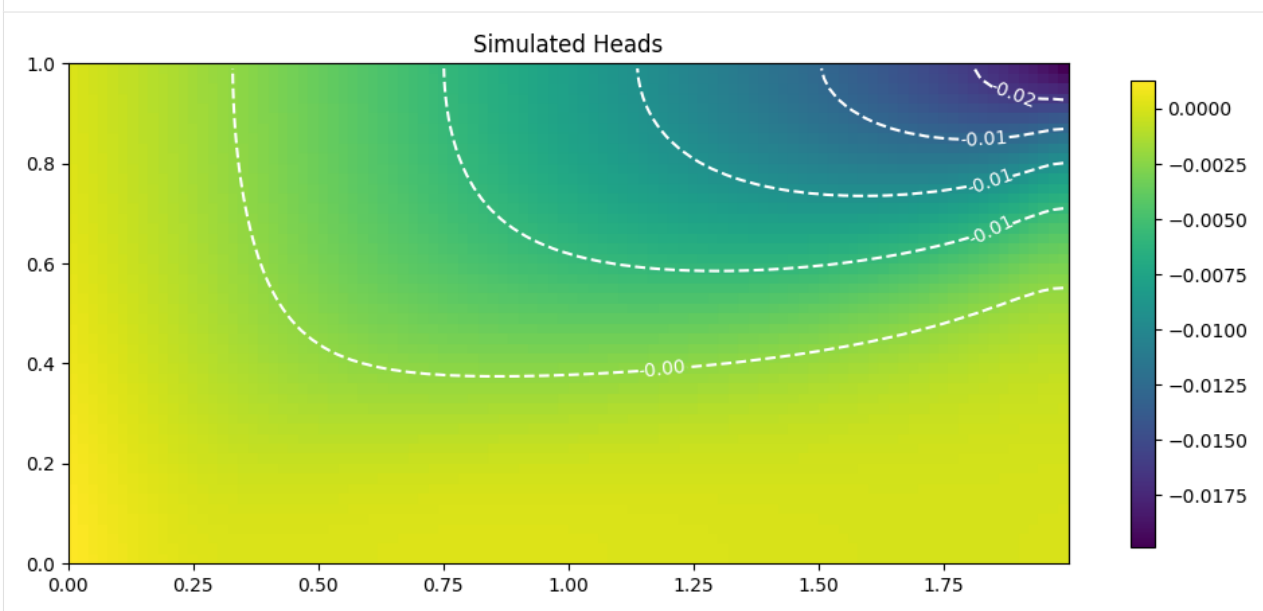
Load the head data

```
[24]: headobj = bf.HeadFile(Path(workspace) / f"{name}.hds")
times = headobj.get_times()
head = headobj.get_data(totim=times[-1])
```

Create a plot with heads

```
[25]: fig = plt.figure(figsize=(12, 9))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
pmv = flopy.plot.PlotCrossSection(model=swt, ax=ax, line={"row": 0})
arr = pmv.plot_array(head)
contours = pmv.contour_array(head, colors="white")
ax.clabel(contours, fmt="%2.2f")
plt.colorbar(arr, shrink=0.5, ax=ax)
ax.set_title("Simulated Heads")
```

```
[25]: Text(0.5, 1.0, 'Simulated Heads')
```



```
[26]: try:
        temp_dir.cleanup()
    except:
        # prevent windows permission error
        pass
```

5.7 PEST

5.7.1 Support for PEST

This notebook demonstrates the current parameter estimation functionality that is available with FloPy. The capability to write a simple template file for PEST is the only capability implemented so far. The plan is to develop functionality for creating PEST instruction files as well as the PEST control file.

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

import numpy as np

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
```

(continues on next page)

(continued from previous page)

```
print("numpy version: {}".format(np.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
flopy version: 3.4.3
```

This notebook will work with a simple model using the dimensions below

```
[2]: # temporary directory
temp_dir = TemporaryDirectory()
workspace = temp_dir.name

# Define the model dimensions
nlay = 3
nrow = 20
ncol = 20

# Create the flopy model object and add the dis and lpf packages
m = flopy.modflow.Modflow(modelname="mymodel", model_ws=workspace)
dis = flopy.modflow.ModflowDis(m, nlay, nrow, ncol)
lpf = flopy.modflow.ModflowLpf(m, hk=10.0)
```

Simple One Parameter Example

In order to create a PEST template file, we first need to define a parameter. For example, let's say we want to parameterize hydraulic conductivity, which is a static variable in flopy and MODFLOW. As a first step, let's define a parameter called HK_LAYER_1 and assign it to all of layer 1. We will not parameterize hydraulic conductivity for layers 2 and 3 and instead leave HK at its value of 10. (as assigned in the block above this one). We can do this as follows.

```
[3]: mfpkg = "lpf"
partype = "hk"
parname = "HK_LAYER_1"
idx = np.empty((nlay, nrow, ncol), dtype=bool)
idx[0] = True
idx[1:] = False

# The span variable defines how the parameter spans the package
span = {"idx": idx}

# These parameters have not affect yet, but may in the future
startvalue = 10.0
lbound = 0.001
ubound = 1000.0
transform = "log"

p = flopy.pest.Params(
    mfpkg, partype, parname, startvalue, lbound, ubound, span
)
```

At this point, we have enough information to write a PEST template file for the LPF package. We can do this using the following statement:

```
[4]: tw = flopy.pest.TemplateWriter(m, [p])
tw.write_template()
```

The following packages will be parameterized: ['LPF']

At this point, the lpf template file will have been created. The following block will print the template file.

```
[5]: lines = open(os.path.join(workspace, "mymodel.lpf.tpl"), "r").readlines()
for l in lines:
    print(l.strip())
```

```
ptf ~
# LPF package for MODFLOW-2005 generated by FloPy 3.4.3
0      -1E+30      0
0      0           0
0      0           0
1.0000000E+00    1.0000000E+00    1.0000000E+00
0      0           0
0      0           0
CONSTANT ~ HK_LAYER_1 ~      #hk Layer 1
CONSTANT      1.0000000E+00      #vka1
CONSTANT 10.0      #hk Layer 2
CONSTANT      1.0000000E+00      #vka2
CONSTANT 10.0      #hk Layer 3
CONSTANT      1.0000000E+00      #vka3
```

The span variable will also accept 'layers', in which the parameter applies to the list of layers, as shown next. When 'layers' is specified in the span dictionary, then the original hk value of 10. remains in the array, and the multiplier is specified on the array control line.

```
[6]: mfpkg = "lpf"
partype = "hk"
parname = "HK_LAYER_1-3"

# Span indicates that the hk parameter applies as a multiplier to layers 0 and 2,
# (MODFLOW layers 1 and 3)
span = {"layers": [0, 2]}

# These parameters have not affect yet, but may in the future
startvalue = 10.0
lbound = 0.001
ubound = 1000.0
transform = "log"

p = flopy.pest.Params(
    mfpkg, partype, parname, startvalue, lbound, ubound, span
)
tw = flopy.pest.templatewriter.TemplateWriter(m, [p])
tw.write_template()
```

The following packages will be parameterized: ['LPF']

[illegible]

5.7. PEST 329

(continued from previous page)

[illegible]

(continues on next page)

[illegible]

(continued from previous page)

10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
↵	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
↵	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
↵	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
↵	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
↵	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
↵	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
↵	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
↵	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
↵	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.0	↵
CONSTANT	1.000000E+00				#vka3					

Multiple Parameter Zoned Approach

The `params` module has a helper function called `zonearray2params` that will take a zone array and some other information and create a list of parameters, which can then be passed to the template writer. This next example shows how to create a slightly more complicated LPF template file in which both HK and VKA are parameterized.

```
[8]: # Create a zone array
zonearray = np.ones((nlay, nrow, ncol), dtype=int)
zonearray[0, 10:, 7:] = 2
zonearray[0, 15:, 9:] = 3
zonearray[1] = 4

[9]: # Create a list of parameters for HK
mfpackage = "lpf"
parzones = [2, 3, 4]
parvals = [56.777, 78.999, 99.0]
lbound = 5
ubound = 500
transform = "log"
plisthk = flopy.pest.zonearray2params(
    mfpackage, "hk", parzones, lbound, ubound, parvals, transform, zonearray
)
```

In this case, Flopy will create three parameters: `hk_2`, `hk_3`, and `hk_4`, which will apply to the horizontal hydraulic conductivity for cells in zones 2, 3, and 4, respectively. Only those zone numbers listed in `parzones` will be parameterized. For example, many cells in `zonearray` have a value of 1. Those cells will not be parameterized. Instead, their hydraulic conductivity values will remain fixed at the value that was specified when the Flopy LPF package was created.

```
[10]: # Create a list of parameters for VKA
parzones = [1, 2]
parvals = [0.001, 0.0005]
zonearray = np.ones((nlay, nrow, ncol), dtype=int)
zonearray[1] = 2
plistvk = flopy.pest.zonearray2params(
    mfpkg, "vka", parzones, lbound, ubound, parvals, transform, zonearray
)
```

```
[11]: # Combine the HK and VKA parameters together
plist = plsthk + plistvk
for p in plist:
    print(p.name, p.mfpkg, p.startvalue)

hk_2 lpf 56.777
hk_3 lpf 78.999
hk_4 lpf 99.0
vka_1 lpf 0.001
vka_2 lpf 0.0005
```

```
[12]: # Write the template file
tw = flopy.pest.templatewriter.TemplateWriter(m, plist)
tw.write_template()

The following packages will be parameterized: ['LPF']
```

```
[13]: # Print contents of template file
lines = open(os.path.join(workspace, "mymodel.lpf.tpl"), "r").readlines()
for l in lines:
    print(l.strip())
```

```
ptf ~
# LPF package for MODFLOW-2005 generated by Flopy 3.4.3
0      -1E+30      0
0      0      0
0      0      0
1.000000E+00  1.000000E+00  1.000000E+00
0      0      0
0      0      0
INTERNAL 1.0 (FREE) -1      #hk Layer 1
10.0      10.0      10.0      10.0      10.0      10.0      10.0
↪      10.0      10.0      10.0      10.0      10.0      10.0      10.0
10.0      10.0      10.0      10.0      10.0      10.0      10.0
↪      10.0      10.0      10.0      10.0      10.0      10.0      10.0
10.0      10.0      10.0      10.0      10.0      10.0      10.0
↪      10.0      10.0      10.0      10.0      10.0      10.0      10.0
10.0      10.0      10.0      10.0      10.0      10.0      10.0
↪      10.0      10.0      10.0      10.0      10.0      10.0      10.0
↪      10.0      10.0      10.0      10.0      10.0      10.0      10.0
```

(continues on next page)

(continued from previous page)

[illegible]

(continues on next page)

(continued from previous page)

```

10.0      10.0      10.0      10.0      10.0      10.0
↪      10.0 ~ hk_2 ~ ~ hk_2 ~ ~ hk_3 ~
~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_
↪3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~
10.0      10.0      10.0      10.0      10.0      10.0
↪      10.0 ~ hk_2 ~ ~ hk_2 ~ ~ hk_3 ~
~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_
↪3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~
10.0      10.0      10.0      10.0      10.0      10.0
↪      10.0 ~ hk_2 ~ ~ hk_2 ~ ~ hk_3 ~
~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_
↪3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~
10.0      10.0      10.0      10.0      10.0      10.0
↪      10.0 ~ hk_2 ~ ~ hk_2 ~ ~ hk_3 ~
~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_
↪3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~ ~ hk_3 ~
CONSTANT ~ vka_1 ~ #vka1
CONSTANT ~ hk_4 ~ #hk Layer 2
CONSTANT ~ vka_2 ~ #vka2
CONSTANT 10.0 #hk Layer 3
CONSTANT ~ vka_1 ~ #vka3

```

Two-Dimensional Transient Arrays

FloPy supports parameterization of transient two dimensional arrays, like recharge. This is similar to the approach for three dimensional static arrays, but there are some important differences in how span is specified. The parameter span here is also a dictionary, and it must contain a 'kper' key, which corresponds to a list of stress periods (zero based, of course) for which the parameter applies. The span dictionary must also contain an 'idx' key. If span['idx'] is None, then the parameter is a multiplier for those stress periods. If span['idx'] is a tuple (iarray, jarray), where iarray and jarray are a list of array indices, or a boolean array of shape (nrow, ncol), then the parameter applies only to the cells specified in idx.

```

[14]: # Define the model dimensions (made smaller for easier viewing)
nlay = 3
nrow = 5
ncol = 5
nper = 3

# Create the flopy model object and add the dis and lpf packages
m = flopy.modflow.Modflow(modelname="mymodel", model_ws=workspace)
dis = flopy.modflow.ModflowDis(m, nlay, nrow, ncol, nper=nper)
lpf = flopy.modflow.ModflowLpf(m, hk=10.0)
rch = flopy.modflow.ModflowRch(m, rech={0: 0.001, 2: 0.003})

```

Next, we create the parameters

```

[15]: plist = []

[16]: # Create a multiplier parameter for recharge
mfpackage = "rch"
partype = "rech"

```

(continues on next page)

(continued from previous page)

```

parname = "RECH_MULT"
startvalue = None
lbound = None
ubound = None
transform = None

# For a recharge multiplier, span[idx] must be None
idx = None
span = {"kpers": [0, 1, 2], "idx": idx}
p = flopy.pest.Params(
    mfpackage, partype, parname, startvalue, lbound, ubound, span
)
plist.append(p)

```

```

[17]: # Write the template file
tw = flopy.pest.TemplateWriter(m, plist)
tw.write_template()

```

The following packages will be parameterized: ['RCH']

```

[18]: # Print the results
lines = open(os.path.join(workspace, "mymodel.rch.tpl"), "r").readlines()
for l in lines:
    print(l.strip())

```

```

ptf ~
# RCH package for MODFLOW-2005 generated by Flopy 3.4.3
3      0
1      -1 # Stress period 1
INTERNAL ~ RECH_MULT ~ (FREE) -1      #rech_1
0.001      0.001      0.001      0.001      0.001
0.001      0.001      0.001      0.001      0.001
0.001      0.001      0.001      0.001      0.001
0.001      0.001      0.001      0.001      0.001
0.001      0.001      0.001      0.001      0.001
1      -1 # Stress period 2
INTERNAL ~ RECH_MULT ~ (FREE) -1      #rech_1
0.001      0.001      0.001      0.001      0.001
0.001      0.001      0.001      0.001      0.001
0.001      0.001      0.001      0.001      0.001
0.001      0.001      0.001      0.001      0.001
0.001      0.001      0.001      0.001      0.001
1      -1 # Stress period 3
INTERNAL ~ RECH_MULT ~ (FREE) -1      #rech_3
0.003      0.003      0.003      0.003      0.003
0.003      0.003      0.003      0.003      0.003
0.003      0.003      0.003      0.003      0.003
0.003      0.003      0.003      0.003      0.003
0.003      0.003      0.003      0.003      0.003

```

Multiplier parameters can also be combined with index parameters as follows.

```
[19]: plist = []

# Create a multiplier parameter for recharge
mfpackage = "rch"
partype = "rech"
parname = "RECH_MULT"
startvalue = None
lbound = None
ubound = None
transform = None

# For a recharge multiplier, span['idx'] must be None
span = {"kpers": [1, 2], "idx": None}
p = flopy.pest.Params(
    mfpackage, partype, parname, startvalue, lbound, ubound, span
)
plist.append(p)
```

```
[20]: # Now create an index parameter
mfpackage = "rch"
partype = "rech"
parname = "RECH_ZONE"
startvalue = None
lbound = None
ubound = None
transform = None

# For a recharge index parameter, span['idx'] must be a boolean array or tuple of array_
↳ indices
idx = np.empty((nrow, ncol), dtype=bool)
idx[0:3, 0:3] = True
span = {"kpers": [1], "idx": idx}
p = flopy.pest.Params(
    mfpackage, partype, parname, startvalue, lbound, ubound, span
)
plist.append(p)
```

```
[21]: # Write the template file
tw = flopy.pest.templatewriter.TemplateWriter(m, plist)
tw.write_template()

# Print the results
lines = open(os.path.join(workspace, "mymodel.rch.tpl"), "r").readlines()
for l in lines:
    print(l.strip())
```

The following packages will be parameterized: ['RCH']

```
ptf ~
# RCH package for MODFLOW-2005 generated by Flopy 3.4.3
3      0
1      -1 # Stress period 1
CONSTANT 1.000000E-03      #rech_1
```

(continues on next page)

(continued from previous page)

```

1      -1 # Stress period 2
INTERNAL ~ RECH_MULT ~ (FREE) -1      #rech_1
~ RECH_ZONE ~ ~ RECH_ZONE ~ ~ RECH_ZONE ~      0.001      0.001
~ RECH_ZONE ~ ~ RECH_ZONE ~ ~ RECH_ZONE ~      0.001      0.001
~ RECH_ZONE ~ ~ RECH_ZONE ~ ~ RECH_ZONE ~      0.001      0.001
0.001      0.001      0.001      0.001      0.001
0.001      0.001      0.001      0.001      0.001
1      -1 # Stress period 3
INTERNAL ~ RECH_MULT ~ (FREE) -1      #rech_3
0.003      0.003      0.003      0.003      0.003
0.003      0.003      0.003      0.003      0.003
0.003      0.003      0.003      0.003      0.003
0.003      0.003      0.003      0.003      0.003
0.003      0.003      0.003      0.003      0.003

```

```

[22]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
except:
    pass

```

EXAMPLES GALLERY

The following examples illustrate the functionality of FloPy. After the [tutorials](#), the examples are the best resource for learning the underlying capabilities of FloPy.

6.1 Preprocessing and Discretization

6.1.1 Triangular mesh example

First set the path and import the required packages. The flopy path doesn't have to be set if you install flopy from a binary installer. If you want to run this notebook, you have to set the path to your own flopy path.

```
[1]: import os
import sys
from pathlib import Path
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

import flopy

temp_dir = TemporaryDirectory()
workspace = Path(temp_dir.name)

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

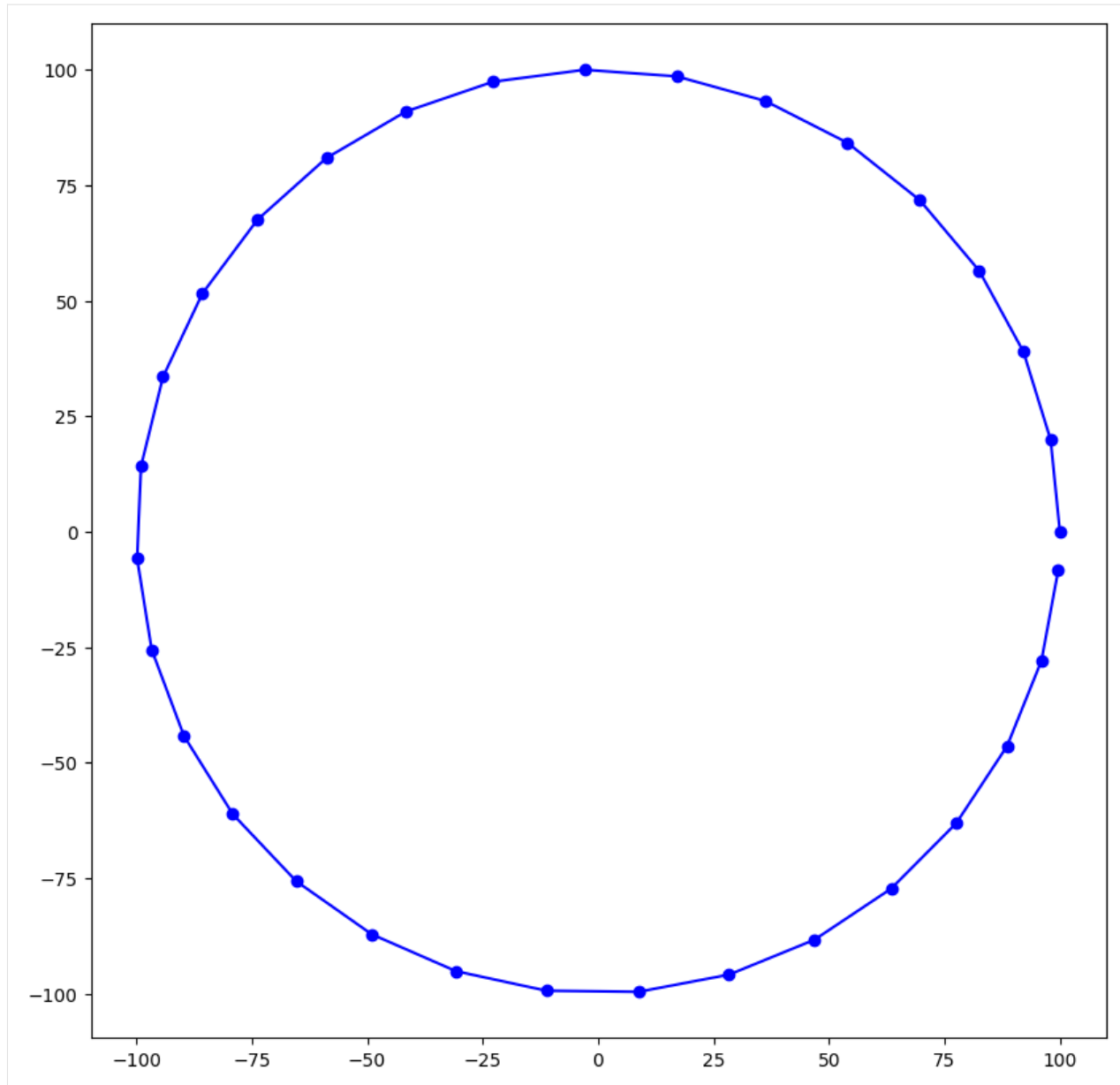
Creating Meshes with the Triangle Class

The FloPy Triangle class at (`flopy.utils.triangle.Triangle`) can be used to generate triangular meshes using the Triangle program (<https://www.cs.cmu.edu/~quake/triangle.html>). The Triangle class is a thin wrapper that builds input files for the Triangle program, reads Triangle output, and makes plots of the mesh. To use the Triangle class, the user must have an executable copy of the triangle program somewhere on their system.

Let's start by making a simple triangular mesh of a circle using the FloPy Triangle class and the triangle program.

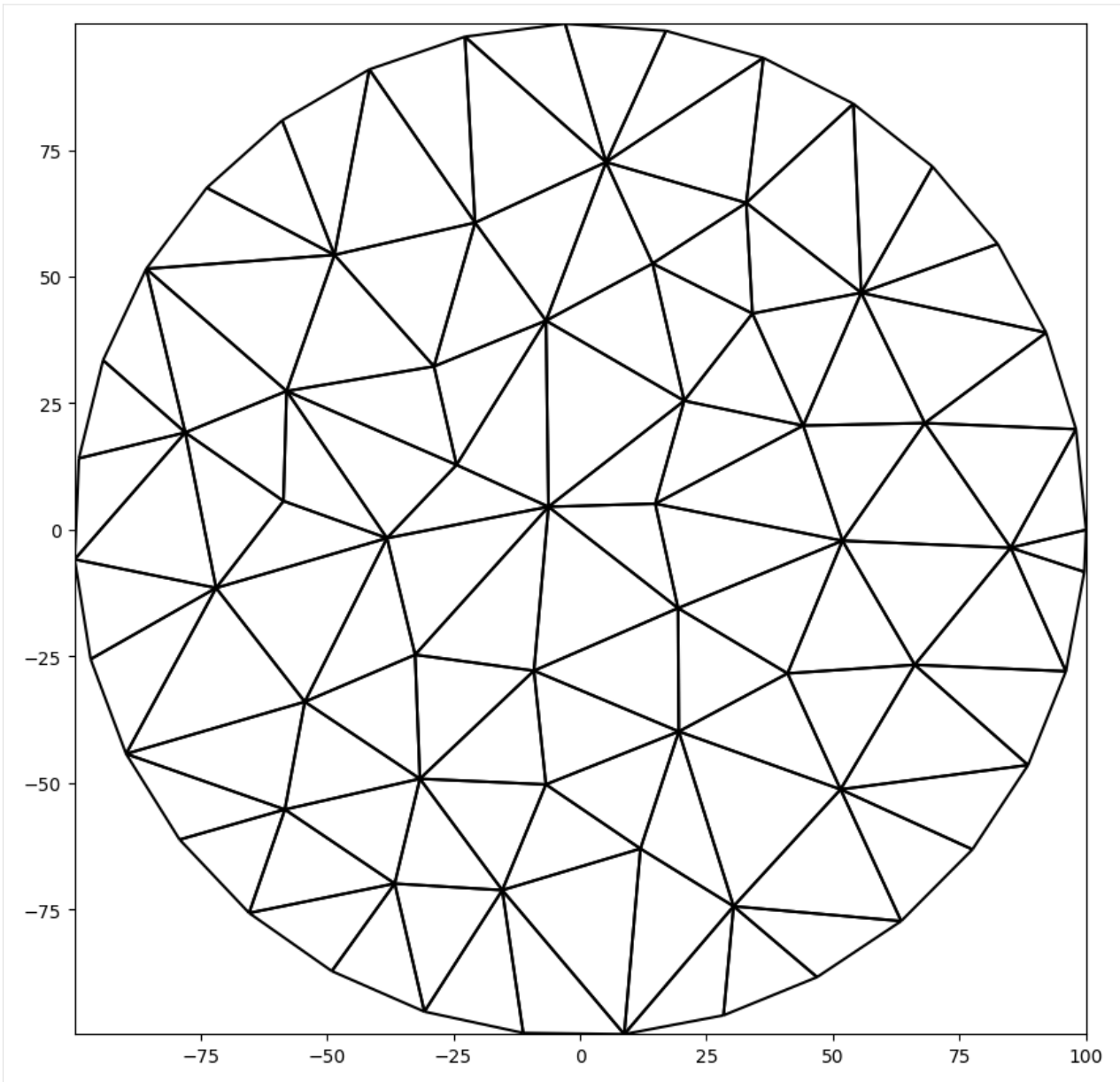
```
[2]: # we start by creating a polygon (circle_poly), which is a list of
      # (x,y) points that define the circle
      theta = np.arange(0.0, 2 * np.pi, 0.2)
      radius = 100.0
      x = radius * np.cos(theta)
      y = radius * np.sin(theta)
      circle_poly = [(x, y) for x, y in zip(x, y)]
      fig = plt.figure(figsize=(10, 10))
      ax = plt.subplot(1, 1, 1, aspect="equal")
      ax.plot(x, y, "bo-")
```

```
[2]: [<matplotlib.lines.Line2D at 0x7ff5069bd910>]
```



```
[3]: from flopy.utils.triangle import Triangle

# We can then use the Triangle class and Triangle program
# to make the mesh, as follows.
tri = Triangle(maximum_area=500, angle=30, model_ws=workspace)
tri.add_polygon(circle_poly)
tri.build(verbose=False)
fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
pc = tri.plot(ax=ax)
```



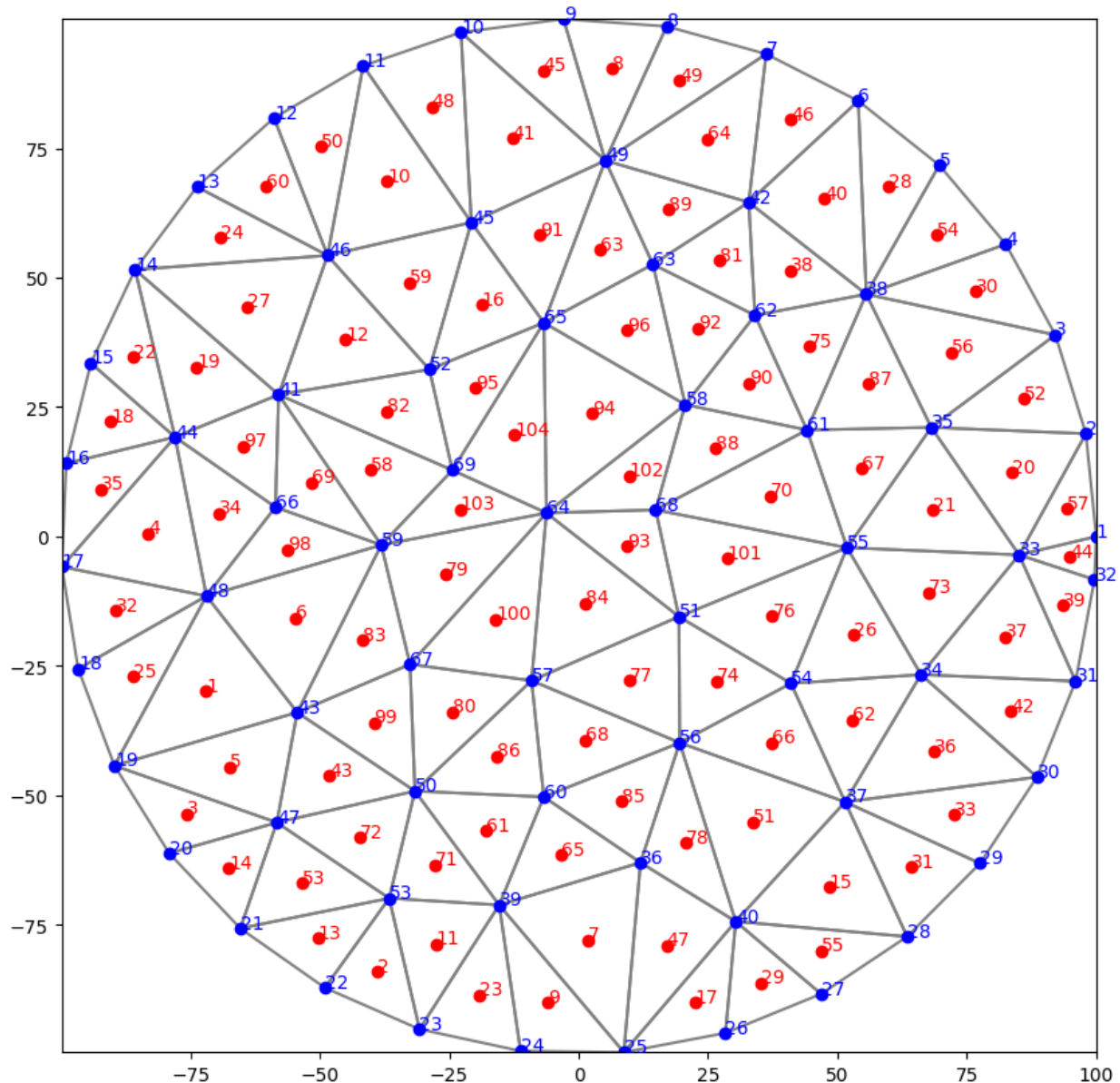
The Triangle class creates a .node and a .poly file as input for the Triangle program. The Triangle class then reads four output files from the Triangle program into numpy structured arrays. These four structured arrays are stored with the object as follows.

```
[4]: print(tri.node.dtype)
      print(tri.ele.dtype)
      print(tri.neigh.dtype)
      print(tri.edge.dtype)

[('ivert', '<i8'), ('x', '<f8'), ('y', '<f8'), ('boundary_marker', '<i8')]
[('icell', '<i8'), ('iv1', '<i8'), ('iv2', '<i8'), ('iv3', '<i8')]
[('icell', '<i8'), ('neighbor1', '<i8'), ('neighbor2', '<i8'), ('neighbor3', '<i8')]
[('iedge', '<i8'), ('endpoint1', '<i8'), ('endpoint2', '<i8'), ('boundary_marker', '<i8'
→ ')]
```



```
[5]: # We can also plot the cells and vertices and label them,
# but this really only works for coarse meshes
fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
tri.plot(ax=ax, edgecolor="gray")
tri.plot_vertices(ax=ax, marker="o", color="blue")
tri.label_vertices(ax=ax, fontsize=10, color="blue")
tri.plot_centroids(ax=ax, marker="o", color="red")
tri.label_cells(ax=ax, fontsize=10, color="red")
```



```
[6]: # What about a hole?
theta = np.arange(0.0, 2 * np.pi, 0.2)
radius = 30.0
x = radius * np.cos(theta) + 25.0
```

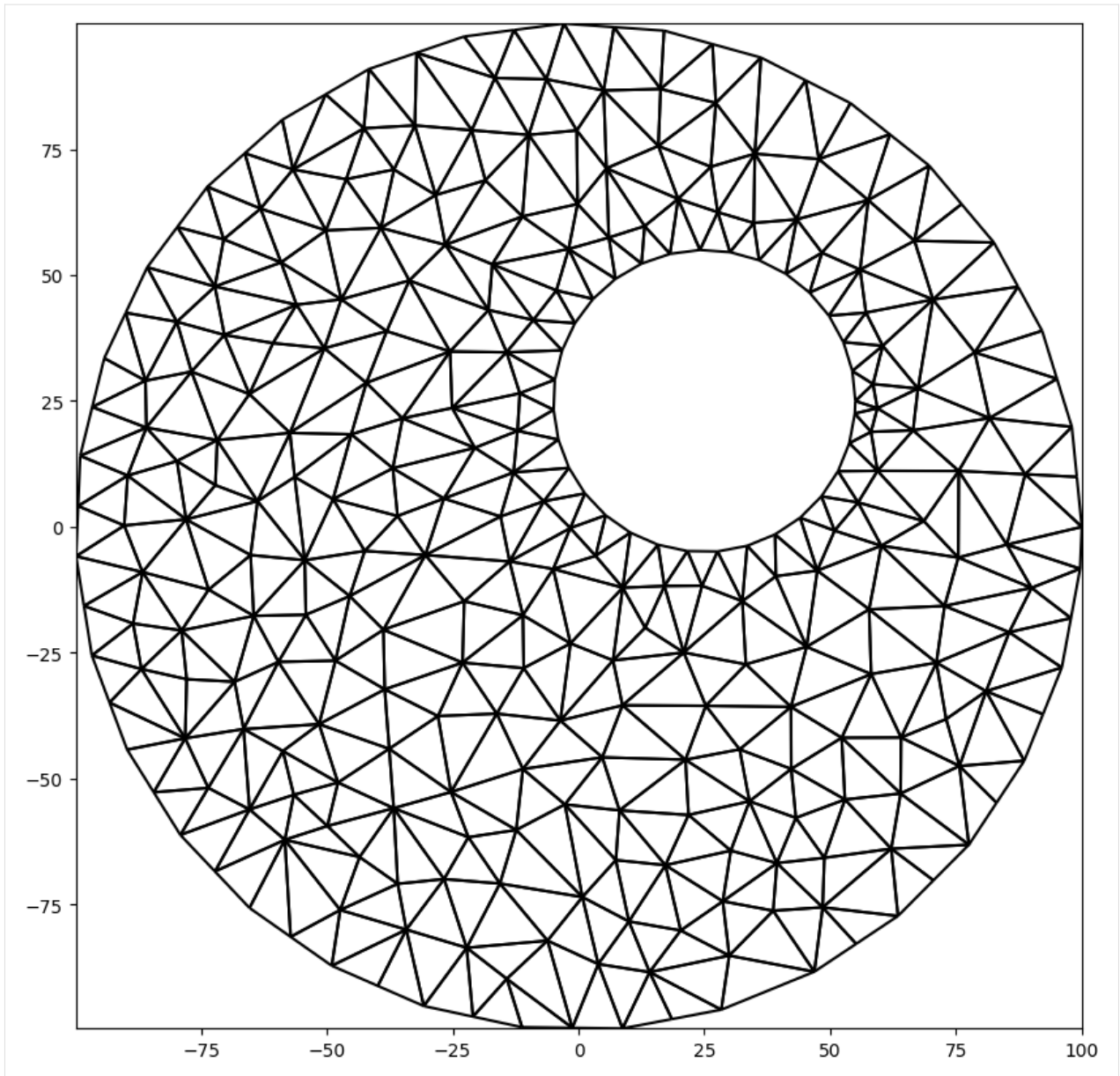
(continues on next page)

(continued from previous page)

```
y = radius * np.sin(theta) + 25.0
inner_circle_poly = [(x, y) for x, y in zip(x, y)]

# The hole is created by passing in another polygon and
# then passing a point inside the hole polygon with the
# add_hole() method.
tri = Triangle(maximum_area=100, angle=30, model_ws=workspace)
tri.add_polygon(circle_poly)
tri.add_polygon(inner_circle_poly)
tri.add_hole((25, 25))
tri.build(verbose=False)
fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
tri.plot(ax=ax)
```

```
[6]: <matplotlib.collections.LineCollection at 0x7ff5062c1640>
```



Specifying Regions with Different Triangle Sizes

Different parts of the domain can be assigned different levels of refinement by adding multiple polygons and then identifying the different polygons as regions with different maximum triangle areas.

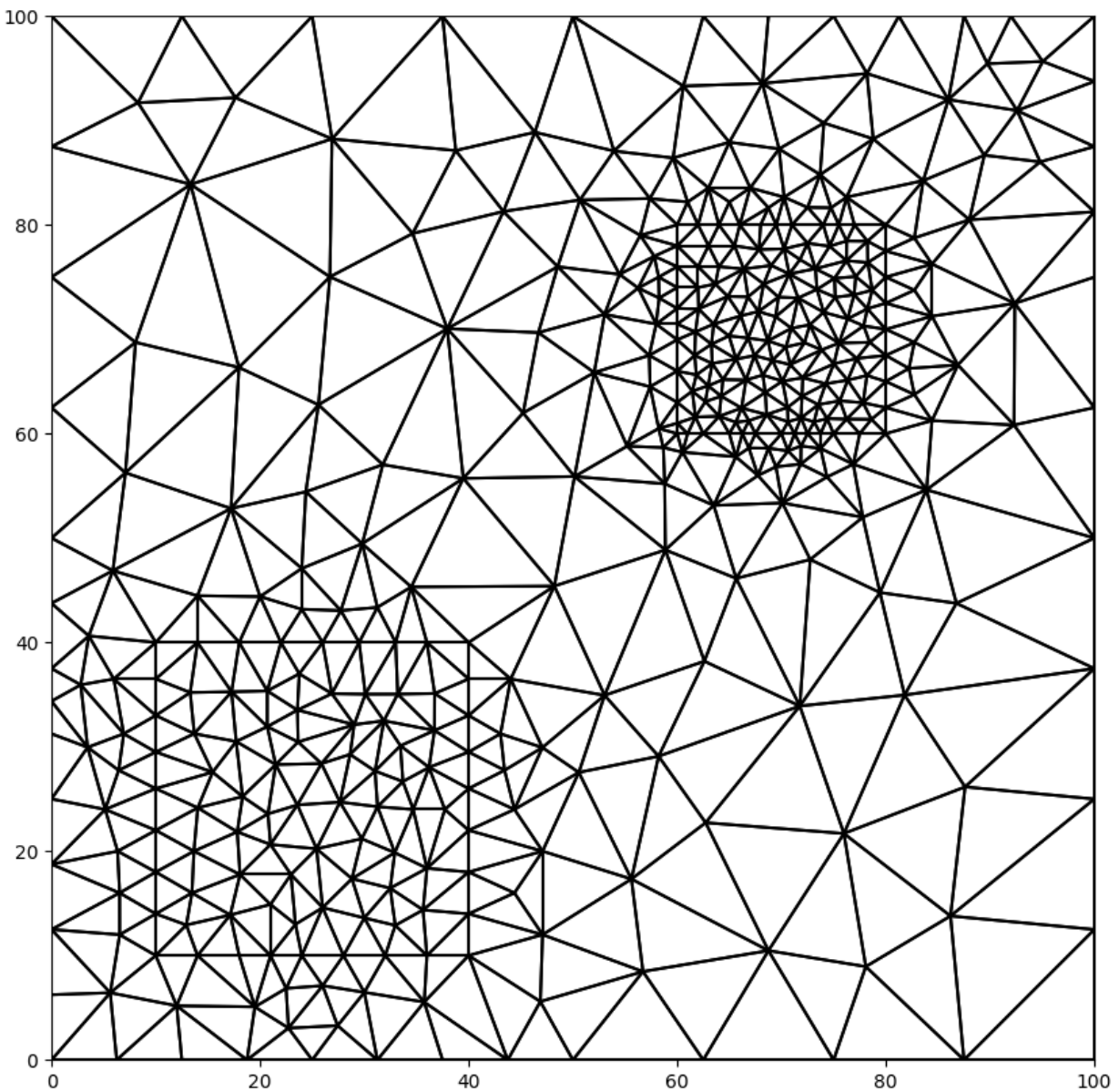
```
[7]: active_domain = [(0, 0), (100, 0), (100, 100), (0, 100)]
    area1 = [(10, 10), (40, 10), (40, 40), (10, 40)]
    area2 = [(60, 60), (80, 60), (80, 80), (60, 80)]
    tri = Triangle(angle=30, model_ws=workspace)
    tri.add_polygon(active_domain)
    tri.add_polygon(area1)
    tri.add_polygon(area2)
    tri.add_region((1, 1), 0, maximum_area=100) # point inside active domain
```

(continues on next page)

(continued from previous page)

```
tri.add_region((11, 11), 1, maximum_area=10) # point inside area1
tri.add_region((61, 61), 2, maximum_area=3)  # point inside area2
tri.build(verbose=False)
fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
tri.plot(ax=ax)
```

[7]: <matplotlib.collections.LineCollection at 0x7ff50629d220>



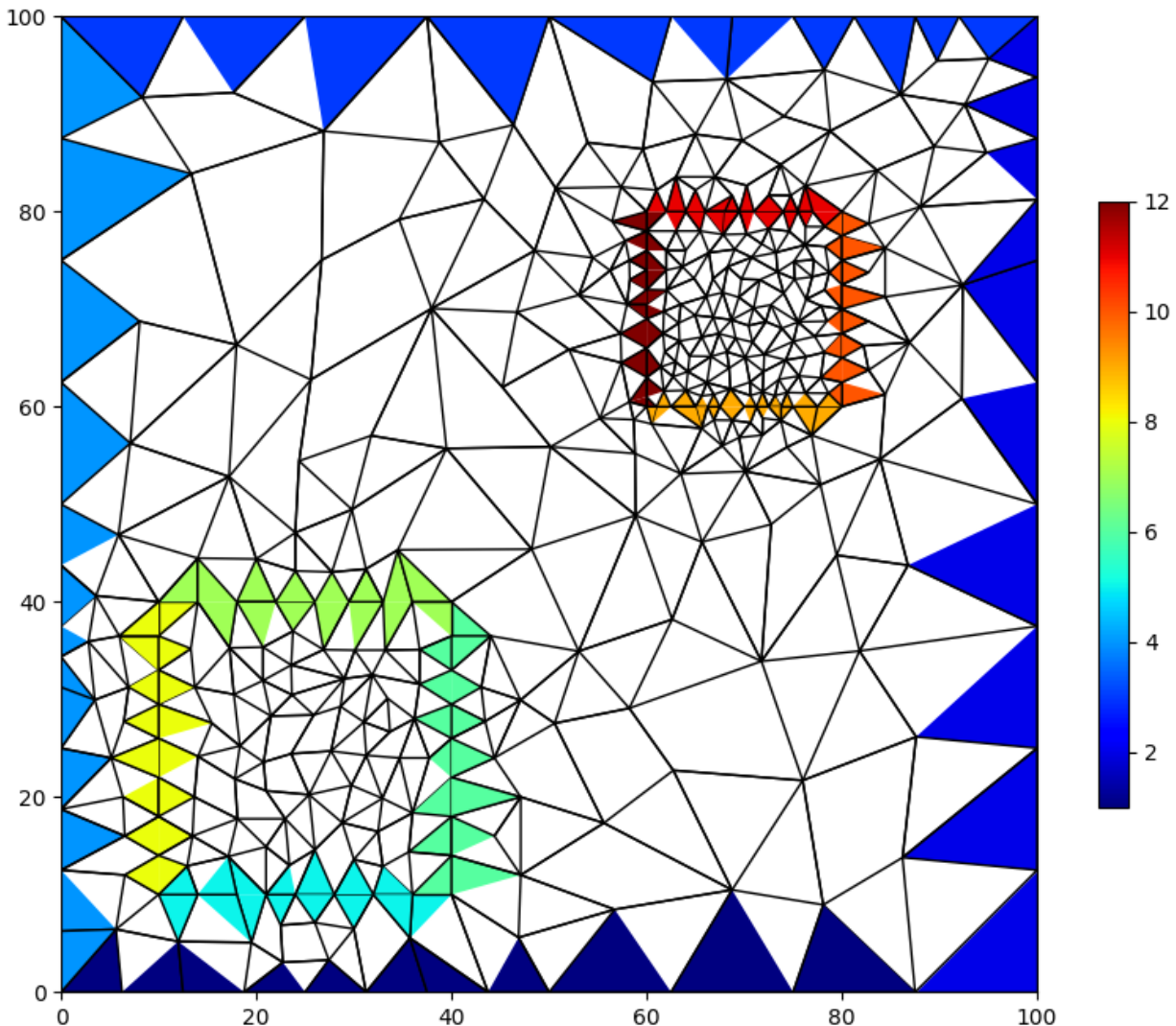
Identifying Boundary Cells

The Triangle class has some limited capabilities for identifying the cells on polygon boundaries. In the example above, three polygons were added to the Triangle class. An integer boundary marker is automatically calculated and assigned by the Triangle class. Boundary marker 1 corresponds to the first line segment of the first polygon added. So in this case, boundary marker 1 corresponds to cells along the line $[(0, 0), (100, 0)]$. Boundary marker 2 corresponds to the next line segment, which is along the right face of the domain.

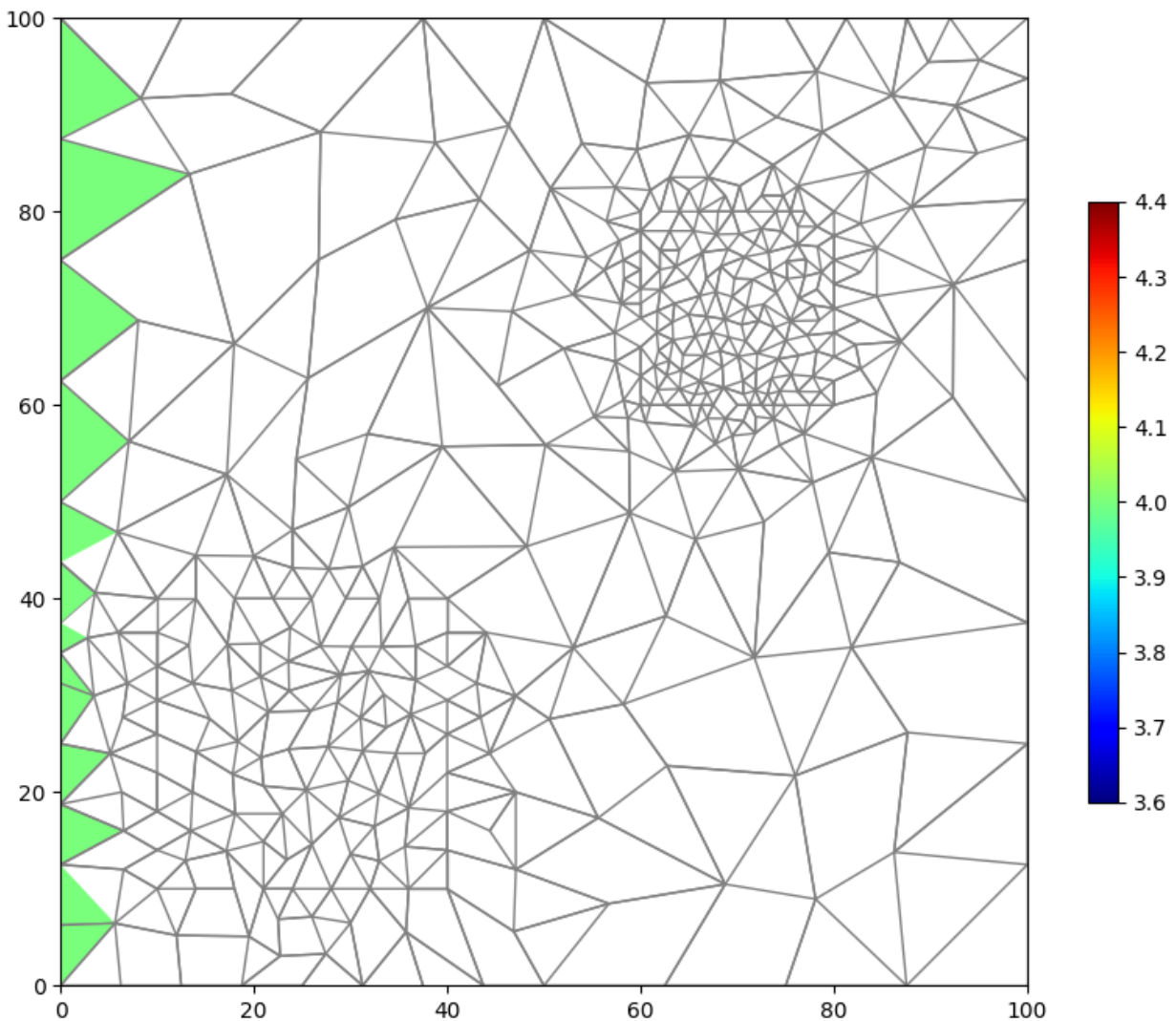
Triangle has a method for getting back an integer array for the mesh that has a boundary marker id for each cell. Values of zero indicate that the cell does not touch a boundary.

```
[8]: # this shows all the boundary cells
ibd = tri.get_boundary_marker_array()
ibd = np.ma.masked_equal(ibd, 0)
fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
pc = tri.plot(a=ibd, cmap="jet")
plt.colorbar(pc, shrink=0.5)
```

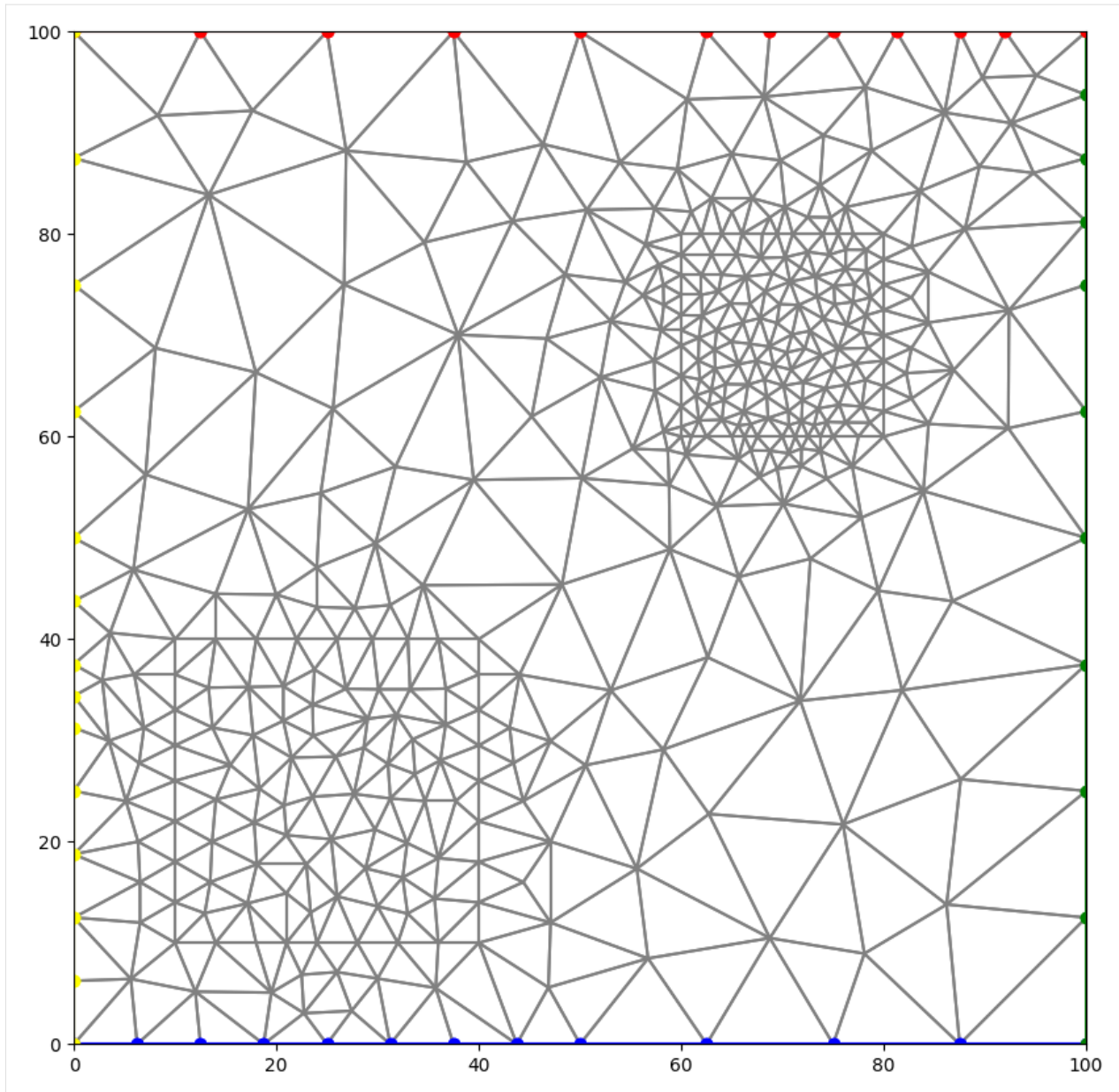
[8]: <matplotlib.colorbar.Colorbar at 0x7ff500c8ec40>




```
[9]: # we could plot just one group of boundary cells
# this shows all the boundary cells
ibd = tri.get_boundary_marker_array()
ibd = np.ma.masked_not_equal(ibd, 4)
fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
pc = tri.plot(a=ibd, cmap="jet", edgecolor="gray")
cb = plt.colorbar(pc, shrink=0.5)
```



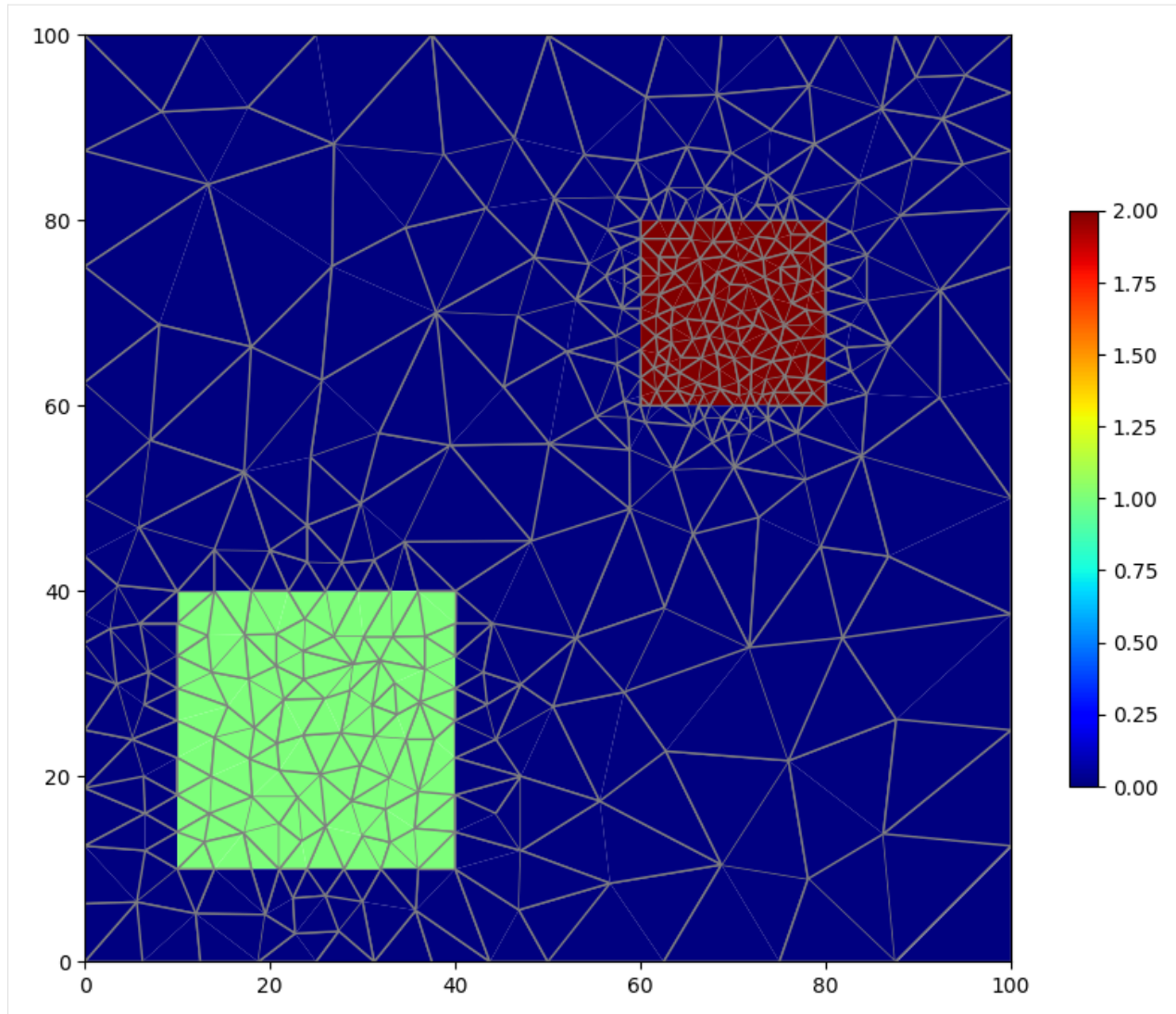
```
[10]: # we can also plot the lines that comprise the boundaries
fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
tri.plot(ax=ax, edgecolor="gray")
for ibm in [1, 2, 3, 4]:
    colors = ["blue", "green", "red", "yellow"]
    tri.plot_boundary(ibm, ax, marker="o", color=colors[ibm - 1])
```



Cell Attributes

If regions (using the `add_region()` method) are used and an attribute value is provided, it is possible to determine the cells that are within each region.

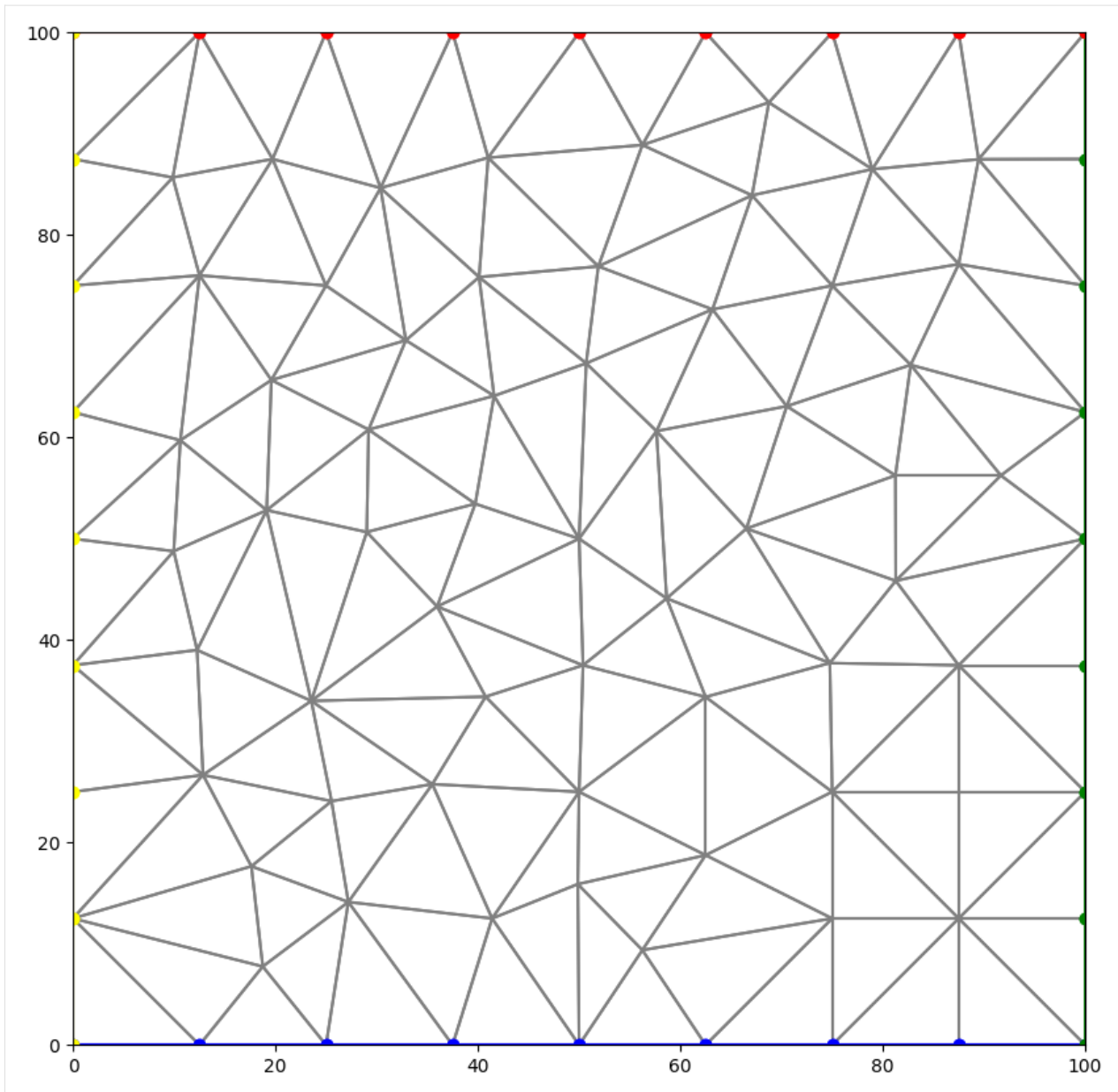
```
[11]: attribute_array = tri.get_attribute_array()
fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
pc = tri.plot(a=attribute_array, cmap="jet", edgecolor="gray")
cb = plt.colorbar(pc, shrink=0.5)
```



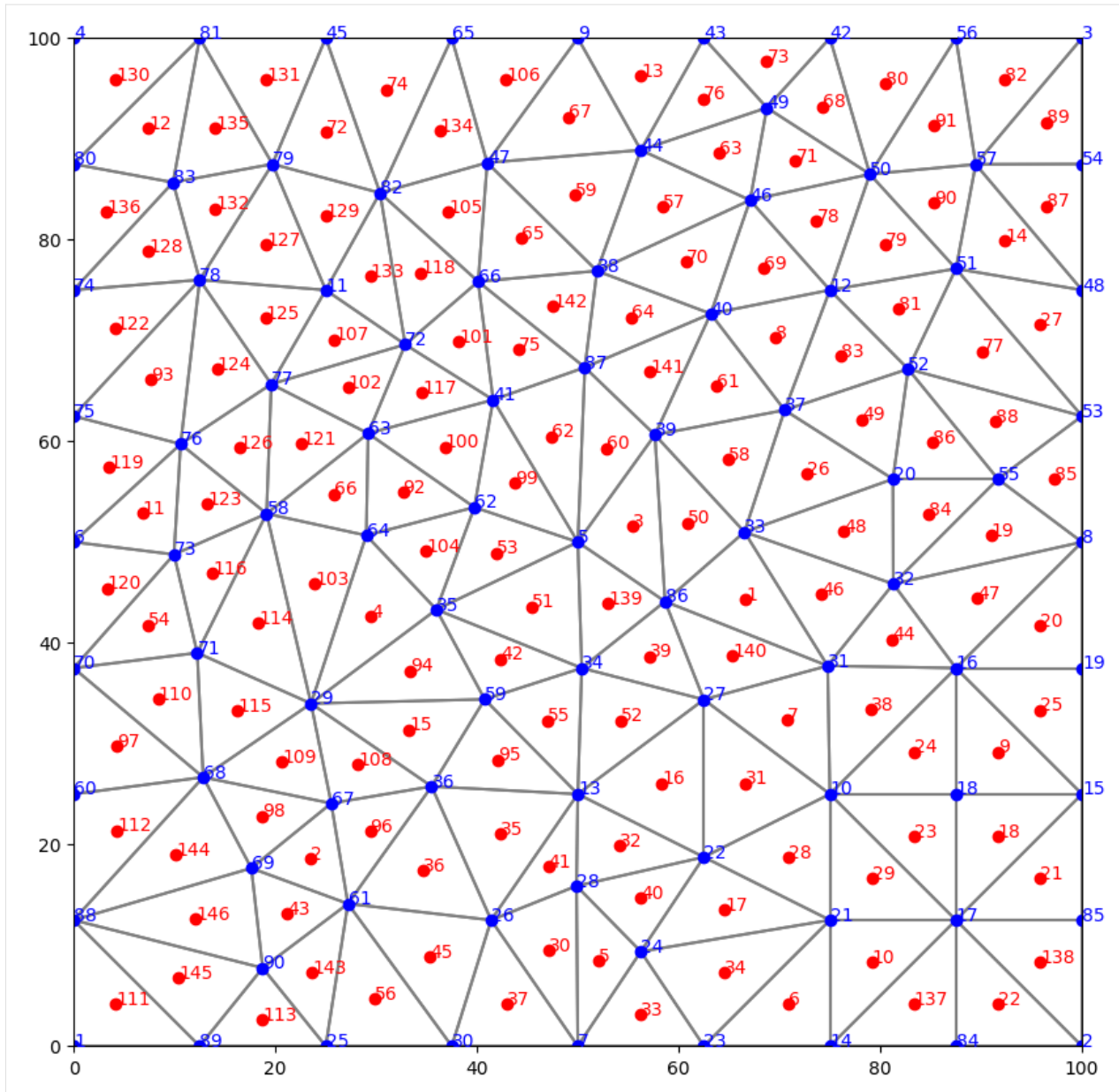
Building a Simple MODFLOW 6 Model

We can use the functionality described so far to build a simple MODFLOW 6 model using FloPy. For demonstration purposes, we'll create a very coarse triangular mesh and impose constant head boundaries on the left and right sides. We will simulate flow as steady state.

```
[12]: active_domain = [(0, 0), (100, 0), (100, 100), (0, 100)]
tri = Triangle(angle=30, maximum_area=100, model_ws=workspace)
tri.add_polygon(active_domain)
tri.build()
fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
tri.plot(edgecolor="gray")
for ibm in [1, 2, 3, 4]:
    colors = ["blue", "green", "red", "yellow"]
    tri.plot_boundary(ibm, ax, marker="o", color=colors[ibm - 1])
```

```
[13]: fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
tri.plot(ax=ax, edgecolor="gray")
tri.plot_vertices(ax=ax, marker="o", color="blue")
tri.label_vertices(ax=ax, fontsize=10, color="blue")
tri.plot_centroids(ax=ax, marker="o", color="red")
tri.label_cells(ax=ax, fontsize=10, color="red")
```



```
[14]: name = "mf"
sim = flopy.mf6.MFSimulation(
    sim_name=name, version="mf6", exe_name="mf6", sim_ws=workspace
)
tdis = flopy.mf6.ModflowTdis(
    sim, time_units="DAYS", perioddata=[[1.0, 1, 1.0]]
)
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, save_flows=True)
ims = flopy.mf6.ModflowIms(
    sim,
    print_option="SUMMARY",
    complexity="complex",
    outer_hclose=1.0e-8,
```

(continues on next page)

(continued from previous page)

```

        inner_hclose=1.0e-8,
    )
    cell2d = tri.get_cell2d()
    vertices = tri.get_vertices()
    xcyc = tri.get_xcyc()
    nlay = 1
    ncpl = tri.ncpl
    nvert = tri.nvert
    top = 1.0
    botm = [0.0]
    dis = flopy.mf6.ModflowGwfdiscv(
        gwf,
        nlay=nlay,
        ncpl=ncpl,
        nvert=nvert,
        top=top,
        botm=botm,
        vertices=vertices,
        cell2d=cell2d,
    )
    npf = flopy.mf6.ModflowGwfnpf(
        gwf, xt3doptions=[(True)], save_specific_discharge=None
    )
    ic = flopy.mf6.ModflowGwfic(gwf)

def chdhead(x):
    return x * 10.0 / 100.0

chdlist = []
leftcells = tri.get_edge_cells(4)
rightcells = tri.get_edge_cells(2)
for icpl in leftcells + rightcells:
    h = chdhead(xcyc[icpl, 0])
    chdlist.append([(0, icpl), h])
chd = flopy.mf6.ModflowGwfchd(gwf, stress_period_data=chdlist)
oc = flopy.mf6.ModflowGwfoc(
    gwf,
    budget_filerecord="{}.cbc".format(name),
    head_filerecord="{}.hds".format(name),
    saverecord=[("HEAD", "LAST"), ("BUDGET", "LAST")],
    printrecord=[("HEAD", "LAST"), ("BUDGET", "LAST")],
)
sim.write_simulation()
success, buff = sim.run_simulation(report=True)

WARNING: Unable to resolve dimension of ('gwf6', 'discv', 'cell2d', 'cell2d', 'icvert')
↳ based on shape "nvert".
writing simulation...
    writing simulation name file...
    writing simulation tdis package...

```

(continues on next page)

(continued from previous page)

```
writing solution package ims_-1...
writing model mf...
  writing model name file...
  writing package disv...
  writing package npf...
  writing package ic...
  writing package chd_0...
INFORMATION: maxbound in ('gwf6', 'chd', 'dimensions') changed to 16 based on size of
↳ stress_period_data
  writing package oc...
FloPy is using the following executable to run the model: ../../home/runner/.local/bin/
↳ modflow/mf6
```

```
MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023
```

```
MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000
```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

```
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:40:13
```

```
Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam
```

```
Solving: Stress period:      1      Time step:      1
```

```
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:40:13
Elapsed run time: 0.024 Seconds
```

WARNING REPORT:

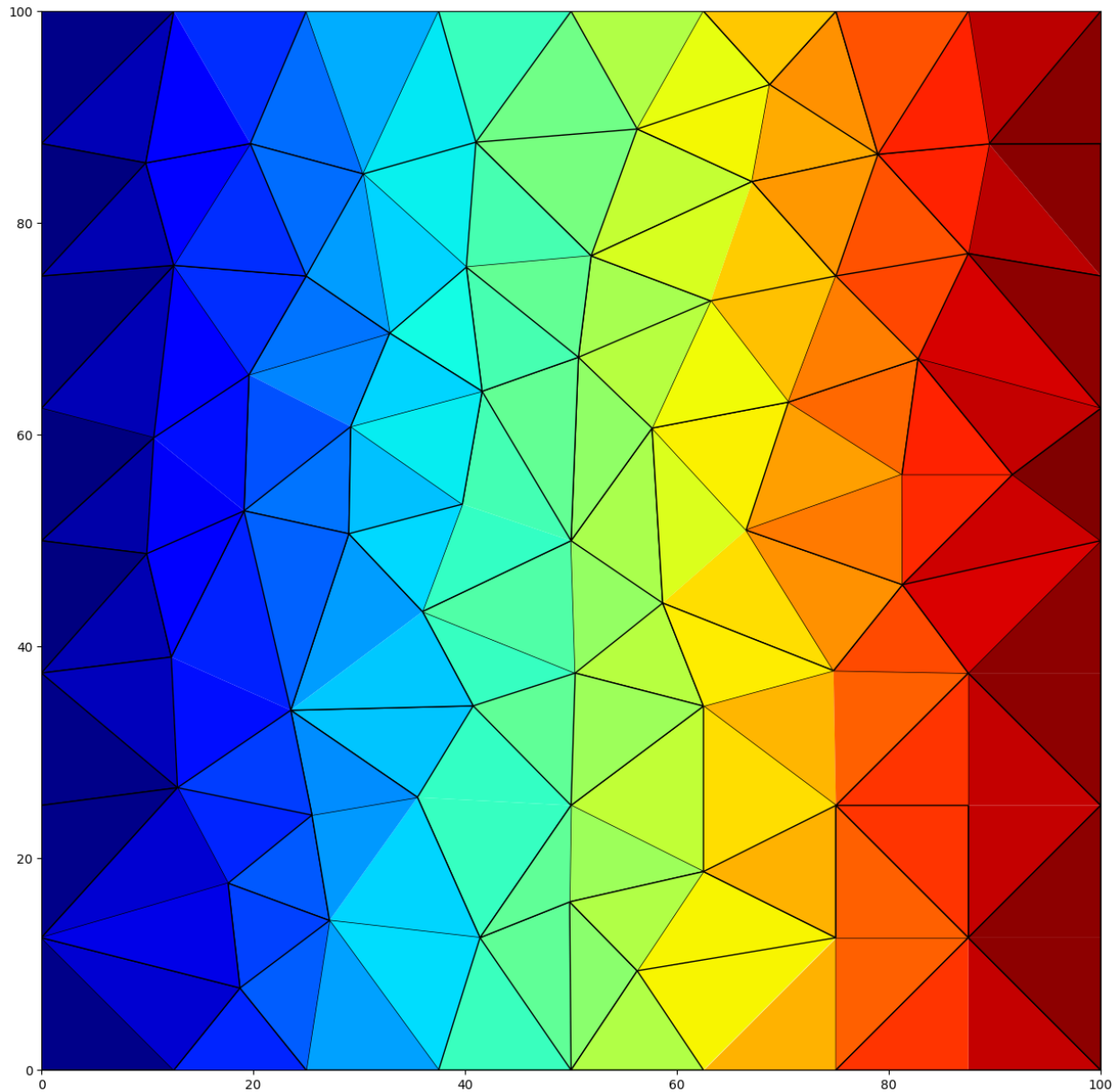
1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'mf.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'mf.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.

```
Normal termination of simulation.
```

```
[15]: fname = workspace / f"{name}.hds"
      hdoobj = flopy.utils.HeadFile(fname, precision="double")
      head = hdoobj.get_data()
      fname = workspace / f"{name}.cbc"
      bdoobj = flopy.utils.CellBudgetFile(fname, precision="double", verbose=False)
      # qxqy = bdoobj.get_data(text='DATA-SPDIS')[0]

      fig = plt.figure(figsize=(15, 15))
      ax = plt.subplot(1, 1, 1, aspect="equal")
      tri.plot(ax=ax, a=head[0, 0, :], cmap="jet")
```

```
[15]: <matplotlib.collections.PathCollection at 0x7ff5009ad130>
```



Clean up the temporary workspace.

```
[16]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass
```

6.1.2 Voronoi Grid and MODFLOW 6 Flow and Transport Example

First set the path and import the required packages. The flopy path doesn't have to be set if you install flopy from a binary installer. If you want to run this notebook, you have to set the path to your own flopy path.

```
[1]: import os
import sys
from pathlib import Path
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
from shapely.geometry import LineString, Point

import flopy
from flopy.discretization import VertexGrid
from flopy.utils.triangle import Triangle as Triangle
from flopy.utils.voronoi import VoronoiGrid

temp_dir = TemporaryDirectory()
workspace = Path(temp_dir.name)

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

Use Triangle to Generate Points for Voronoi Grid

```
[2]: # set domain extents
xmin = 0.0
xmax = 2000.0
ymin = 0.0
ymax = 1000.0

# set minimum angle
angle_min = 30
```

(continues on next page)

(continued from previous page)

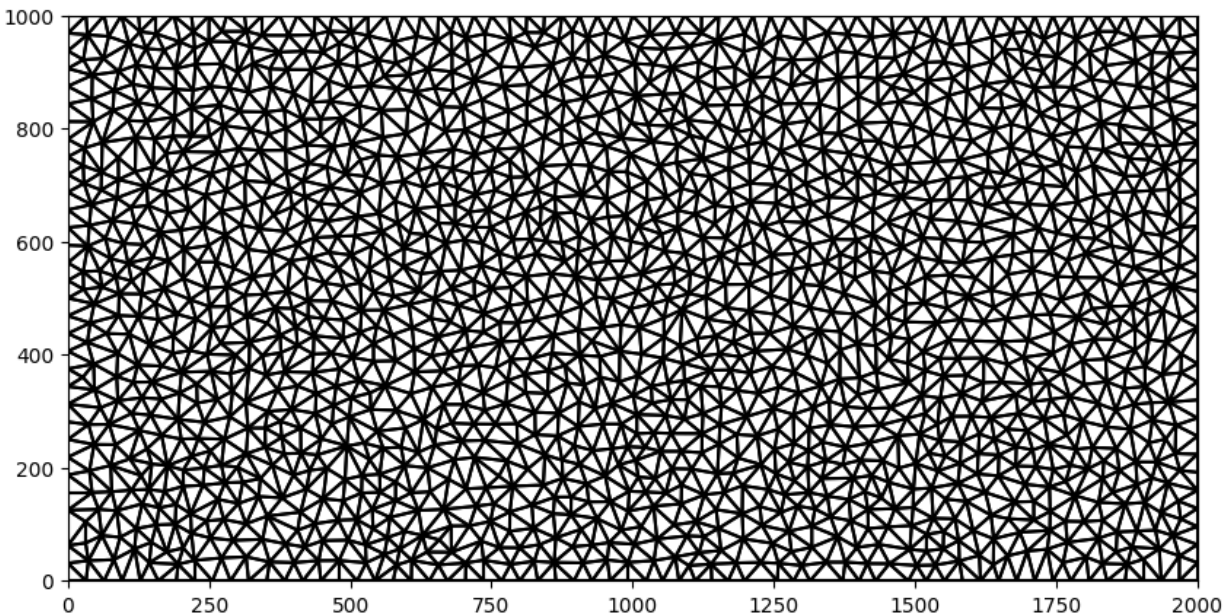
```
# set maximum area
area_max = 1000.0

delr = area_max**0.5
ncol = xmax / delr
nrow = ymax / delr
nodes = ncol * nrow
print("equivalent delr: ", delr)
print("equivalent nodes, ncol, nrow: ", int(nodes), ncol, nrow)

equivalent delr:  31.622776601683793
equivalent nodes, ncol, nrow:  2000 63.245553203367585 31.622776601683793
```

```
[3]: tri = Triangle(maximum_area=area_max, angle=angle_min, model_ws=workspace)
poly = np.array(((xmin, ymin), (xmax, ymin), (xmax, ymax), (xmin, ymax)))
tri.add_polygon(poly)
tri.build(verbose=False)

fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
pc = tri.plot(ax=ax)
```

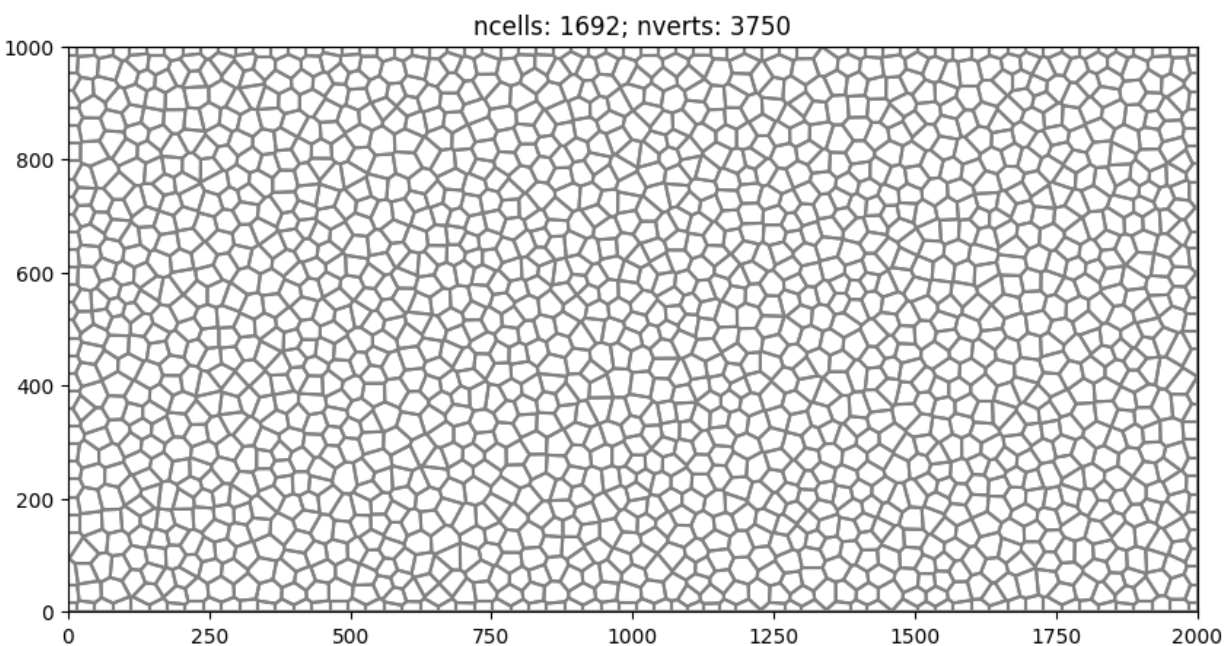


Create and Plot FloPy Voronoi Grid

The FloPy VoronoiGrid class can be used to generate voronoi grids using the `scipy.spatial.Voronoi` class. The VoronoiGrid class is a thin wrapper that makes sure edge cells are closed and provides methods for obtaining the information needed to make FloPy MODFLOW models. It works by passing in the flopy Triangle object generated in the previous cell.

```
[4]: voronoi_grid = VoronoiGrid(tri)
fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
voronoi_grid.plot(ax=ax, facecolor="none")
```

```
[4]: <Axes: title={'center': 'ncells: 1692; nverts: 3750'}>
```



Use the VertexGrid Representation to Identify Boundary Cells

```
[5]: gridprops = voronoi_grid.get_gridprops_vertexgrid()
vgrid = flopy.discretization.VertexGrid(**gridprops, nlay=1)
ibd = np.zeros(vgrid.ncpl, dtype=int)
gi = flopy.utils.GridIntersect(vgrid)

# identify cells on left edge
line = LineString([(xmin, ymin), (xmin, ymax)])
cells0 = gi.intersect(line)["cellids"]
cells0 = np.array(list(cells0))
ibd[cells0] = 1

# identify cells on right edge
line = LineString([(xmax, ymin), (xmax, ymax)])
cells1 = gi.intersect(line)["cellids"]
cells1 = np.array(list(cells1))
```

(continues on next page)

(continued from previous page)

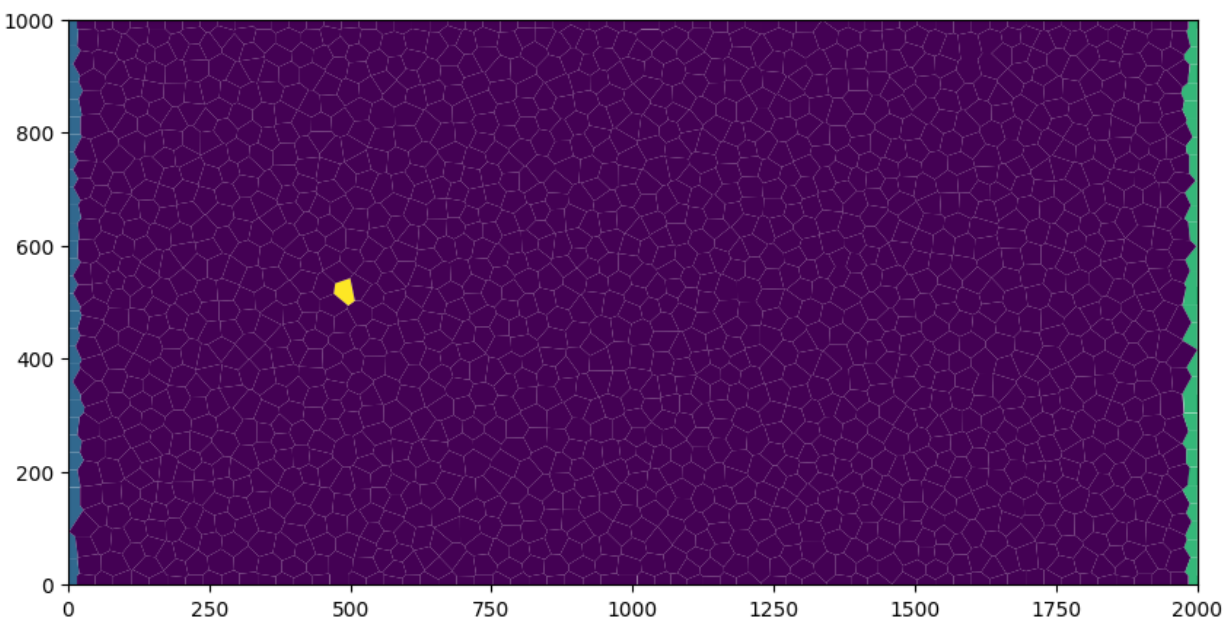
```

ibd[cells1] = 2

# identify cell for a constant concentration condition
point = Point((500, 500))
cells2 = gi.intersect(point)["cellids"]
cells2 = np.array(list(cells2))
ibd[cells2] = 3

if True:
    fig = plt.figure(figsize=(10, 10))
    ax = plt.subplot(1, 1, 1, aspect="equal")
    pmv = flopy.plot.PlotMapView(modelgrid=vgrid)
    pmv.plot_array(ibd)

```



Create Run and Post Process a MODFLOW 6 Flow Model

```

[6]: name = "mf"
sim_ws = os.path.join(workspace, "flow")
sim = flopy.mf6.MFSimulation(
    sim_name=name, version="mf6", exe_name="mf6", sim_ws=sim_ws
)
tdis = flopy.mf6.ModflowTdis(
    sim, time_units="DAYS", perioddata=[[1.0, 1, 1.0]]
)
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, save_flows=True)
ims = flopy.mf6.ModflowIms(
    sim,
    print_option="SUMMARY",
    complexity="complex",
    outer_dvclose=1.0e-8,

```

(continues on next page)

(continued from previous page)

```

        inner_dvclose=1.0e-8,
    )
    disv_gridprops = voronoi_grid.get_disv_gridprops()
    nlay = 1
    top = 1.0
    botm = [0.0]
    disv = flopy.mf6.ModflowGwfdisc(
        gwf, nlay=nlay, **disv_gridprops, top=top, botm=botm
    )
    npf = flopy.mf6.ModflowGwfnpf(
        gwf,
        xt3doptions=[(True)],
        k=10.0,
        save_saturation=True,
        save_specific_discharge=True,
    )
    ic = flopy.mf6.ModflowGwfic(gwf)

    chdlist = []
    for icpl in cells0:
        chdlist.append([(0, icpl), 1.0])
    for icpl in cells1:
        chdlist.append([(0, icpl), 0.0])
    chd = flopy.mf6.ModflowGwfchd(gwf, stress_period_data=chdlist)
    oc = flopy.mf6.ModflowGwfoc(
        gwf,
        budget_filerecord="{}.bud".format(name),
        head_filerecord="{}.hds".format(name),
        saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
        printrecord=[("HEAD", "LAST"), ("BUDGET", "LAST")],
    )
    sim.write_simulation()
    success, buff = sim.run_simulation(report=True, silent=True)

    head = gwf.output.head().get_data()
    bdoobj = gwf.output.budget()
    spdis = bdoobj.get_data(text="DATA-SPDIS")[0]

    fig = plt.figure(figsize=(15, 15))
    ax = plt.subplot(1, 1, 1, aspect="equal")
    pmv = flopy.plot.PlotMapView(gwf)
    pmv.plot_array(head, cmap="jet", alpha=0.5)
    pmv.plot_vector(spdis["qx"], spdis["qy"], alpha=0.25)

```

```

WARNING: Unable to resolve dimension of ('gwf6', 'disv', 'cell2d', 'cell2d', 'icvert')_
↳ based on shape "ncvert".

```

```

writing simulation...
    writing simulation name file...
    writing simulation tdis package...
    writing solution package ims_-1...
    writing model mf...
    writing model name file...

```

(continues on next page)

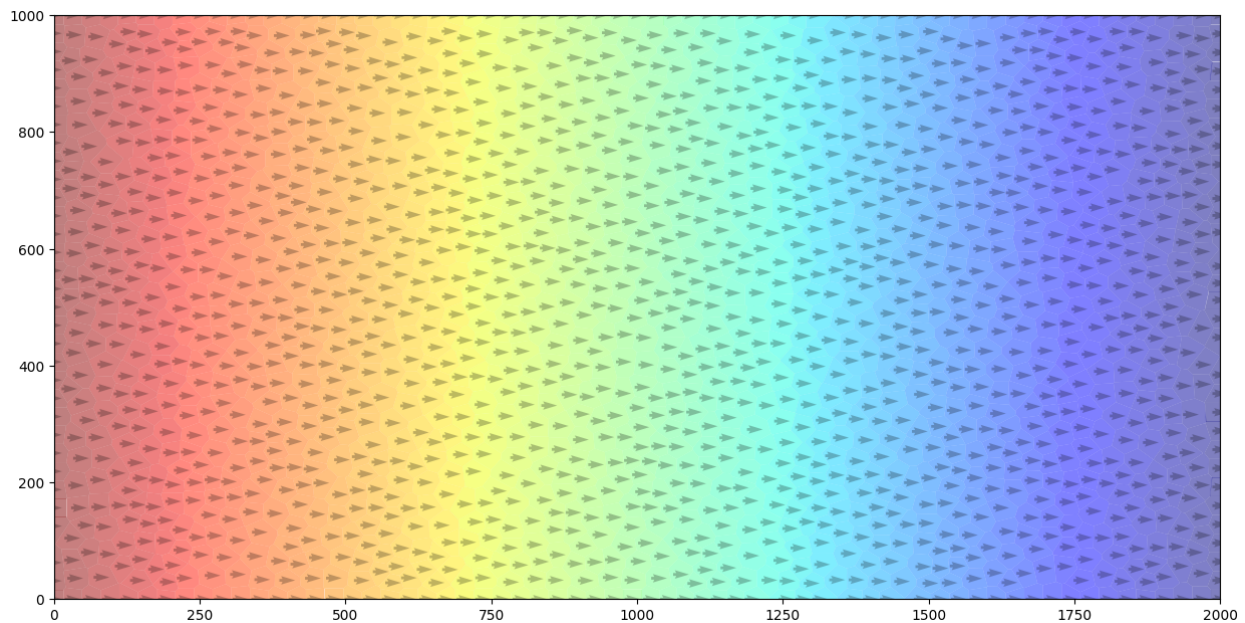
(continued from previous page)

```

writing package disv...
writing package npf...
writing package ic...
writing package chd_0...
INFORMATION: maxbound in ('gwf6', 'chd', 'dimensions') changed to 62 based on size of
↳ stress_period_data
writing package oc...

```

[6]: <matplotlib.quiver.Quiver at 0x7f4a3d926250>



Create Run and Post Process a MODFLOW 6 Transport Model

```

[7]: name = "mf"
sim_ws = os.path.join(workspace, "transport")
sim = flopy.mf6.MFSimulation(
    sim_name=name, version="mf6", exe_name="mf6", sim_ws=sim_ws
)
tdis = flopy.mf6.ModflowTdis(
    sim, time_units="DAYS", perioddata=[[100 * 365.0, 100, 1.0]]
)
gwt = flopy.mf6.ModflowGwt(sim, modelname=name, save_flows=True)
ims = flopy.mf6.ModflowIms(
    sim,
    print_option="SUMMARY",
    complexity="simple",
    linear_acceleration="bicgstab",
    outer_dvclose=1.0e-6,
    inner_dvclose=1.0e-6,
)
disv_gridprops = voronoi_grid.get_disv_gridprops()
nlay = 1

```

(continues on next page)

(continued from previous page)

```

top = 1.0
botm = [0.0]
disv = flopy.mf6.ModflowGwtDisv(
    gwt, nlay=nlay, **disv_gridprops, top=top, botm=botm
)
ic = flopy.mf6.ModflowGwtIc(gwt, strt=0.0)
sto = flopy.mf6.ModflowGwtMst(gwt, porosity=0.2)
adv = flopy.mf6.ModflowGwtAdv(gwt, scheme="TVD")
dsp = flopy.mf6.ModflowGwtDsp(gwt, alh=5.0, ath1=0.5)
sourcerearray = [()]
ssm = flopy.mf6.ModflowGwtSsm(gwt, sources=sourcerearray)
cnclist = [
    [(0, cells2[0]), 1.0],
]
cnc = flopy.mf6.ModflowGwtCnc(
    gwt, maxbound=len(cnclist), stress_period_data=cnclist, pname="CNC-1"
)
pd = [
    ("GWFHEAD", "../flow/mf.hds"),
    ("GWFBUDGET", "../flow/mf.bud"),
]
fmi = flopy.mf6.ModflowGwtFmi(gwt, packagedata=pd)
oc = flopy.mf6.ModflowGwtOc(
    gwt,
    budget_filerecord="{}.cbc".format(name),
    concentration_filerecord="{}.ucn".format(name),
    saverecord=[("CONCENTRATION", "ALL"), ("BUDGET", "ALL")],
)

sim.write_simulation()
success, buff = sim.run_simulation(report=True, silent=True)

conc = gwt.output.concentration().get_data()

fig = plt.figure(figsize=(10, 10))
ax = plt.subplot(1, 1, 1, aspect="equal")
pmv = flopy.plot.PlotMapView(gwf)
c = pmv.plot_array(conc, cmap="jet")
pmv.contour_array(conc, levels=(0.0001, 0.001, 0.01, 0.1), colors="y")
plt.colorbar(c, shrink=0.5)

```

```

WARNING: Unable to resolve dimension of ('gwt6', 'disv', 'cell2d', 'cell2d', 'icvert')_
↳ based on shape "ncvert".

```

```

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims_1...
  writing model mf...
    writing model name file...
    writing package disv...
    writing package ic...
    writing package mst...

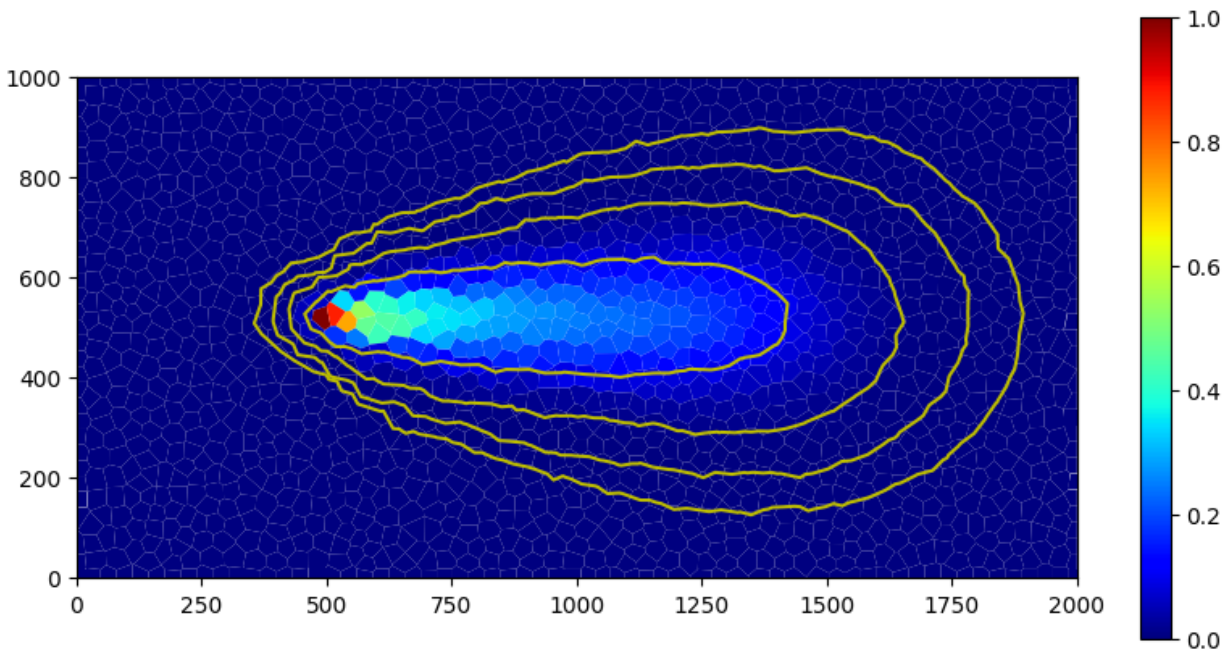
```

(continues on next page)

(continued from previous page)

```
writing package adv...
writing package dsp...
writing package ssm...
writing package cnc-1...
writing package fmi...
writing package oc...
```

[7]: <matplotlib.colorbar.Colorbar at 0x7f4a3d8944f0>



Building Voronoi Grid Examples

Irregular Domain Boundary

```
[8]: domain = [
    [1831.381546, 6335.543757],
    [4337.733475, 6851.136153],
    [6428.747084, 6707.916043],
    [8662.980804, 6493.085878],
    [9350.437333, 5891.561415],
    [9235.861245, 4717.156511],
    [8963.743036, 3685.971717],
    [8691.624826, 2783.685023],
    [8047.13433, 2038.94045],
    [7416.965845, 578.0953252],
    [6414.425073, 105.4689614],
    [5354.596258, 205.7230386],
    [4624.173696, 363.2651598],
    [3363.836725, 563.7733141],
    [1330.11116, 1809.788273],
```

(continues on next page)

(continued from previous page)

```

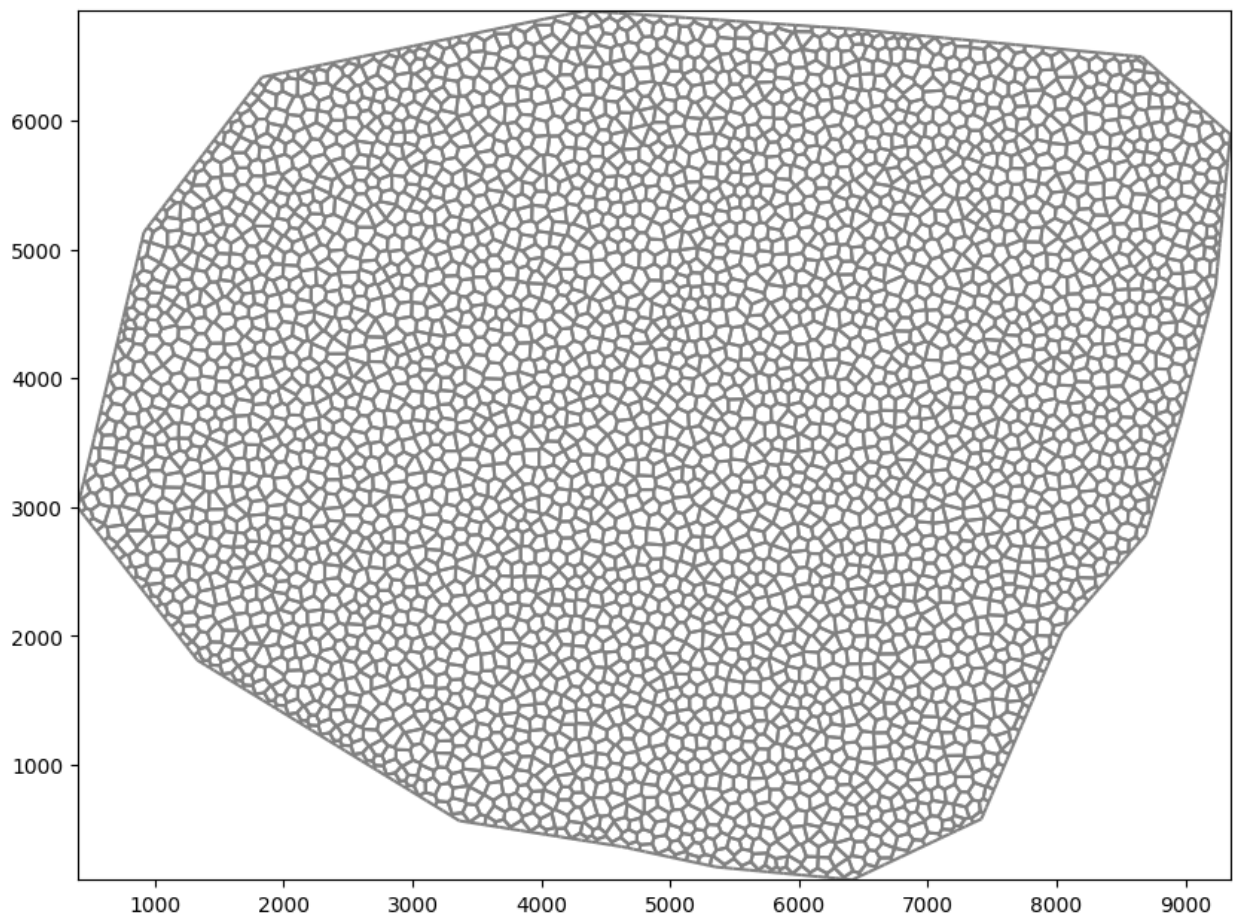
    [399.1804436, 2998.515188],
    [914.7728404, 5132.494831],
]
area_max = 100.0**2
tri = Triangle(maximum_area=area_max, angle=30, model_ws=workspace)
poly = np.array(domain)
tri.add_polygon(poly)
tri.build(verbose=False)

vor = VoronoiGrid(tri)
gridprops = vor.get_gridprops_vertexgrid()
voronoi_grid = VertexGrid(**gridprops, nlay=1)

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot()
ax.set_aspect("equal")
voronoi_grid.plot(ax=ax)

```

[8]: <matplotlib.collections.LineCollection at 0x7f4a3d8b4910>



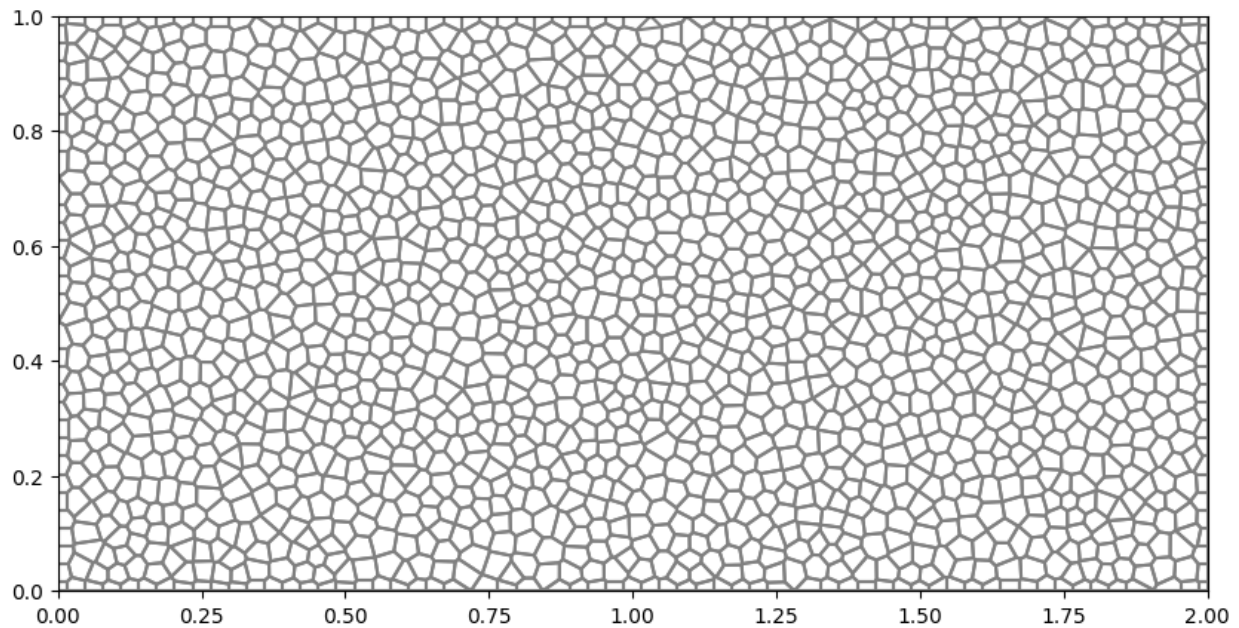
Simple Rectangular Domain

```
[9]: xmin = 0.0
      xmax = 2.0
      ymin = 0.0
      ymax = 1.0
      area_max = 0.001
      tri = Triangle(maximum_area=area_max, angle=30, model_ws=workspace)
      poly = np.array(((xmin, ymin), (xmax, ymin), (xmax, ymax), (xmin, ymax)))
      tri.add_polygon(poly)
      tri.build(verbose=False)

      vor = VoronoiGrid(tri)
      gridprops = vor.get_gridprops_vertexgrid()
      voronoi_grid = VertexGrid(**gridprops, nlay=1)

      fig = plt.figure(figsize=(10, 10))
      ax = fig.add_subplot()
      ax.set_aspect("equal")
      voronoi_grid.plot(ax=ax)
```

```
[9]: <matplotlib.collections.LineCollection at 0x7f4a3de24ac0>
```



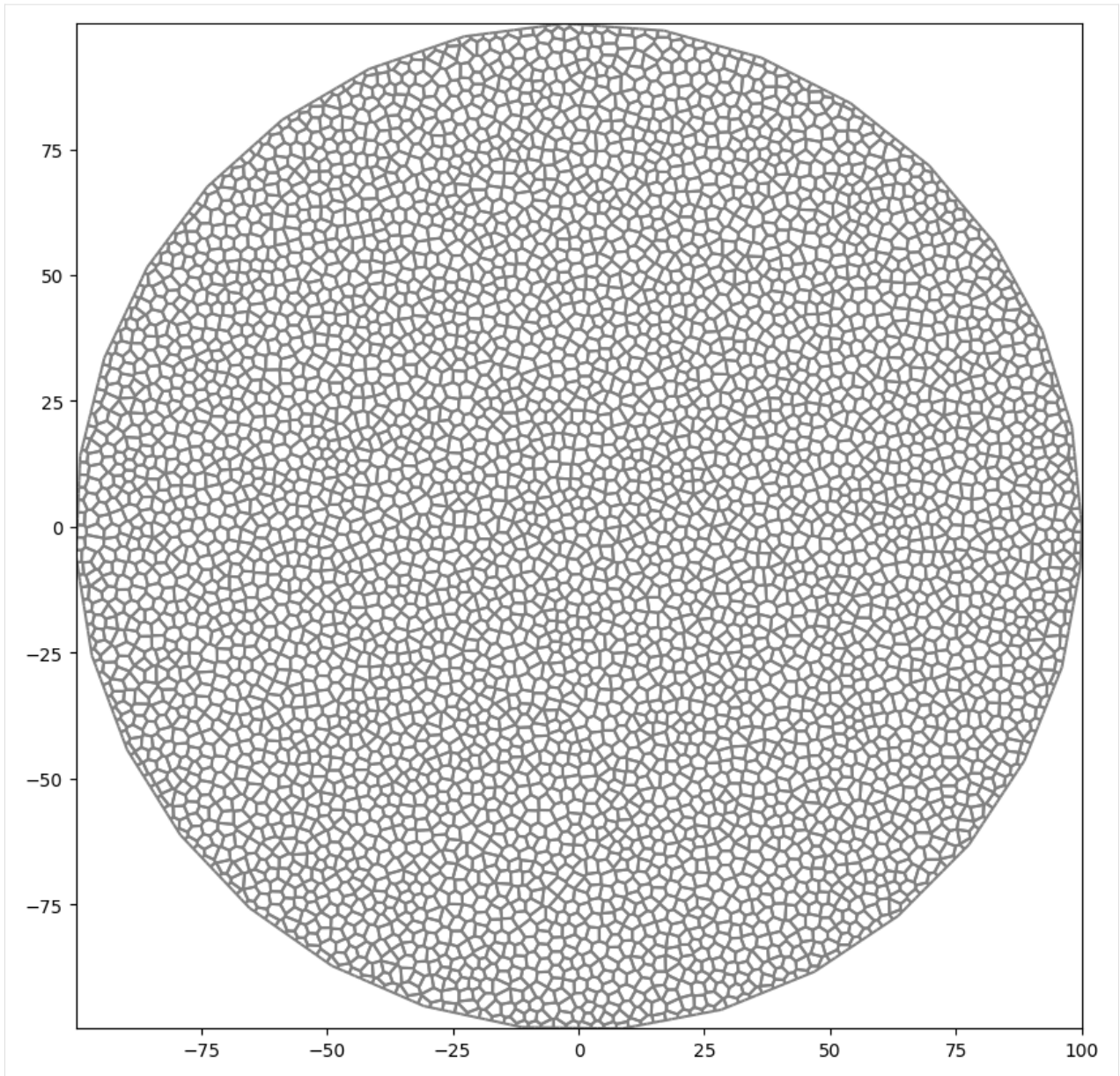
Circular Grid

```
[10]: theta = np.arange(0.0, 2 * np.pi, 0.2)
      radius = 100.0
      x = radius * np.cos(theta)
      y = radius * np.sin(theta)
      circle_poly = [(x, y) for x, y in zip(x, y)]
      tri = Triangle(maximum_area=5, angle=30, model_ws=workspace)
      tri.add_polygon(circle_poly)
      tri.build(verbose=False)

      vor = VoronoiGrid(tri)
      gridprops = vor.get_gridprops_vertexgrid()
      voronoi_grid = VertexGrid(**gridprops, nlay=1)

      fig = plt.figure(figsize=(10, 10))
      ax = fig.add_subplot()
      ax.set_aspect("equal")
      voronoi_grid.plot(ax=ax)

[10]: <matplotlib.collections.LineCollection at 0x7f4a3c81c430>
```

Circular Grid with Hole

```
[11]: theta = np.arange(0.0, 2 * np.pi, 0.2)
      radius = 30.0
      x = radius * np.cos(theta) + 25.0
      y = radius * np.sin(theta) + 25.0
      inner_circle_poly = [(x, y) for x, y in zip(x, y)]

      tri = Triangle(maximum_area=10, angle=30, model_ws=workspace)
      tri.add_polygon(circle_poly)
      tri.add_polygon(inner_circle_poly)
      tri.add_hole((25, 25))
```

(continues on next page)

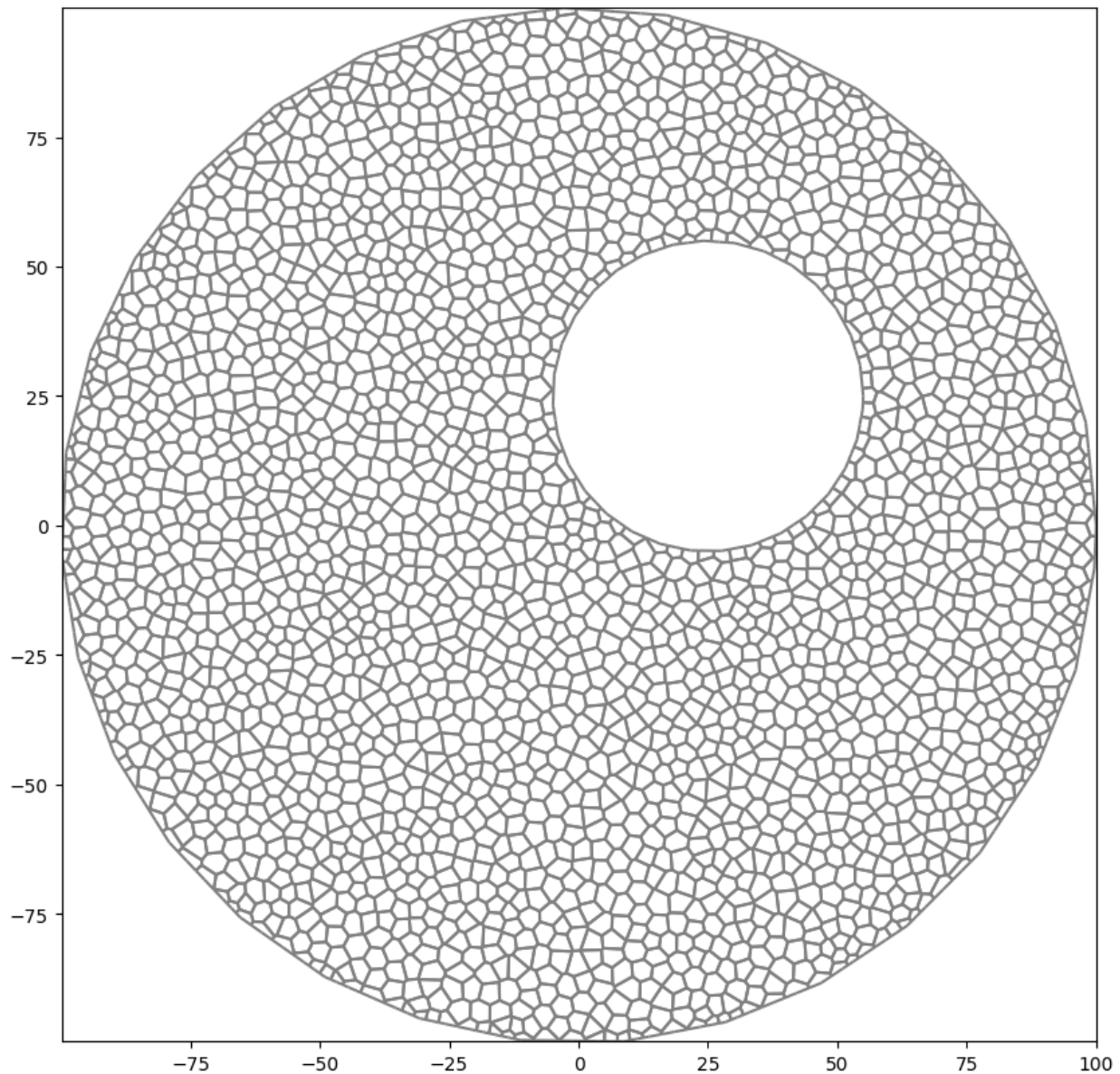
(continued from previous page)

```
tri.build(verbose=False)

vor = VoronoiGrid(tri)
gridprops = vor.get_gridprops_vertexgrid()
voronoi_grid = VertexGrid(**gridprops, nlay=1)

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot()
ax.set_aspect("equal")
voronoi_grid.plot(ax=ax)
```

[11]: <matplotlib.collections.LineCollection at 0x7f4a3bc190a0>



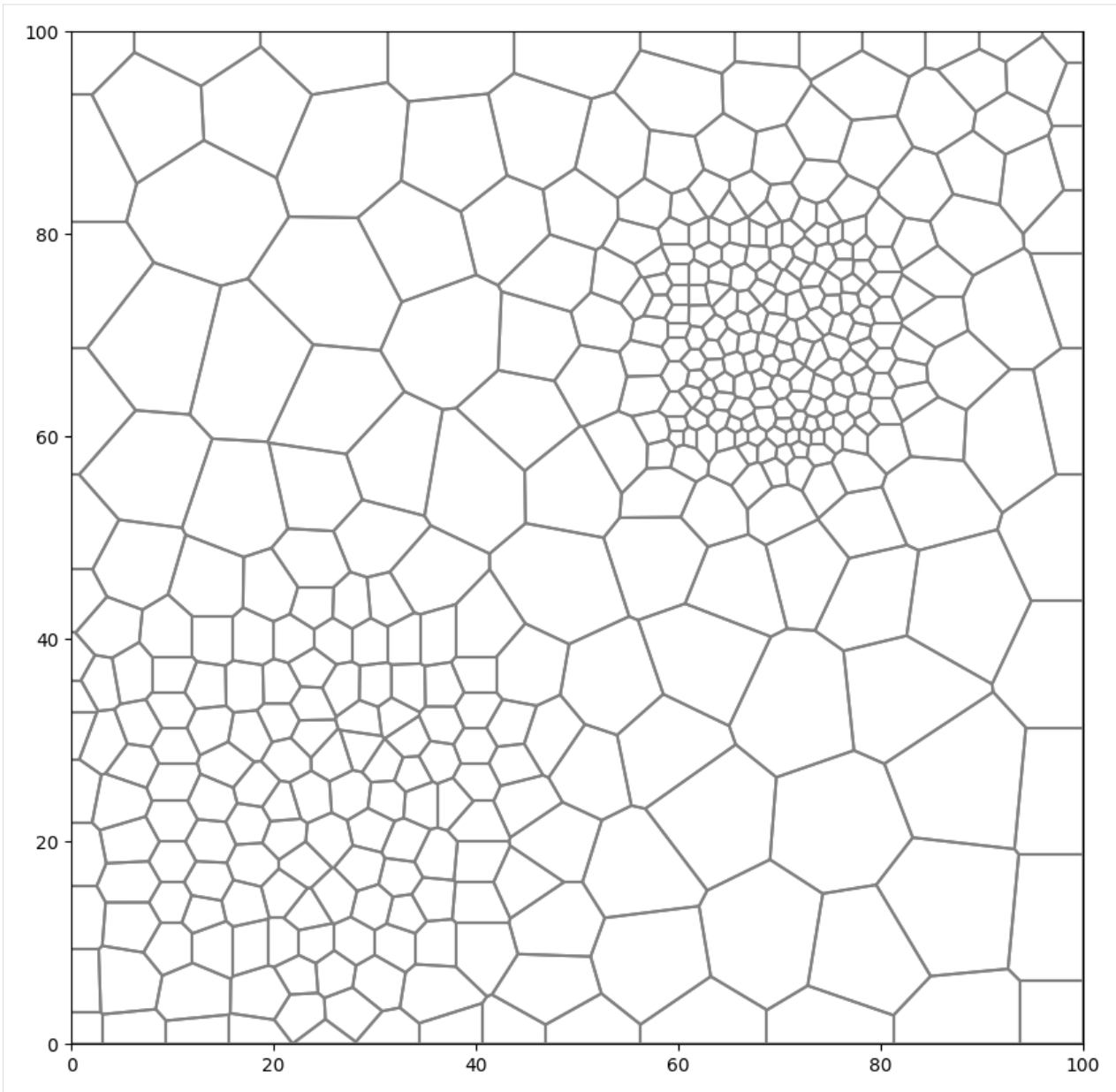
Regions with Different Refinement

```
[12]: active_domain = [(0, 0), (100, 0), (100, 100), (0, 100)]
      area1 = [(10, 10), (40, 10), (40, 40), (10, 40)]
      area2 = [(60, 60), (80, 60), (80, 80), (60, 80)]
      tri = Triangle(angle=30, model_ws=workspace)
      tri.add_polygon(active_domain)
      tri.add_polygon(area1)
      tri.add_polygon(area2)
      tri.add_region((1, 1), 0, maximum_area=100) # point inside active domain
      tri.add_region((11, 11), 1, maximum_area=10) # point inside area1
      tri.add_region((61, 61), 2, maximum_area=3) # point inside area2
      tri.build(verbose=False)

      vor = VoronoiGrid(tri)
      gridprops = vor.get_gridprops_vertexgrid()
      voronoi_grid = VertexGrid(**gridprops, nlay=1)

      fig = plt.figure(figsize=(10, 10))
      ax = fig.add_subplot()
      ax.set_aspect("equal")
      voronoi_grid.plot(ax=ax)

[12]: <matplotlib.collections.LineCollection at 0x7f4a3b2d9520>
```



Regions with Different Refinement and Hole

```
[13]: active_domain = [(0, 0), (100, 0), (100, 100), (0, 100)]
      area1 = [(10, 10), (40, 10), (40, 40), (10, 40)]
      area2 = [(70, 70), (90, 70), (90, 90), (70, 90)]

      tri = Triangle(angle=30, model_ws=workspace)

      # requirement that active_domain is first polygon to be added
      tri.add_polygon(active_domain)

      # requirement that any holes be added next
```

(continues on next page)

(continued from previous page)

```

theta = np.arange(0.0, 2 * np.pi, 0.2)
radius = 10.0
x = radius * np.cos(theta) + 50.0
y = radius * np.sin(theta) + 70.0
circle_poly0 = [(x, y) for x, y in zip(x, y)]
tri.add_polygon(circle_poly0)
tri.add_hole((50, 70))

# Add a polygon to force cells to conform to it
theta = np.arange(0.0, 2 * np.pi, 0.2)
radius = 10.0
x = radius * np.cos(theta) + 70.0
y = radius * np.sin(theta) + 20.0
circle_poly1 = [(x, y) for x, y in zip(x, y)]
tri.add_polygon(circle_poly1)
# tri.add_hole((70, 20))

# add line through domain to force conforming cells
line = [(x, x) for x in np.linspace(11, 89, 100)]
tri.add_polygon(line)

# then regions and other polygons should follow
tri.add_polygon(area1)
tri.add_polygon(area2)
tri.add_region((1, 1), 0, maximum_area=100) # point inside active domain
tri.add_region((11, 11), 1, maximum_area=10) # point inside area1
tri.add_region((70, 70), 2, maximum_area=1) # point inside area2

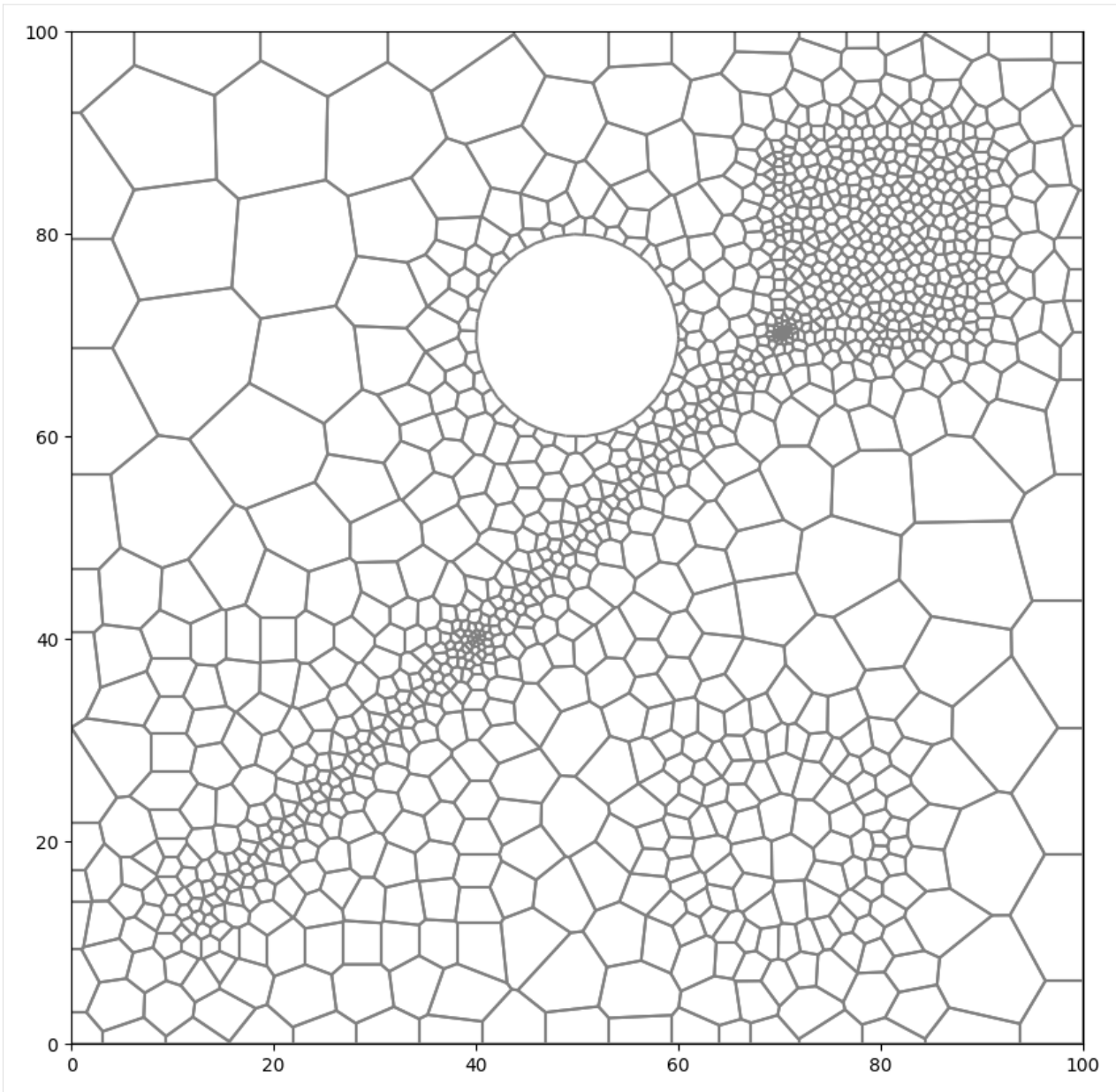
tri.build(verbose=False)

vor = VoronoiGrid(tri)
gridprops = vor.get_gridprops_vertexgrid()
voronoi_grid = VertexGrid(**gridprops, nlay=1)

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot()
ax.set_aspect("equal")
voronoi_grid.plot(ax=ax)

```

[13]: <matplotlib.collections.LineCollection at 0x7f4a3c81c7c0>



```
[14]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
except:
    pass
```

6.1.3 Intersecting model grids with shapes

Note: This feature requires the shapely package (which is an optional FloPy dependency).

This notebook shows the grid intersection functionality in flopy. The intersection methods are available through the `GridIntersect` object. A flopy modelgrid is passed to instantiate the object. Then the modelgrid can be intersected with Points, LineStrings and Polygons and their Multi variants.

Table of Contents

- `GridIntersect` Class
- Rectangular regular grid
 - Polygon with regular grid
 - `MultiLineString` with regular grid
 - `MultiPoint` with regular grid
- Vertex grid
 - Polygon with triangular grid
 - `MultiLineString` with triangular grid
 - `MultiPoint` with triangular grid

Import packages

```
[1]: import os
import sys

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import shapely
from shapely.geometry import (
    LineString,
    MultiLineString,
    MultiPoint,
    MultiPolygon,
    Point,
    Polygon,
)

import flopy
import flopy.discretization as fgrid
import flopy.plot as fplot
from flopy.utils import GridIntersect

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))
print("shapely version: {}".format(shapely.__version__))
```

```
3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
shapely version: 2.0.1
```

GridIntersect Class

The GridIntersect class is constructed by passing a flopy modelgrid object to the constructor. There are options users can select to change how the intersection is calculated.

- **method**: derived from model grid type or defined by the user: can be either "vertex" or "structured". If "structured" is passed, the intersections are performed using structured methods. These methods use information about the regular grid to limit the search space for intersection calculations. Note that **method="vertex"** also works for structured grids.
- **rtree**: either True (default) or False, only read when **method="vertex"**. When True, an STR-tree is built, which allows for fast spatial queries. Building the STR-tree does take some time however. Setting the option to False avoids building the STR-tree but requires the intersection calculation to loop through all grid cells.

In general the “vertex” option is robust and fast and is therefore recommended in most situations. In some rare cases building the STR-tree might not be worth the time, in which case it can be avoided by passing **rtree=False**. If you are working with a structured grid, then the **method="structured"** can speed up intersection operations in some situations (e.g. for (multi)points) with the added advantage of not having to build an STR-tree.

The important methods in the GridIntersect object are:

- **intersects()**: returns cellids for gridcells that intersect a shape (accepts shapely geometry objects, flopy geometry object, shapefile.Shape objects, and geojson objects)
- **intersect()**: for intersecting the modelgrid with point, linestrings, and polygon geometries (accepts shapely geometry objects, flopy geometry object, shapefile.Shape objects, and geojson objects)
- **plot_point()**: for plotting point intersection results
- **plot_linestring()**: for plotting linestring intersection results
- **plot_polygon()**: for plotting polygon intersection results

In the following sections examples of intersections are shown for structured and vertex grids for different types of shapes (Polygon, LineString and Point).

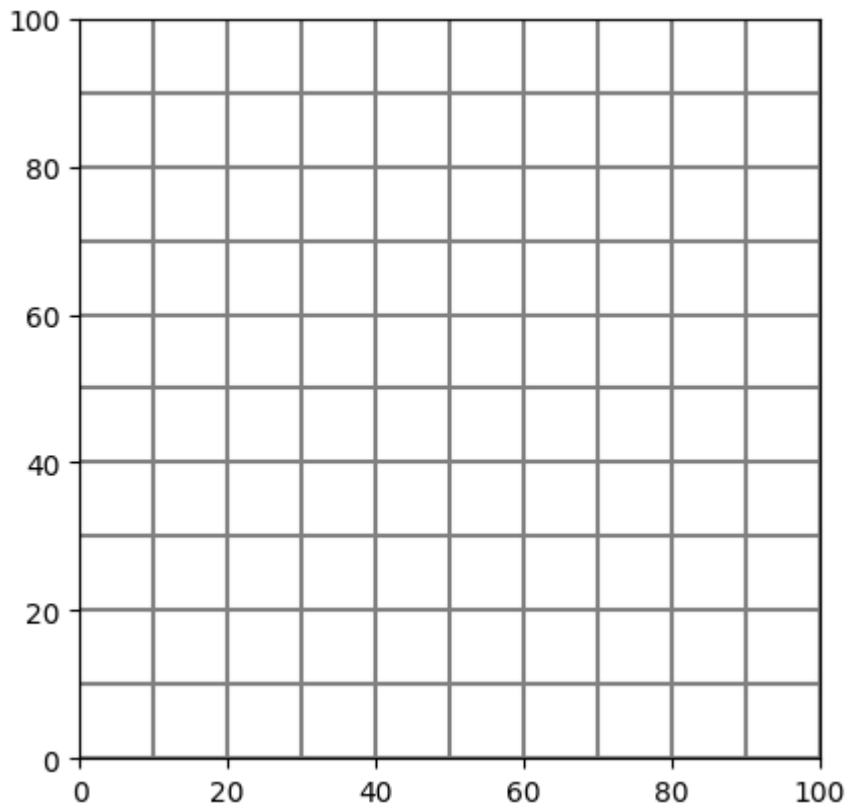
Rectangular regular grid

```
[2]: delc = 10 * np.ones(10, dtype=float)
delr = 10 * np.ones(10, dtype=float)
```

```
[3]: xoff = 0.0
yoff = 0.0
angrot = 0.0
sgr = fgrid.StructuredGrid(
    delc, delr, top=None, botm=None, xoff=xoff, yoff=yoff, angrt=angrot
)
```



```
[4]: sgr.plot()
[4]: <matplotlib.collections.LineCollection at 0x7f1e0d612ee0>
```



Polygon with regular grid

Polygon to intersect with:

```
[5]: p = Polygon(
    shell=[
        (15, 15),
        (20, 50),
        (35, 80.0),
        (80, 50),
        (80, 40),
        (40, 5),
        (15, 12),
    ],
    holes=[[ (25, 25), (25, 45), (45, 45), (45, 25) ]],
)
```

Create the GridIntersect class for our modelgrid. The method kwarg is passed to force GridIntersect to use the "vertex" intersection methods.

```
[6]: ix = GridIntersect(sgr, method="vertex")
```

Do the intersect operation for a polygon

```
[7]: result = ix.intersect(p)
```

The results are returned as a numpy.recarray containing several fields based on the intersection performed. An explanation of the data in each of the possible fields is given below: - **cellids**: contains the cell ids of the intersected grid cells - **vertices**: contains the vertices of the intersected shape - **areas**: contains the area of the polygon in that grid cell (only for polygons) - **lengths**: contains the length of the linestring in that grid cell (only for linestrings) - **ixshapes**: contains the shapely object representing the intersected shape (useful for plotting the result)

Looking at the first few entries of the results of the polygon intersection (convert to pandas.DataFrame for prettier formatting)

```
[8]: result[:5]
# pd.DataFrame(result) # recommended for prettier formatting and working with result

[8]: rec.array([(2, 3), <POLYGON ((35 80, 40 76.667, 40 70, 30 70, 35 80))>, 66.66666667),
              ((2, 4), <POLYGON ((50 70, 40 70, 40 76.667, 50 70))>, 33.33333333),
              ((3, 2), <POLYGON ((30 70, 30 60, 25 60, 30 70))>, 25.          ),
              ((3, 3), <POLYGON ((40 70, 40 60, 30 60, 30 70, 40 70))>, 100.          ),
              ((3, 4), <POLYGON ((50 70, 50 60, 40 60, 40 70, 50 70))>, 100.          )],
              dtype=[('cellids', '0'), ('ixshapes', '0'), ('areas', '<f8')])
```

The cellids can be easily obtained

```
[9]: result.cellids

[9]: array([(2, 3), (2, 4), (3, 2), (3, 3), (3, 4), (3, 5), (3, 6), (4, 2),
              (4, 3), (4, 4), (4, 5), (4, 6), (4, 7), (5, 1), (5, 2), (5, 3),
              (5, 4), (5, 5), (5, 6), (5, 7), (6, 1), (6, 2), (6, 4), (6, 5),
              (6, 6), (6, 7), (7, 1), (7, 2), (7, 3), (7, 4), (7, 5), (7, 6),
              (8, 1), (8, 2), (8, 3), (8, 4), (8, 5), (9, 2), (9, 3), (9, 4)],
              dtype=object)
```

Or the areas

```
[10]: result.areas

[10]: array([ 66.66666667,  33.33333333,  25.          , 100.          ,
              100.          ,  66.66666667,   8.33333333,  75.          ,
              100.          , 100.          , 100.          ,  91.66666667,
              33.33333333,   7.14285714,  75.          ,  50.          ,
              75.          , 100.          , 100.          , 100.          ,
              21.42857143,  50.          ,  50.          , 100.          ,
              99.10714286,  43.75         ,  35.71428571,  75.          ,
              50.          ,  75.          ,  96.42857143,  32.14285714,
              41.71428571,  99.35714286, 100.          ,  91.96428571,
              22.32142857,   8.64285714,  36.          ,  14.28571429])
```

If a user is only interested in which cells the shape intersects (and not the areas or the actual shape of the intersected object) with there is also the `intersects()` method. This method works for all types of shapely geometries.

```
[11]: ix.intersects(p)

[11]: rec.array([(8, 1), (7, 2), (7, 1), (6, 1), (6, 2),
              (9, 4), (9, 2), (9, 3), (8, 2), (8, 3),
              (8, 4), (7, 3), (7, 4), (6, 4), (5, 1),
```

(continues on next page)

(continued from previous page)

```

((4, 1),), ((4, 2),), ((3, 2),), ((5, 2),), ((5, 3),),
((5, 4),), ((4, 3),), ((4, 4),), ((3, 4),), ((3, 3),),
((2, 4),), ((2, 3),), ((2, 2),), ((1, 3),), ((8, 5),),
((7, 5),), ((7, 6),), ((6, 5),), ((6, 6),), ((6, 7),),
((6, 8),), ((5, 5),), ((5, 7),), ((5, 6),), ((4, 6),),
((4, 5),), ((3, 6),), ((3, 5),), ((2, 5),), ((5, 8),),
((4, 8),), ((4, 7),)],
dtype=[('cellids', '0')])

```

The results of an intersection can be visualized with the plotting methods in the GridIntersect object: - plot_polygon - plot_linestring - plot_point

```

[12]: # create a figure and plot the grid
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
sgr.plot(ax=ax)

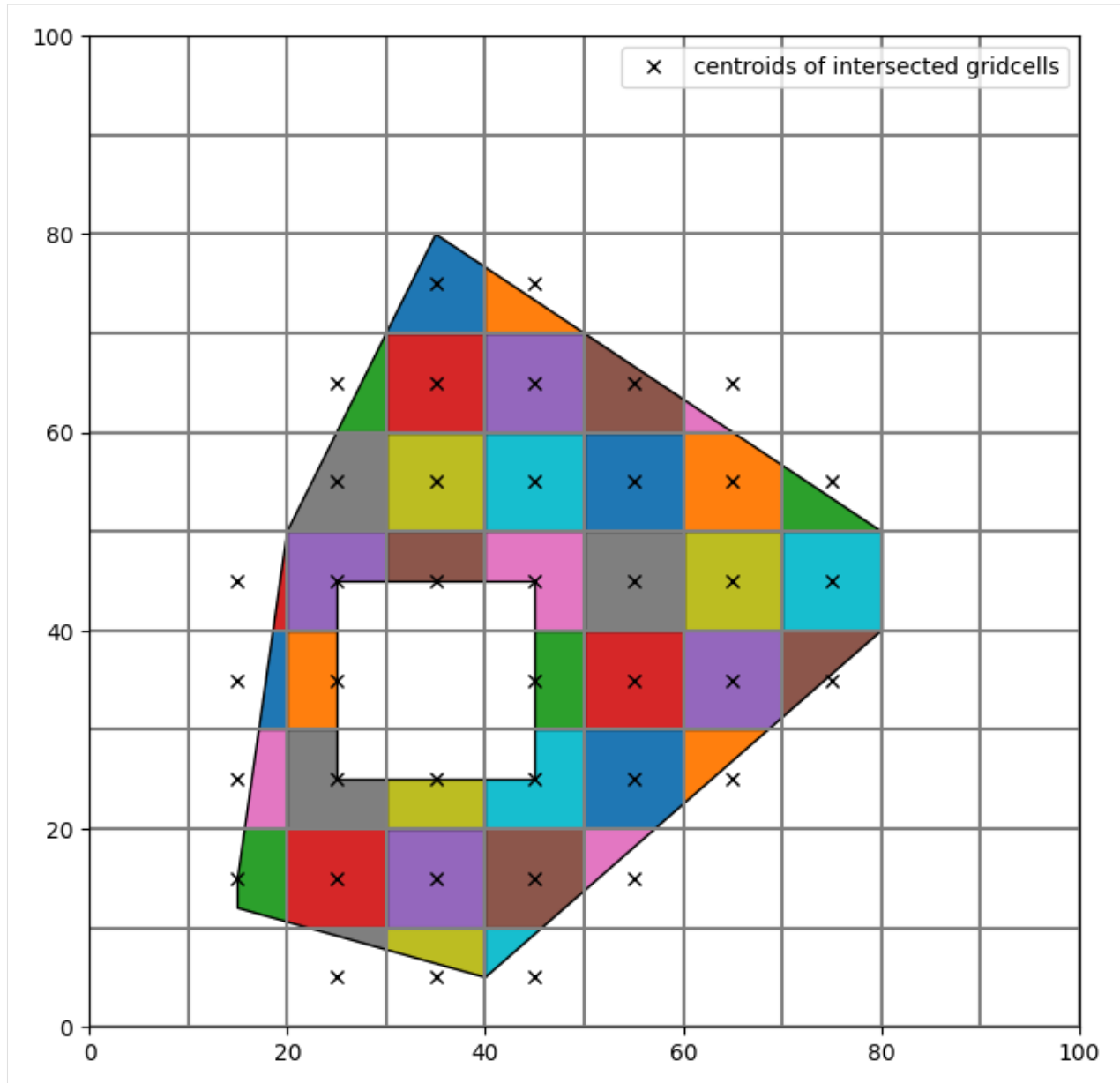
# the intersection object contains some helpful plotting commands
ix.plot_polygon(result, ax=ax)

# add black x at cell centers
for irow, icol in result.cellids:
    (h2,) = ax.plot(
        sgr.xcellcenters[0, icol],
        sgr.ycellcenters[irow, 0],
        "kx",
        label="centroids of intersected gridcells",
    )

# add legend
ax.legend([h2], [i.get_label() for i in [h2]], loc="best")

[12]: <matplotlib.legend.Legend at 0x7f1e0d49cb20>

```



The `intersect()` method contains several keyword arguments that specifically deal with polygons:

- `contains_centroid`: only store intersection result if cell centroid is contained within polygon
- `min_area_fraction`: minimal intersecting cell area (expressed as a fraction of the total cell area) to include cells in intersection result

Two examples showing the usage of these keyword arguments are shown below.

Example with `contains_centroid` set to `True`, only cells in which centroid is within the intersected polygon are stored. Note the difference with the previous result.

```
[13]: # contains_centroid example

result2 = ix.intersect(p, contains_centroid=True)
```

(continues on next page)

(continued from previous page)

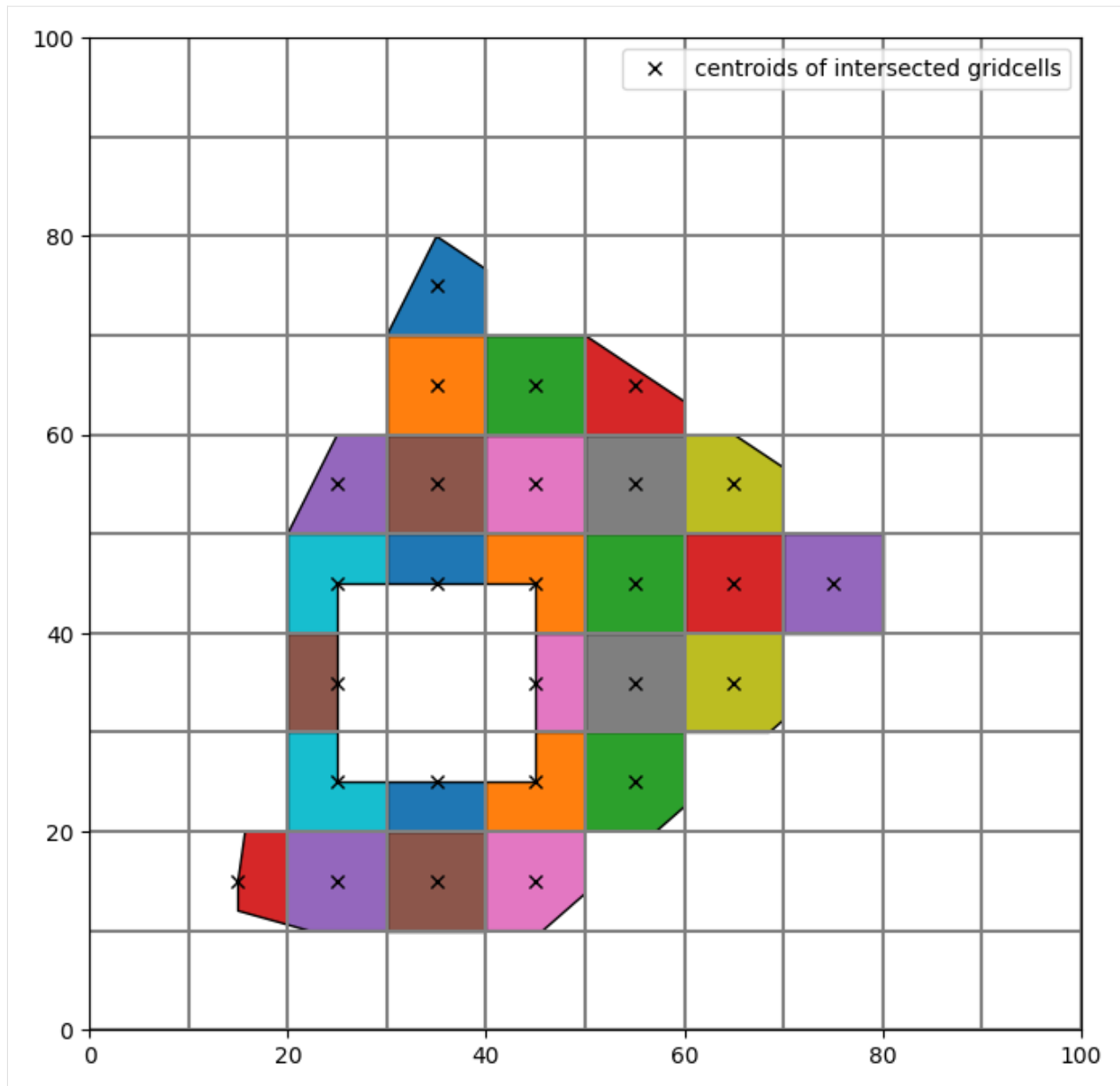
```
# create a figure and plot the grid
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
sgr.plot(ax=ax)

# the intersection object contains some helpful plotting commands
ix.plot_polygon(result2, ax=ax)

# add black x at cell centers
for irow, icol in result2.cellids:
    (h2,) = ax.plot(
        sgr.xcellcenters[0, icol],
        sgr.ycellcenters[irow, 0],
        "kx",
        label="centroids of intersected gridcells",
    )

# add legend
ax.legend([h2], [i.get_label() for i in [h2]], loc="best")
```

```
[13]: <matplotlib.legend.Legend at 0x7f1e0b1656d0>
```



Example with `min_area_threshold` set to 0.35, the intersection result in a cell should cover 35% or more of the cell area.

[14]: *# min_area_threshold example*

```
result3 = ix.intersect(p, min_area_fraction=0.35)
```

```
# create a figure and plot the grid
```

```
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
```

```
sgr.plot(ax=ax)
```

```
# the intersection object contains some helpful plotting commands
```

```
ix.plot_polygon(result3, ax=ax)
```

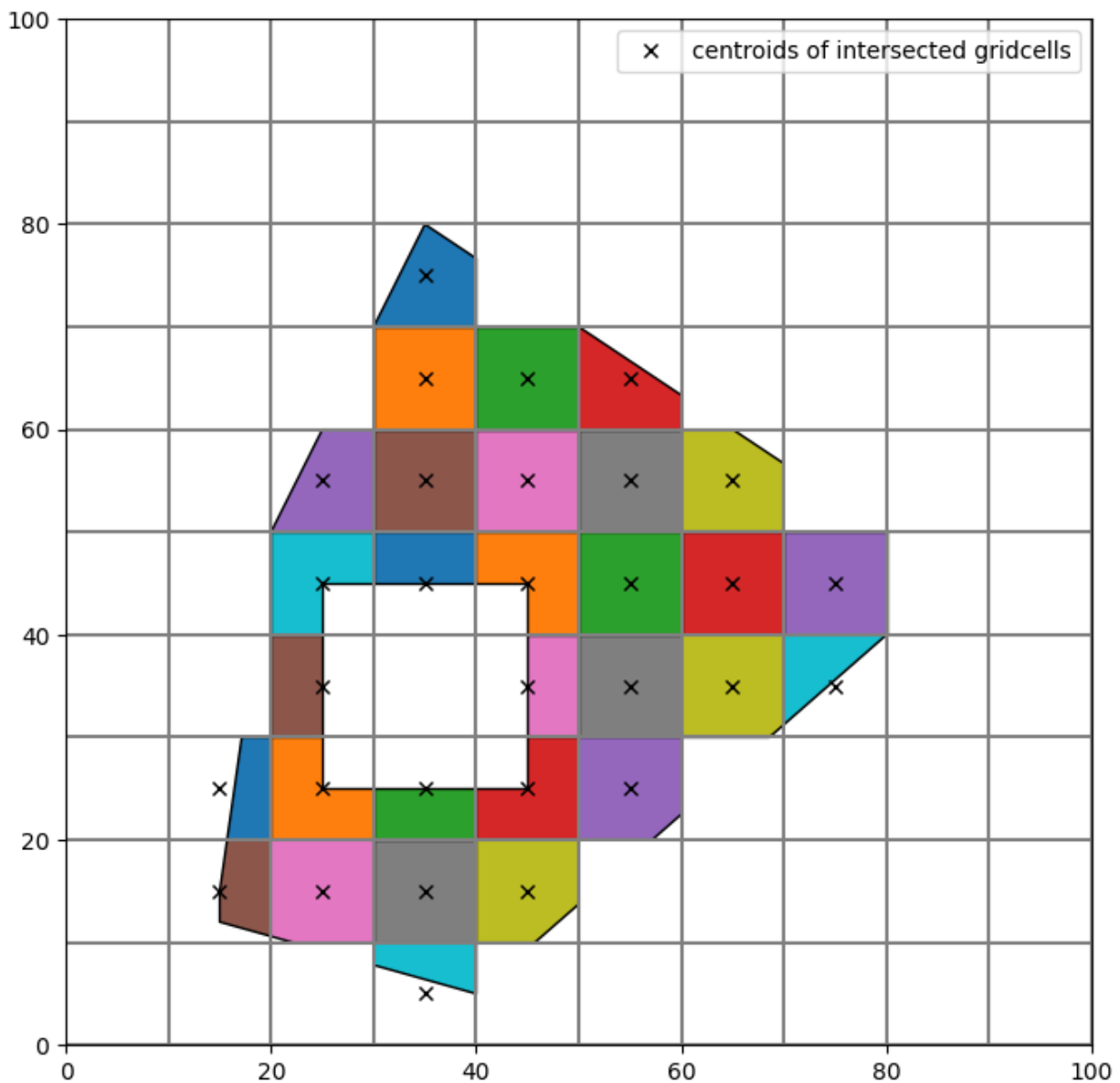
(continues on next page)

(continued from previous page)

```
# add black x at cell centers
for irow, icol in result3.cellids:
    (h2,) = ax.plot(
        sgr.xcellcenters[0, icol],
        sgr.ycellcenters[irow, 0],
        "kx",
        label="centroids of intersected gridcells",
    )

# add legend
ax.legend([h2], [i.get_label() for i in [h2]], loc="best")
```

[14]: <matplotlib.legend.Legend at 0x7f1e0b0ad700>



Alternatively, the intersection can be calculated using special methods optimized for structured grids. Access these

methods by instantiating the GridIntersect class with the method="structured" keyword argument.

```
[15]: ix = GridIntersect(sgr, method="structured")
      result4 = ix.intersect(p)
```

The result is the same as before:

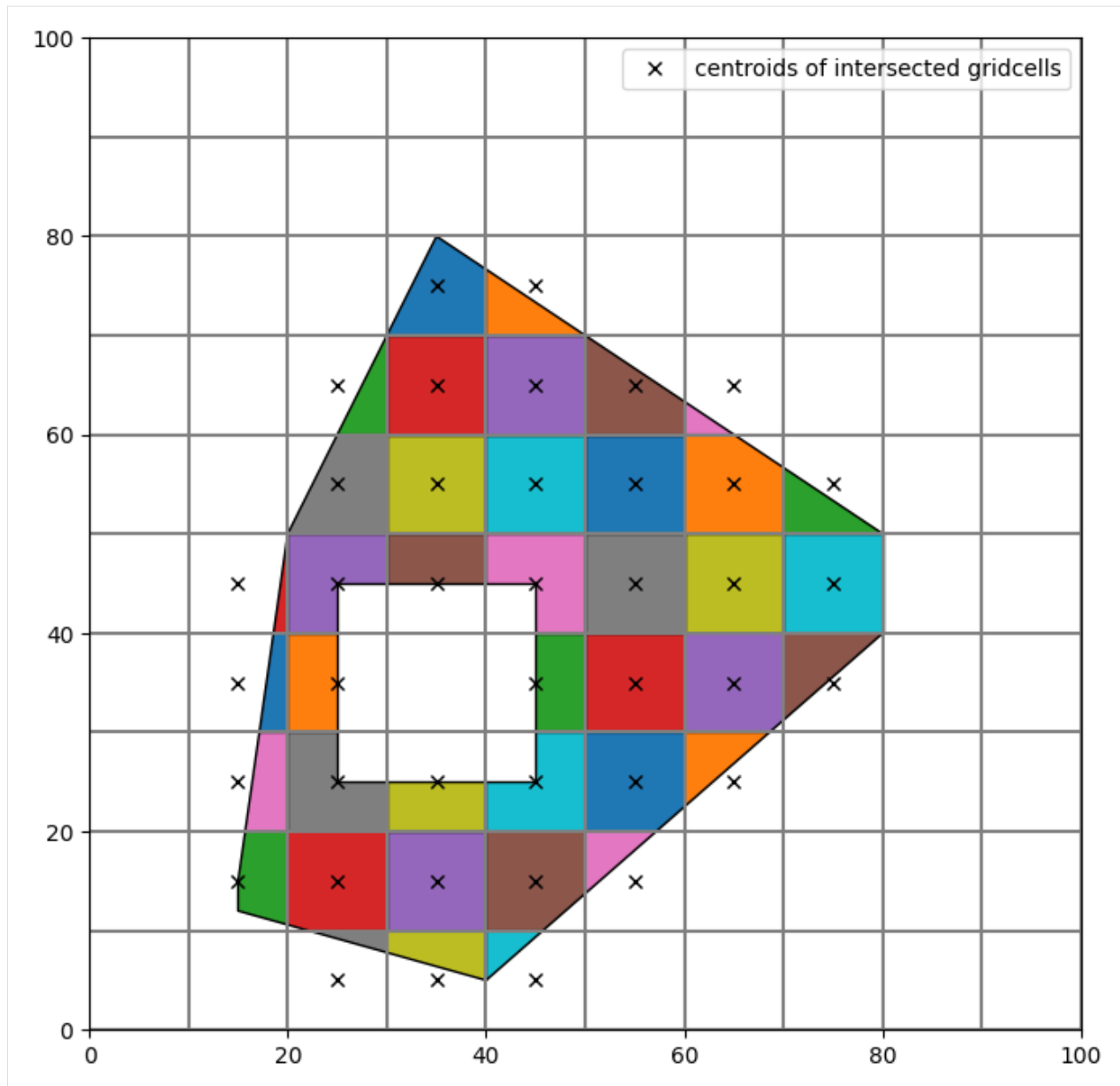
```
[16]: # create a figure and plot the grid
      fig, ax = plt.subplots(1, 1, figsize=(8, 8))
      sgr.plot(ax=ax)

      # the intersection object contains some helpful plotting commands
      ix.plot_polygon(result4, ax=ax)

      # add black x at cell centers
      for irow, icol in result4.cellids:
          (h2,) = ax.plot(
              sgr.xcellcenters[0, icol],
              sgr.ycellcenters[irow, 0],
              "kx",
              label="centroids of intersected gridcells",
          )

      # add legend
      ax.legend([h2], [i.get_label() for i in [h2]], loc="best")

[16]: <matplotlib.legend.Legend at 0x7f1e0afff550>
```

Polyline with regular grid

MultiLineString to intersect with:

```
[17]: ls1 = LineString([(95, 105), (30, 50)])
ls2 = LineString([(30, 50), (90, 22)])
ls3 = LineString([(90, 22), (0, 0)])
mls = MultiLineString(lines=[ls1, ls2, ls3])
```

```
[18]: result = ix.intersect(mls)
```

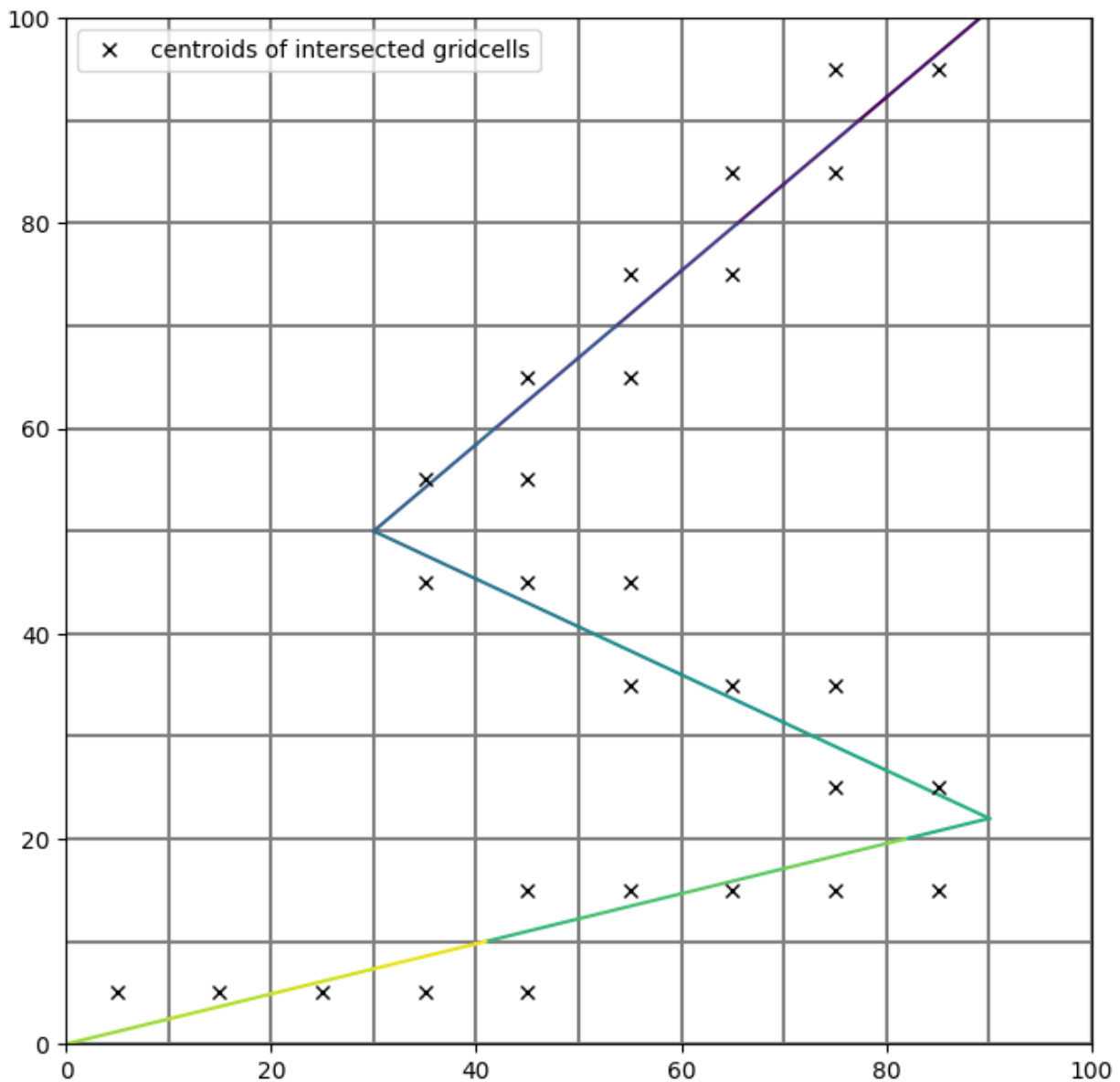
Plot the result

```
[19]: fig, ax = plt.subplots(1, 1, figsize=(8, 8))
sgr.plot(ax=ax)
ix.plot_linestring(result, ax=ax, cmap="viridis")

for irow, icol in result.cellids:
    (h2,) = ax.plot(
        sgr.xcellcenters[0, icol],
        sgr.ycellcenters[irow, 0],
        "kx",
        label="centroids of intersected gridcells",
    )

ax.legend([h2], [i.get_label() for i in [h2]], loc="best")

[19]: <matplotlib.legend.Legend at 0x7f1e0ac91d00>
```



Same as before, the intersect for structured grids can also be performed with a different method optimized for structured grids

```
[20]: ix = GridIntersect(sgr, method="structured")
```

```
[21]: result2 = ix.intersect(mls)
```

```
# ordering is different so compare sets to check equality
check = len(set(result2.cellids) - set(result.cellids)) == 0
print(
    "Intersection result with method='structured' and "
    f"method='vertex' are equal: {check}"
)
```

```
Intersection result with method='structured' and method='vertex' are equal: True
```

MultiPoint with regular grid

MultiPoint to intersect with

```
[22]: mp = MultiPoint(
    points=[
        Point(50.0, 0.0),
        Point(45.0, 45.0),
        Point(10.0, 10.0),
        Point(150.0, 100.0),
    ]
)
```

For points and linestrings there is a keyword argument `return_all_intersections` which will return multiple intersection results for points or (parts of) linestrings on cell boundaries. As an example, the difference is shown with the MultiPoint intersection. Note the number of red “+” symbols indicating the centroids of intersected cells, in the bottom left case, there are 4 results because the point lies exactly on the intersection between 4 grid cells.

```
[23]: result = ix.intersect(mp)
result_all = ix.intersect(mp, return_all_intersections=True)
```

```
[24]: fig, ax = plt.subplots(1, 1, figsize=(8, 8))
sgr.plot(ax=ax)
ix.plot_point(result, ax=ax, s=50, color="C0")
ix.plot_point(result_all, ax=ax, s=50, marker=".", color="C3")

for irow, icol in result.cellids:
    (h2,) = ax.plot(
        sgr.xcellcenters[0, icol],
        sgr.ycellcenters[irow, 0],
        "kx",
        ms=15,
        label="centroids of intersected cells",
    )

for irow, icol in result_all.cellids:
```

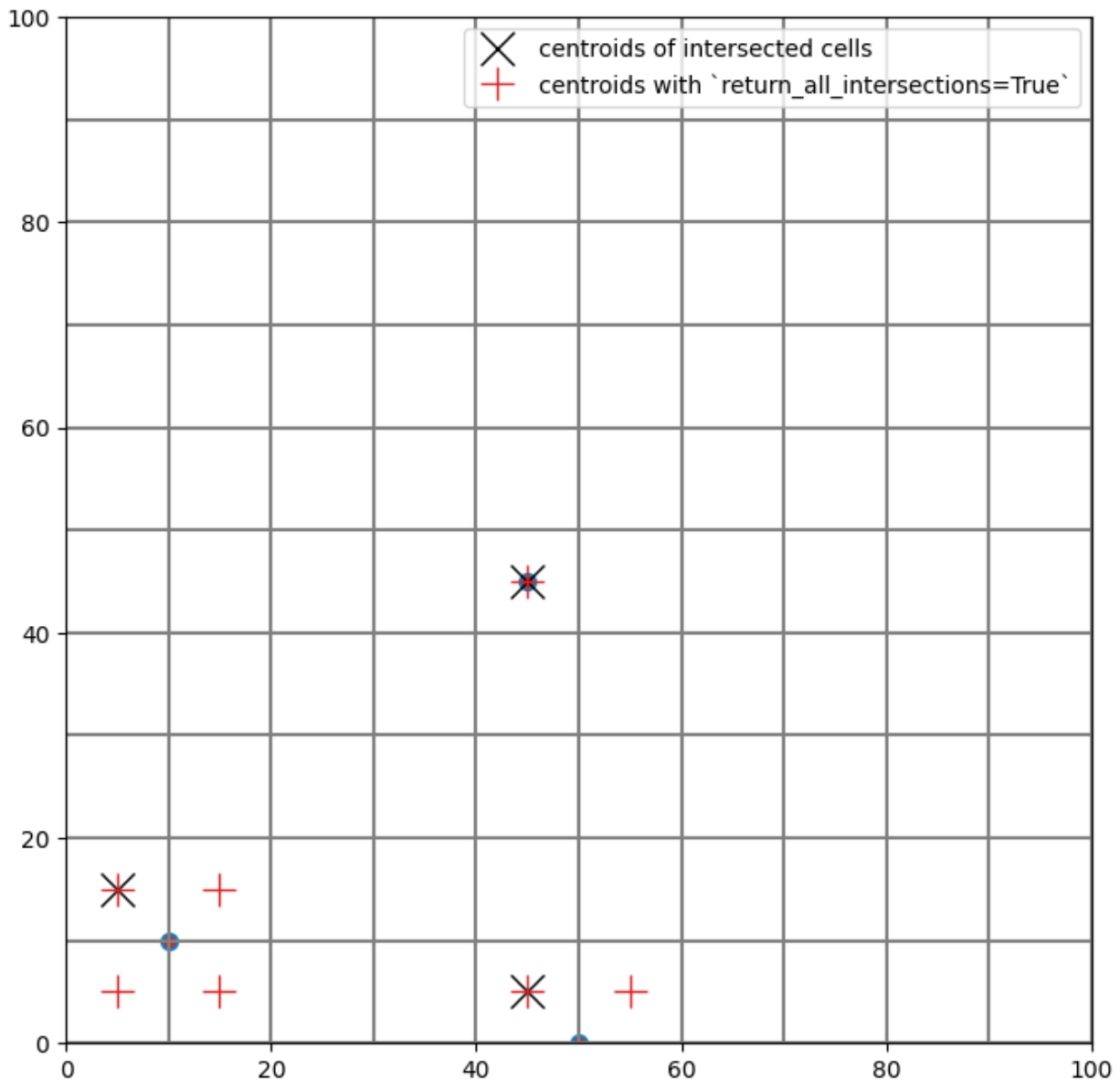
(continues on next page)

(continued from previous page)

```
(h3,) = ax.plot(
    sgr.xcellcenters[0, icol],
    sgr.ycellcenters[irow, 0],
    "C3+",
    ms=15,
    label="centroids with `return_all_intersections=True`",
)

ax.legend([h2, h3], [i.get_label() for i in [h2, h3]], loc="best")
```

[24]: <matplotlib.legend.Legend at 0x7f1e0abb1b20>



Same as before, the intersect for structured grids can also be performed with a different method written specifically for structured grids.

```
[25]: ix = GridIntersect(sgr, method="structured")
```

```
[26]: result2 = ix.intersect(mp, return_all_intersections=False)
```

```
# ordering is different so compare sets to check equality
check = len(set(result2.cellids) - set(result.cellids)) == 0
print(
    "Intersection result with method='structured' and "
    f"method='vertex' are equal: {check}"
)
```

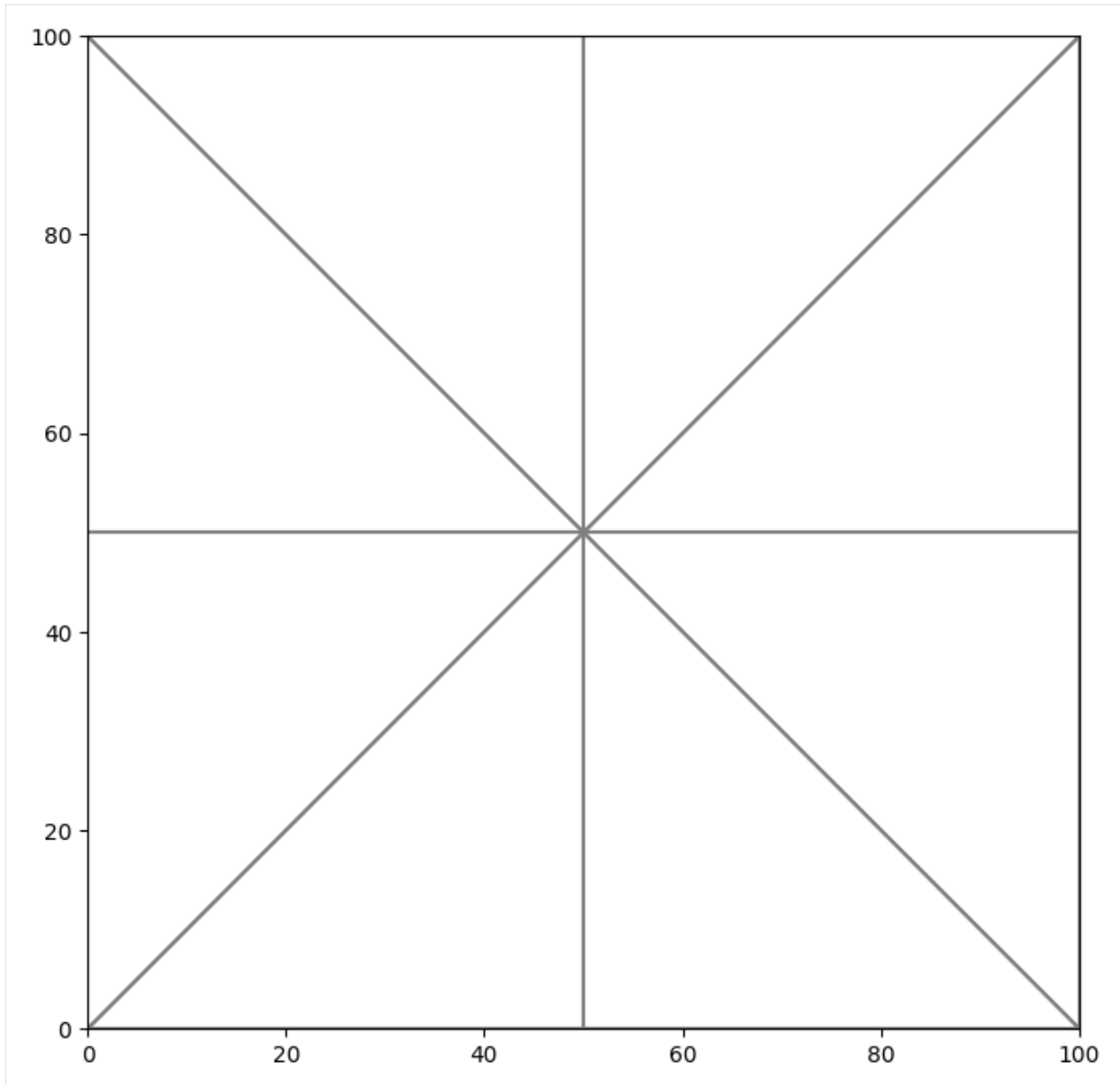
```
Intersection result with method='structured' and method='vertex' are equal: True
```

Vertex Grid

```
[27]: cell2d = [
    [0, 83.33333333333333, 66.66666666666667, 3, 4, 2, 7],
    [1, 16.666666666666668, 33.333333333333336, 3, 4, 0, 5],
    [2, 33.333333333333336, 83.33333333333333, 3, 1, 8, 4],
    [3, 16.666666666666668, 66.66666666666667, 3, 5, 1, 4],
    [4, 33.333333333333336, 16.666666666666668, 3, 6, 0, 4],
    [5, 66.66666666666667, 16.666666666666668, 3, 4, 3, 6],
    [6, 83.33333333333333, 33.333333333333336, 3, 7, 3, 4],
    [7, 66.66666666666667, 83.33333333333333, 3, 8, 2, 4],
]
vertices = [
    [0, 0.0, 0.0],
    [1, 0.0, 100.0],
    [2, 100.0, 100.0],
    [3, 100.0, 0.0],
    [4, 50.0, 50.0],
    [5, 0.0, 50.0],
    [6, 50.0, 0.0],
    [7, 100.0, 50.0],
    [8, 50.0, 100.0],
]
tgr = fgrid.VertexGrid(vertices, cell2d)
```

```
[28]: fig, ax = plt.subplots(1, 1, figsize=(8, 8))
pmv = fplot.PlotMapView(modelgrid=tgr)
pmv.plot_grid(ax=ax)
```

```
[28]: <matplotlib.collections.LineCollection at 0x7f1e0aafb400>
```



Polygon with triangular grid

```
[29]: ix2 = GridIntersect(tgr)
```

```
[30]: result = ix2.intersect(p)
```

```
[31]: fig, ax = plt.subplots(1, 1, figsize=(8, 8))
      pmv = fplot.PlotMapView(ax=ax, modelgrid=tgr)
      pmv.plot_grid()
      ix.plot_polygon(result, ax=ax)
```

```
# only cells that intersect with shape
```

(continues on next page)

(continued from previous page)

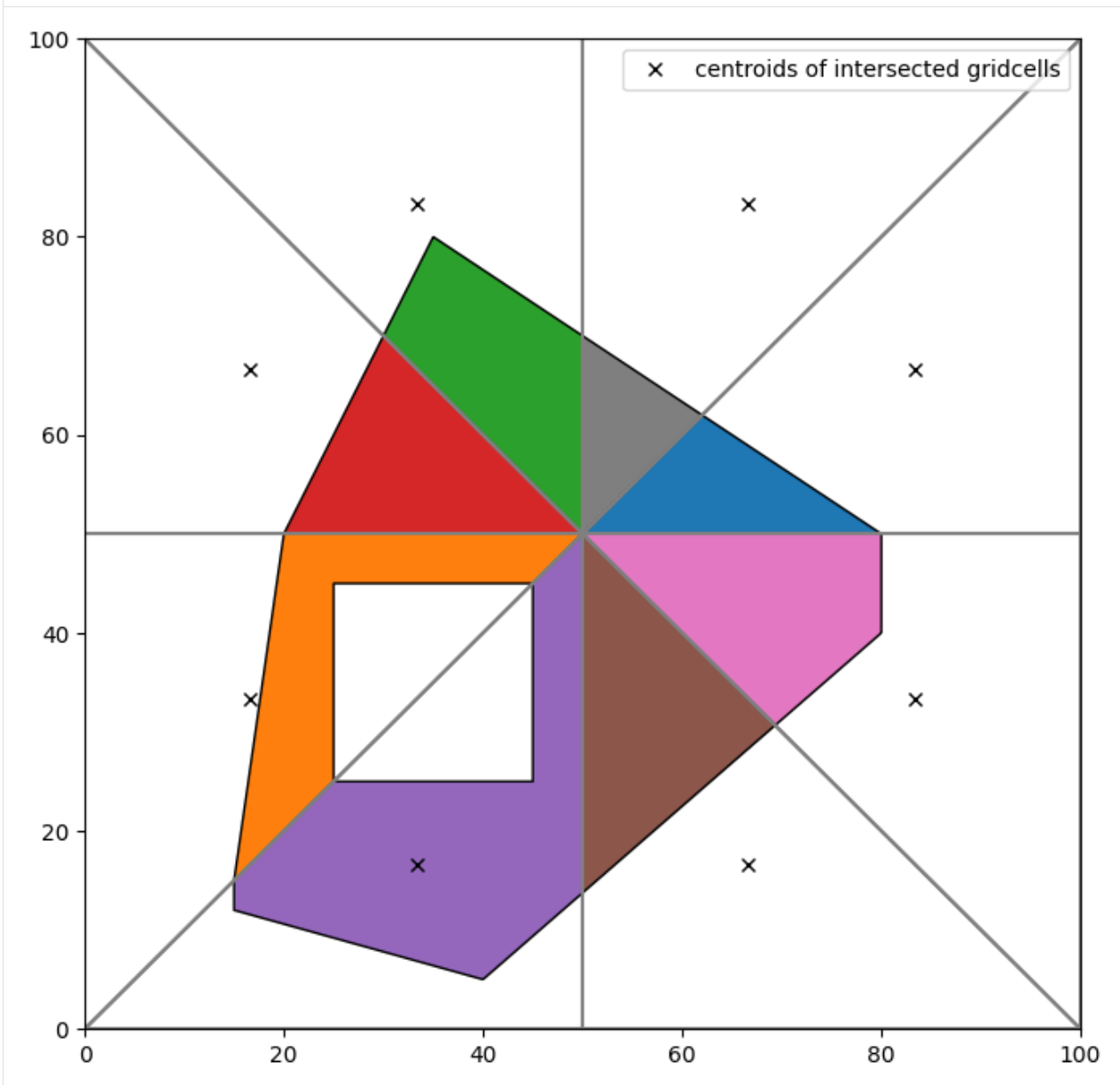
```

for cellid in result.cellids:
    (h2,) = ax.plot(
        tgr.xcellcenters[cellid],
        tgr.ycellcenters[cellid],
        "kx",
        label="centroids of intersected gridcells",
    )

ax.legend([h2], [i.get_label() for i in h2], loc="best")

```

[31]: <matplotlib.legend.Legend at 0x7f1e0aa955b0>



LineString with triangular grid

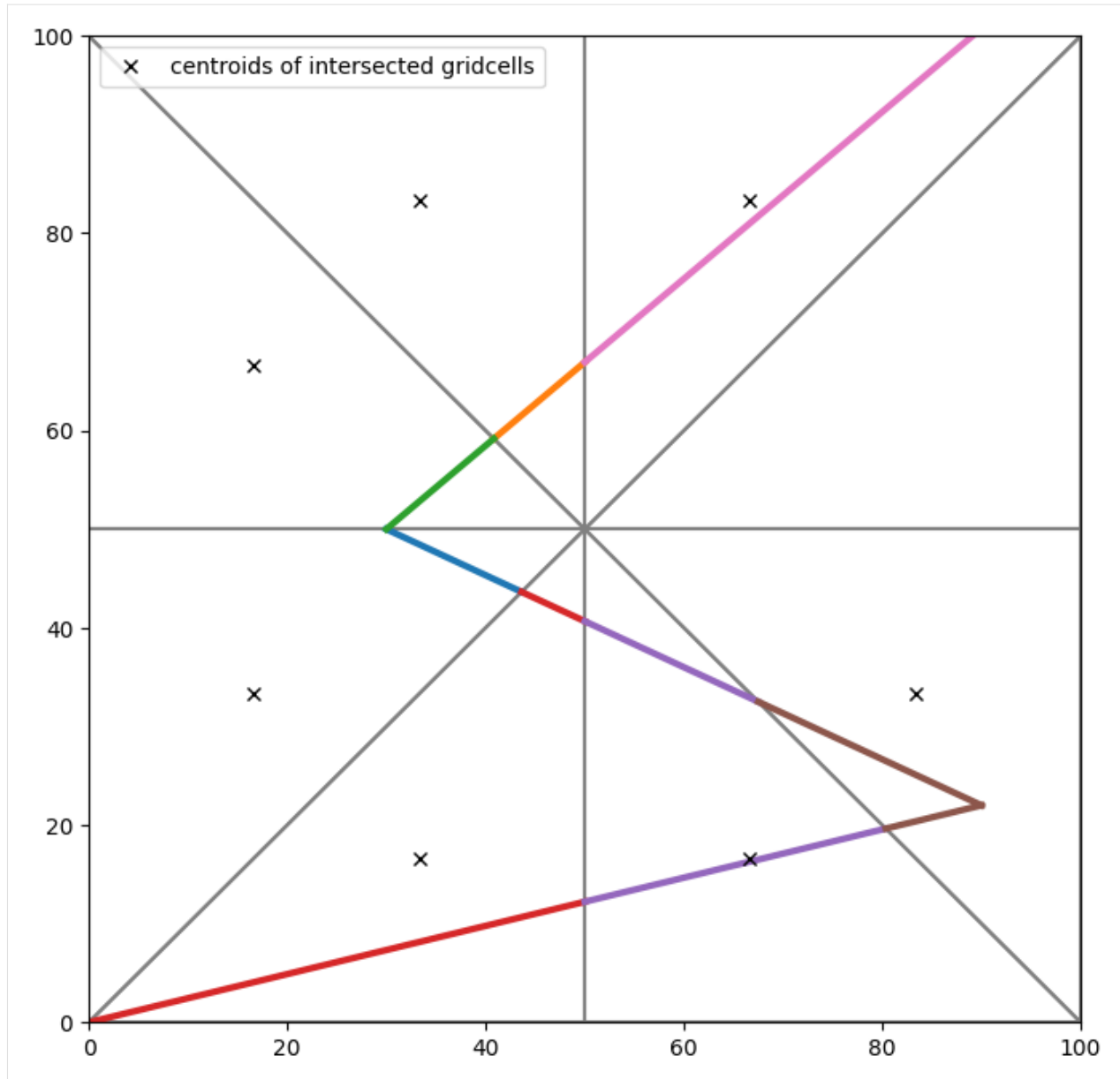
```
[32]: result = ix2.intersect(mls)

[33]: fig, ax = plt.subplots(1, 1, figsize=(8, 8))
      pmv = fplot.PlotMapView(ax=ax, modelgrid=tgr)
      pmv.plot_grid()
      ix2.plot_linestring(result, ax=ax, lw=3)

      for cellid in result.cellids:
          (h2,) = ax.plot(
              tgr.xcellcenters[cellid],
              tgr.ycellcenters[cellid],
              "kx",
              label="centroids of intersected gridcells",
          )

      ax.legend([h2], [i.get_label() for i in [h2]], loc="best")

[33]: <matplotlib.legend.Legend at 0x7f1e0aa95640>
```

MultiPoint with triangular grid

```
[34]: result = ix2.intersect(mp)
      result_all = ix2.intersect(mp, return_all_intersections=True)
```

```
[35]: fig, ax = plt.subplots(1, 1, figsize=(8, 8))
      pmv = fplot.PlotMapView(ax=ax, modelgrid=tgr)
      pmv.plot_grid()
      ix2.plot_point(result, ax=ax, color="k", zorder=5, s=80)

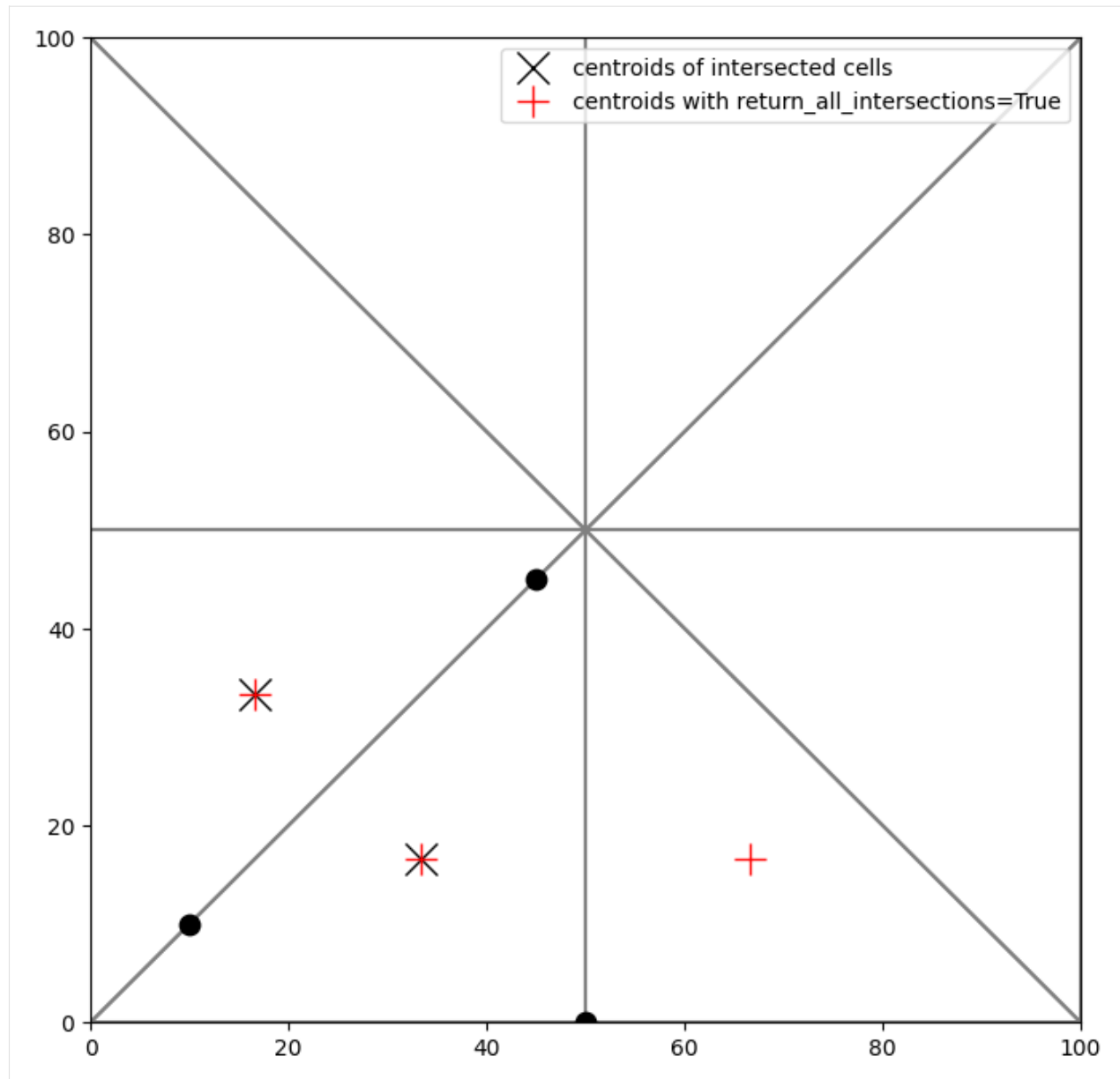
      for cellid in result.cellids:
          (h2,) = ax.plot(
```

(continues on next page)

(continued from previous page)

```
        tgr.xcellcenters[cellid],
        tgr.ycellcenters[cellid],
        "kx",
        ms=15,
        label="centroids of intersected cells",
    )
    for cellid in result_all.cellids:
        (h3,) = ax.plot(
            tgr.xcellcenters[cellid],
            tgr.ycellcenters[cellid],
            "r+",
            ms=15,
            label="centroids with return_all_intersections=True",
        )
    ax.legend([h2, h3], [i.get_label() for i in [h2, h3]], loc="best")
```

```
[35]: <matplotlib.legend.Legend at 0x7f1e0896f370>
```



6.1.4 Creating Layered Quadtree Grids with GRIDGEN

FloPy has a module that can be used to drive the GRIDGEN program. This notebook shows how it works.

Import Modules and Locate Gridgen Executable

```
[1]: import os
import shutil
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

from flopy.utils import flopy_io
from flopy.utils.gridgen import Gridgen

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

The Flopy GRIDGEN module requires that the gridgen executable can be called using subprocess (i.e., **gridgen** is in your path).

```
[2]: gridgen_exe = flopy.which("gridgen")
if gridgen_exe is None:
    msg = (
        "Warning, gridgen is not in your path. "
        "When you create the gridgen object you will need to "
        "provide a full path to the gridgen binary executable."
    )
    print(msg)
else:
    print(
        "gridgen executable was found at: {}".format(
            flopy_io.relpath_safe(gridgen_exe)
        )
    )

gridgen executable was found at: ../../../../local/bin/modflow/gridgen
```

```
[3]: # temporary directory
temp_dir = TemporaryDirectory()
model_ws = temp_dir.name

gridgen_ws = os.path.join(model_ws, "gridgen")
if not os.path.exists(gridgen_ws):
    os.makedirs(gridgen_ws, exist_ok=True)
print("Model workspace is : {}".format(flopy_io.scrub_login(model_ws)))
print("Gridgen workspace is : {}".format(flopy_io.scrub_login(gridgen_ws)))

Model workspace is : /tmp/tmpo9cwtze3
Gridgen workspace is : /tmp/tmpo9cwtze3/gridgen
```

Basic Gridgen Operations

Setup Base MODFLOW Grid

GRIDGEN works off of a base MODFLOW grid. The following information defines the basegrid.

```
[4]: Lx = 100.0
Ly = 100.0
nlay = 2
nrow = 51
ncol = 51
delr = Lx / ncol
delc = Ly / nrow
h0 = 10
h1 = 5
top = h0
botm = np.zeros((nlay, nrow, ncol), dtype=np.float32)
botm[1, :, :] = -10.0
```

```
[5]: ms = flopy.modflow.Modflow(rotation=-20.0)
dis = flopy.modflow.ModflowDis(
    ms,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=delr,
    delc=delc,
    top=top,
    botm=botm,
)
```

Create the Gridgen Object

```
[6]: g = Gridgen(ms.modelgrid, model_ws=gridgen_ws)
```

Add an Optional Active Domain

Cells outside of the active domain will be clipped and not numbered as part of the final grid. If this step is not performed, then all cells will be included in the final grid.

```
[7]: # setup the active domain
adshp = os.path.join(gridgen_ws, "ad0")
adpoly = [[(0, 0), (0, 60), (40, 80), (60, 0), (0, 0)]]
# g.add_active_domain(adpoly, range(nlay))
```

Refine the Grid

```
[8]: x = Lx * np.random.random(10)
y = Ly * np.random.random(10)
wells = list(zip(x, y))
g.add_refinement_features(wells, "point", 3, range(nlay))
rf0shp = os.path.join(gridgen_ws, "rf0")
```

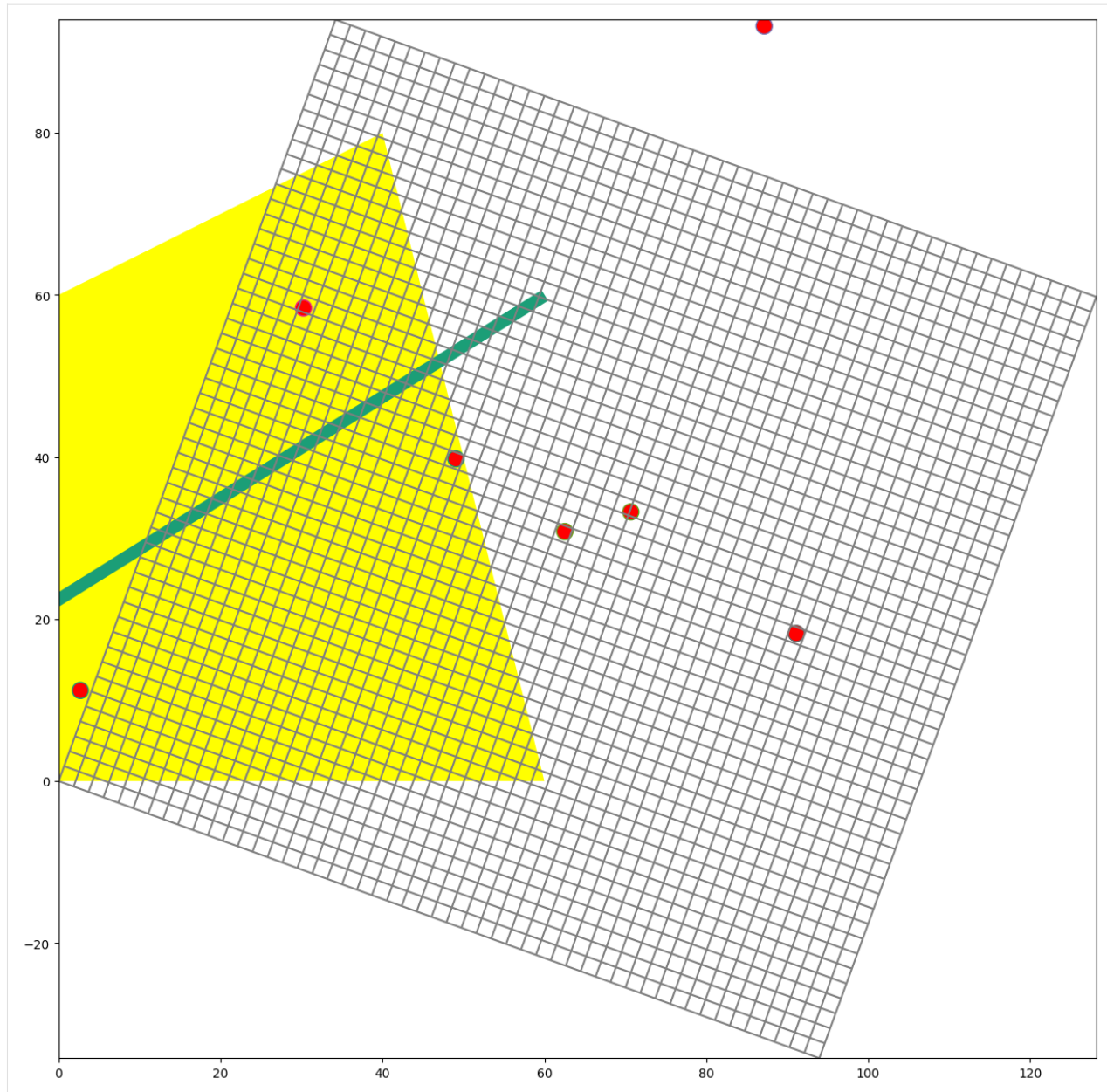
```
[9]: river = [(-20, 10), (60, 60)]
g.add_refinement_features(river, "line", 3, range(nlay))
rf1shp = os.path.join(gridgen_ws, "rf1")
```

```
[10]: g.add_refinement_features(adpoly, "polygon", 1, range(nlay))
rf2shp = os.path.join(gridgen_ws, "rf2")
```

Plot the Gridgen Input

```
[11]: fig = plt.figure(figsize=(15, 15))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mm = flopy.plot.PlotMapView(model=ms)
mm.plot_grid()
flopy.plot.plot_shapefile(rf2shp, ax=ax, facecolor="yellow", edgecolor="none")
flopy.plot.plot_shapefile(rf1shp, ax=ax, linewidth=10)
flopy.plot.plot_shapefile(rf0shp, ax=ax, facecolor="red", radius=1)
```

```
[11]: <matplotlib.collections.PatchCollection at 0x7f43c74c0fd0>
```



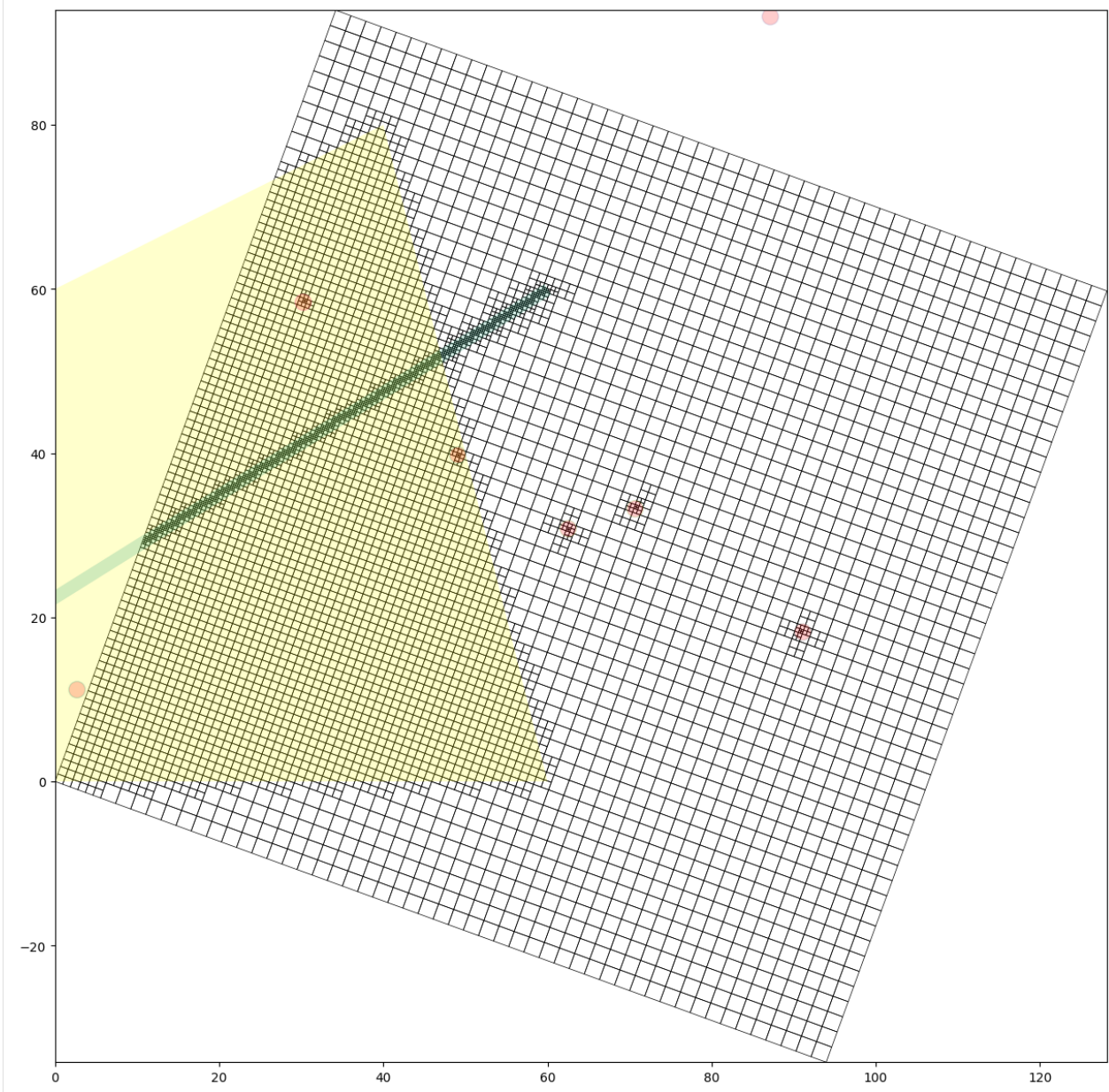
Build the Grid

```
[12]: g.build(verbose=False)
```

Plot the Grid

```
[13]: fig = plt.figure(figsize=(15, 15))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      g.plot(ax, linewidth=0.5)
      flopy.plot.plot_shapefile(
          rf2shp, ax=ax, facecolor="yellow", edgecolor="none", alpha=0.2
      )
      flopy.plot.plot_shapefile(rf1shp, ax=ax, linewidth=10, alpha=0.2)
      flopy.plot.plot_shapefile(rf0shp, ax=ax, facecolor="red", radius=1, alpha=0.2)
```

```
[13]: <matplotlib.collections.PatchCollection at 0x7f43bac359d0>
```



Create a FloPy ModflowDisu Object

```
[14]: mu = flopy.mfusg.MfUsq(model_ws=gridgen_ws, modelname="mfusg")
      disu = g.get_disu(mu)
      disu.write_file()
      # print(disu)
```

Intersect Features with the Grid

```
[15]: adpoly_intersect = g.intersect(adpoly, "polygon", 0)
      print(adpoly_intersect.dtype.names)
      print(adpoly_intersect)
      print(adpoly_intersect.nodenumbers)

('nodenumbers', 'polyid', 'totalarea', 'SHAPEID')
[( 322, 0, 0.961169 , 0) ( 382, 0, 0.961169 , 0) ( 325, 0, 0.961169 , 0)
 ... (5946, 0, 0.0977309, 0) (5948, 0, 0.348072 , 0)
 (5949, 0, 0.0428516, 0)]
[ 322  382  325 ... 5946 5948 5949]
```

```
[16]: well_intersect = g.intersect(wells, "point", 0)
      print(well_intersect.dtype.names)
      print(well_intersect)
      print(well_intersect.nodenumbers)

('nodenumbers', 'pointid', 'SHAPEID')
[(2403, 1, 1) (1464, 4, 4) (2730, 5, 5) (2261, 6, 6) (2918, 9, 9)]
[2403 1464 2730 2261 2918]
```

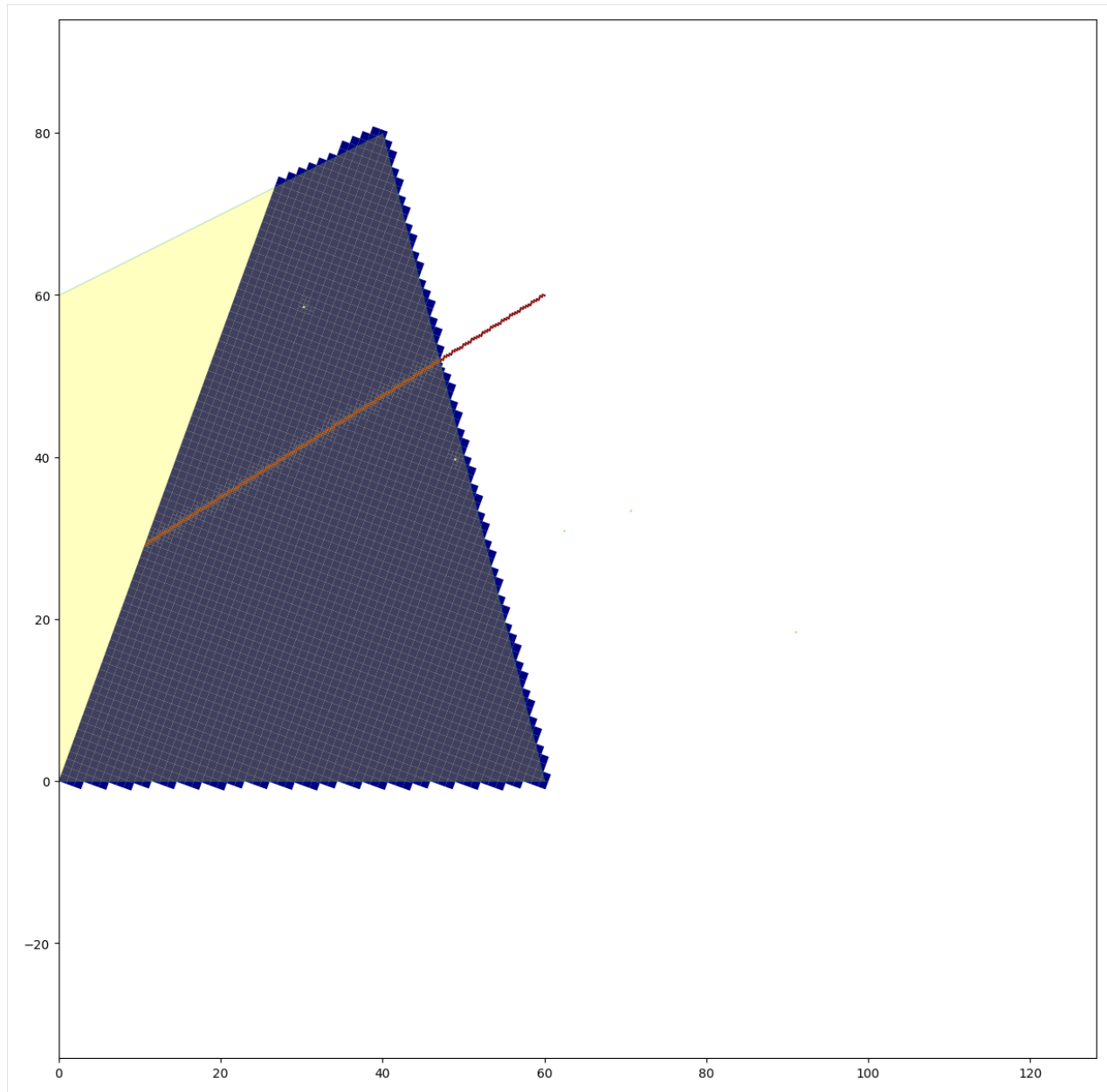
```
[17]: river_intersect = g.intersect(river, "line", 0)
      print(river_intersect.dtype.names)
      # print(river_intersect)
      # print(river_intersect.nodenumbers)

('nodenumbers', 'arcid', 'length', 'starting_distance', 'ending_distance', 'SHAPEID')
```

Plot Intersected Features

```
[18]: a = np.zeros((g.nodes), dtype=int)
      a[adpoly_intersect.nodenumbers] = 1
      a[well_intersect.nodenumbers] = 2
      a[river_intersect.nodenumbers] = 3
      fig = plt.figure(figsize=(15, 15))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      g.plot(ax, a=a, masked_values=[0], edgecolor="none", cmap="jet")
      flopy.plot.plot_shapefile(rf2shp, ax=ax, facecolor="yellow", alpha=0.25)
```

```
[18]: <matplotlib.collections.PatchCollection at 0x7f43bb3fd3a0>
```



Use Gridgen to Build MODFLOW 6 DISV Model

In this section, we will reproduce the MODFLOW 6 Quick Start example that is shown on the main page of the flopy repository (<https://github.com/modflowpy/flopy>).

A main idea for DISV in MODFLOW 6 is that each layer must have the same spatial grid. Gridgen allows the creation of a grid that has a different number of cells within each layer. This type of grid cannot be used with the DISV Package. To make sure that the resulting grid is the same for each layer, refinement should be added to all layers when using the flopy Gridgen wrapper.

```
[19]: from shapely.geometry import Polygon
```

(continues on next page)

(continued from previous page)

```

name = "dummy"
nlay = 3
nrow = 10
ncol = 10
delr = delc = 1.0
top = 1
bot = 0
dz = (top - bot) / nlay
botm = [top - k * dz for k in range(1, nlay + 1)]

# Create a dummy model and regular grid to use as the base grid for gridgen
sim = flopy.mf6.MFSimulation(sim_name=name, sim_ws=gridgen_ws, exe_name="mf6")
gwf = flopy.mf6.ModflowGwf(sim, modelname=name)

dis = flopy.mf6.ModflowGwfdis(
    gwf,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=delr,
    delc=delc,
    top=top,
    botm=botm,
)

# Create and build the gridgen model with a refined area in the middle
g = Gridgen(gwf.modelgrid, model_ws=gridgen_ws)
polys = [Polygon([(4, 4), (6, 4), (6, 6), (4, 6)])]
g.add_refinement_features(polys, "polygon", 3, range(nlay))
g.build()

```

```

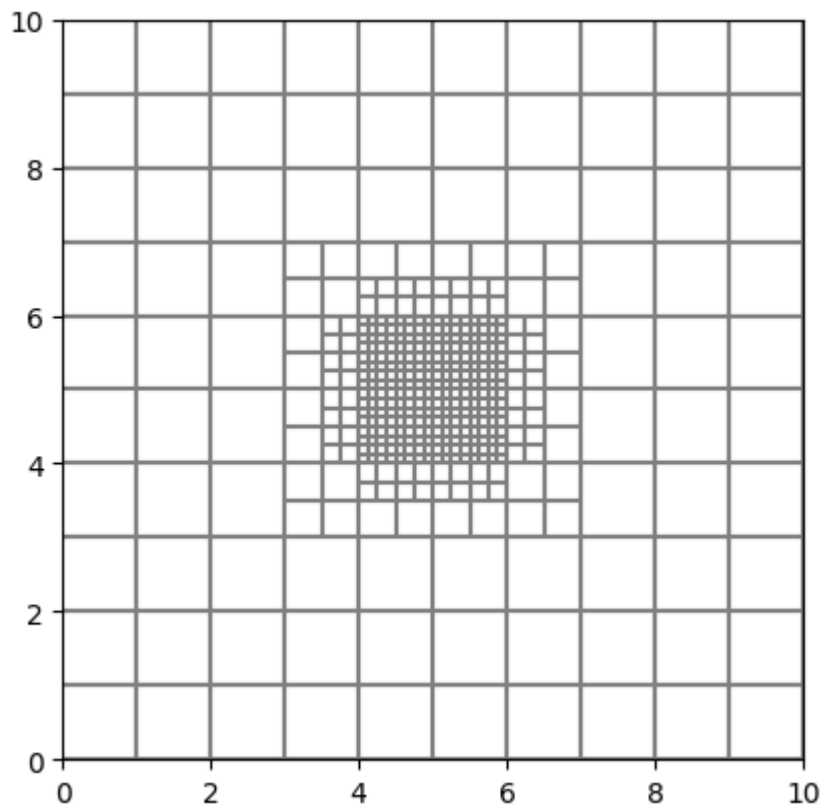
[20]: # Create and plot a flopy VertexGrid object
      # Note that this is not necessary, because it can
      # be created automatically after the disv package
      # is added to the gwf model (gwf.modelgrid should exist)
      gridprops_vg = g.get_gridprops_vertexgrid()
      vgrid = flopy.discretization.VertexGrid(**gridprops_vg)
      vgrid.plot()

```

```

[20]: <matplotlib.collections.LineCollection at 0x7f43bb33a160>

```



```
[21]: # retrieve a dictionary of arguments to be passed
      # directly into the flopy disv constructor
      disv_gridprops = g.get_gridprops_disv()
      disv_gridprops.keys()
```

```
[21]: dict_keys(['nlay', 'ncpl', 'top', 'botm', 'nvert', 'vertices', 'cell2d'])
```

```
[22]: # find the cell numbers for constant heads
      chdspd = []
      ilay = 0
      for x, y, head in [(0, 10, 1.0), (10, 0, 0.0)]:
          ra = g.intersect([(x, y)], "point", ilay)
          ic = ra["nodenumber"][0]
          chdspd.append([(ilay, ic), head])
      chdspd
```

```
[22]: [(0, 0), 1.0], [(0, 435), 0.0]
```

```
[23]: # build uun and post-process the MODFLOW 6 model
      ws = os.path.join(model_ws, "gridgen_disv")
      name = "mymodel"
      sim = flopy.mf6.MFSimulation(
          sim_name=name, sim_ws=ws, exe_name="mf6", verbosity_level=0
      )
      tdis = flopy.mf6.ModflowTdis(sim)
      ims = flopy.mf6.ModflowIms(sim, linear_acceleration="bicgstab")
```

(continues on next page)

(continued from previous page)

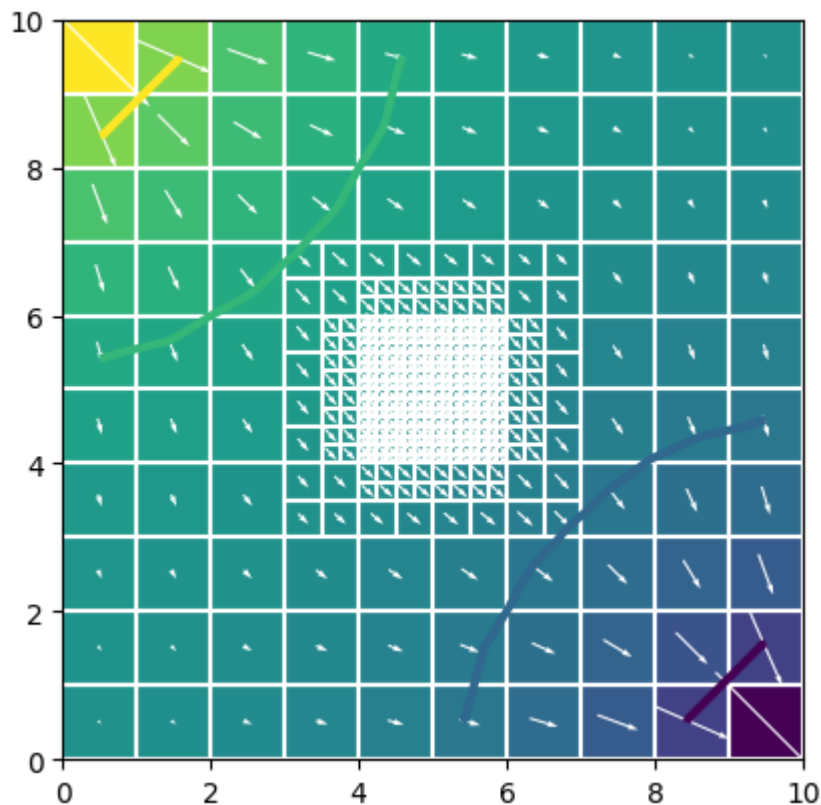
```

gwf = flopy.mf6.ModflowGwf(sim, modelname=name, save_flows=True)
disv = flopy.mf6.ModflowGwfdsv(gwf, **disv_gridprops)
ic = flopy.mf6.ModflowGwfic(gwf)
npf = flopy.mf6.ModflowGwfnpf(
    gwf, xt3doptions=True, save_specific_discharge=True
)
chd = flopy.mf6.ModflowGwfchd(gwf, stress_period_data=chdspd)
budget_file = name + ".bud"
head_file = name + ".hds"
oc = flopy.mf6.ModflowGwfoc(
    gwf,
    budget_filerecord=budget_file,
    head_filerecord=head_file,
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
)
sim.write_simulation()
sim.run_simulation(silent=True)
head = gwf.output.head().get_data()
bud = gwf.output.budget()
spdis = bud.get_data(text="DATA-SPDIS")[0]

pmv = flopy.plot.PlotMapView(gwf)
pmv.plot_array(head)
pmv.plot_grid(colors="white")
pmv.contour_array(head, levels=[0.2, 0.4, 0.6, 0.8], linewidths=3.0)
pmv.plot_vector(spdis["qx"], spdis["qy"], color="white")

```

[23]: <matplotlib.quiver.Quiver at 0x7f43ba910fd0>



Use Gridgen to Build MODFLOW 6 DISU Model

In this section, we will reproduce the MODFLOW 6 Quick Start example that is shown on the main page of the flopy repository (<https://github.com/modflowpy/flopy>).

DISU is the most general grid form that can be used with MODFLOW 6. It does not have the requirement that each layer must use the same grid

```
[24]: from shapely.geometry import Polygon

name = "dummy"
nlay = 3
nrow = 10
ncol = 10
delr = delc = 1.0
top = 1
bot = 0
dz = (top - bot) / nlay
botm = [top - k * dz for k in range(1, nlay + 1)]

# Create a dummy model and regular grid to use as the base grid for gridgen
sim = flopy.mf6.MFSimulation(sim_name=name, sim_ws=gridgen_ws, exe_name="mf6")
gwf = flopy.mf6.ModflowGwf(sim, modelname=name)

dis = flopy.mf6.ModflowGwfdiis(
```

(continues on next page)

(continued from previous page)

```

    gwf,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=delr,
    delc=delc,
    top=top,
    botm=botm,
)

# Create and build the gridgen model with a refined area in the middle
g = Gridgen(gwf.modelgrid, model_ws=gridgen_ws)
polys = [Polygon([(4, 4), (6, 4), (6, 6), (4, 6)])]
g.add_refinement_features(polys, "polygon", 3, layers=[0])
g.build()

```

[25]: *# retrieve a dictionary of arguments to be passed
directly into the flopy disu constructor*

```

disu_gridprops = g.get_gridprops_disu6()
disu_gridprops.keys()

```

[25]: dict_keys(['nodes', 'top', 'bot', 'area', 'iac', 'nja', 'ja', 'cl12', 'ihc', 'hwva',
→ 'angldegx', 'nvert', 'vertices', 'cell2d'])

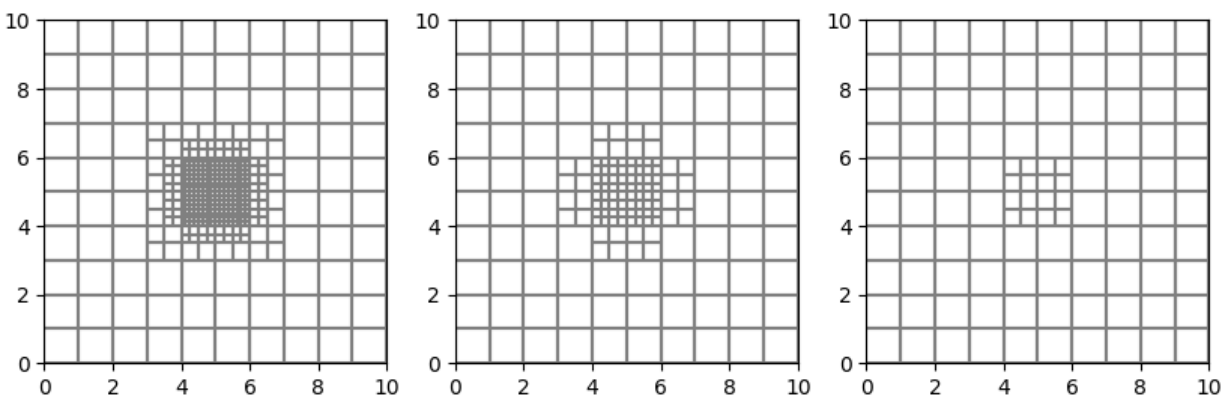
[26]: *# Create and plot a flopy UnstructuredGrid object*

```

gridprops_ug = g.get_gridprops_unstructuredgrid()
ugrid = flopy.discretization.UnstructuredGrid(**gridprops_ug)

f = plt.figure(figsize=(10, 10))
for ilay in range(g.nlay):
    ax = plt.subplot(1, g.nlay, ilay + 1)
    ugrid.plot(layer=ilay, ax=ax)

```



[27]: *# find the cell numbers for constant heads*

```

chdspd = []
for x, y, head in [(0, 10, 1.0), (10, 0, 0.0)]:
    ra = g.intersect([(x, y)], "point", 0)
    ic = ra["nodenumber"][0]

```

(continues on next page)

(continued from previous page)

```
chdspd.append([(ic,), head])
chdspd
```

```
[27]: [[(0,), 1.0], [(435,), 0.0]]
```

```
[28]: # build run and post-process the MODFLOW 6 model
ws = os.path.join(model_ws, "gridgen_disu")
name = "mymodel"
sim = flopy.mf6.MFSimulation(
    sim_name=name, sim_ws=ws, exe_name="mf6", verbosity_level=1
)
tdis = flopy.mf6.ModflowTdis(sim)
ims = flopy.mf6.ModflowIms(sim, linear_acceleration="bicgstab")
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, save_flows=True)
disu = flopy.mf6.ModflowGwfdisu(gwf, **disu_gridprops)
ic = flopy.mf6.ModflowGwfic(gwf)
npf = flopy.mf6.ModflowGwfnpf(
    gwf, xt3doptions=True, save_specific_discharge=True
)
chd = flopy.mf6.ModflowGwfchd(gwf, stress_period_data=chdspd)
budget_file = name + ".bud"
head_file = name + ".hds"
oc = flopy.mf6.ModflowGwfoc(
    gwf,
    budget_filerecord=budget_file,
    head_filerecord=head_file,
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
)
sim.write_simulation()
sim.run_simulation(silent=True)
head = gwf.output.head().get_data()
bud = gwf.output.budget()
spdis = bud.get_data(text="DATA-SPDIS")[0]

gwf.modelgrid.set_coord_info(angrot=15)

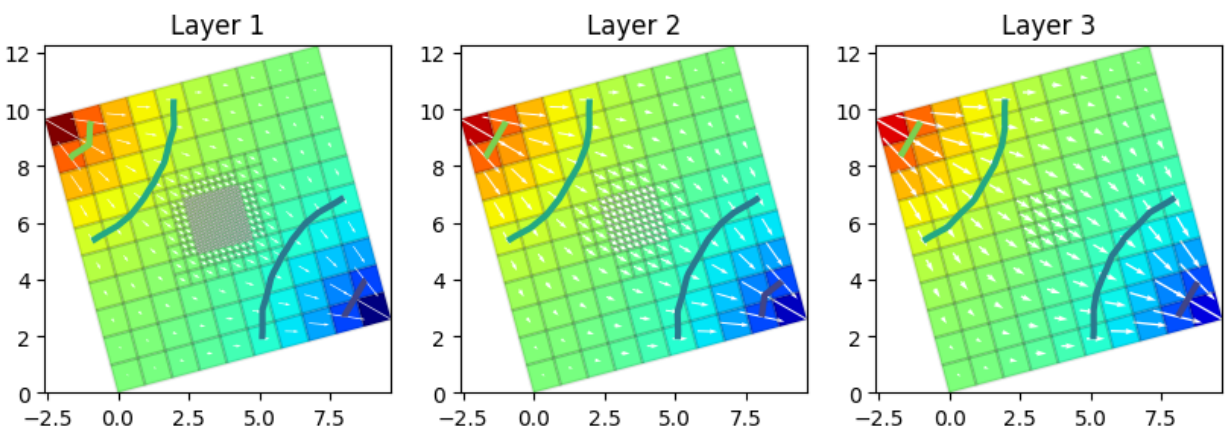
f = plt.figure(figsize=(10, 10))
vmin = head.min()
vmax = head.max()
for ilay in range(gwf.modelgrid.nlay):
    ax = plt.subplot(1, g.nlay, ilay + 1)
    pmv = flopy.plot.PlotMapView(gwf, layer=ilay, ax=ax)
    ax.set_aspect("equal")
    pmv.plot_array(head.flatten(), cmap="jet", vmin=vmin, vmax=vmax)
    pmv.plot_grid(colors="k", alpha=0.1)
    pmv.contour_array(
        head, levels=[0.2, 0.4, 0.6, 0.8], linewidths=3.0, vmin=vmin, vmax=vmax
    )
    ax.set_title("Layer {}".format(ilay + 1))
    pmv.plot_vector(spdis["qx"], spdis["qy"], color="white")
```

```
WARNING: Unable to resolve dimension of ('gwf6', 'disu', 'cell2d', 'cell2d', 'icvert')_
↳ based on shape "ncvert".
```

(continues on next page)

(continued from previous page)

```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims_-1...
  writing model mymodel...
    writing model name file...
    writing package disu...
    writing package ic...
    writing package npf...
    writing package chd_0...
INFORMATION: maxbound in ('gwf6', 'chd', 'dimensions') changed to 2 based on size of
↪ stress_period_data
  writing package oc...
```



Use Gridgen to Build MODFLOW-USG DISU Model

In this section, we will reproduce the MODFLOW 6 Quick Start example that is shown on the main page of the flopy repository (<https://github.com/modflowpy/flopy>).

In this last example, MODFLOW-USG will be used to simulate the problem.

```
[29]: # build run and post-process the MODFLOW 6 model
ws = os.path.join(model_ws, "gridgen_mfusg")
name = "mymodel"

chdspd = []
for x, y, head in [(0, 10, 1.0), (10, 0, 0.0)]:
    ra = g.intersect([(x, y)], "point", 0)
    ic = ra["nodenumbr"][0]
    chdspd.append([ic, head, head])

gridprops = g.get_gridprops_disu5()

# create the mfusg model
m = flopy.mfusg.MfUsG(
    modelname=name,
    model_ws=ws,
```

(continues on next page)

(continued from previous page)

```

version="mfusg",
exe_name="mfusg",
structured=False,
)
disu = flopy.mfusg.MfUsgDisU(m, **gridprops)
bas = flopy.modflow.ModflowBas(m)
lpf = flopy.mfusg.MfUsgLpf(m)
chd = flopy.modflow.ModflowChd(m, stress_period_data=chdspd)
sms = flopy.mfusg.MfUsgSms(m)
oc = flopy.modflow.ModflowOc(m)
m.write_input()
success, buff = m.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

# head is returned as a list of head arrays for each layer
head_file = os.path.join(ws, name + ".hds")
head = flopy.utils.HeadUFile(head_file).get_data()

# MODFLOW-USG does not have vertices, so we need to create
# and unstructured grid and then assign it to the model. This
# will allow plotting and other features to work properly.
gridprops_ug = g.get_gridprops_unstructuredgrid()
ugrid = flopy.discretization.UnstructuredGrid(**gridprops_ug)
m.modelgrid = ugrid

f = plt.figure(figsize=(10, 10))
vmin = 0.0
vmax = 1.0
for ilay in range(disu.nlay):
    ax = plt.subplot(1, g.nlay, ilay + 1)
    pmv = flopy.plot.PlotMapView(m, layer=ilay, ax=ax)
    ax.set_aspect("equal")
    pmv.plot_array(head[ilay], cmap="jet", vmin=vmin, vmax=vmax)
    pmv.plot_grid(colors="k", alpha=0.1)
    pmv.contour_array(head[ilay], levels=[0.2, 0.4, 0.6, 0.8], linewidths=3.0)
    ax.set_title("Layer {}".format(ilay + 1))

```

MODFLOW-USG
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER FLOW MODEL
Version 1.5.00 02/27/2019

Using NAME file: mymodel.nam

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:42:24

Solving: Stress period: 1 Time step: 1 Groundwater Flow Eqn.

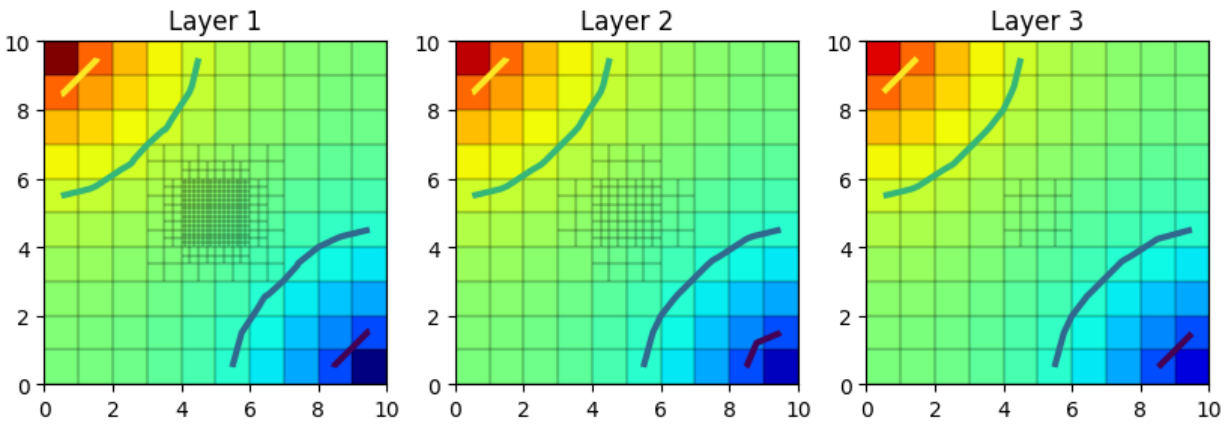
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:42:24

Elapsed run time: 0.005 Seconds

(continues on next page)

(continued from previous page)

Normal termination of simulation



```
[30]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.1.5 ModelGrid classes demo

The three modelgrid classes `StructuredGrid`, `VertexGrid`, and `UnstructuredGrid` will be demonstrated in this notebook.

All three classes behave similarly, as they inherit their base functionality from the same “parent” object. Even though they behave similarly, there are also some differences between the three classes based upon the specific grid type.

This notebook will cover:

- 1) How to access the modelgrid object from a model and common usages for the modelgrid
- 2) How to build modelgrid objects from scratch
- 3) Useful methods and features

```
[1]: import os
      import shutil
```

```
[2]: import sys
      from tempfile import TemporaryDirectory

      import matplotlib as mpl
      import matplotlib.pyplot as plt
      import numpy as np

      # run installed version of flopy or add local path
```

(continues on next page)

(continued from previous page)

```
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

import flopy.utils.binaryfile as bf
from flopy.discretization import StructuredGrid, UnstructuredGrid, VertexGrid

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[3]: # set the names of our modflow executables
      # assumes that the executable is in users path statement
      mf6_exe = "mf6"
      gridgen_exe = "gridgen"
```

```
[4]: # set paths to each of our model types for this example notebook
      spth = os.path.join(
          "..", "..", "examples", "data", "freyberg_multilayer_transient"
      )
      spth6 = os.path.join("..", "..", "examples", "data", "mf6-freyberg")
      vpth = os.path.join("..", "..", "examples", "data")
      upth = os.path.join("..", "..", "examples", "data")
      u_data_ws = os.path.join("..", "..", "examples", "data", "unstructured")

      # temporary workspace
      temp_dir = TemporaryDirectory()
      gridgen_ws = temp_dir.name
```

Accessing the modelgrid and common usage

How to access the modelgrid object from a model

FloPy model objects have a built in method that dynamically assembles a modelgrid from model discretization information. Therefore, if the user updates their DIS file, when they call the modelgrid property the new discretization information will be included within it.

Modflow-2005 example

```
[5]: # Load a modflow-2005 model
ml = flopy.modflow.Modflow.load("freyberg.nam", model_ws=spth)

# access the modelgrid object
modelgrid = ml.modelgrid

print(type(modelgrid))

<class 'flopy.discretization.structuredgrid.StructuredGrid'>
```

Spatial reference information that is stored in the NAM file, such as:

xll : geographic location of lower left model corner x-coordinate yll : geographic location of lower left model corner
y-coordinate rotation : modelgrid rotation in degrees epsg : epsg code of modelgrid coordinate system proj4_str
: proj4 projection information

can be automatically read in and applied to the modelgrid. FloPy will also write this information out when the user saves their model to file.

This information can be seen by printing the modelgrid

```
[6]: print(modelgrid)

xll:622241.1904510253; yll:3343617.741737109; rotation:15.0; crs:EPSG:32614; units:
meters; lenuni:2
```

Modflow-6 example

Modflow-6 models also have modelgrid objects attached to them. These grids function in the way. Here is an example.

```
[7]: # load a modflow-6 simulation
sim = flopy.mf6.MFSimulation.load(sim_ws=spth6, verbosity_level=0)

# get a model object from the simulation
ml = sim.get_model("freyberg")

# access the modelgrid
modelgrid1 = ml.modelgrid

print(type(modelgrid1))
print(modelgrid1)

<class 'flopy.discretization.structuredgrid.StructuredGrid'>
xll:0.0; yll:0.0; rotation:0.0; units:meters; lenuni:2
```

Note how there is no spatial reference information associated with this modelgrid. This is because, there it was not specified in the name file. In the following section, the notebook will show how to access this information from a modelgrid and how to set this information in a modelgrid object.

Accessing and setting modelgrid reference information

Accessing modelgrid reference information

There are properties attached to the modelgrid that allows the user to access reference information:

- `xoffset` : returns the x-coordinate for the modelgrid's lower left corner
- `yoffset` : returns the y-coordinate for the modelgrid's lower left corner
- `angrot` : returns the rotation of the modelgrid in degrees
- `epsg` : returns the modelgrid epsg code
- `proj4` : returns the modelgrid proj4_str information

```
[8]: xoff = modelgrid.xoffset
yoff = modelgrid.yoffset
angrot = modelgrid.angrot
epsg = modelgrid.epsg
proj4 = modelgrid.proj4

print(
    "xoff: {} \nyoff: {} \nangrot: {} \nepsg: {} \nproj4: {}".format(
        xoff, yoff, angrout, epsg, proj4
    )
)
```

```
xoff: 622241.1904510253
yoff: 3343617.741737109
angrot: 15.0
epsg: 32614
proj4: +proj=utm +zone=14 +datum=WGS84 +units=m +no_defs +type=crs
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/pyproj/crs/crs.py:
↳1286: UserWarning: You will likely lose important projection information when
↳converting to a PROJ string from another format. See: https://proj.org/faq.html#what-
↳is-the-best-format-for-describing-coordinate-reference-systems
proj = self._crs.to_proj4(version=version)
```

Setting modelgrid reference information

The `set_coord_info()` method allows the user to set some or all of the modelgrid's coordinate reference information. Here is an example using the `modflow6` modelgrid:

```
[9]: # show the coordinate info before setting it
print("Before: {} \n".format(modelgrid1))

# set reference information
modelgrid1.set_coord_info(
```

(continues on next page)

(continued from previous page)

```

    xoff=xoff, yoff=yoff, angrot=angrot, epsg=epsg, proj4=proj4
)

print("After: {}".format(modelgrid1))

Before: xll:0.0; yll:0.0; rotation:0.0; units:meters; lenuni:2

After: xll:622241.1904510253; yll:3343617.741737109; rotation:15.0; crs:EPSG:32614;
↪units:meters; lenuni:2

/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/utils/crs.py:
↪119: PendingDeprecationWarning: the epsg argument will be deprecated and will be
↪removed in version 3.4. Use crs instead.
warnings.warn(

```

The user can also set individual parts of the coordinate information if they do not want to supply all of the fields

```

[10]: # change the offsets and rotation of the modelgrid
      angrot = 55
      xll = 100
      yll = 1000
      modelgrid1.set_coord_info(xoff=xll, yoff=yll, angrot=angrot)

      print(modelgrid1)

      xll:100; yll:1000; rotation:55; crs:EPSG:32614; units:meters; lenuni:2

```

Using the modelgrid with plotting routines

FloPy's plotting routines accept a model or modelgrid object to determine cell locations for plotting. Here is an example of plotting with the modelgrid object

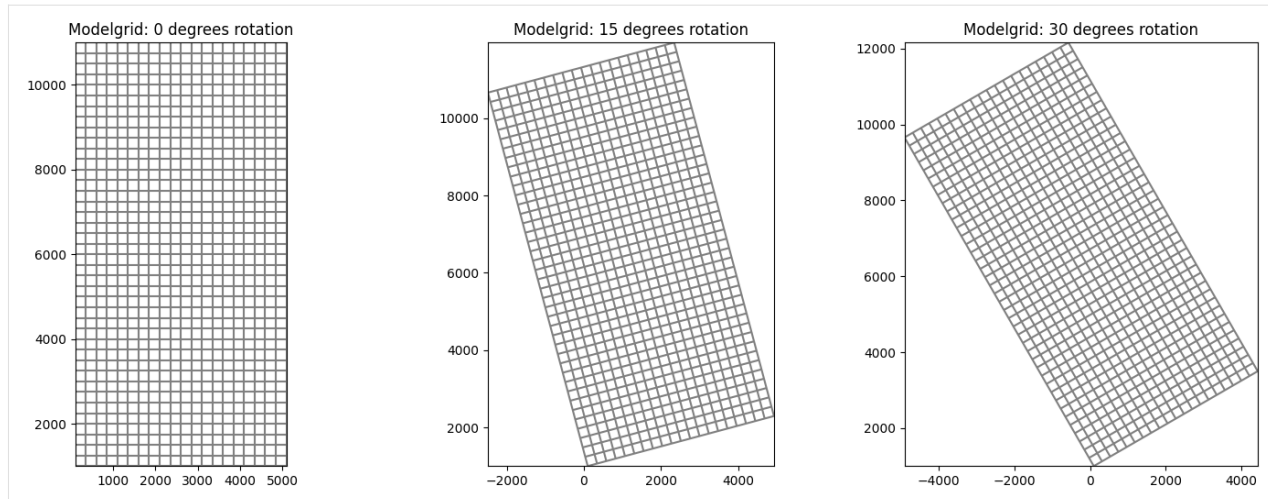
```

[11]: fig, axs = plt.subplots(
      nrows=1, ncols=3, figsize=(18, 6), subplot_kw={"aspect": "equal"}
)

rotation = [0, 15, 30]

for ix, ax in enumerate(axs):
    modelgrid1.set_coord_info(angrot=rotation[ix])
    pmv = flopy.plot.PlotMapView(modelgrid=modelgrid1, ax=ax)
    pmv.plot_grid()
    ax.set_title("Modelgrid: {} degrees rotation".format(rotation[ix]))

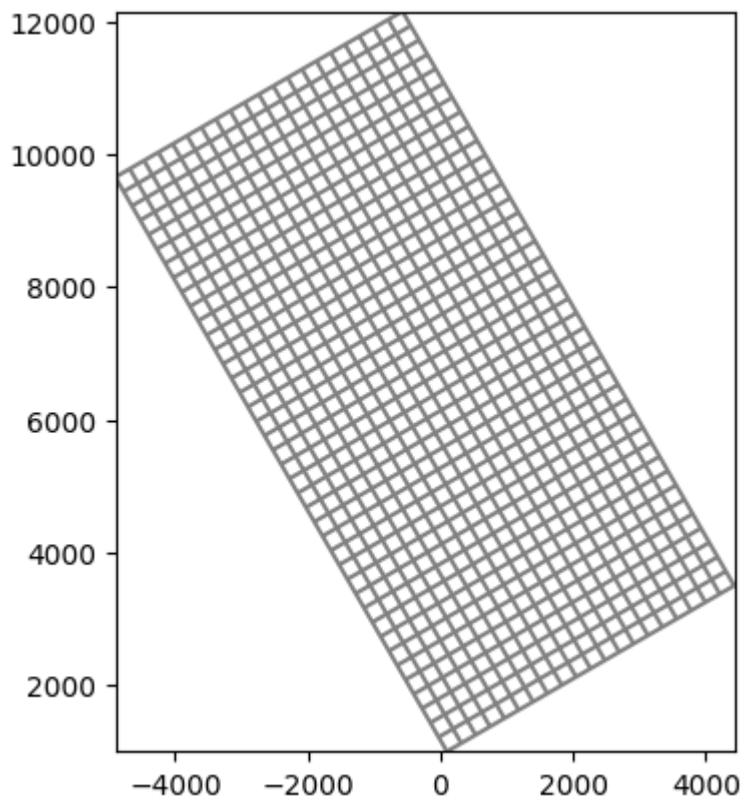
```



The grid lines can also be plotted directly from the modelgrid object

```
[12]: modelgrid1.plot()
```

```
[12]: <matplotlib.collections.LineCollection at 0x7fe0b6adcb20>
```



Building modelgrid objects from scratch

StructuredGrid example

The StructuredGrid class can accept a number of parameters, however for a minimal working StructuredGrid the user only needs to provide:

- `delc` : Array of cell widths along columns
- `delr` : Array of cell widths along rows

Other optional, but useful parameters the user can supply include: - `top` : Array of model Top elevations - `botm` : Array of layer Botm elevations - `idomain` : An ibound or idomain array that specifies active and inactive cells - `lenuni` : Model length unit integer - `epsg` : epsg code of model coordinate system - `proj4` : proj4 str describing model coordinate system - `prj` : path to “.prj” projection file that describes the model coordinate system - `xoff` : x-coordinate of the lower-left corner of the modelgrid - `yoff` : y-coordinate of the lower-left corner of the modelgrid - `angrot` : model grid rotation - `nlay` : number of model layers - `nrow` : number of model rows - `ncol` : number of model columns - `laycbd` : array of length, nlay indicating if Quasi-3D confining layers exist

In this example, some of the more common parameters are used to create a “StructuredGrid”

```
[13]: xll = 3579
      yll = 10000
      rotation = 0

      nrow = 40
      ncol = 20
      nlay = 1
      Lx = 4000
      Ly = 8000

      # create delr and delc arrays
      delr = np.ones((ncol,)) * (Lx / ncol)
      delc = np.ones((nrow,)) * (Ly / nrow)

      # create a sloped top and bottom
      slope = np.linspace(100, 0, nrow)
      slope.shape = (1, nrow)

      top = np.ones((nrow, ncol)) * slope.T
      botm = np.ones((nlay, nrow, ncol)) * (top - 100)

      # create an ibound array
      ibound = np.ones((nrow, ncol), dtype=int)
      ibound[-1, 0:5] = 0
      ibound[-1, 15:] = 0

      modelgrid = StructuredGrid(
          delr=delr,
          delc=delc,
          top=top,
          botm=botm,
```

(continues on next page)

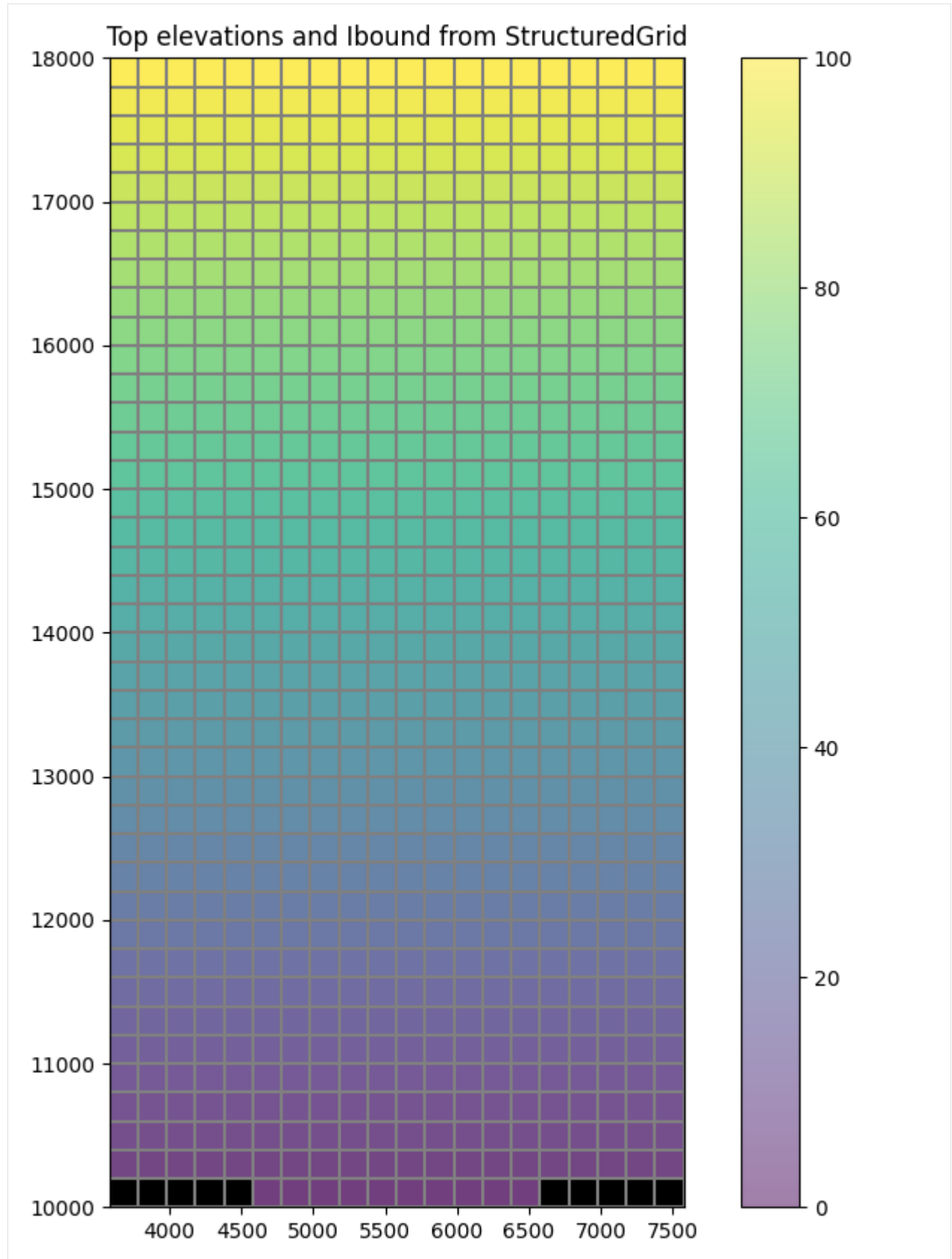
(continued from previous page)

```
    idomain=ibound,  
    nlay=nlay,  
    xoff=xll,  
    yoff=yll,  
    angrot=rotation,  
)  
modelgrid
```

```
[13]: xll:3579; yll:10000; rotation:0; units:undefined; lenuni:0
```

Plot some of the modelgrid information

```
[14]: fig, ax = plt.subplots(figsize=(10, 10))  
  
pmv = flopy.plot.PlotMapView(modelgrid=modelgrid)  
pc = pmv.plot_array(modelgrid.top, alpha=0.5)  
pmv.plot_grid()  
pmv.plot_ibound()  
  
plt.colorbar(pc)  
ax.set_title("Top elevations and Ibound from StructuredGrid")  
  
[14]: Text(0.5, 1.0, 'Top elevations and Ibound from StructuredGrid')
```



Plot a CrossSection of the structured grid

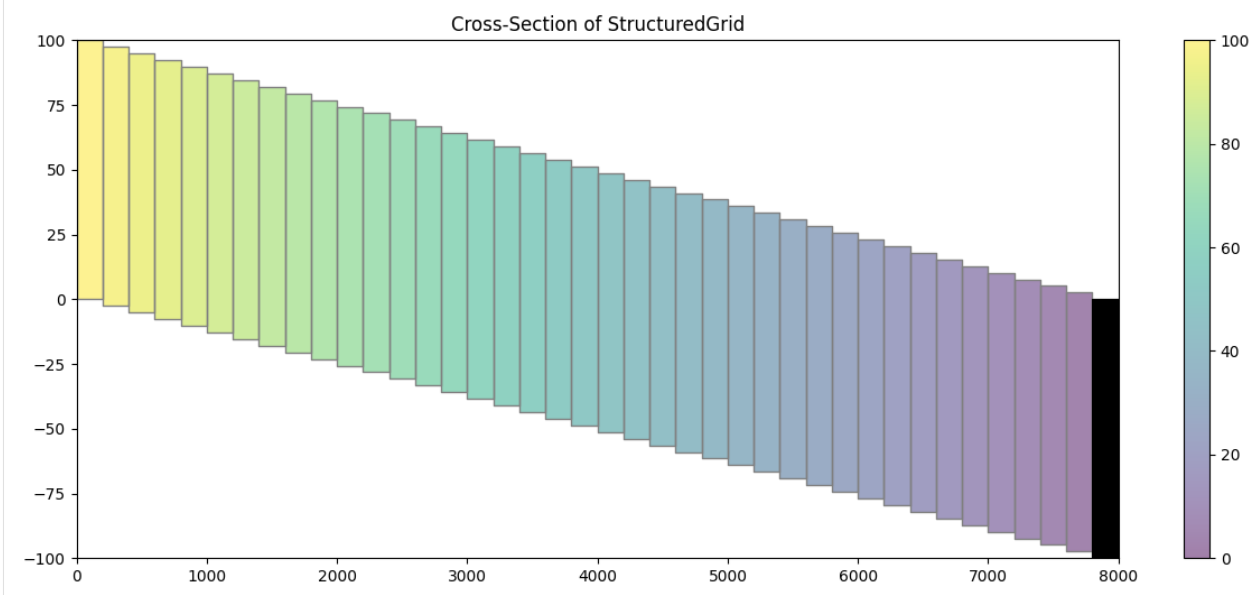
```
[15]: col = 4

figure, ax = plt.subplots(figsize=(15, 6))

xc = flopy.plot.PlotCrossSection(modelgrid=modelgrid, line={"column": col})
pc = xc.plot_array(modelgrid.top, alpha=0.5)
xc.plot_grid()
xc.plot_ibound()

plt.colorbar(pc)
plt.title("Cross-Section of StructuredGrid")
```

```
[15]: Text(0.5, 1.0, 'Cross-Section of StructuredGrid')
```



VertexGrid example

Before building the `VertexGrid` class we must first develop a grid. This example uses the `Gridgen` class and executable to produce the same grid as the MODFLOW 6 Quick Start example that is shown on the main page of the flopy repository (<https://github.com/modflowpy/flopy>).

NOTE: The `Gridgen` class requires that either a path to the executable is provided, that the executable exists in the same directory as the script, or that the executable is in the machine's PATH variables to run properly.

```
[16]: from flopy.utils.geometry import Polygon
```

```
[17]: from flopy.utils.gridgen import Gridgen
```

```
name = "dummy"
nlay = 3
nrow = 10
ncol = 10
```

(continues on next page)

(continued from previous page)

```

delr = delc = 1.0
top = 1
bot = 0
dz = (top - bot) / nlay
botm = [top - k * dz for k in range(1, nlay + 1)]

# Create a dummy model and regular grid to use as the base grid for gridgen
sim = flopy.mf6.MFSimulation(sim_name=name, sim_ws=gridgen_ws, exe_name="mf6")
gwf = flopy.mf6.ModflowGwf(sim, modelname=name)

dis = flopy.mf6.ModflowGwfdis(
    gwf,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=delr,
    delc=delc,
    top=top,
    botm=botm,
)

# Create and build the gridgen model with a refined area in the middle
g = Gridgen(gwf.modelgrid, model_ws=gridgen_ws)
polys = [Polygon([(4, 4), (6, 4), (6, 6), (4, 6), (4, 4)])]
g.add_refinement_features(polys, "polygon", 3, range(nlay))
g.build()

```

```

[18]: # get the vertex grid properties from gridgen and examine them
      gridprops = g.get_gridprops_vertexgrid()
      gridprops.keys()

```

```

[18]: dict_keys(['nlay', 'ncpl', 'top', 'botm', 'vertices', 'cell2d'])

```

Building the VertexGrid

VertexGrid has many similar parameters as the previous StructuredGrid example. For a minimal working model-grid, VertexGrid requires:

- **vertices** : list of vertex number, xvertex, yvertex that make up the grid
- **cell2d** : list containing node number, xcenter, ycenter, and vertex numbers (from vertices)

Other optional, but useful parameters include:

- **top** : Array of model Top elevations
- **botm** : Array of layer Botm elevations
- **idomain** : An ibound or idomain array that specifies active and inactive cells
- **lenuni** : Model length unit integer
- **epsg** : epsg code of model coordinate system
- **proj4** : proj4 str describing model coordinate system

- `prj` : path to “.prj” projection file that describes the model coordinate system
- `xoff` : x-coordinate of the lower-left corner of the modelgrid
- `yoff` : y-coordinate of the lower-left corner of the modelgrid
- `angrot` : model grid rotation
- `nlay` : number of model layers
- `ncpl` : number of cells per model layer

In this example, some of the more common parameters are used to create a “VertexGrid”

```
[19]: vertices = gridprops["vertices"]
      cell2d = gridprops["cell2d"]
      top = gridprops["top"]
      botm = gridprops["botm"]
      nlay = gridprops["nlay"]
      ncpl = gridprops["ncpl"]
      idomain = np.ones((nlay, ncpl), dtype=int)

      modelgrid = VertexGrid(
          vertices=vertices,
          cell2d=cell2d,
          top=top,
          idomain=idomain,
          botm=botm,
          nlay=nlay,
          ncpl=ncpl,
      )
      print(modelgrid)

xll:0.0; yll:0.0; rotation:0.0; units:undefined; lenuni:0
```

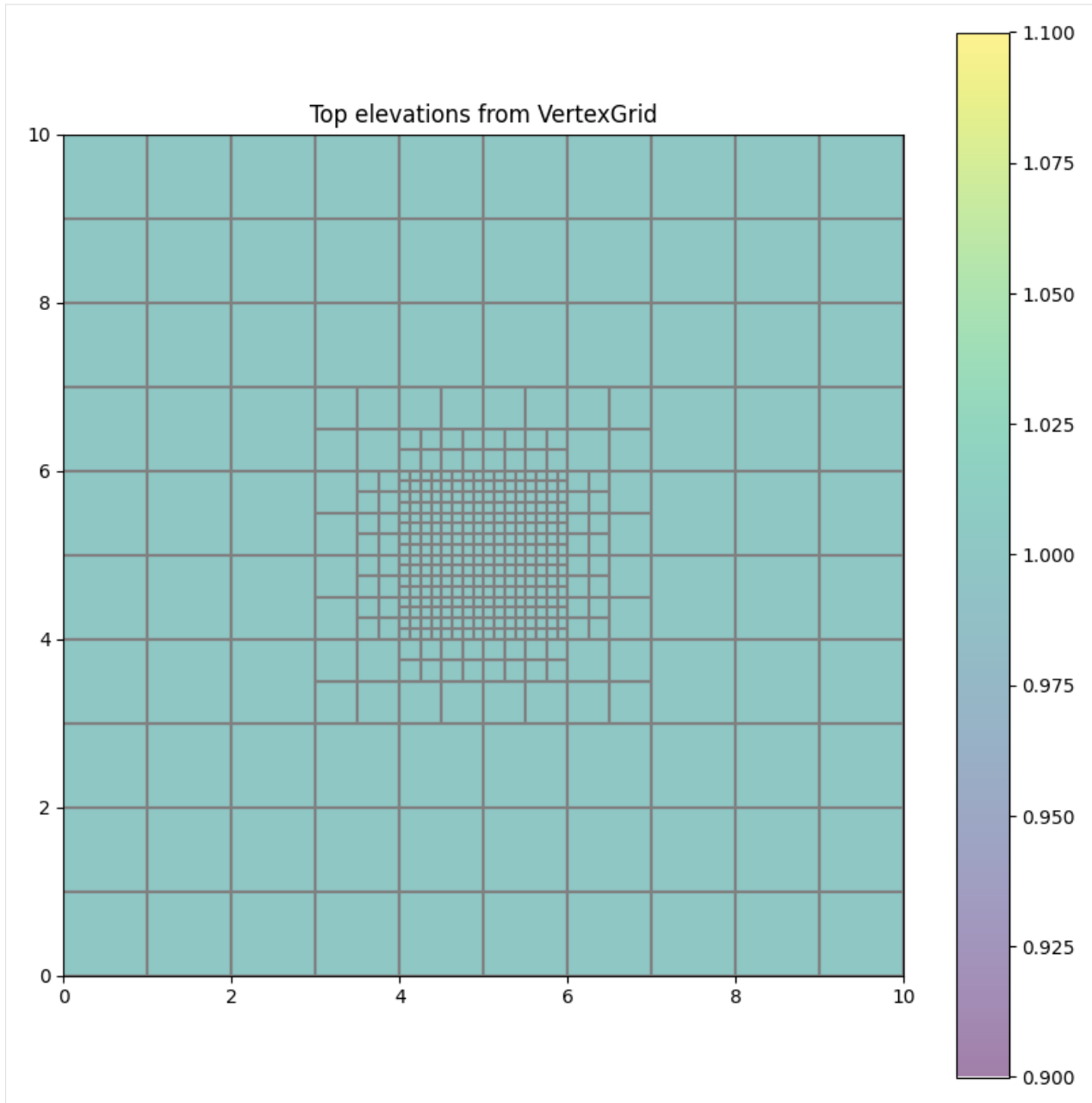
Plot some of the modelgrid information

```
[20]: fig, ax = plt.subplots(figsize=(10, 10))

      pmv = flopy.plot.PlotMapView(modelgrid=modelgrid)
      pc = pmv.plot_array(modelgrid.top, alpha=0.5)
      pmv.plot_grid()
      pmv.plot_ibound()

      plt.colorbar(pc)
      ax.set_title("Top elevations from VertexGrid")

[20]: Text(0.5, 1.0, 'Top elevations from VertexGrid')
```



Plot a CrossSection of the vertex grid

```
[21]: line = [(4.45, 0), (4.45, 10)]

figure, ax = plt.subplots(figsize=(15, 6))

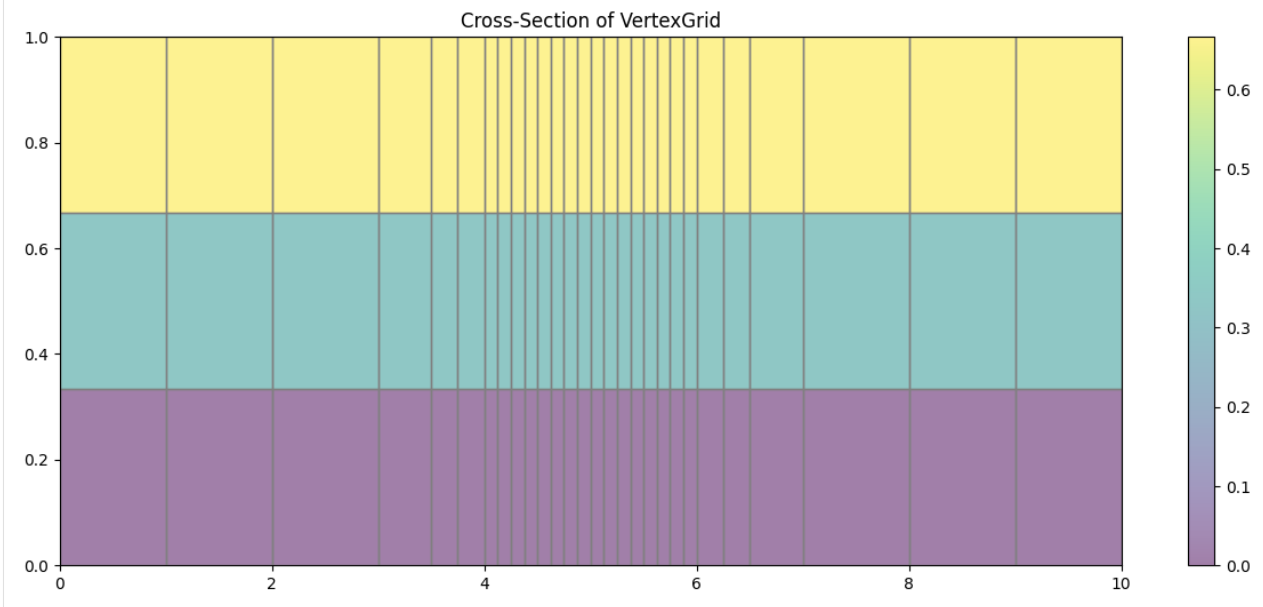
xc = flopy.plot.PlotCrossSection(modelgrid=modelgrid, line={"line": line})
pc = xc.plot_array(modelgrid.botm, alpha=0.5)
xc.plot_grid()
xc.plot_ibound()
```

(continues on next page)

(continued from previous page)

```
plt.colorbar(pc)
plt.title("Cross-Section of VertexGrid")
```

```
[21]: Text(0.5, 1.0, 'Cross-Section of VertexGrid')
```



UnstructuredGrid example

Before building an `UnstructuredGrid` we must first develop a grid. This example loads saved grid information from data files to create an `UnstructuredGrid`.

Note: Gridgen can also be used to develop an unstructured grid and examples of how to do this can be found in the notebook `flopy3_gridgen.ipynb`

```
[22]: # simple functions to load vertices and indice lists
def load_verts(fname):
    verts = np.genfromtxt(
        fname, dtype=[int, float, float], names=["iv", "x", "y"]
    )
    verts["iv"] -= 1 # zero based
    return verts

def load_iverts(fname):
    f = open(fname, "r")
    iverts = []
    xc = []
    yc = []
    for line in f:
        ll = line.strip().split()
        iverts.append([int(i) - 1 for i in ll[4:]])
        xc.append(float(ll[1]))
```

(continues on next page)

(continued from previous page)

```

        yc.append(float(l1[2]))
    return iverts, np.array(xc), np.array(yc)

```

Building the UnstructuredGrid

UnstructuredGrid has many similar parameters as the previous VertexGrid example. For a minimal working, but “incomplete” modelgrid, UnstructuredGrid requires:

- `vertices` : list of vertex number, xvertex, yvertex that make up the grid
- `iverts` : list of vertex numbers that make up each cell
- `xccenters` : list of x center coordinates for all cells in the grid if the grid varies by layer or for all cells in a layer if the same grid is used for all layers
- `yccenters` : list of y center coordinates for all cells in the grid if the grid varies by layer or for all cells in a layer if the same grid is used for all layers

For a “complete” UnstructuredGrid these parameters must be provided in addition to the ones listed above:

- `top` : Array of model Top elevations
- `botm` : Array of layer Botm elevations

Other optional, but useful parameters include:

- `idomain` : An ibound or idomain array that specifies active and inactive cells
- `lenuni` : Model length unit integer
- `ncpl` : one dimensional array of number of cells per model layer
- `epsg` : epsg code of model coordinate system
- `proj4` : proj4 str describing model coordinate system
- `prj` : path to “.prj” projection file that describes the model coordinate system
- `xoff` : x-coordinate of the lower-left corner of the modelgrid
- `yoff` : y-coordinate of the lower-left corner of the modelgrid
- `angrot` : model grid rotation

In this example, some of the more common parameters are used to create a “UnstructuredGrid”

```

[23]: # load vertices
fname = os.path.join(u_data_ws, "ugrid_verts.dat")
verts = load_verts(fname)

# load the index list into iverts, xc, and yc
fname = os.path.join(u_data_ws, "ugrid_iverts.dat")
iverts, xc, yc = load_iverts(fname)

# create a 3 layer model grid
ncpl = np.array(3 * [len(iverts)])
nnodes = np.sum(ncpl)

```

(continues on next page)

(continued from previous page)

```
top = np.ones((nnodes))
botm = np.ones((nnodes))

# set top and botm elevations
i0 = 0
i1 = ncpl[0]
elevs = [100, 0, -100, -200]
for ix, cpl in enumerate(ncpl):
    top[i0:i1] *= elevs[ix]
    botm[i0:i1] *= elevs[ix + 1]
    i0 += cpl
    i1 += cpl

# create the modelgrid
modelgrid = UnstructuredGrid(
    vertices=verts,
    iverts=iverts,
    xcenters=xc,
    ycenters=yc,
    top=top,
    botm=botm,
    ncpl=ncpl,
)
```

Plot some of the modelgrid information

```
[24]: # rotate the modelgrid for the fun of it
modelgrid.set_coord_info(angrot=10)

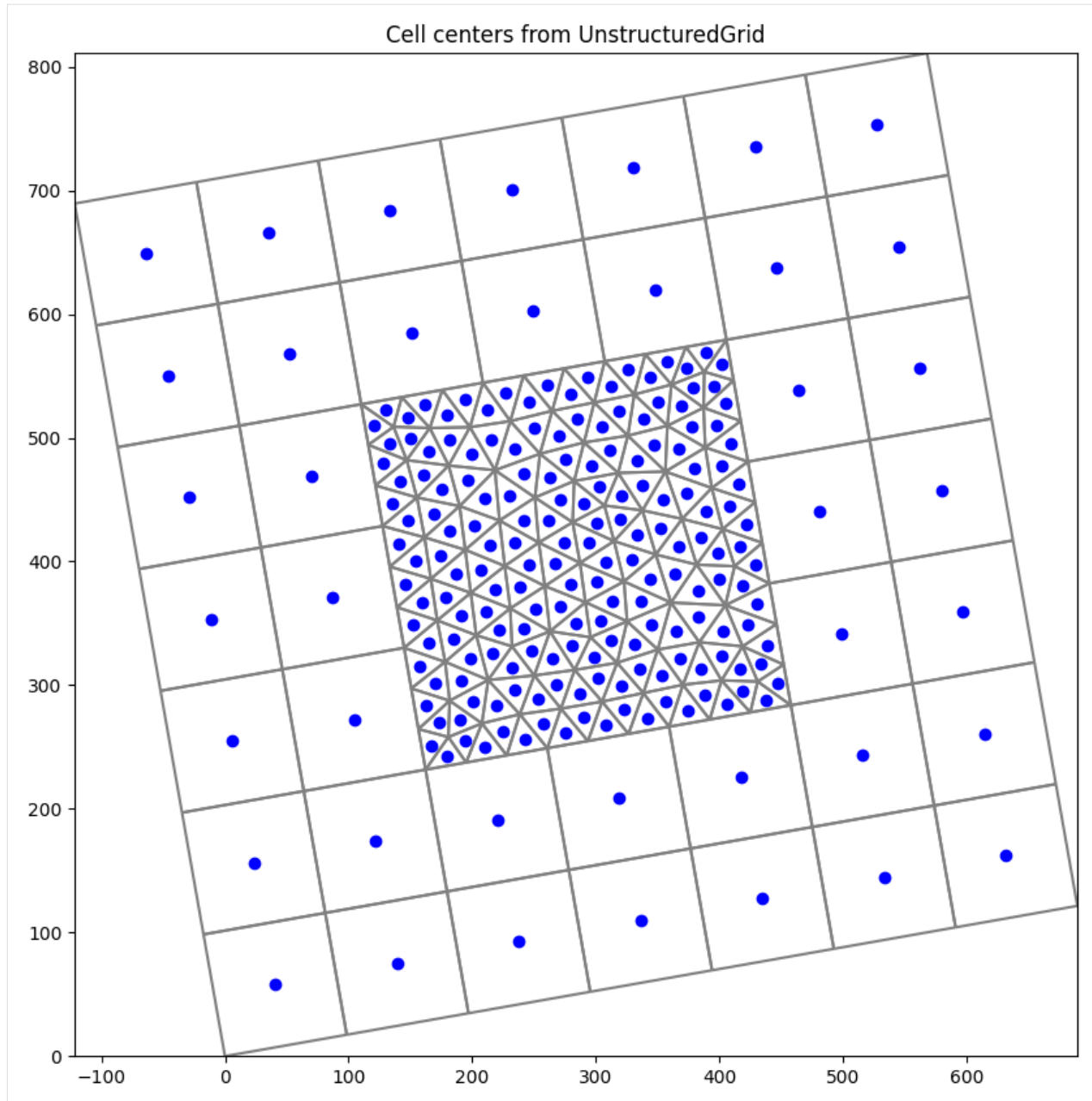
fig, ax = plt.subplots(figsize=(10, 10))

pmv = flopy.plot.PlotMapView(modelgrid=modelgrid)
pmv.plot_grid()

# plot cell centers
plt.plot(modelgrid.xcellcenters, modelgrid.ycellcenters, "bo")

ax.set_title("Cell centers from UnstructuredGrid")

[24]: Text(0.5, 1.0, 'Cell centers from UnstructuredGrid')
```



Plot a CrossSection of the unstructured grid

```
[25]: line = [(350, 0), (300, 800)]

figure, ax = plt.subplots(figsize=(15, 6))

# note geographic_coords=True should be set for unstructured grid cross-sections
xc = flopy.plot.PlotCrossSection(
    modelgrid=modelgrid, line={"line": line}, geographic_coords=True
)
```

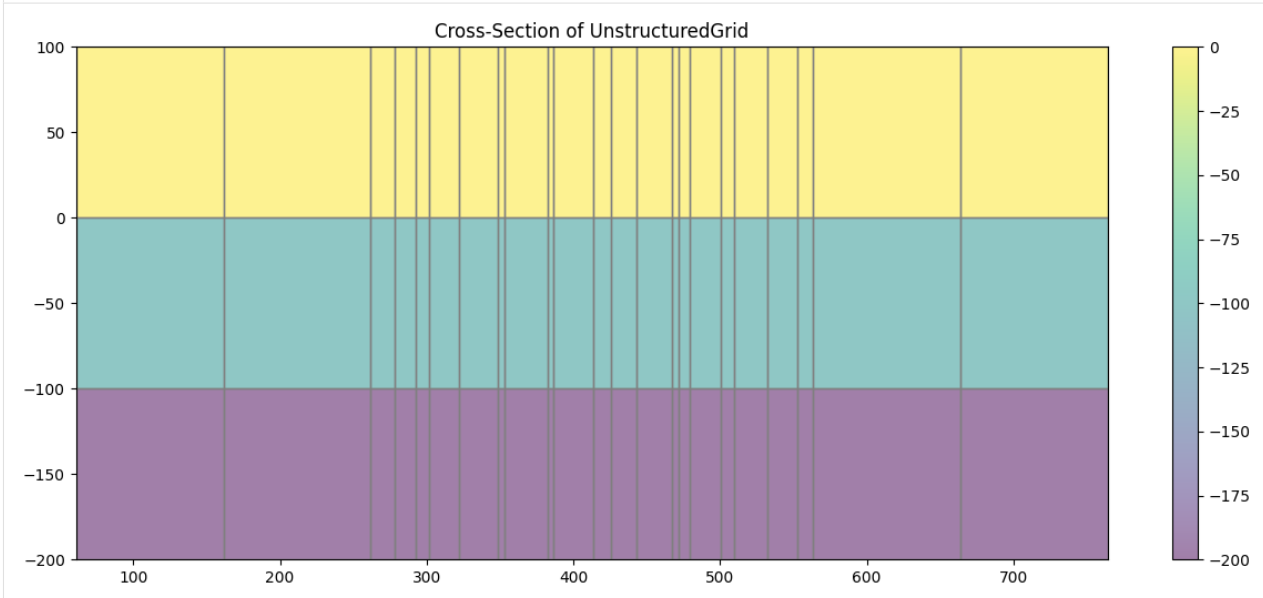
(continues on next page)

(continued from previous page)

```
pc = xc.plot_array(modelgrid.botm, alpha=0.5)
xc.plot_grid()
```

```
plt.colorbar(pc)
plt.title("Cross-Section of UnstructuredGrid")
```

[25]: `Text(0.5, 1.0, 'Cross-Section of UnstructuredGrid')`



Useful methods and properties of the modelgrid classes

Many of the useful methods and features of the modelgrid classes will be presented in this section of the Notebook. The modelgrid classes have a large number of common properties and methods that we will explore. Let's first load a model and look at all of the common features.

```
[26]: # load a modflow-6 freyberg simulation
sim = flopy.mf6.MFSimulation.load(
    sim_ws=spth6, verbosity_level=0, exe_name=mf6_exe
)

# get a model object from the simulation
ml = sim.get_model("freyberg")

# access the modelgrid
modelgrid = ml.modelgrid

# and set the coordinate info
modelgrid.set_coord_info(
    xoff=622241.1904510253,
    yoff=3343617.741737109,
    angrot=15.0,
    proj4="+proj=utm +zone=14 +ellps=WGS84 +datum=WGS84 +units=m +no_defs",
)
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/utils/crs.py:
↳137: PendingDeprecationWarning: the proj4 argument will be deprecated and will be
↳removed in version 3.4. Use crs instead.
warnings.warn(
```

Common properties

There are many common (shared) properties that are present in `StructuredGrid`, `VertexGrid`, and `UnstructuredGrid`:

Grid info properties

- `is_valid` : returns True if modelgrid is valid
- `is_complete` : returns True if modelgrid has all basic information and `top`, `botm`, and `idomain` have been set
- `grid_type` : returns a string representation of the grid type (“structured”, “vertex”, or “unstructured”)
- `units` : returns a string representation of model units
- `lenuni` : returns an integer representation of model units
- `extent` : returns the modelgrid extent as (xmin, xmax, ymin, ymax)

```
[27]: # print grid info properties
if modelgrid.is_valid and modelgrid.is_complete:
    print("{} modelgrid is valid and complete\n".format(modelgrid.grid_type))

print("lenuni: {}, units: {}\n".format(modelgrid.lenuni, modelgrid.units))

print(
    "lower left corner: ({0}, {2}), upper right corner: ({1}, {3})".format(
        *modelgrid.extent
    )
)

structured modelgrid is valid and complete

lenuni: 2, units: meters

lower left corner: (619653.0, 3343617.741737109), upper right corner: (627070.8195824706,
↳ 3354571.0952255125)
```

Spatial reference properties

- `xoffset` : returns the current xoffset of the modelgrid
- `yoffset` : returns the current yoffset of the modelgrid
- `angrot` : returns the angle of rotation of the modelgrid in degrees
- `angrot_radians` : returns the angle of rotation of the modelgrid in radians
- `epsg` : returns the modelgrid epsg code if it is set

- `proj4` : returns the modelgrid proj4 string if it is set
- `prj` : returns the path to the modelgrid projection file if it is set

```
[28]: # Access and print some of these properties
print(
    "xoffset: {}, yoffset: {}\n".format(modelgrid.xoffset, modelgrid.yoffset)
)
print(
    "rotation (deg): {:.1f}, (rad): {:.4f}\n".format(
        modelgrid.angrot, modelgrid.angrot_radians
    )
)
print("proj4_str: {}".format(modelgrid.proj4))

xoffset: 622241.1904510253, yoffset: 3343617.741737109

rotation (deg): 15.0, (rad): 0.2618

proj4_str: +proj=utm +zone=14 +datum=WGS84 +units=m +no_defs +type=crs
```

Model discretization properties

- `shape` : returns the shape of the modelgrid (tuple)
- `ncpl` : returns the number of cells per layer
- `nnodes` : returns the number of cells in the model
- `top` : returns an array of the top elevation of the model or for unstructured grids the top elevation of all cells
- `botm` : returns an array of the botm elevations of model cells
- `cell_thickness` : returns an array of model cell thickness
- `idomain` : returns the idomain array associated with the model
- `xvertices` : returns an array of x-vertices for model cells
- `yvertices` : returns an array of y-vertices for model cells
- `zvertices` : returns an array of z-vertices for model cells
- `xcellcenters` : returns an array of x-vertices for model cell centers
- `ycellcenters` : returns an array of y-vertices for model cell centers
- `verts` : returns a list of vertex number, x-coordinate, y-coordinate
- `iverts` : returns a list of cell number, vertex numbers for each cell

```
[29]: # look at some model discretization information

print("Grid shape: {}\n".format(modelgrid.shape))
print("number of cells per layer: {}\n".format(modelgrid.ncpl))
print("number of cells in model: {}".format(modelgrid.nnodes))

Grid shape: (1, 40, 20)

number of cells per layer: 800
```

(continues on next page)

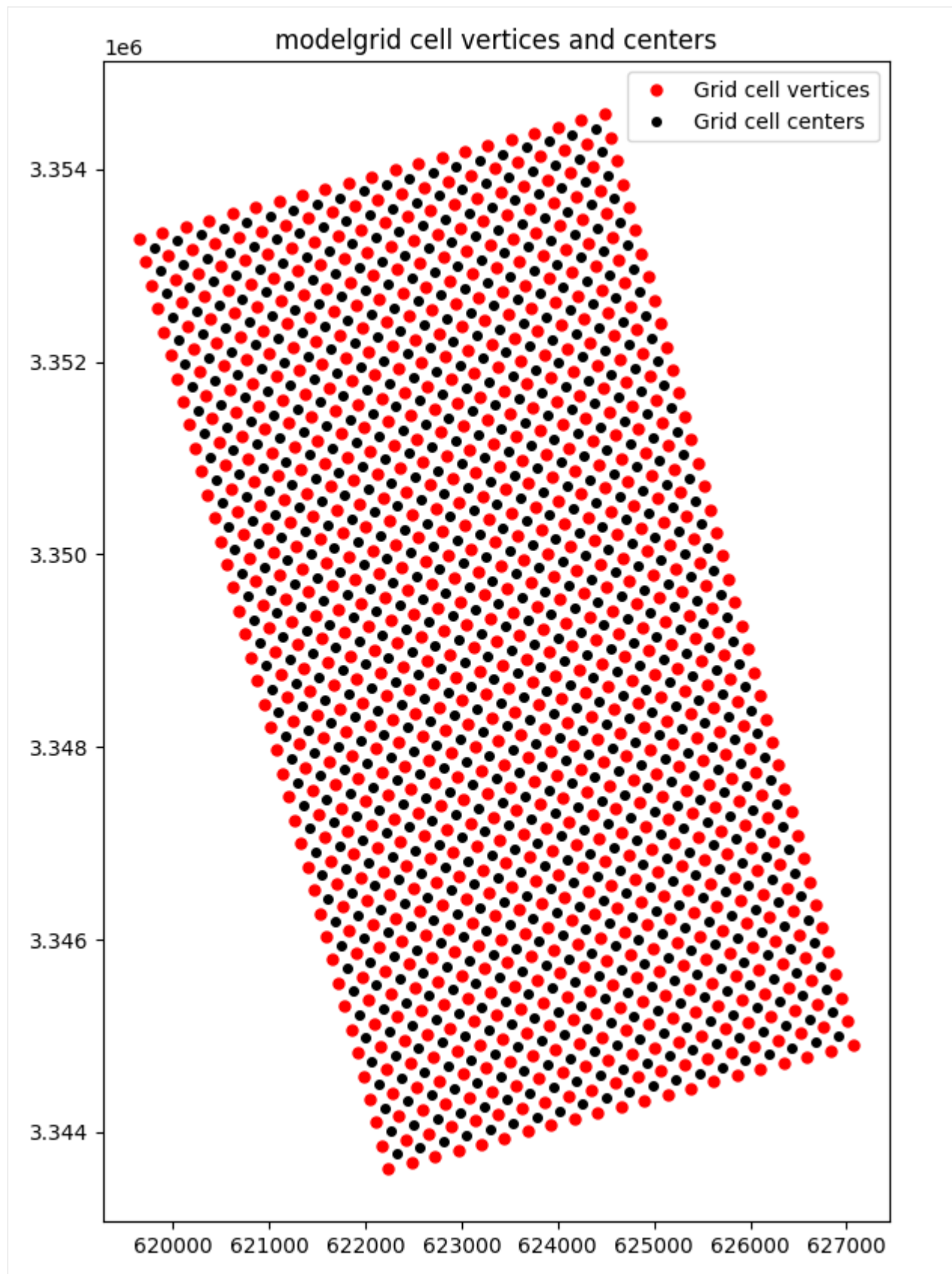
(continued from previous page)

```
number of cells in model: 800
```

```
[30]: # plot the model cell vertices and cell centers
fig, ax = plt.subplots(figsize=(10, 10), subplot_kw={"aspect": "equal"})

ax.plot(
    np.ravel(modelgrid.xvertices),
    np.ravel(modelgrid.yvertices),
    "ro",
    label="Grid cell vertices",
    ms=5,
)
ax.plot(
    np.ravel(modelgrid.xcellcenters),
    np.ravel(modelgrid.ycellcenters),
    "ko",
    label="Grid cell centers",
    ms=4,
)
plt.legend(
    loc=0,
)
plt.title("modelgrid cell vertices and centers")
```

```
[30]: Text(0.5, 1.0, 'modelgrid cell vertices and centers')
```




```
[31]: # plot layer 1 top, botm, and thickness with ibound overlain
fig, axs = plt.subplots(
    nrows=1, ncols=3, figsize=(18, 6), subplot_kw={"aspect": "equal"}
)

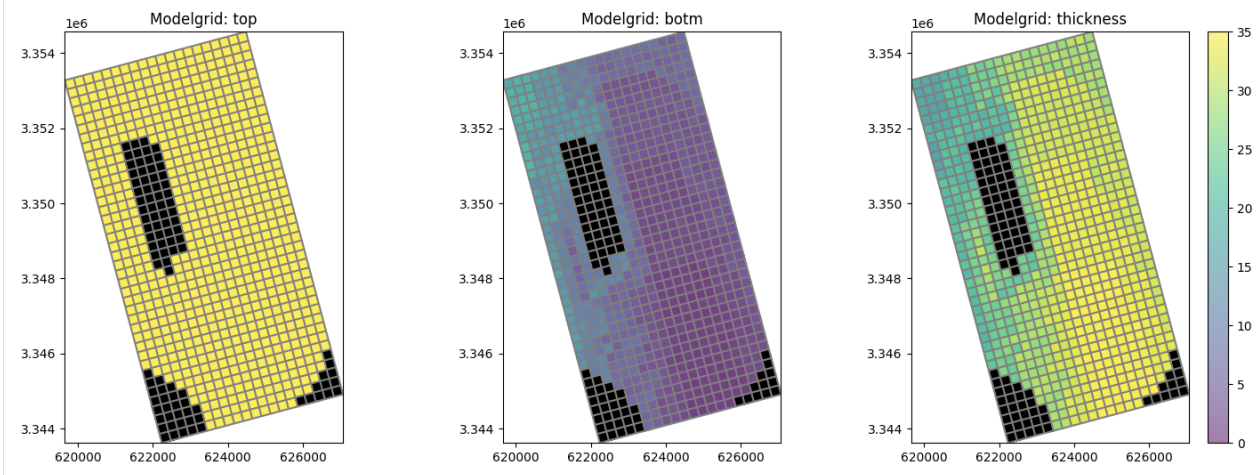
arrays = [modelgrid.top, modelgrid.botm, modelgrid.cell_thickness]

labels = ["top", "botm", "thickness"]

# plot arrays
for ix, ax in enumerate(axs):
    pmv = flopy.plot.PlotMapView(modelgrid=modelgrid, ax=ax)
    pc = pmv.plot_array(
        arrays[ix], masked_values=[1e30], vmin=0, vmax=35, alpha=0.5
    )
    pmv.plot_grid()
    pmv.plot_inactive()
    ax.set_title("Modelgrid: {}".format(labels[ix]))

plt.colorbar(pc)
```

```
[31]: <matplotlib.colorbar.Colorbar at 0x7fe0b49d6af0>
```



Common methods

There are also many common (shared) methods that are present in `StructuredGrid`, `VertexGrid`, and `UnstructuredGrid`:

Overview of methods

- `set_coord_info()` : Method to set coordinate reference information for the modelgrid
- `get_coords()` : Method to convert model coordinates to real-world coordinates based on coordinate reference information
- `get_local_coords()` : Method to convert real-world coordinates to model coordinates based on coordinate reference information
- `intersect()` : Method to get the cellid (StructuredGrid=(row, column) OR VertexGrid & UnstrucuturedGrid=node number) from either model coordinates or from real-world coordinates
- `saturated_thickness()` : Method to get the saturated thickness
- `write_shapefile()` : Method to write a shapefile of the grid with just the cellid attributes

set_coord_info()

Method to set coordinate reference information for the modelgrid. This method accepts the following optional parameters:

- `xoff` : lower-left corner of modelgrid x-coordinate location
- `yoff` : lower-left corner of modelgrid y-coordinate location
- `angrot` : rotation of model grid in degrees
- `epsg` : epsg code for model grid projection
- `proj4` : proj4 string describing the model grid projection
- `merge_coord_info` : boolean flag to either merge changes with the existing coordinate info or clear existing coordinate info before applying changes.

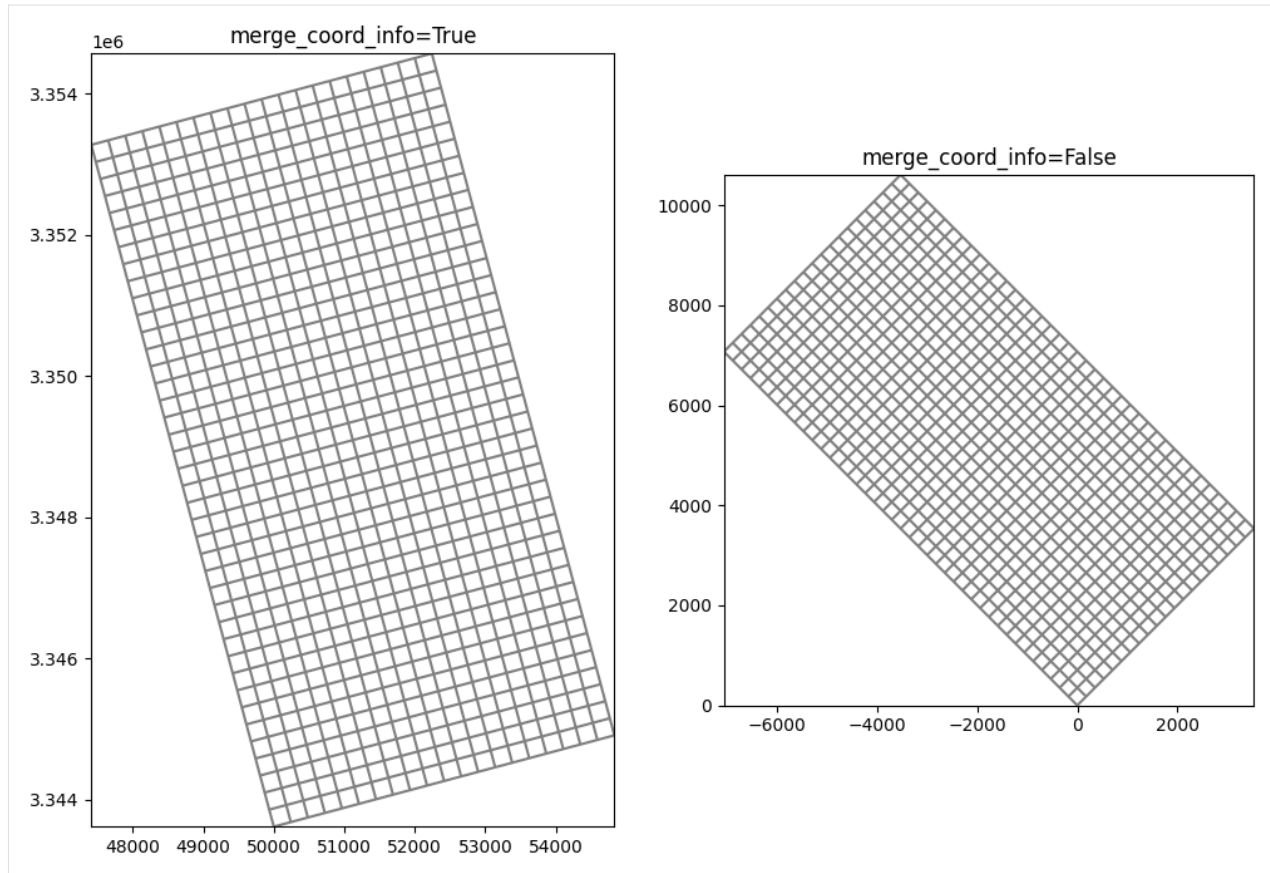
```
[32]: fig, ax = plt.subplots(
        nrows=1, ncols=2, figsize=(12, 8), subplot_kw={"aspect": "equal"}
    )

    # example usage with merge_coord_info=True
    modelgrid.set_coord_info(xoff=500000)
    print(modelgrid)
    modelgrid.plot(ax=ax[0])
    ax[0].set_title("merge_coord_info=True")

    # example usage with merge_coord_info=False
    modelgrid.set_coord_info(angrot=45, merge_coord_info=False)
    print(modelgrid)
    modelgrid.plot(ax=ax[1])
    ax[1].set_title("merge_coord_info=False")

    xll:500000; yll:3343617.741737109; rotation:15.0; crs:EPSG:32614; units:meters; lenuni:2
    xll:0.0; yll:0.0; rotation:45; units:meters; lenuni:2

[32]: Text(0.5, 1.0, 'merge_coord_info=False')
```



get_coords()

Method to convert model coordinates to real-world coordinates based on coordinate reference information. Input parameters include:

- **x** : float, list, or array of x coordinate values
- **y** : float, list, or array of y coordinate values

```
[33]: # create some synthetic pumping data
x = modelgrid.xoffset + np.random.rand(10) * 4000
y = modelgrid.yoffset + np.random.rand(10) * 10000
q = np.random.rand(10) * -1000

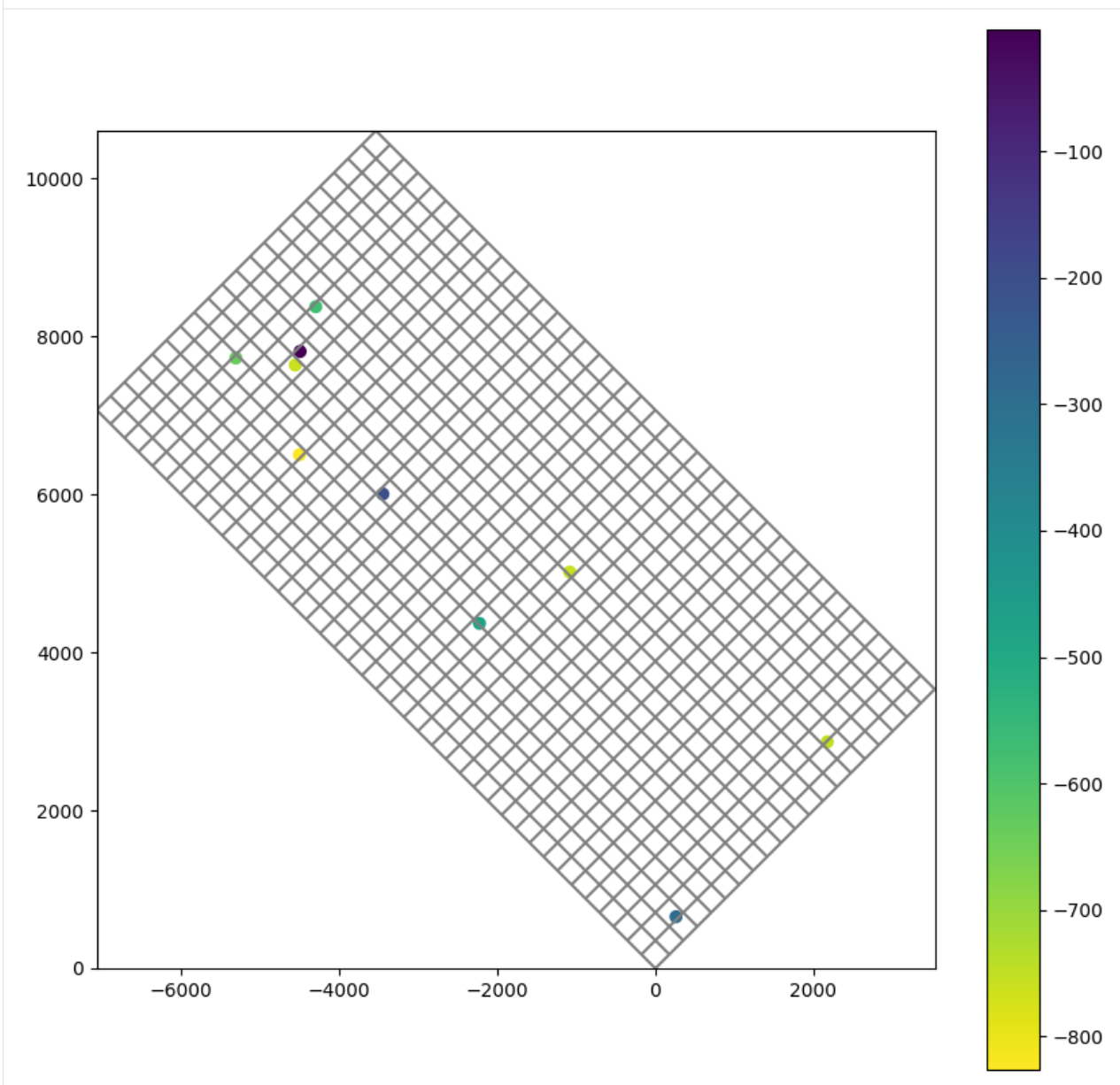
# get "real-world coordinates"
rw_coords = modelgrid.get_coords(x, y)

fig, ax = plt.subplots(figsize=(10, 10), subplot_kw={"aspect": "equal"})

pmv = flopy.plot.PlotMapView(modelgrid=modelgrid, ax=ax)
pmv.plot_grid()

s = ax.scatter(rw_coords[0], rw_coords[1], c=q, cmap="viridis_r")
plt.colorbar(s)
```

[33]: <matplotlib.colorbar.Colorbar at 0x7fe0b4606790>



get_local_coords()

Method to convert real-world coordinates to local coordinates based on coordinate reference information. Input parameters include:

- **x** : float, list, or array of x coordinate values
- **y** : float, list, or array of y coordinate values

```
[34]: # reuse the "real world" coordinates from the previous cell
l_coords = modelgrid.get_local_coords(rw_coords[0], rw_coords[1])
```

(continues on next page)

(continued from previous page)

```
# show that these have been properly translated
print(np.allclose(x, l_coords[0]))
print(np.allclose(y, l_coords[1]))

# plot the local coords on a modelgrid that is in model units
modelgrid.set_coord_info(merge_coord_info=False)

fig, ax = plt.subplots(figsize=(10, 10), subplot_kw={"aspect": "equal"})

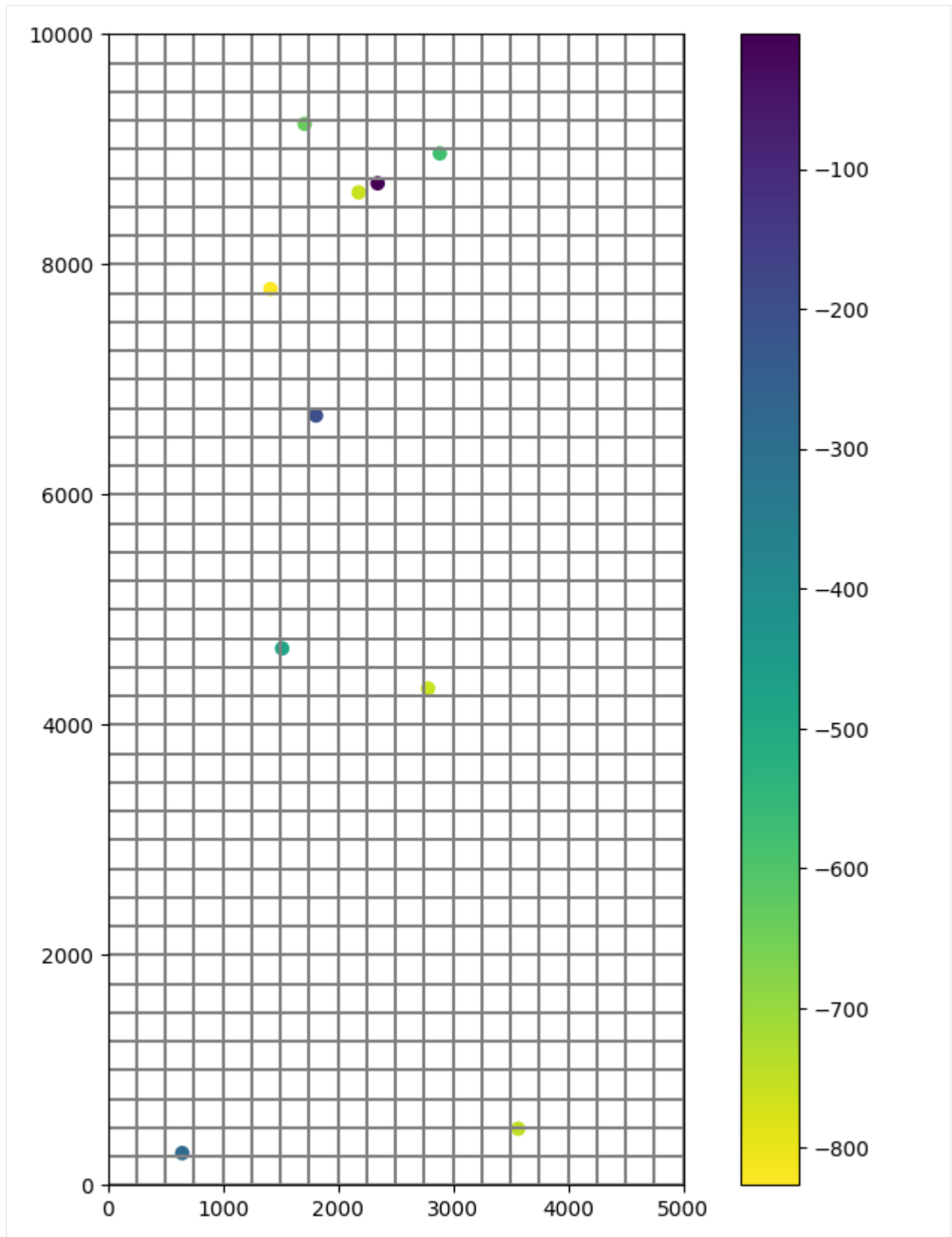
pmv = flopy.plot.PlotMapView(modelgrid=modelgrid, ax=ax)
pmv.plot_grid()

s = ax.scatter(l_coords[0], l_coords[1], c=q, cmap="viridis_r")
plt.colorbar(s)
```

True

True

[34]: <matplotlib.colorbar.Colorbar at 0x7fe0b44f7040>



intersect()

Method to get the cellid (StructuredGrid=(row, column) OR VertexGrid & UnstructuredGrid=node number) from either model coordinates or from real-world coordinates. Parameters include:

- **x** : x coordinate value
- **y** : y coordinate value
- **local** : if True x and y are in local coordinates (default is False)
- **forgive** : forgive x, y coordinate pairs and return NaN when coordinate pairs fall outside the model domain (default is False)

```
[35]: # apply rotation, xoffset, and yoffset to modelgrid
modelgrid.set_coord_info(
    xoff=622241.1904510253,
    yoff=3343617.741737109,
    angrot=15.0,
)

# generate some synthetic pumping data
x = modelgrid.xoffset + np.random.rand(10) * 4000
y = modelgrid.yoffset + np.random.rand(10) * 10000
q = np.random.rand(10) * -0.02

# intersect to get row and column locations and create pumping data
pdata = []
for ix, xc in enumerate(x):
    i, j = modelgrid.intersect(xc, y[ix], forgive=True)
    if not np.isnan(i) and not np.isnan(j):
        if modelgrid.idomain[0, i, j]:
            pdata.append([(0, i, j), q[ix]])

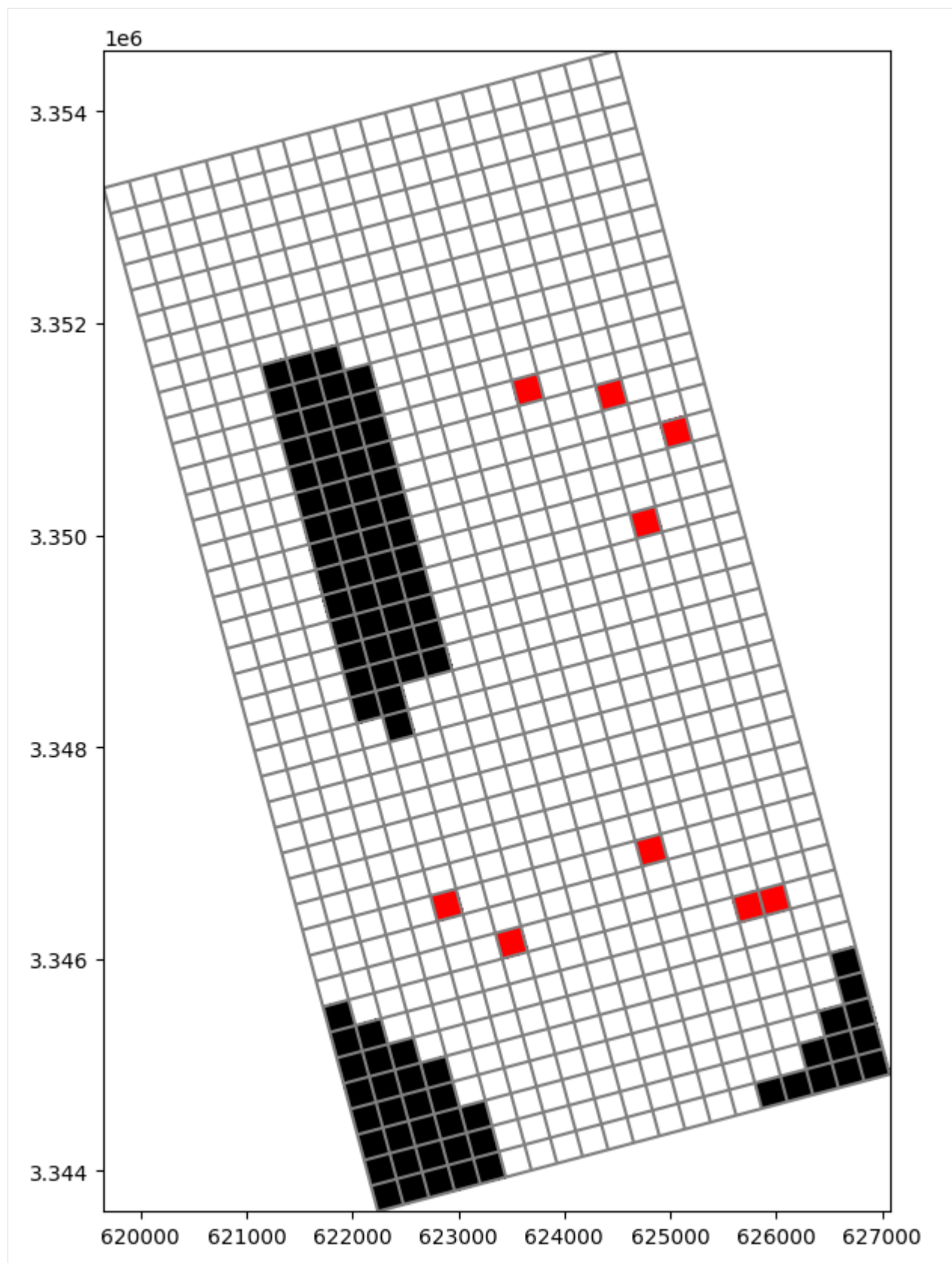
# remove existing wel package
ml.remove_package("wel_0")

# create a mf6 WEL package and add it to the existing model
stress_period_data = {0: pdata}
wel = flopy.mf6.modflow.ModflowGwfwel(
    ml, stress_period_data=stress_period_data
)

# plot the locations from the new WEL package on the modelgrid
fig, ax = plt.subplots(figsize=(10, 10), subplot_kw={"aspect": "equal"})

pmv = flopy.plot.PlotMapView(modelgrid=modelgrid, ax=ax)
pmv.plot_grid()
pmv.plot_inactive()
pmv.plot_bc(package=wel)
```

```
[35]: <matplotlib.collections.QuadMesh at 0x7fe0b4432490>
```



saturated_thickness()

Method to calculate the saturated thickness of each model cell from an array of elevations. Input parameters to this method include:

- `array` : array of elevations that will be used to adjust the cell thickness
- `mask` : float, list, tuple, or array of values to replace with a NaN value (ex. `hdry` and `hnoFlo` values)

```
[36]: # write inputs and run the mf6 simulation with the new wells we added above
sim.set_sim_path(gridgen_ws)
sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

# load the binary head file from the model
ml = sim.freyberg
hds = ml.output.head()
head = hds.get_alldata()[0]

# calculate saturated thickness
sat_thk = ml.modelgrid.saturated_thickness(head, mask=[1e30])

# get thick for comparison
thk = ml.modelgrid.cell_thickness

# plot thickness and saturated thickness for comparison
fig, ax = plt.subplots(
    nrows=1, ncols=2, figsize=(16, 7), subplot_kw={"aspect": "equal"}
)

pmv = flopy.plot.PlotMapView(modelgrid=ml.modelgrid, ax=ax[0])
pc = pmv.plot_array(thk, vmin=5, vmax=35)
pmv.plot_ibound()
pmv.plot_grid()
ax[0].set_title("Modelgrid thickness")

pmv = flopy.plot.PlotMapView(modelgrid=ml.modelgrid, ax=ax[1])
pc = pmv.plot_array(sat_thk, vmin=5, vmax=35)
pmv.plot_ibound()
pmv.plot_grid()
ax[1].set_title("Saturated thickness")

plt.tight_layout()
plt.colorbar(pc)
```

MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64

(continues on next page)

(continued from previous page)

Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:19

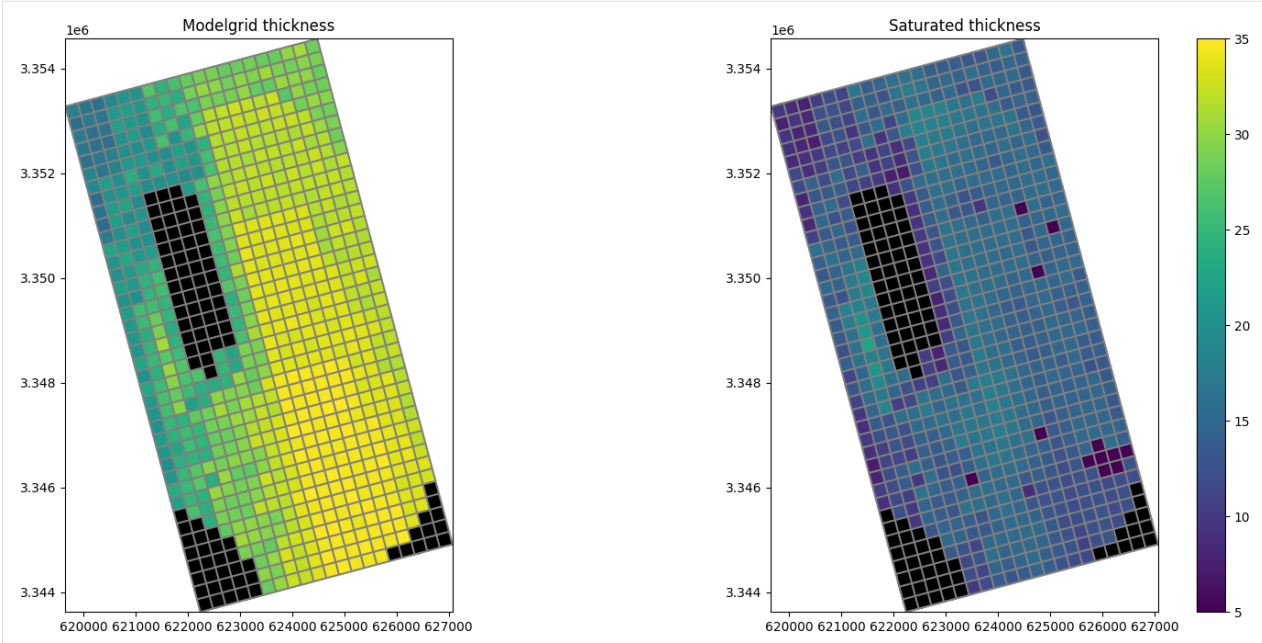
Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

Solving: Stress period: 1 Time step: 1

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:19
Elapsed run time: 0.079 Seconds

Normal termination of simulation.

[36]: <matplotlib.colorbar.Colorbar at 0x7fe0b42f9e20>



write_shapefile()

Method to write a shapefile of the grid with just the cellid attributes. Input parameters include:

- filename : shapefile name
- epsg : optional epsg code of the coordinate system projection
- prj : optional, input projection file to be used to define the coordinate system projection

```
[37]: # write a shapefile
shp_name = os.path.join(gridgen_ws, "freyberg-6_grid.shp")
epsg = "32614"

ml.modelgrid.write_shapefile(shp_name, epsg)

/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/utils/crs.py:
→119: PendingDeprecationWarning: the epsg argument will be deprecated and will be
→removed in version 3.4. Use crs instead.
warnings.warn(
```

```
[38]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass
```

6.1.6 Intersecting rasters with modelgrids using FloPy's Raster class

A Raster class was developed as a wrapper that leverages RasterIO, RasterStats, and SciPy built in methods for easy raster intersections and cropping.

This notebook will show some of the basic functionality of the Raster class with structured and unstructured model grid examples.

The Raster class accepts Tiff and GeoTiff, ASCII Grid (ESRI ASCII), and Erdas Imagine .img files.

Ideally this can be used to easily snap DEM rasters, PET, PPT, recharge and other rasters to a modflow grid for further processing and/or to apply as fluxes and boundary conditions to a MODFLOW model

```
[1]: import os
import sys
import time
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import shapefile
import shapely

try:
    import flopy
    from flopy.utils import Raster
```

(continues on next page)

(continued from previous page)

```
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy
    from flopy.utils import Raster

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("pandas version: {}".format(pd.__version__))
print("shapely version: {}".format(shapely.__version__))
print("flopy version: {}".format(flopy.__version__))
```

```
3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
pandas version: 2.0.3
shapely version: 2.0.1
flopy version: 3.4.3
```

```
[2]: # temporary directory
temp_dir = TemporaryDirectory()
workspace = temp_dir.name
```

Raster files can be loaded using the `Raster.load` method

```
[3]: raster_ws = os.path.join("..", "..", "examples", "data", "options", "dem")
raster_name = "dem.img"

rio = Raster.load(os.path.join(raster_ws, raster_name))
```

The bands within the raster can be viewed by calling the parameter `bands`; there is only one band in this raster

```
[4]: rio.bands
```

```
[4]: (1,)
```

```
[5]: arr = rio.get_array(1)
idx = np.isfinite(arr)

vmin, vmax = arr[idx].min(), arr[idx].max()
vmin, vmax
```

```
[5]: (1920.5467529296875, 2663.162109375)
```

Using the built in `.plot` method, we can take a look at the DEM raster data before we start manipulating it

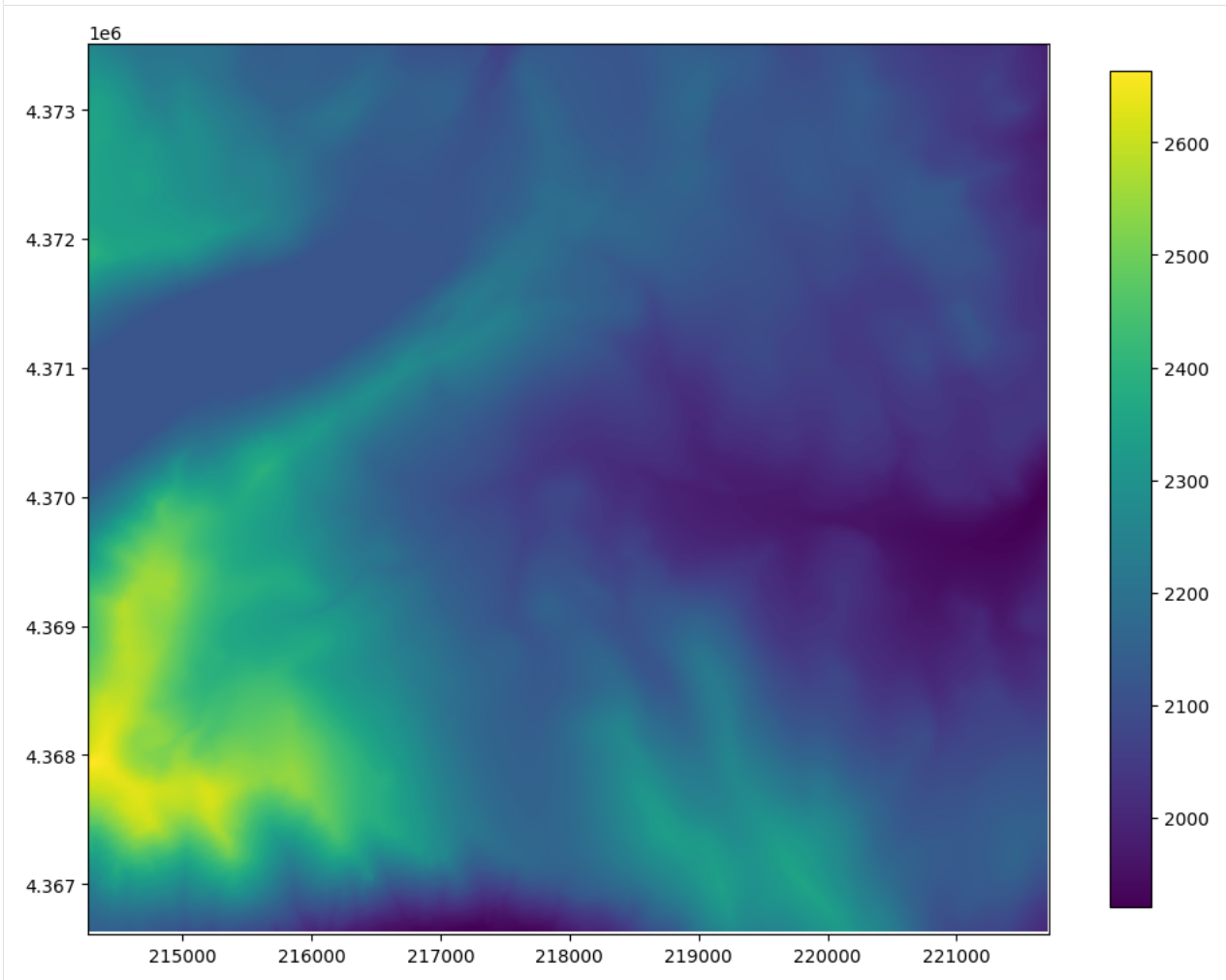
```
[6]: fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
```

(continues on next page)

(continued from previous page)

```
ax = rio.plot(ax=ax, vmin=vmin, vmax=vmax)
plt.colorbar(ax.images[0], shrink=0.7)
```

[6]: <matplotlib.colorbar.Colorbar at 0x7f981f8edeb0>



Intersecting and resampling a data using the FloPy ModelGrid

Structured Grid Example

The structured grid example uses the DIS file from the GSFLOW Sagehen example problem to create a modelgrid

```
[7]: model_ws = os.path.join("../", "..", "examples", "data", "options", "sagehen")
ml = flopy.modflow.Modflow.load(
    "sagehen.nam", version="mfwt", model_ws=model_ws
)

xoff = 214110
yoff = 4366620
ml.modelgrid.set_coord_info(xoff=xoff, yoff=yoff)
```

```

loading iuzfbnd array...
loading vks array...
loading eps array...
loading thts array...
stress period 1:
  loading finf array...
stress period 2:

```

```

[8]: fig = plt.figure(figsize=(12, 12))
    ax = fig.add_subplot(1, 1, 1, aspect="equal")

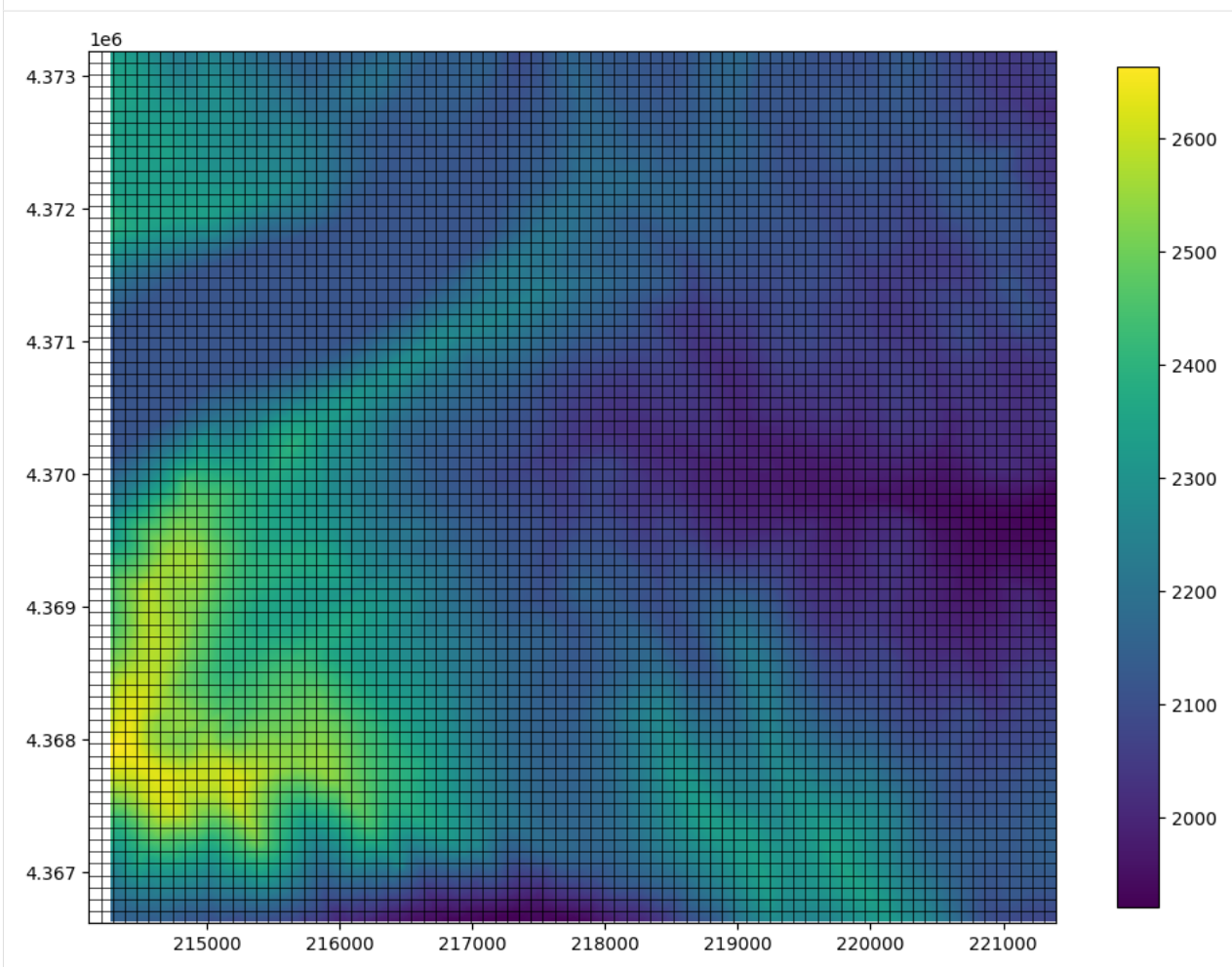
    ax = rio.plot(ax=ax, vmin=vmin, vmax=vmax)
    plt.colorbar(ax.images[0], shrink=0.7)
    pmv = flopy.plot.PlotMapView(modelgrid=ml.modelgrid)
    pmv.plot_grid(ax=ax, lw=0.5, color="black")

```

```

[8]: <matplotlib.collections.LineCollection at 0x7f981f14ffa0>

```



Once a `modelgrid` has been loaded, the `resample_to_grid()` method can be used to re-sample the data to an array consistent with the model grid.

Inputs to `resample_to_grid()` include:

- `modelgrid`: flopy Grid object

- `band` : raster band to resample
- `method` : resampling method, options include:
 - "nearest" for nearest neighbor
 - "linear" for bilinear sampling
 - "cubic" for bicubic sampling
 - "mean" for mean value sampling
 - "median" for median value sampling
 - "min" for minimum value sampling
 - "max" for maximum value sampling
 - "mode" for most often (dominant) sampling
- `extrapolate_edges` : boolean flag to extrapolate edges using the "nearest" resampling method. For all of the sampling methods except "nearest", interpolation is only performed in areas bounded by data; nodata values are returned in areas without data. This option has no effect when the "nearest" interpolation method is used.

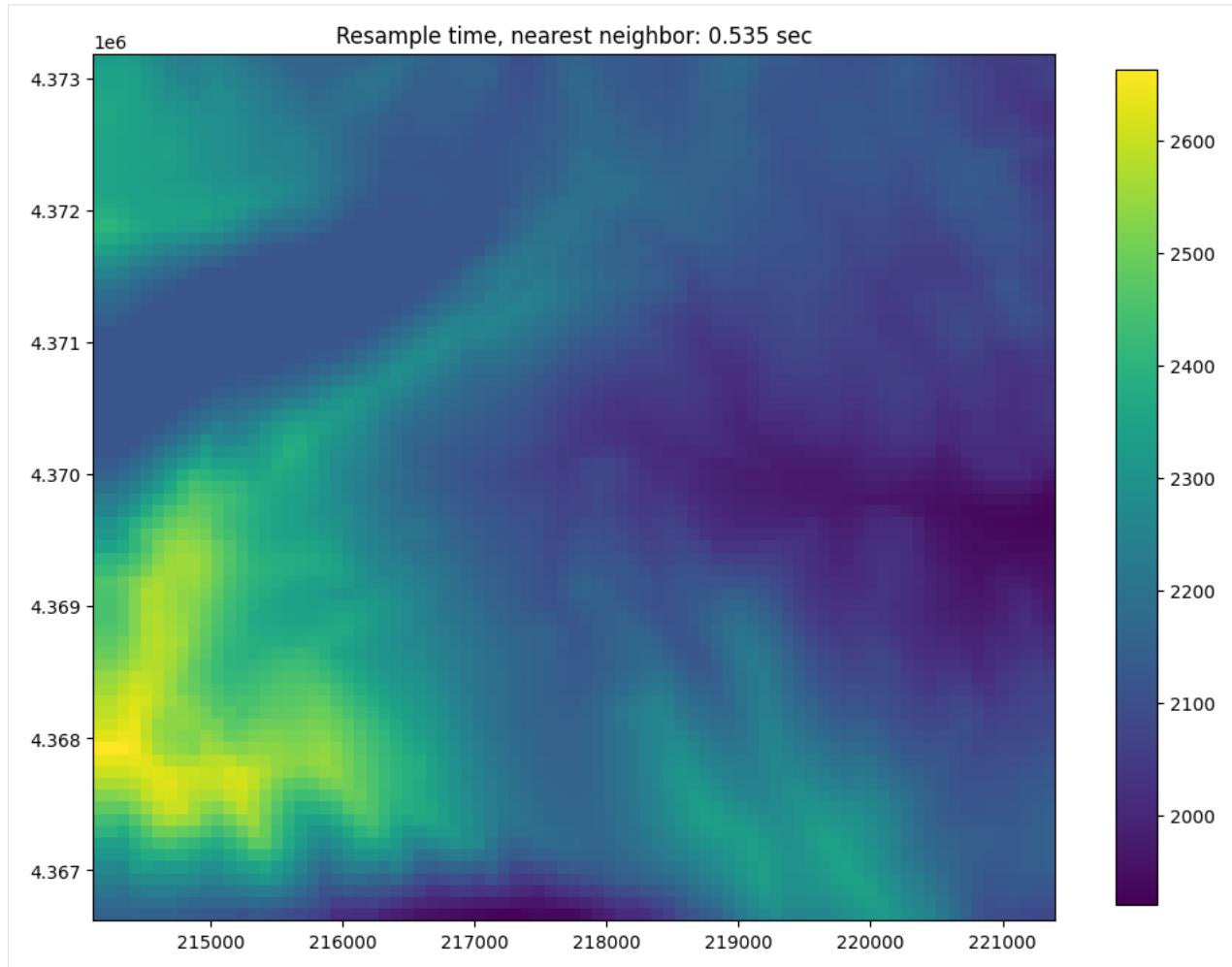
Note: Bottlenecks in sampling time depend on the resampling method used: + "nearest", "linear", and "cubic" bottlenecks are due to raster resolution. + "mean", "median", "min", "max", and "mode" are a function of the number of grid cells.

```
[9]: t0 = time.time()
dem_data = rio.resample_to_grid(
    ml.modelgrid, band=rio.bands[0], method="nearest"
)
resample_time = time.time() - t0
```

```
[10]: # now to visualize using flopy and matplotlib
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

pmv = flopy.plot.PlotMapView(modelgrid=ml.modelgrid, ax=ax)
ax = pmv.plot_array(
    dem_data, masked_values=rio.nodatavals, vmin=vmin, vmax=vmax
)
plt.title("Resample time, nearest neighbor: {:.3f} sec".format(resample_time))
plt.colorbar(ax, shrink=0.7)
```

```
[10]: <matplotlib.colorbar.Colorbar at 0x7f980ca36550>
```

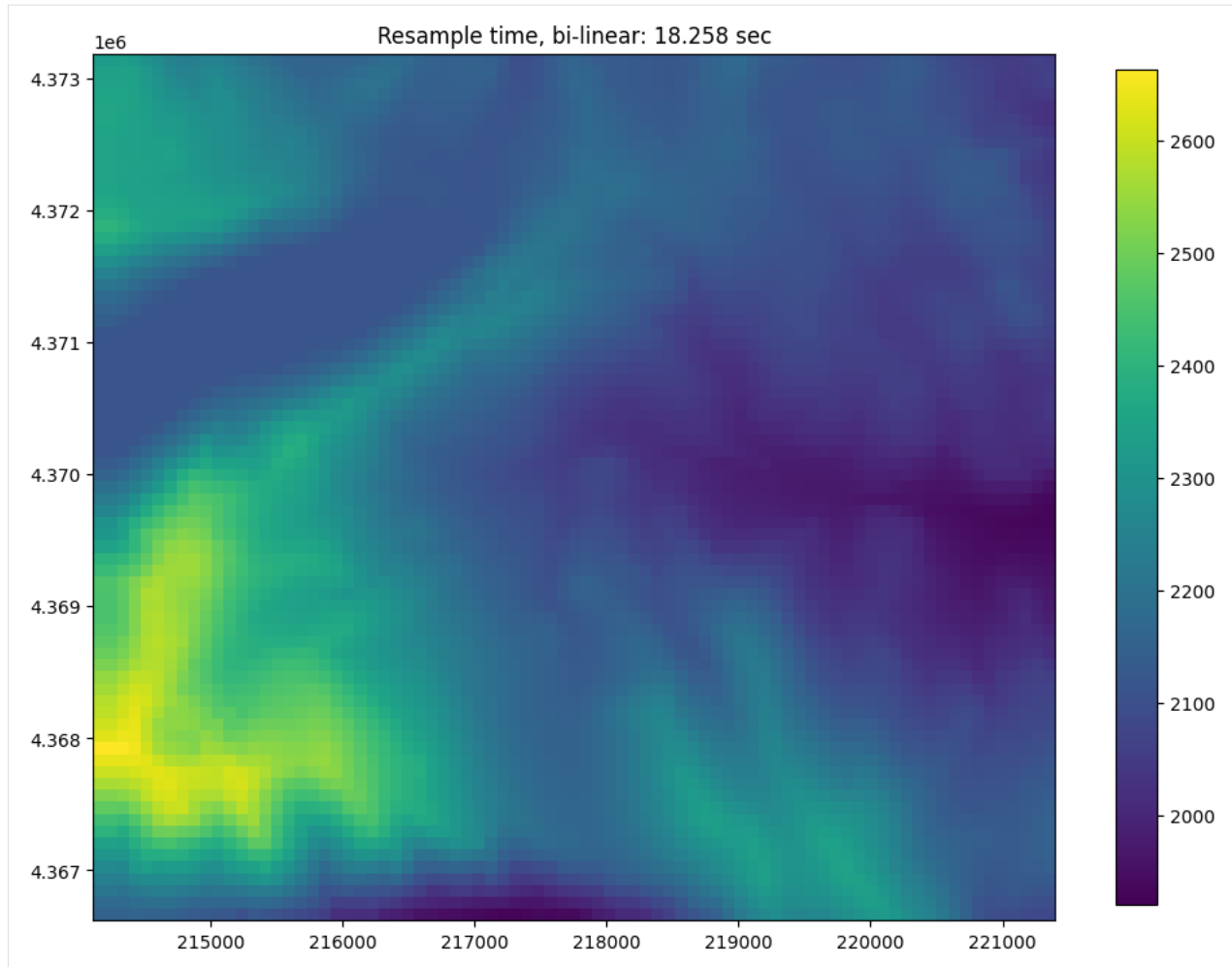


```
[11]: t0 = time.time()
dem_data = rio.resample_to_grid(
    ml.modelgrid, band=rio.bands[0], method="linear", extrapolate_edges=True
)
resample_time = time.time() - t0
```

```
[12]: # now to visualize using flopy and matplotlib
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

pmv = flopy.plot.PlotMapView(modelgrid=ml.modelgrid, ax=ax)
ax = pmv.plot_array(
    dem_data, masked_values=rio.nodatavals, vmin=vmin, vmax=vmax
)
plt.title("Resample time, bi-linear: {:.3f} sec".format(resample_time))
plt.colorbar(ax, shrink=0.7)
```

```
[12]: <matplotlib.colorbar.Colorbar at 0x7f980c926220>
```

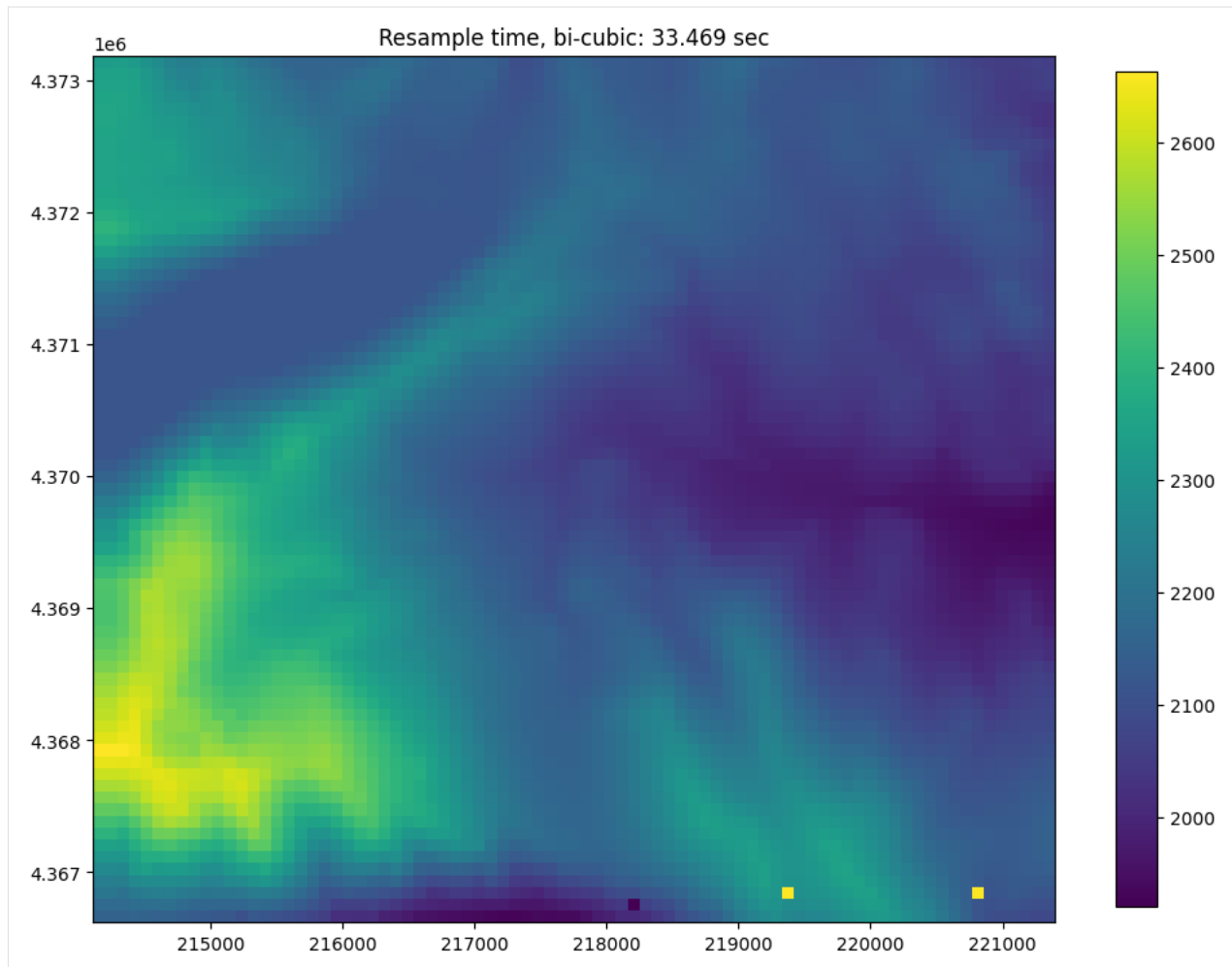



```
[13]: t0 = time.time()
dem_data = rio.resample_to_grid(
    ml.modelgrid, band=rio.bands[0], method="cubic", extrapolate_edges=True
)
resample_time = time.time() - t0
```

```
[14]: # now to visualize using flopy and matplotlib
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

pmv = flopy.plot.PlotMapView(modelgrid=ml.modelgrid, ax=ax)
ax = pmv.plot_array(
    dem_data, masked_values=rio.nodatavals, vmin=vmin, vmax=vmax
)
plt.title("Resample time, bi-cubic: {:.3f} sec".format(resample_time))
plt.colorbar(ax, shrink=0.7)
```

```
[14]: <matplotlib.colorbar.Colorbar at 0x7f980c87cee0>
```

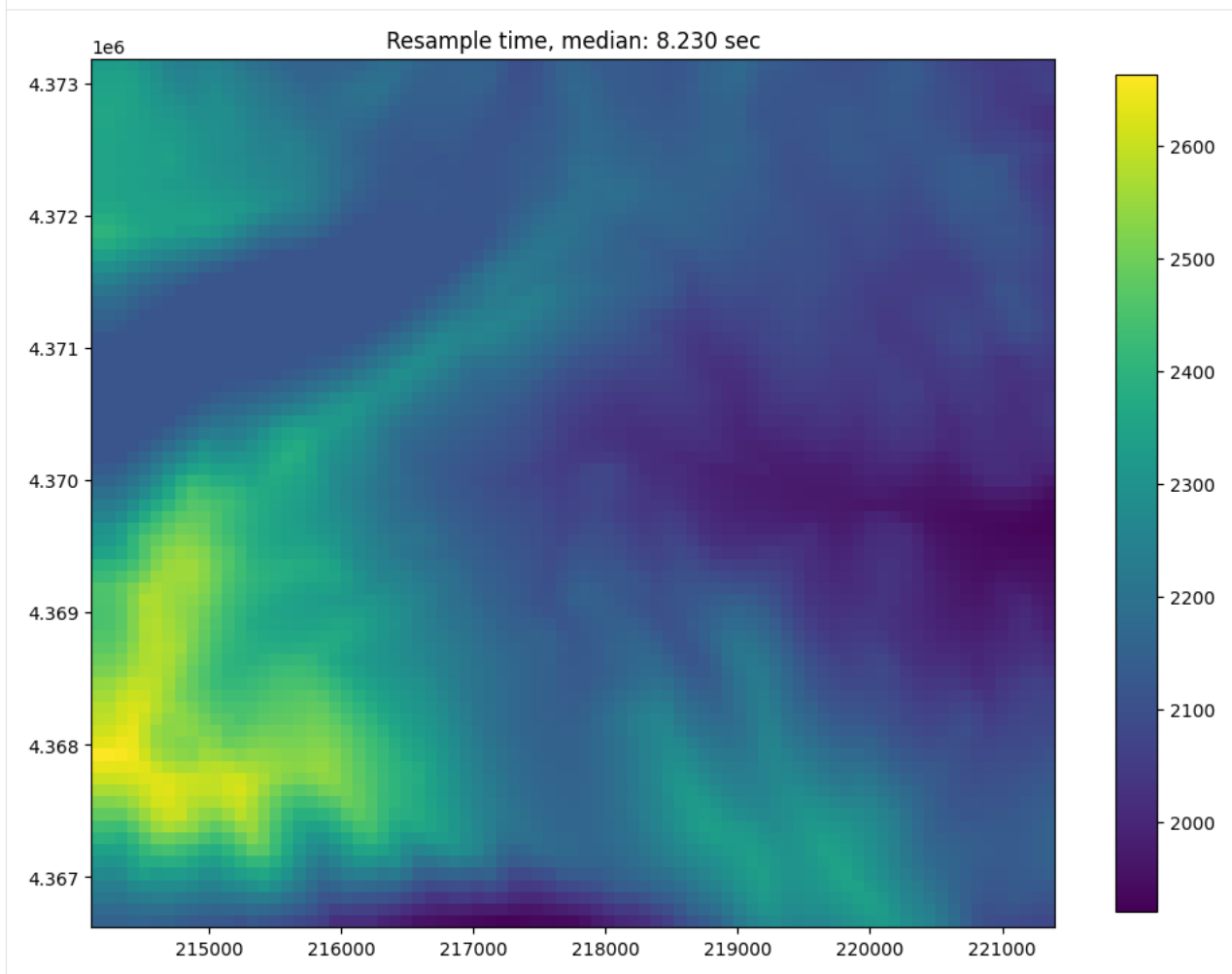


```
[15]: t0 = time.time()
dem_data = rio.resample_to_grid(
    ml.modelgrid,
    band=rio.bands[0],
    method="median",
    extrapolate_edges=True,
)
resample_time = time.time() - t0
```

```
[16]: # now to visualize using flopy and matplotlib
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

pmv = flopy.plot.PlotMapView(modelgrid=ml.modelgrid, ax=ax)
ax = pmv.plot_array(
    dem_data, masked_values=rio.nodatavals, vmin=vmin, vmax=vmax
)
plt.title("Resample time, median: {:.3f} sec".format(resample_time))
plt.colorbar(ax, shrink=0.7)
```

[16]: <matplotlib.colorbar.Colorbar at 0x7f980c4a8970>



Vertex and Unstructured grid example

The user can also use either a vertex grid or an unstructured grid and resample raster data to it using the same `resample_to_grid()` method

Here is an example of building a triangular mesh and creating an unstructured grid instance to use for Raster resampling

```
[17]: from flopy.utils.triangle import Triangle
```

```
maximum_area = 30000.0 # 30000.
```

```
extent = rio.bounds
```

```
domainpoly = [
    (extent[0], extent[2]),
    (extent[1], extent[2]),
    (extent[1], extent[3]),
    (extent[0], extent[3]),
]
```

(continues on next page)

(continued from previous page)

```
tri = Triangle(maximum_area=maximum_area, angle=30, model_ws=workspace)

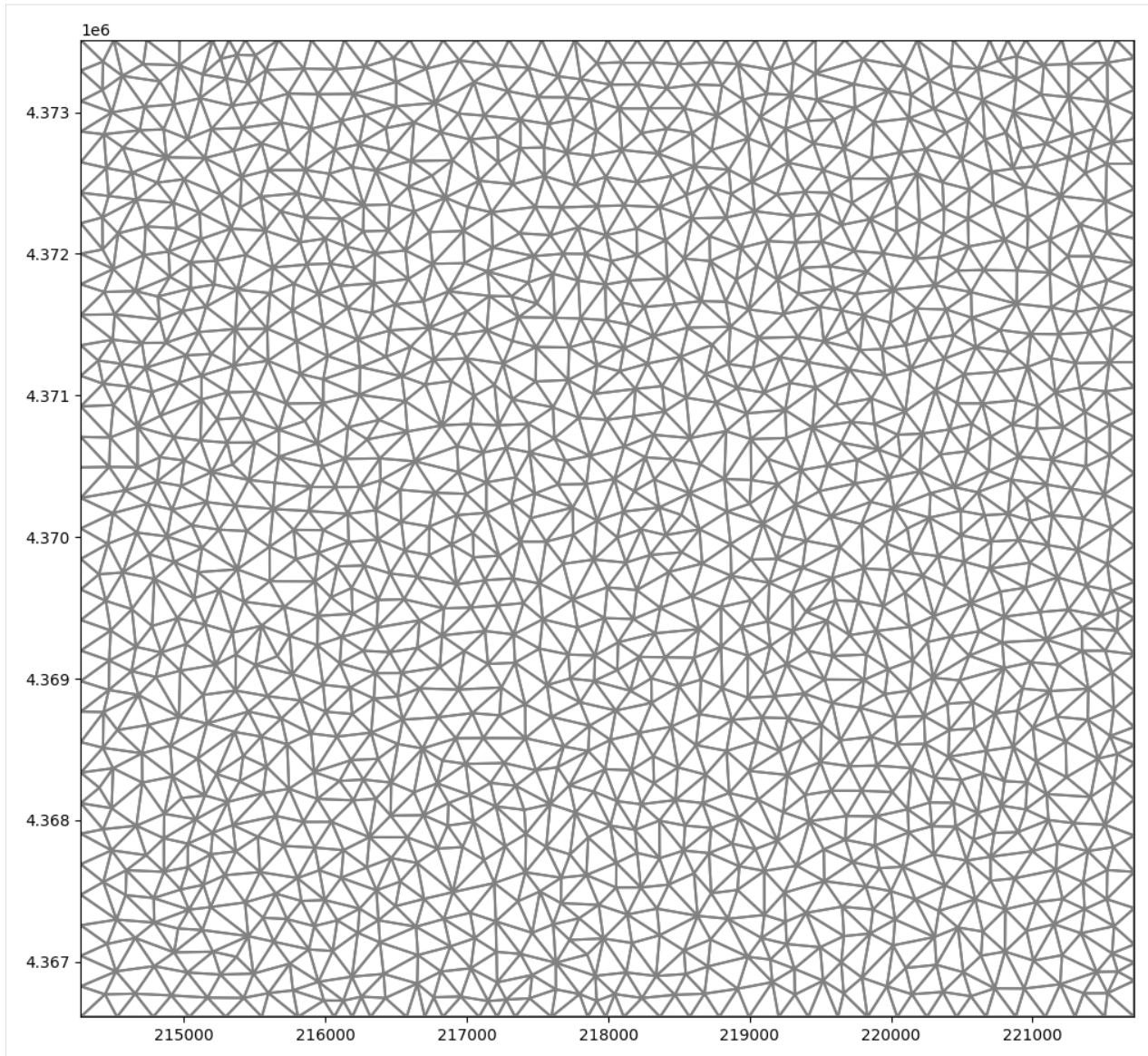
tri.add_polygon(domainpoly)
tri.build(verbose=False)
xc, yc = tri.get_xcyc().T
verts = [[iv, x, y] for iv, (x, y) in enumerate(tri.verts)]
iverts = tri.iverts
ncpl = np.array([len(iverts)])

mg_unstruct = flopy.discretization.UnstructuredGrid(
    vertices=verts, iverts=iverts, ncpl=ncpl, xcenters=xc, ycenters=yc
)
```

```
[18]: # now to visualize using matplotlib
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

pmv = flopy.plot.PlotMapView(modelgrid=mg_unstruct, ax=ax)
pmv.plot_grid()
```

```
[18]: <matplotlib.collections.LineCollection at 0x7f980c965160>
```



Once a grid object is created, the raster can be resampled to the grid using the same `resample_to_grid()` method as the structured grid example

```
[19]: t0 = time.time()
dem_data = rio.resample_to_grid(
    mg_unstruct, band=rio.bands[0], method="nearest"
)

resample_time = time.time() - t0
```

```
[20]: # now to visualize using flopy and matplotlib
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

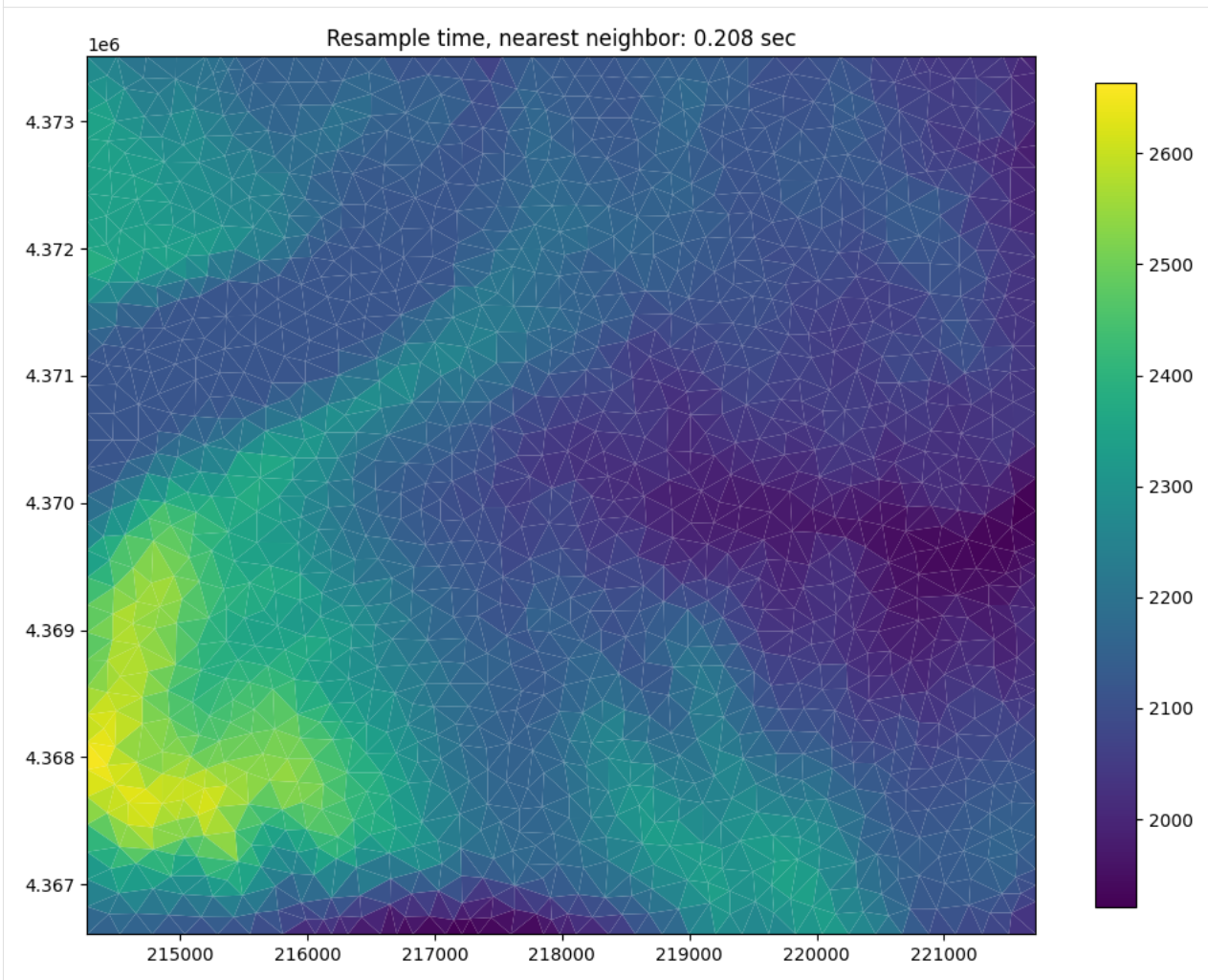
pmv = flopy.plot.PlotMapView(modelgrid=mg_unstruct, ax=ax)
```

(continues on next page)

(continued from previous page)

```
ax = pmv.plot_array(
    dem_data,
    masked_values=rio.nodatavals,
    cmap="viridis",
    vmin=vmin,
    vmax=vmax,
)
plt.title("Resample time, nearest neighbor: {:.3f} sec".format(resample_time))
plt.colorbar(ax, shrink=0.7)
```

[20]: <matplotlib.colorbar.Colorbar at 0x7f980bf40940>



```
[21]: t0 = time.time()
dem_data = rio.resample_to_grid(
    mg_unstruct, band=rio.bands[0], method="linear"
)

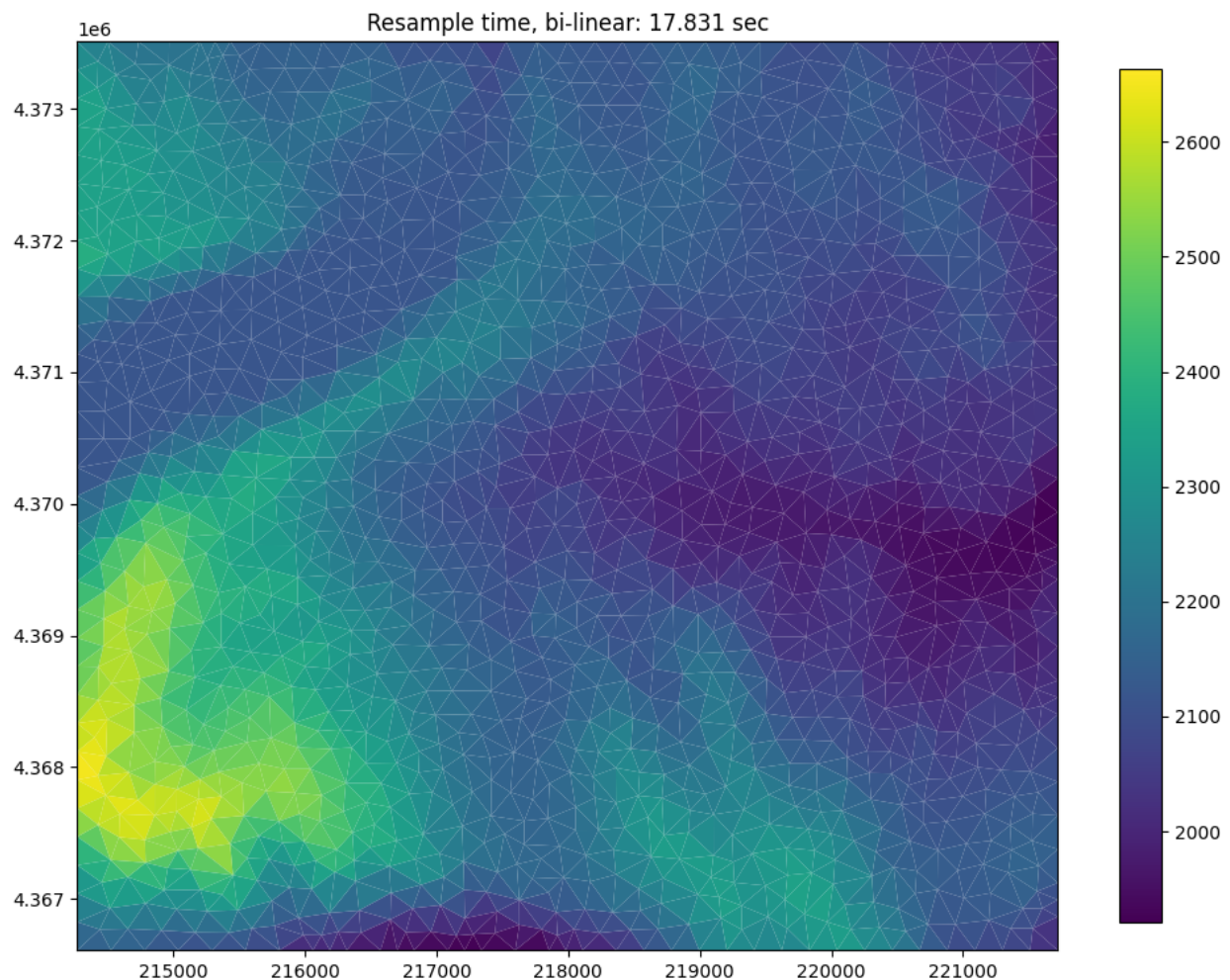
resample_time = time.time() - t0
```



```
[22]: # now to visualize using flopy and matplotlib
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

pmv = flopy.plot.PlotMapView(modelgrid=mg_unstruct, ax=ax)
ax = pmv.plot_array(
    dem_data,
    masked_values=rio.nodatavals,
    cmap="viridis",
    vmin=vmin,
    vmax=vmax,
)
plt.title("Resample time, bi-linear: {:.3f} sec".format(resample_time))
plt.colorbar(ax, shrink=0.7)
```

```
[22]: <matplotlib.colorbar.Colorbar at 0x7f980be1f940>
```



```
[23]: t0 = time.time()
dem_data = rio.resample_to_grid(
    mg_unstruct,
    band=rio.bands[0],
```

(continues on next page)

(continued from previous page)

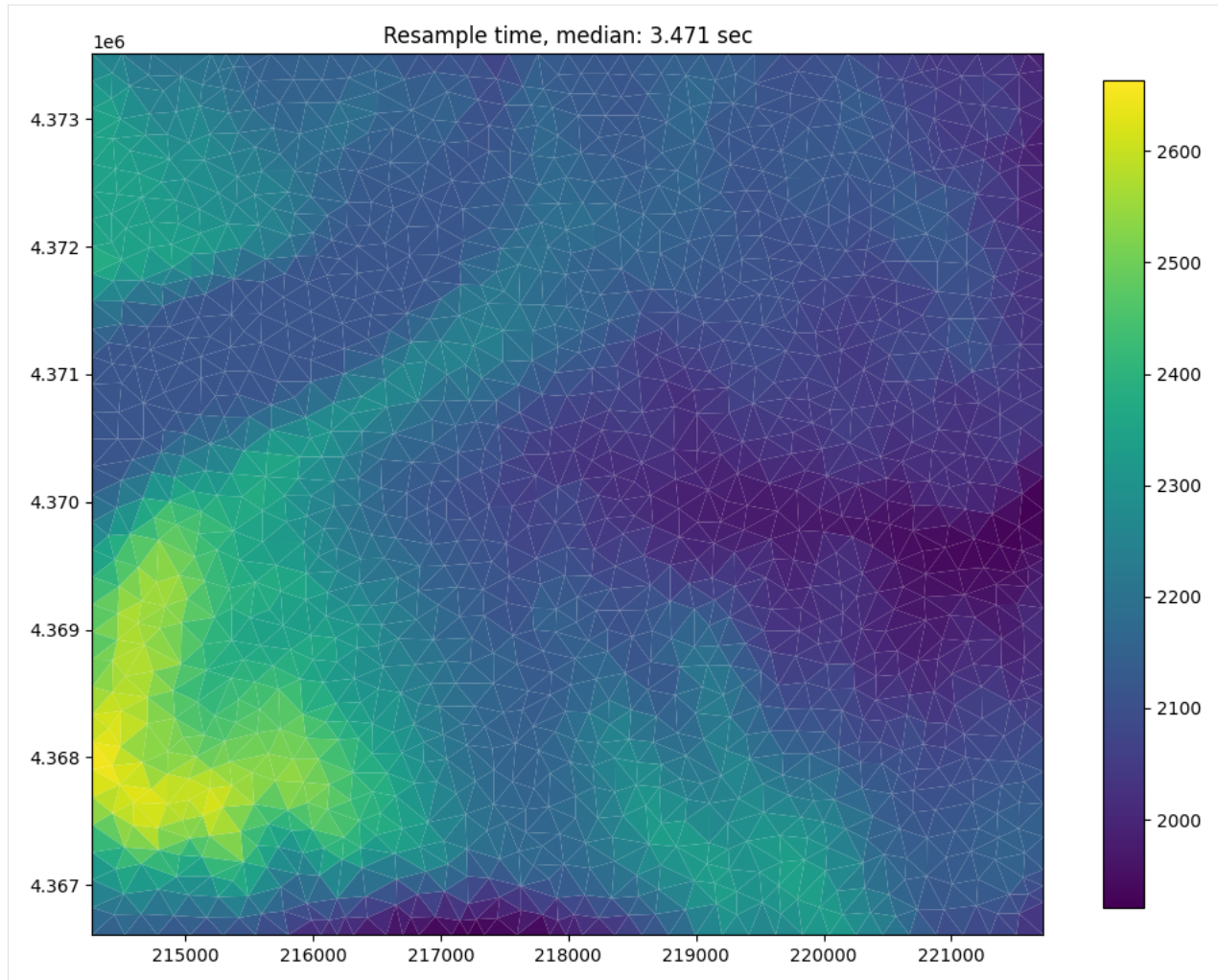
```
        method="median",
    )

    resample_time = time.time() - t0
```

```
[24]: # now to visualize using flopy and matplotlib
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

pmv = flopy.plot.PlotMapView(modelgrid=mg_unstruct, ax=ax)
ax = pmv.plot_array(
    dem_data,
    masked_values=rio.nodatavals,
    cmap="viridis",
    vmin=vmin,
    vmax=vmax,
)
plt.title("Resample time, median: {:.3f} sec".format(resample_time))
plt.colorbar(ax, shrink=0.7)
```

```
[24]: <matplotlib.colorbar.Colorbar at 0x7f980be50af0>
```

Note: bi-cubic sampling does not work well with triangular meshes and is not recommended for unstructured grids

Sampling points, Cropping, and performing intersections using raster data

The `Raster` class contains useful methods for sampling single points, cross sections, cropping and performing intersections.

The `sample_point()` method can be used to sample a single raster value or to sample a cross section

The `sample_polygon()` method can be used to sample all raster values within an arbitrary polygon

The `crop()` method allows the user to crop the raster in-place. This method can also be used to perform intersections.

The `crop()` and `sample_polygon()` methods apply a modified binary ray casting algorithm for extremely fast intersections. The raster data that's used for this example contains over 500,000 points. For each intersection every point must be segmented as inside or outside of an arbitrary polygon.

Sampling points or a cross section from the raster

The user can also sample from a points within the raster using the `sample_point()` method.

This can be used to create simple cross sections of data, such as an elevation profile

```
[25]: d = {"easting": [], "northing": [], "elevation": []}

for adj in range(1, 10000, 100):
    easting = xoff + adj
    northing = yoff + adj
    val = rio.sample_point(xoff + adj, yoff + adj, band=1)
    d["easting"].append(easting)
    d["northing"].append(northing)
    d["elevation"].append(val)

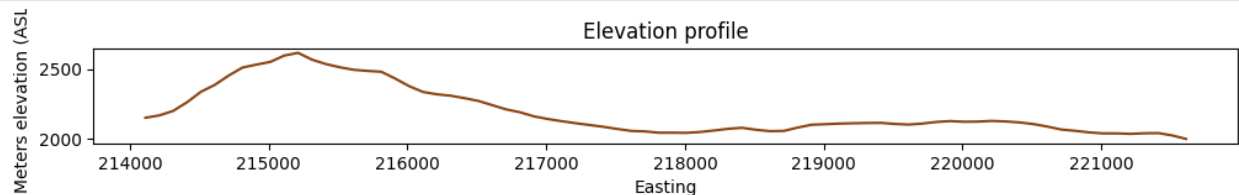
df = pd.DataFrame(d)

[26]: fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

ax.plot(df.easting.values, df.elevation.values, color="saddlebrown")
ax.set_ylabel("Meters elevation (ASL)")
ax.set_xlabel("Easting")
ax.set_title("Elevation profile")

df.head()
```

```
[26]:   easting  northing  elevation
0   214111   436621   2149.143311
1   214211   4366721  2165.882812
2   214311   4366821  2198.365234
3   214411   4366921  2260.733643
4   214511   4367021  2335.735596
```



Sampling all points within a polygon in the raster

The user can also sample all points within an arbitrary polygon within the raster using the `sample_polygon()` method.

The `sample_polygon()` method returns an unordered array of raster values that can be used to perform statical analysis on a chunk of the raster data

```
[27]: x0, x1, y0, y1 = rio.bounds

# let's create an a square to use for sampling and cropping
x0 += 1000
```

(continues on next page)

(continued from previous page)

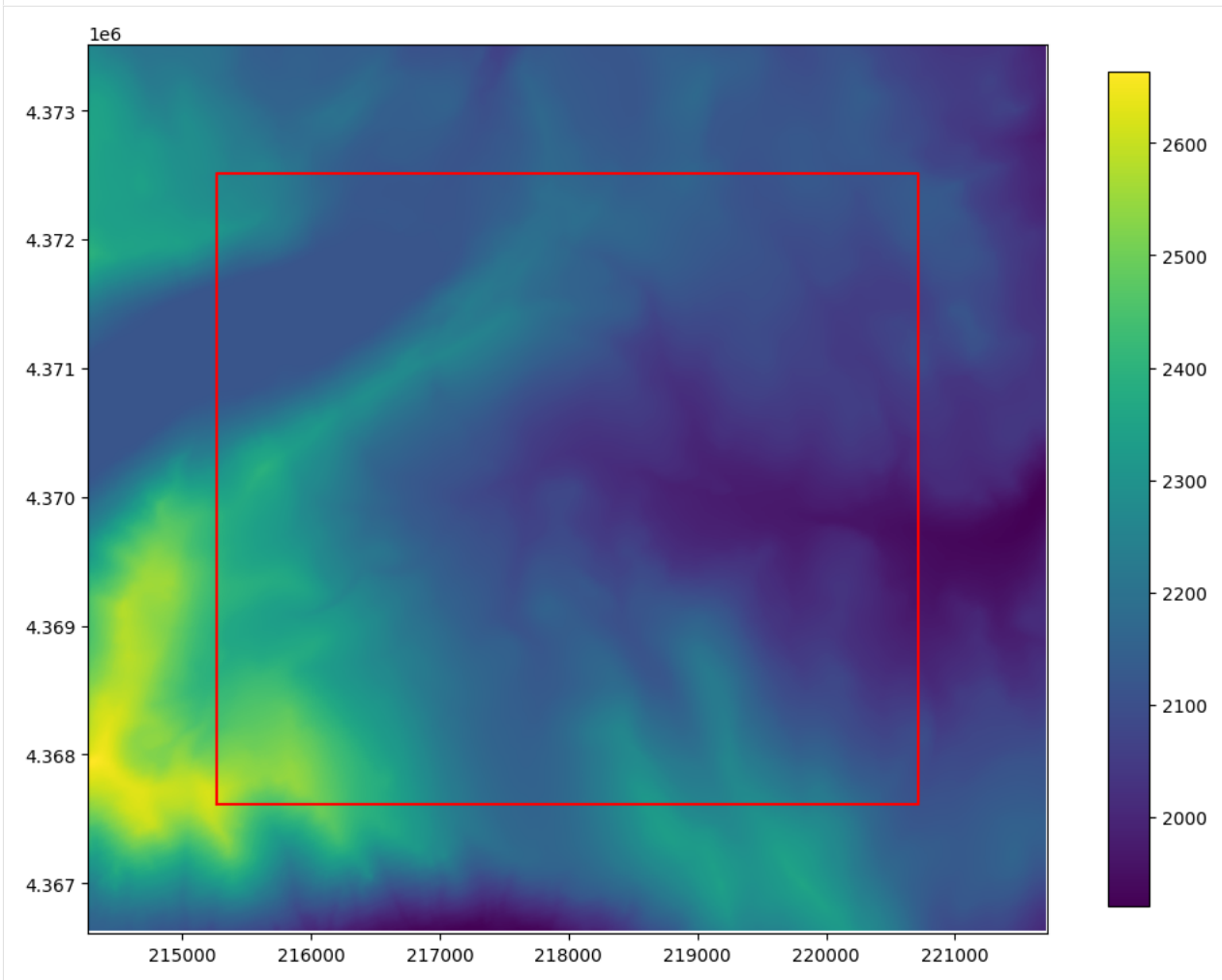
```
y0 += 1000
x1 -= 1000
y1 -= 1000

shape = np.array([(x0, y0), (x0, y1), (x1, y1), (x1, y0), (x0, y0)])
```

```
[28]: fig = plt.figure(figsize=(12, 12))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")

      ax = rio.plot(ax=ax, vmin=vmin, vmax=vmax)
      ax.plot(shape.T[0], shape.T[1], "r-")
      plt.colorbar(ax.images[0], shrink=0.7)
```

```
[28]: <matplotlib.colorbar.Colorbar at 0x7f980bcd88e0>
```



```
[29]: data = rio.sample_polygon(shape, band=rio.bands[0])

      mean = np.mean(data)
      dmin = np.min(data)
      dmax = np.max(data)
```

(continues on next page)

(continued from previous page)

```
stdv = np.std(data)

s = "Minimum elevation: {:.2f}\nMaximum elevation: {:.2f}\nMean elevation: {:.2f}\nStandard deviation: {:.2f}"
print(s.format(dmin, dmax, mean, stdv))

Minimum elevation: 1942.17
Maximum elevation: 2608.56
Mean elevation: 2151.67
Standard deviation: 113.60
```

Cropping and resampling to a modelgrid

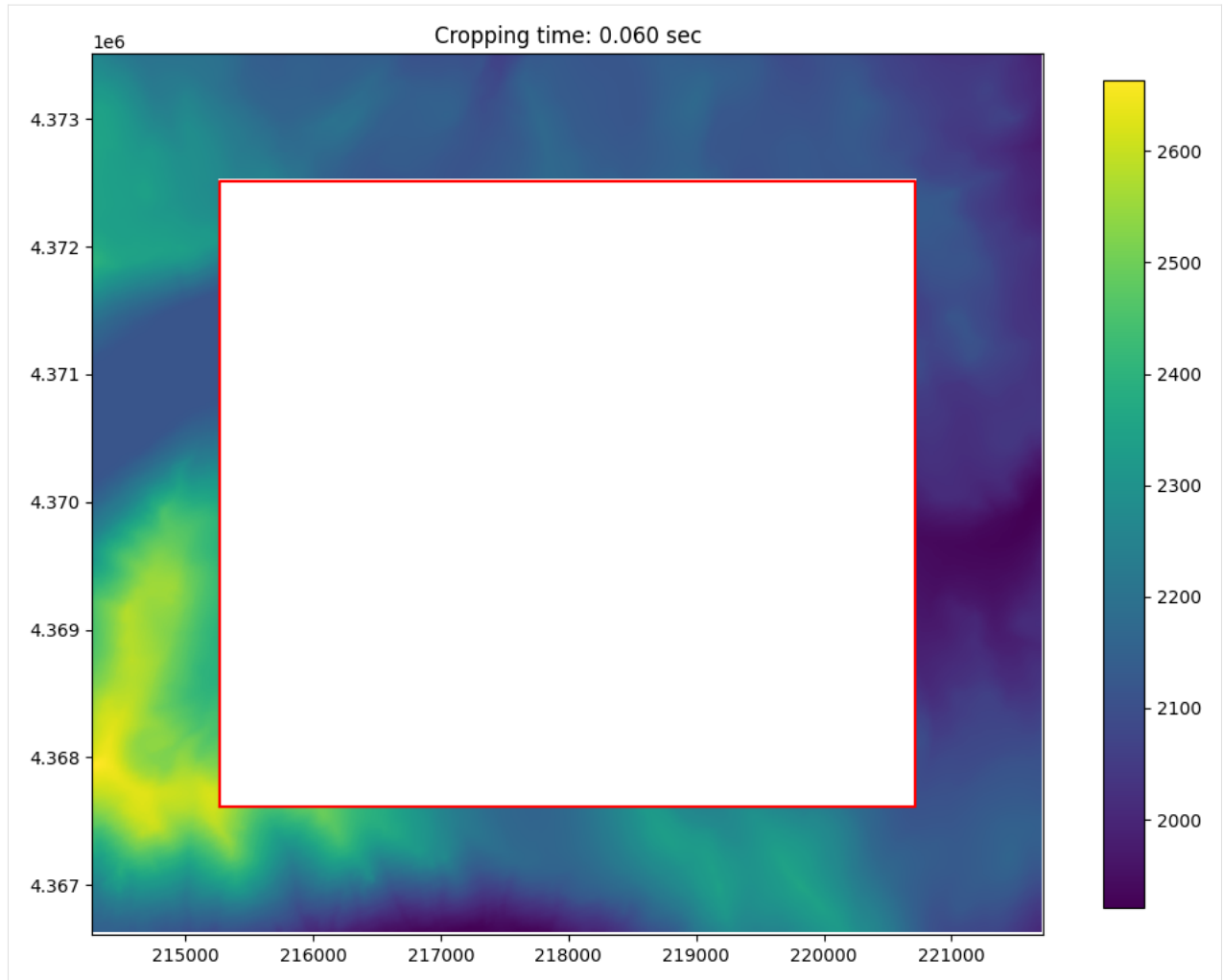
The `crop()` method can accept a list or `np.array` of vertices, a shapely Polygon object, or a GeoJSON dictionary. The crop can also be inverted, using `invert=True`.

```
[30]: t0 = time.time()
      rio.crop(shape, invert=True)
      crop_time = time.time() - t0

[31]: fig = plt.figure(figsize=(12, 12))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")

      ax = rio.plot(ax=ax, vmin=vmin, vmax=vmax)
      ax.plot(shape.T[0], shape.T[1], "r-")
      plt.title("Cropping time: {:.3f} sec".format(crop_time))
      plt.colorbar(ax.images[0], shrink=0.7)

[31]: <matplotlib.colorbar.Colorbar at 0x7f980bd8e3d0>
```



And then this can be re-sampled to a ModelGrid Object

```
[32]: t0 = time.time()
dem_data = rio.resample_to_grid(
    mg_unstruct, band=rio.bands[0], method="nearest"
)

resample_time = time.time() - t0

[33]: # now to visualize using fropy and matplotlib
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

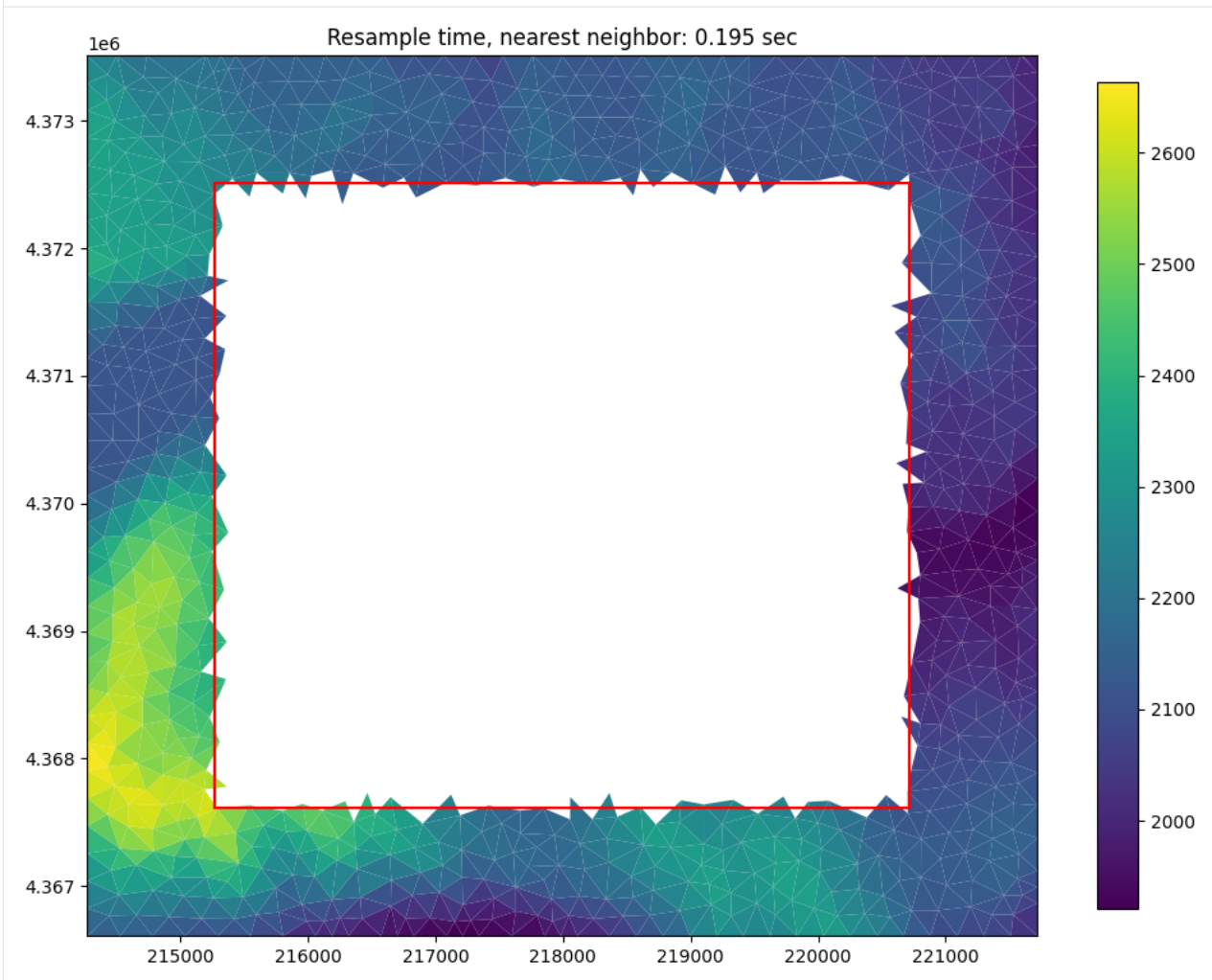
pmv = fropy.plot.PlotMapView(modelgrid=mg_unstruct, ax=ax)
ax = pmv.plot_array(
    dem_data,
    masked_values=rio.nodatavals,
    cmap="viridis",
    vmin=vmin,
    vmax=vmax,
```

(continues on next page)

(continued from previous page)

```
)
plt.plot(shape.T[0], shape.T[1], "r-")
plt.title("Resample time, nearest neighbor: {:.3f} sec".format(resample_time))
plt.colorbar(ax, shrink=0.7)
```

[33]: <matplotlib.colorbar.Colorbar at 0x7f980ba7bf70>



```
[34]: t0 = time.time()
dem_data = rio.resample_to_grid(
    mg_unstruct, band=rio.bands[0], method="linear"
)

resample_time = time.time() - t0
```

```
[35]: # now to visualize using flopy and matplotlib
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

pmv = flopy.plot.PlotMapView(modelgrid=mg_unstruct, ax=ax)
ax = pmv.plot_array(
```

(continues on next page)

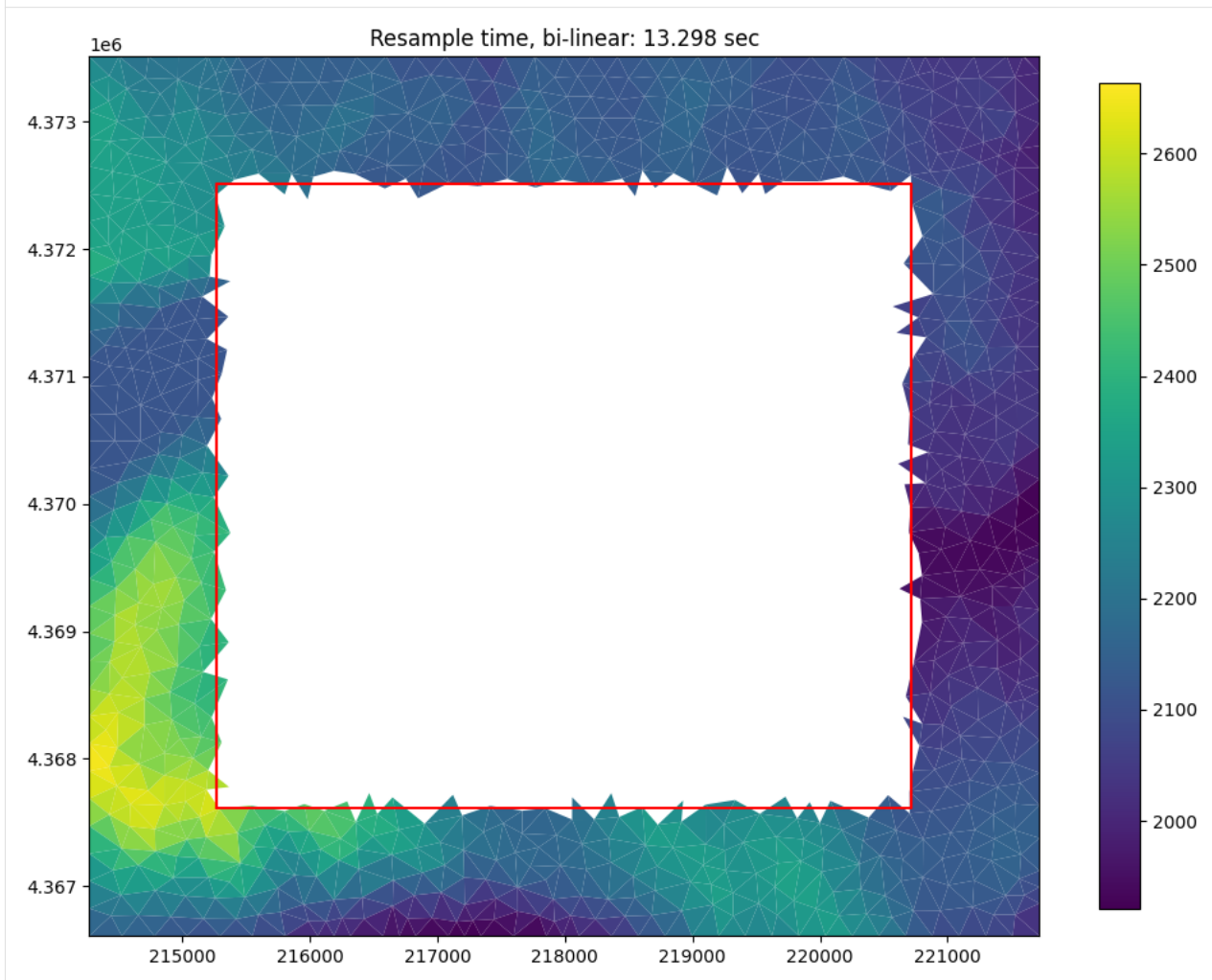
(continued from previous page)

```

dem_data,
masked_values=rio.nodatavals,
cmap="viridis",
vmin=vmin,
vmax=vmax,
)
plt.plot(shape.T[0], shape.T[1], "r-")
plt.title("Resample time, bi-linear: {:.3f} sec".format(resample_time))
plt.colorbar(ax, shrink=0.7)

```

[35]: <matplotlib.colorbar.Colorbar at 0x7f980b963a90>



Arbitrary-shaped model boundaries

In the example pyshp and shapely are used to get geometry information and then we create a top array and an ibound array using that geometry information

First let's reload the raster (since operations are done in-place) and then load our shapefile data

```
[36]: rio = Raster.load(os.path.join(raster_ws, raster_name))

shp_name = os.path.join(raster_ws, "model_boundary.shp")

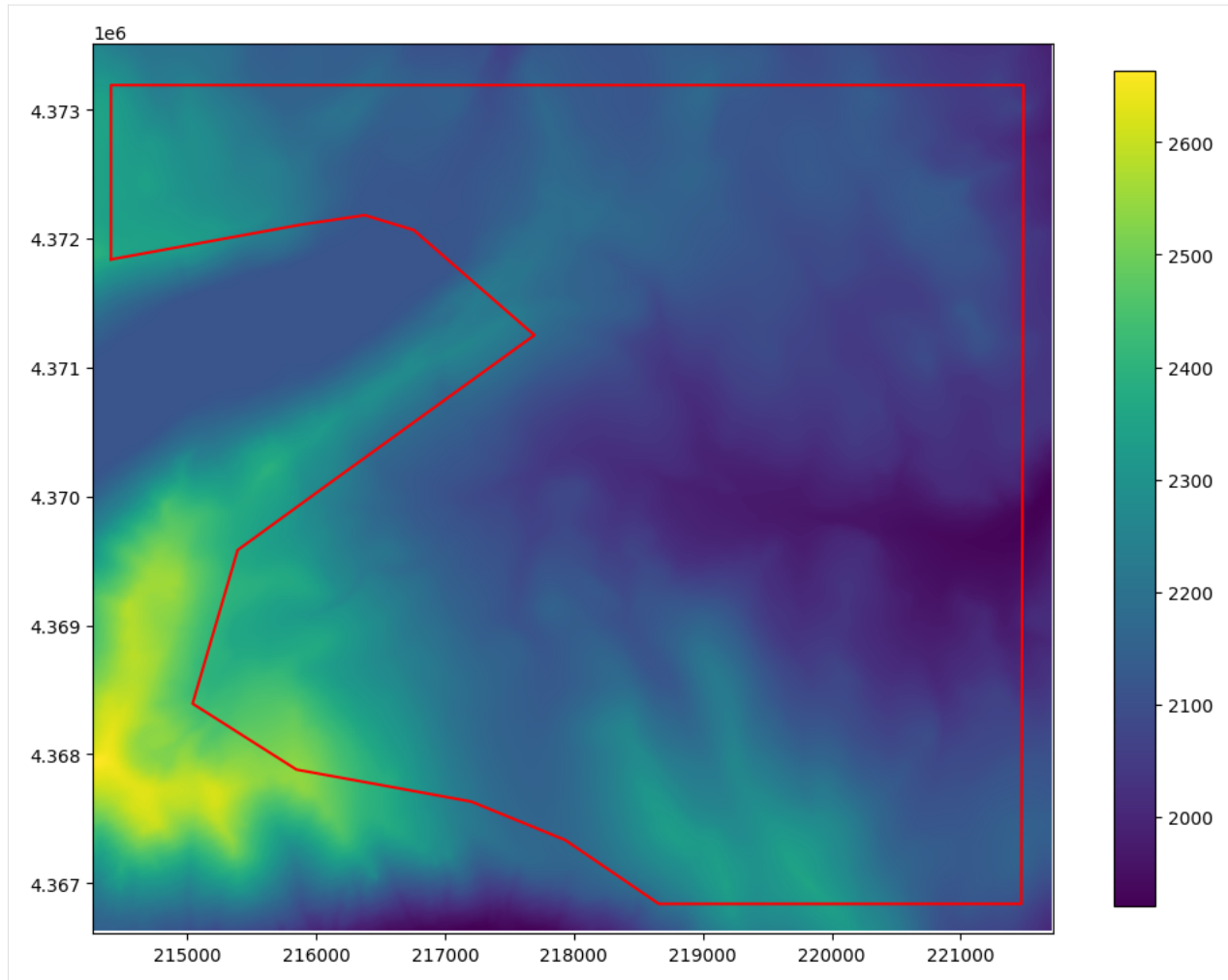
# read in the shapefile
sf = shapefile.Reader(shp_name)
shapes = sf.shapes()
```

```
[37]: fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

ax = rio.plot(ax=ax, vmin=vmin, vmax=vmax)

# plot the shapes for visualization
for shp in shapes:
    shp = np.array(shp.points).T
    plt.plot(shp[0], shp[1], "r-")
plt.colorbar(ax.images[0], shrink=0.7)
```

```
[37]: <matplotlib.colorbar.Colorbar at 0x7f980b8cb8e0>
```

Now we can apply an intersection using the point data directly from the shapefile class

```
[38]: polygon = shapes[0].points
```

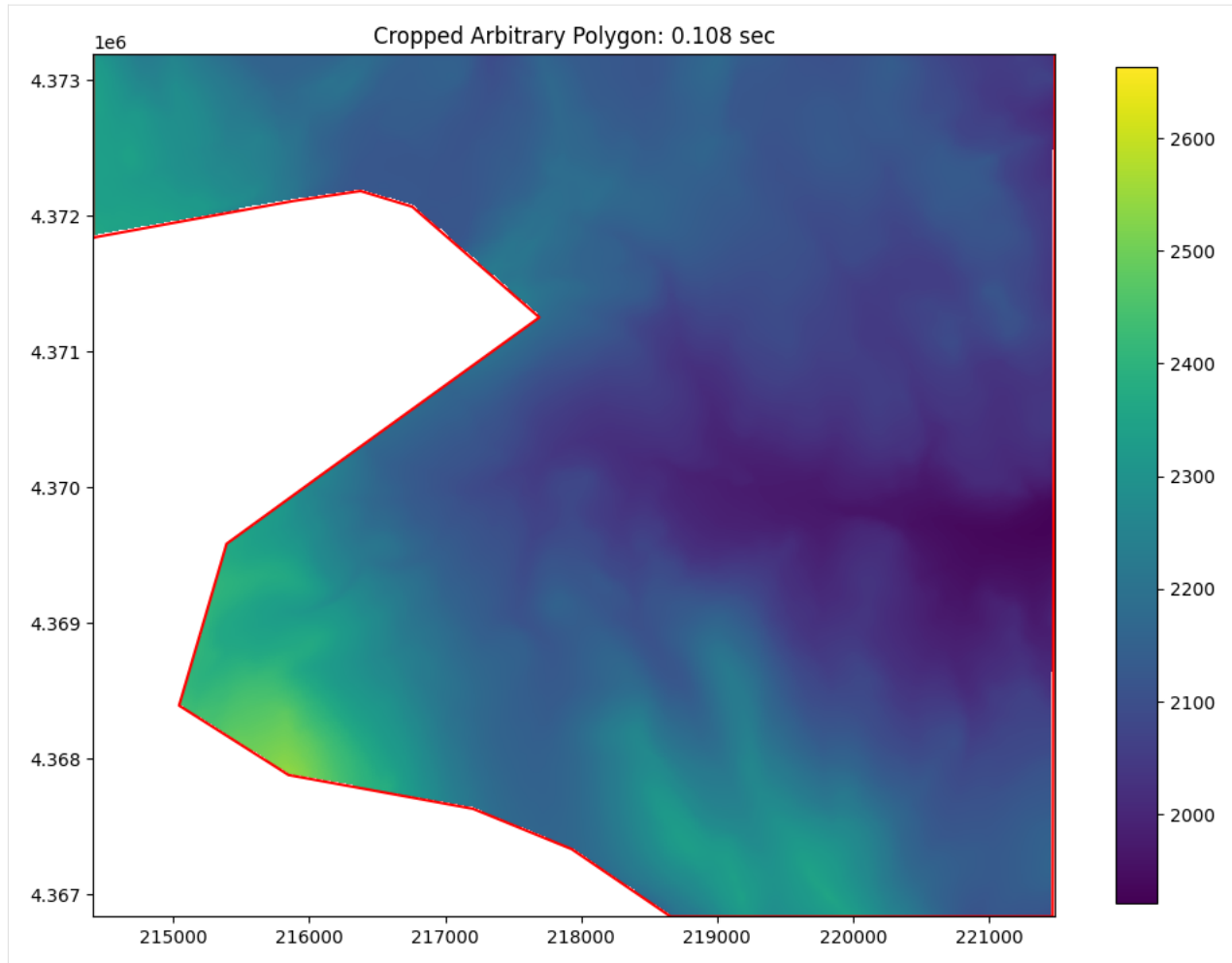
```
t0 = time.time()
rio.crop(polygon)
crop_time = time.time() - t0
```

```
[39]: fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

ax = rio.plot(ax=ax, vmin=vmin, vmax=vmax)

shape = np.array(polygon).T
plt.plot(shape[0], shape[1], "r-")
plt.title("Cropped Arbitrary Polygon: {:.3f} sec".format(crop_time))
plt.colorbar(ax.images[0], shrink=0.7)
```

```
[39]: <matplotlib.colorbar.Colorbar at 0x7f981fe577f0>
```



Now the data can be re-sampled to the modelgrid

```
[40]: top = rio.resample_to_grid(mg_unstruct, band=rio.bands[0], method="linear")
```

```
# apply a "realistic" nodataval to top cells outside the model domain
```

```
for val in rio.nodatavals:
```

```
    top[top == val] = 3500
```

```
# create an ibound array
```

```
ibound = np.ones(top.shape, dtype=int)
```

```
ibound[top == 3500] = 0
```

```
[41]: # now to visualize using flopy and matplotlib
```

```
fig = plt.figure(figsize=(12, 12))
```

```
ax = fig.add_subplot(1, 1, 1, aspect="equal")
```

```
pmv = flopy.plot.PlotMapView(modelgrid=mg_unstruct, ax=ax)
```

```
ax = pmv.plot_array(
```

```
    top,
```

```
    masked_values=[
```

```
        3500,
```

(continues on next page)

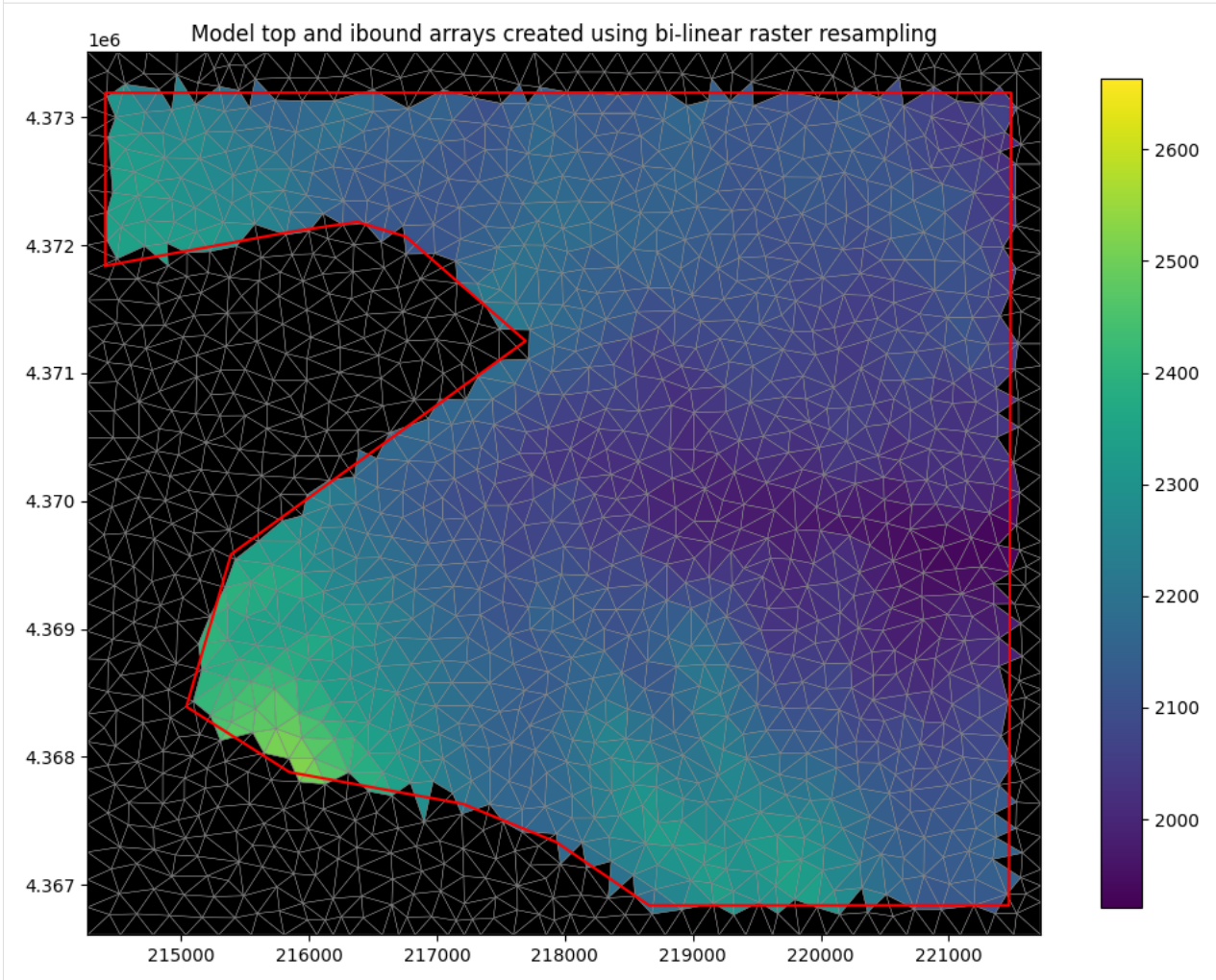
(continued from previous page)

```

],
cmap="viridis",
vmin=vmin,
vmax=vmax,
)
ib = pmv.plot_ibound(ibound)
pmv.plot_grid(linewidth=0.3)
plt.plot(shape[0], shape[1], "r-")
plt.title(
    "Model top and ibound arrays created using bi-linear raster resampling"
)
plt.colorbar(ax, shrink=0.7)

```

[41]: <matplotlib.colorbar.Colorbar at 0x7f980b2e9940>



The ibound array and the top array can be used to build or edit the BAS and DIS file objects in FloPy

Future development

Potential features that draw on this functionality could include: + intersection with multiple polygons + flow accumulation to develop SFR networks + streambed topology from raster layers + intersection with layers of derived parameters based on multiple raster bands

```
[42]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.2 Postprocessing and Visualization

6.2.1 Plotting SWR Process Results

This notebook demonstrates the use of the `SwrObs` and `SwrStage`, `SwrBudget`, `SwrFlow`, and `SwrExchange`, `SwrStructure`, classes to read binary SWR Process observation, stage, budget, reach to reach flows, reach-aquifer exchange, and structure files. It demonstrates these capabilities by loading these binary file types and showing examples of plotting SWR Process data. An example showing how the simulated water surface profile at a selected time along a selection of reaches can be plotted is also presented.

```
[1]: import os
     import sys
```

```
[2]: import matplotlib as mpl
     import matplotlib.pyplot as plt
     import numpy as np
```

```
[3]: from IPython.display import Image

     # run installed version of flopy or add local path
     try:
         import flopy
     except:
         fpth = os.path.abspath(os.path.join("..", ".."))
         sys.path.append(fpth)
         import flopy

     print(sys.version)
     print("numpy version: {}".format(np.__version__))
     print("matplotlib version: {}".format(mpl.__version__))
     print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[4]: # Set the paths
datapth = os.path.join("../", "..", "examples", "data", "swr_test")

# SWR Process binary files
files = ("SWR004.obs", "SWR004.vel", "SWR004.str", "SWR004.stg", "SWR004.flow")
```

Load SWR Process observations

Create an instance of the SwrObs class and load the observation data.

```
[5]: subj = flopy.utils.SwrObs(os.path.join(datapth, files[0]))

ts = subj.get_data()
```

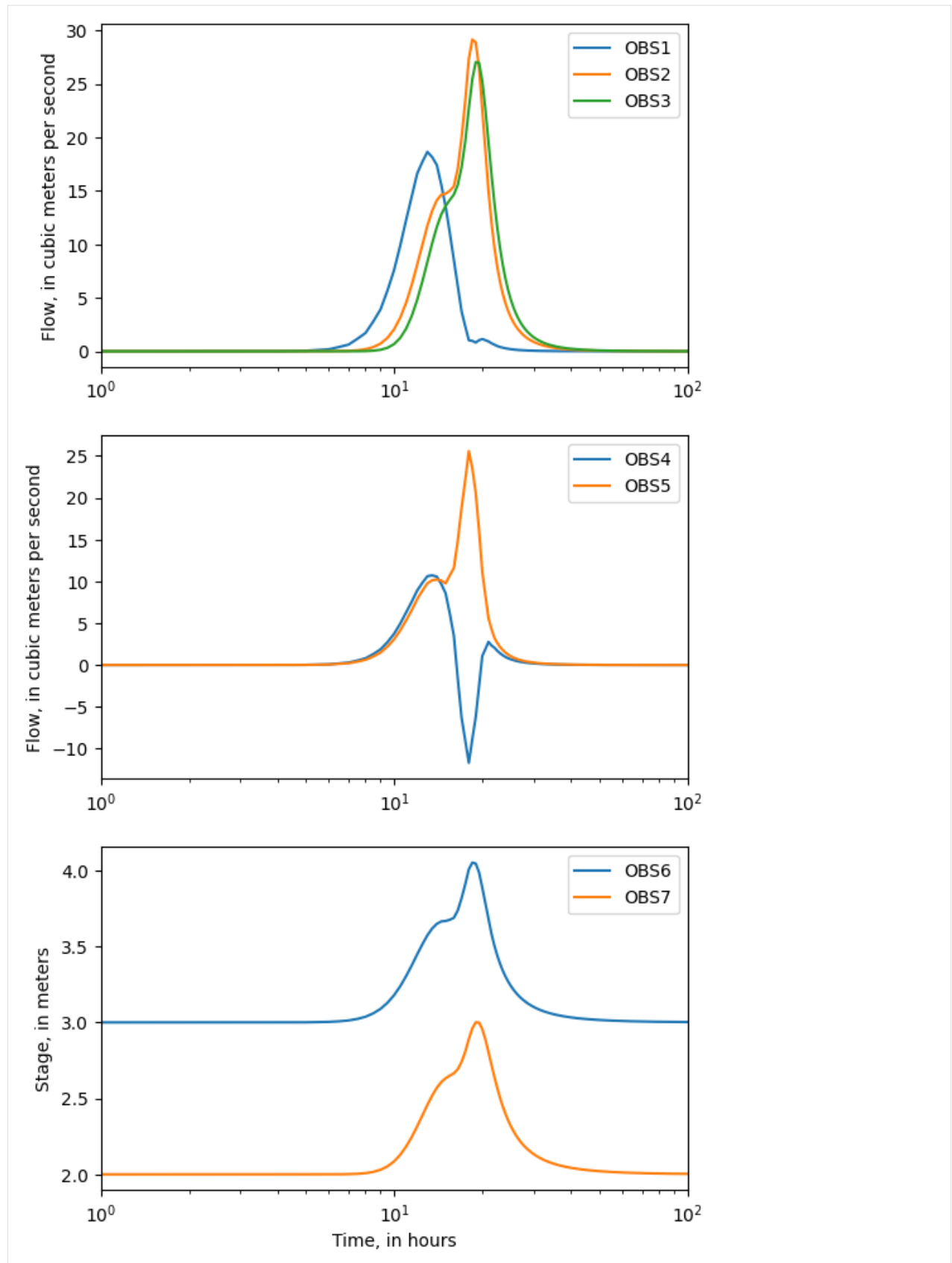
Plot the data from the binary SWR Process observation file

```
[6]: fig = plt.figure(figsize=(6, 12))
ax1 = fig.add_subplot(3, 1, 1)
ax1.semilogx(ts["totim"] / 3600.0, -ts["OBS1"], label="OBS1")
ax1.semilogx(ts["totim"] / 3600.0, -ts["OBS2"], label="OBS2")
ax1.semilogx(ts["totim"] / 3600.0, -ts["OBS9"], label="OBS3")
ax1.set_ylabel("Flow, in cubic meters per second")
ax1.legend()

ax = fig.add_subplot(3, 1, 2, sharex=ax1)
ax.semilogx(ts["totim"] / 3600.0, -ts["OBS4"], label="OBS4")
ax.semilogx(ts["totim"] / 3600.0, -ts["OBS5"], label="OBS5")
ax.set_ylabel("Flow, in cubic meters per second")
ax.legend()

ax = fig.add_subplot(3, 1, 3, sharex=ax1)
ax.semilogx(ts["totim"] / 3600.0, ts["OBS6"], label="OBS6")
ax.semilogx(ts["totim"] / 3600.0, ts["OBS7"], label="OBS7")
ax.set_xlim(1, 100)
ax.set_ylabel("Stage, in meters")
ax.set_xlabel("Time, in hours")
ax.legend()

[6]: <matplotlib.legend.Legend at 0x7fd91bfcfb80>
```



Load the same data from the individual binary SWR Process files

Load discharge data from the flow file. The flow file contains the simulated flow between connected reaches for each connection in the model.

```
[7]: subj = flopy.utils.SwrFlow(os.path.join(datapth, files[1]))
times = np.array(subj.get_times()) / 3600.0
obs1 = subj.get_ts(irec=1, iconn=0)
obs2 = subj.get_ts(irec=14, iconn=13)
obs4 = subj.get_ts(irec=4, iconn=3)
obs5 = subj.get_ts(irec=5, iconn=4)
```

Load discharge data from the structure file. The structure file contains the simulated structure flow for each reach with a structure.

```
[8]: subj = flopy.utils.SwrStructure(os.path.join(datapth, files[2]))
obs3 = subj.get_ts(irec=17, istr=0)
```

Load stage data from the stage file. The flow file contains the simulated stage for each reach in the model.

```
[9]: subj = flopy.utils.SwrStage(os.path.join(datapth, files[3]))
obs6 = subj.get_ts(irec=13)
```

Load budget data from the budget file. The budget file contains the simulated budget for each reach group in the model. The budget file also contains the stage data for each reach group. In this case the number of reach groups equals the number of reaches in the model.

```
[10]: subj = flopy.utils.SwrBudget(os.path.join(datapth, files[4]))
obs7 = subj.get_ts(irec=17)
```

Plot the data loaded from the individual binary SWR Process files.

Note that the plots are identical to the plots generated from the binary SWR observation data.

```
[11]: fig = plt.figure(figsize=(6, 12))
ax1 = fig.add_subplot(3, 1, 1)
ax1.semilogx(times, obs1["flow"], label="OBS1")
ax1.semilogx(times, obs2["flow"], label="OBS2")
ax1.semilogx(times, -obs3["strflow"], label="OBS3")
ax1.set_ylabel("Flow, in cubic meters per second")
ax1.legend()

ax = fig.add_subplot(3, 1, 2, sharex=ax1)
ax.semilogx(times, obs4["flow"], label="OBS4")
ax.semilogx(times, obs5["flow"], label="OBS5")
ax.set_ylabel("Flow, in cubic meters per second")
ax.legend()

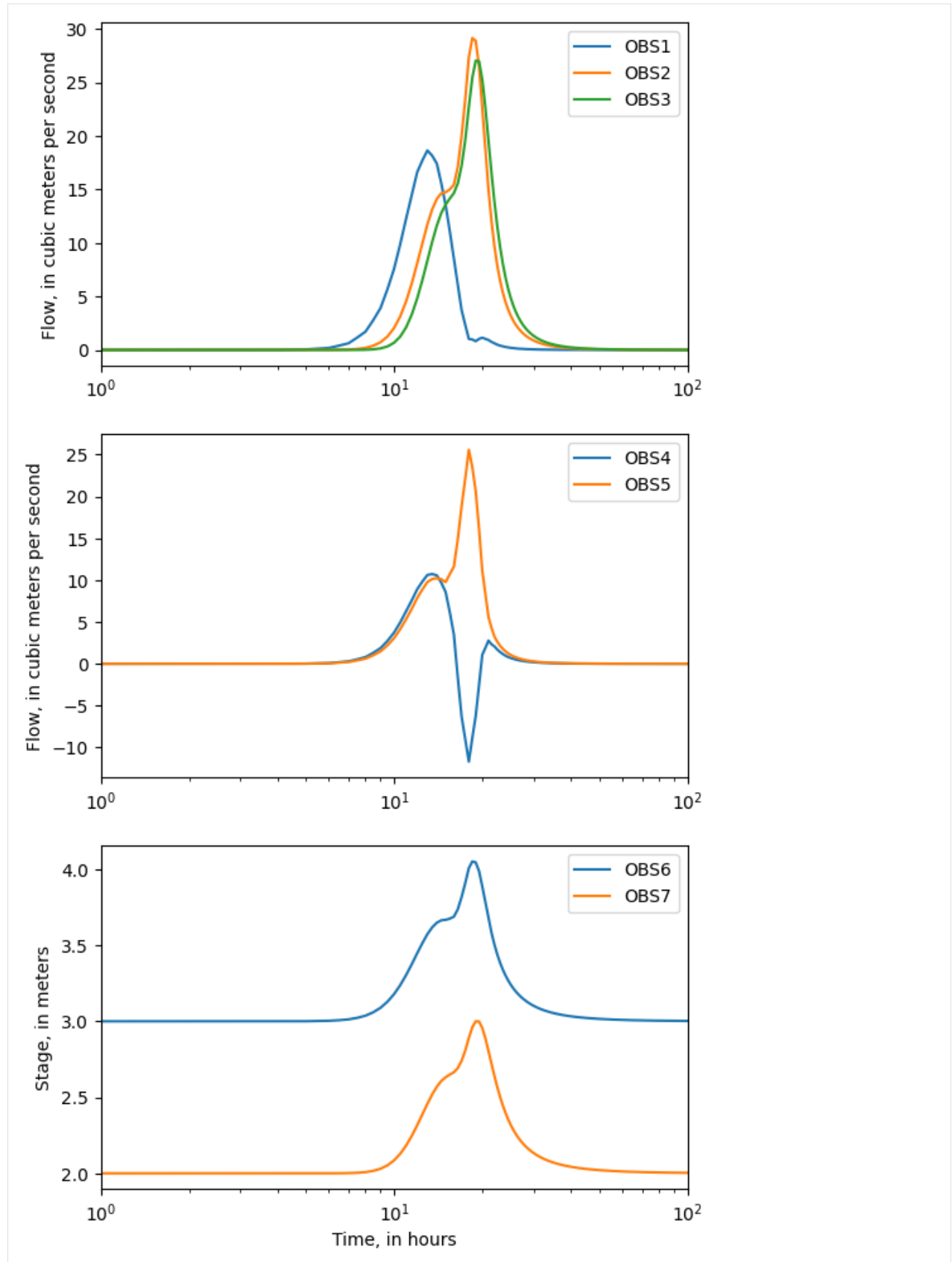
ax = fig.add_subplot(3, 1, 3, sharex=ax1)
ax.semilogx(times, obs6["stage"], label="OBS6")
ax.semilogx(times, obs7["stage"], label="OBS7")
ax.set_xlim(1, 100)
ax.set_ylabel("Stage, in meters")
```

(continues on next page)

(continued from previous page)

```
ax.set_xlabel("Time, in hours")  
ax.legend()
```

```
[11]: <matplotlib.legend.Legend at 0x7fd918723c10>
```

Plot simulated water surface profiles

Simulated water surface profiles can be created using the `ModelCrossSection` class.

Several things that we need in addition to the stage data include reach lengths and bottom elevations. We load these data from an existing file.

```
[12]: sd = np.genfromtxt(os.path.join(datapth, "SWR004.dis.ref"), names=True)
```

The contents of the file are shown in the cell below.

```
[13]: fc = open(os.path.join(datapth, "SWR004.dis.ref")).readlines()
fc
```

```
[13]: ['  IRCH  RLEN  BELEV\n',
'      1 1000.    2.0\n',
'      2  500.    1.83\n',
'      3  500.    1.83\n',
'      4 1000.    1.67\n',
'      5 1000.    1.50\n',
'      6 1000.    1.33\n',
'      7 1000.    1.17\n',
'      8  500.    1.0\n',
'      9  500.    1.83\n',
'     10 1118.    1.65\n',
'     11 1118.    1.41\n',
'     12 1000.    1.17\n',
'     13  500.    1.0\n',
'     14  500.    1.0\n',
'     15 1000.    0.75\n',
'     16 1000.    0.50\n',
'     17 1000.    0.25\n',
'     18 1000.    0.0']
```

Create an instance of the `SwrStage` class for SWR Process stage data.

```
[14]: sobj = flopy.utils.SwrStage(os.path.join(datapth, files[3]))
```

Create a selection condition (`iprof`) that can be used to extract data for the reaches of interest (reaches 0, 1, and 8 through 17). Use this selection condition to extract reach lengths (from `sd['RLEN']`) and the bottom elevation (from `sd['BELEV']`) for the reaches of interest. The selection condition will also be used to extract the stage data for reaches of interest.

```
[15]: iprof = sd["IRCH"] > 0
iprof[2:8] = False
dx = np.extract(iprof, sd["RLEN"])
belev = np.extract(iprof, sd["BELEV"])
```

Create a fake model instance so that the `ModelCrossSection` class can be used.

```
[16]: ml = flopy.modflow.Modflow()
dis = flopy.modflow.ModflowDis(
    ml,
    nrow=1,
    ncol=dx.shape[0],
```

(continues on next page)

(continued from previous page)

```

    delr=dx,
    top=4.5,
    botm=belev.reshape(1, 1, 12),
)

```

Create an array with the x position at the downstream end of each reach, which will be used to color the plots below each reach.

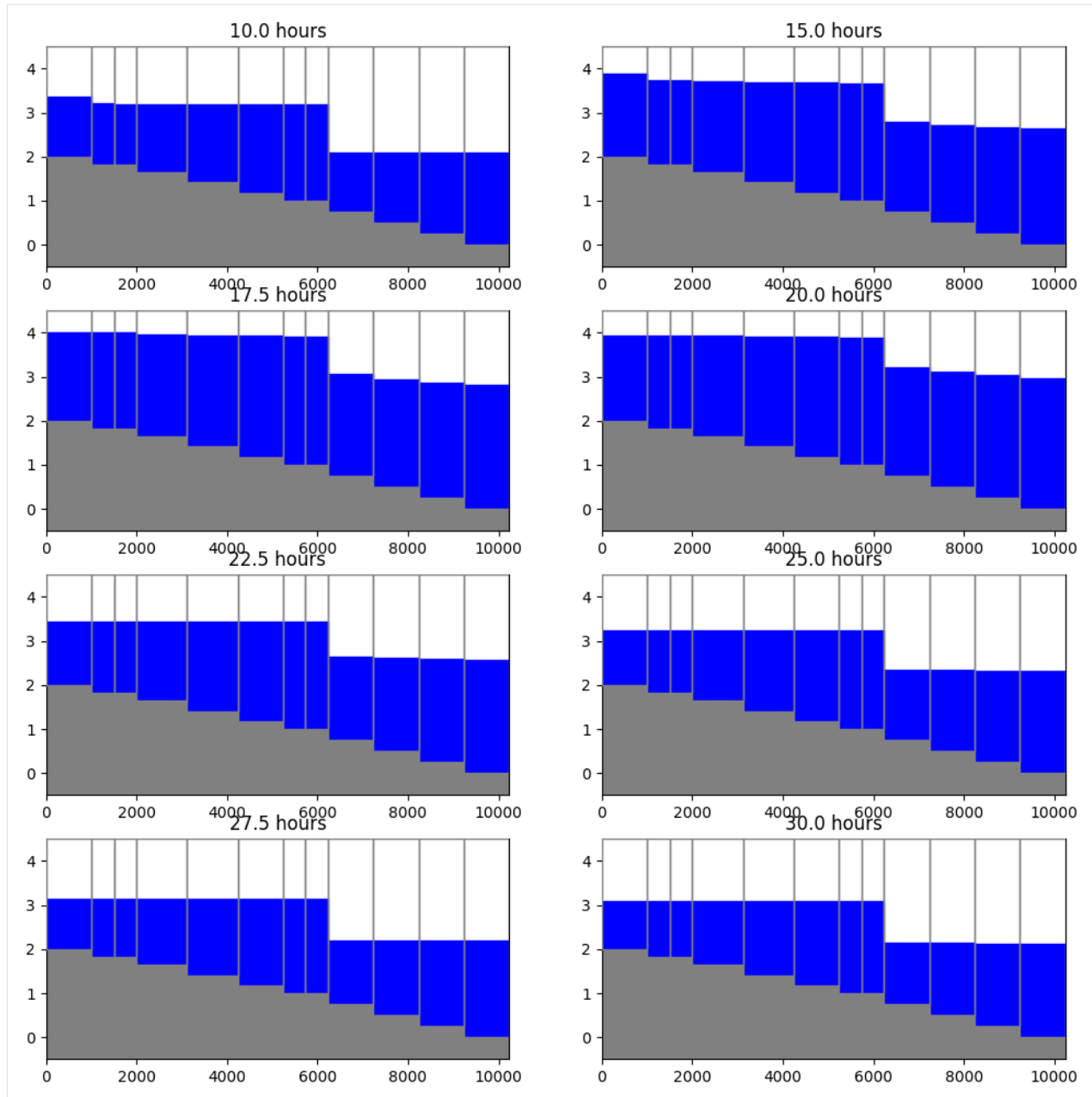
```
[17]: x = np.cumsum(dx)
```

Plot simulated water surface profiles for 8 times.

```

[18]: fig = plt.figure(figsize=(12, 12))
    for idx, v in enumerate([19, 29, 34, 39, 44, 49, 54, 59]):
        ax = fig.add_subplot(4, 2, idx + 1)
        s = sobj.get_data(idx=v)
        stage = np.extract(iprof, s["stage"])
        xs = flopy.plot.PlotCrossSection(model=m1, line={"Row": 0})
        xs.plot_fill_between(
            stage.reshape(1, 1, 12),
            colors=["none", "blue"],
            ax=ax,
            edgecolors="none",
        )
        linecollection = xs.plot_grid(ax=ax, zorder=10)
        ax.fill_between(
            np.append(0.0, x),
            y1=np.append(belev[0], belev),
            y2=-0.5,
            facecolor="0.5",
            edgecolor="none",
            step="pre",
        )
        ax.set_title("{} hours".format(times[v]))
        ax.set_ylim(-0.5, 4.5)

```



This notebook demonstrates flopy functionality for reading binary output generated by the SWR Process. Binary files that can be read include observations, stages, budgets, flow, reach-aquifer exchanges, and structure data. The binary stage data can also be used to create water-surface profiles.

Hope this gets you started!

6.2.2 Plotting Model Arrays and Results

This notebook demonstrates the simple array and results plotting capabilities of flopy. It demonstrates these capabilities by loading and running an existing model, and then showing how the `.plot()` method can be used to make simple plots of the model data and model results.

```
[1]: import os
import sys
from tempfile import TemporaryDirectory
```

```
[2]: import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
```

```
[3]: from IPython.display import Image

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[4]: # Set name of MODFLOW exe
# assumes executable is in users path statement
version = "mf2005"
exe_name = "mf2005"

# Set the paths
loadpth = os.path.join("..", "..", "examples", "data", "secp")

# temporary directory
temp_dir = TemporaryDirectory()
modelpth = temp_dir.name

# make sure modelpth directory exists
if not os.path.isdir(modelpth):
    os.makedirs(modelpth, exist_ok=True)

files = ["secp.hds"]
```

Load and Run an Existing Model

A model called the “Southeast Coastal Plain Model” is located in the loadpth folder. In the following code block, we load that model, then change into a new workspace (modelpth) where we recreate and run the model. For this to work properly, the MODFLOW-2005 executable (mf2005) must be in the path. We verify that it worked correctly by checking for the presence of secp.hds.

```
[5]: ml = flopy.modflow.Modflow.load(
    "secp.nam", model_ws=loadpth, exe_name=exe_name, version=version
)
ml.change_model_ws(new_pth=modelpth)
ml.write_input()

success, buff = ml.run_model(silent=True)
if not success:
    print("Something bad happened.")

# confirm that the model files have been created
for f in files:
    if os.path.isfile(os.path.join(modelpth, f)):
        msg = "Output file located: {}".format(f)
        print(msg)
    else:
        errmsg = "Error. Output file cannot be found: {}".format(f)
        print(errmsg)
```

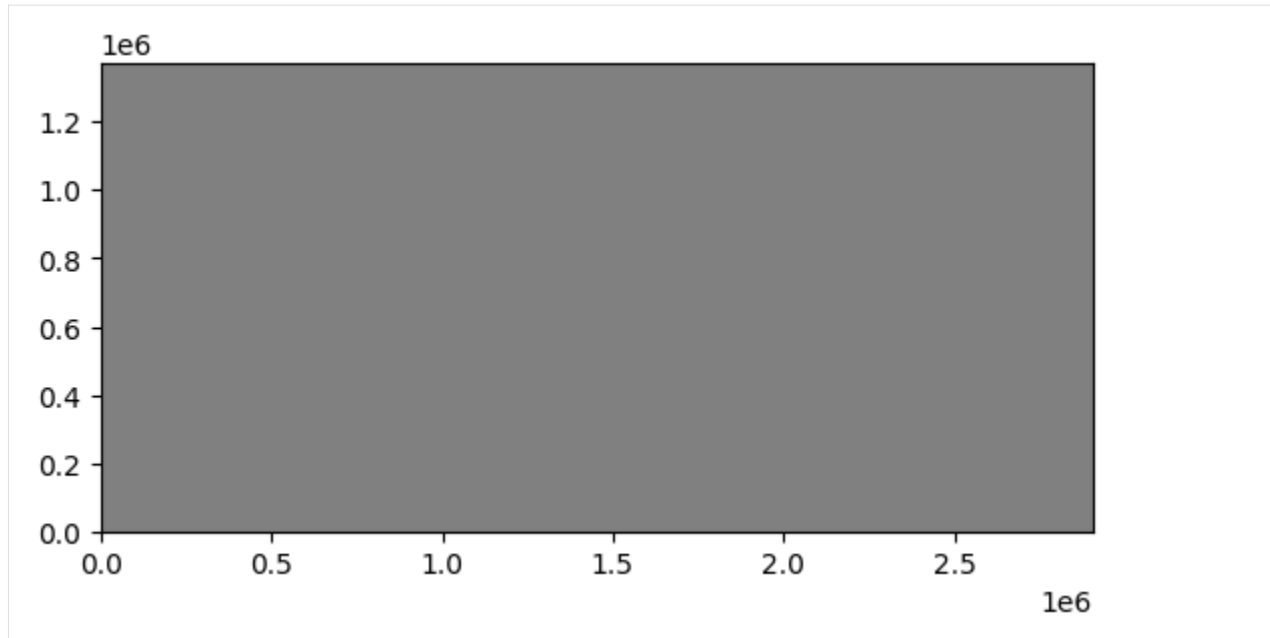
```
Output file located: secp.hds
```

Plotting Model Data

Once a model object is created MODFLOW package data can be plotted using the `.plot()` method.

Two-dimensional data (for example the model top) can be plotted by calling the `.plot()` method for each data array.

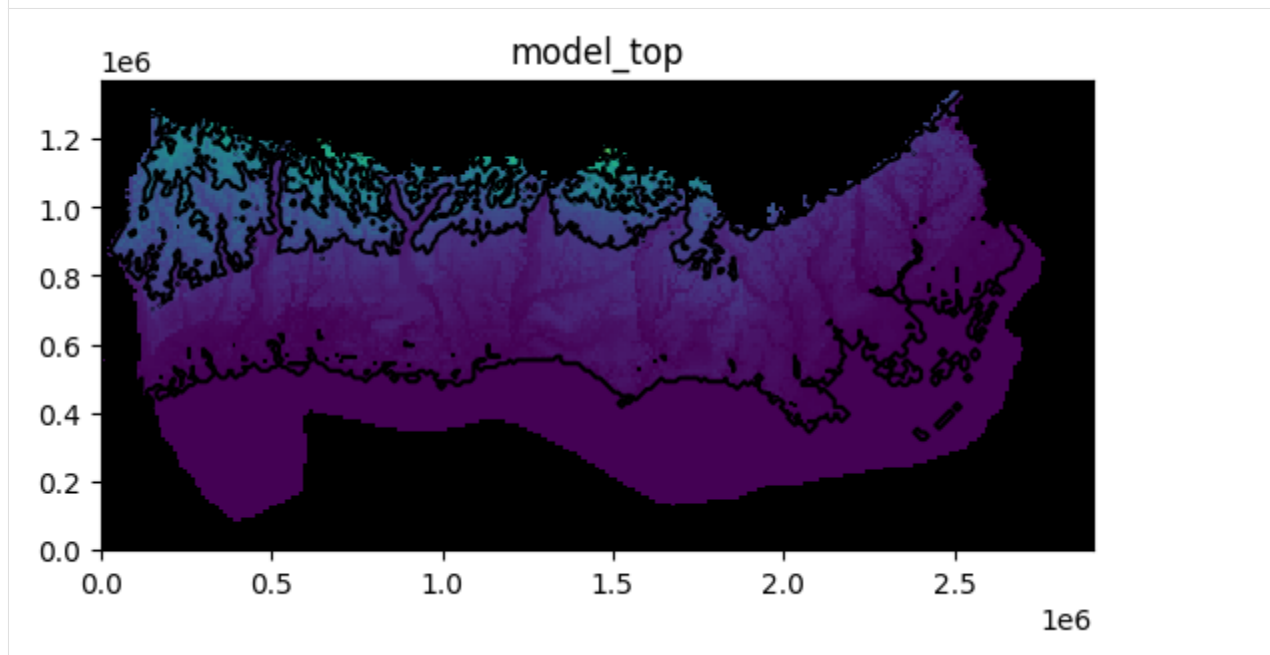
```
[6]: ml.modelgrid.plot()
[6]: <matplotlib.collections.LineCollection at 0x7f76fa864160>
```



As you can see, the `.plot()` methods returns a `matplotlib.pyplot` axis object, which can be used to add additional data to the figure. Below we will add black contours to the axis returned in the first line.

```
[7]: ax = ml.dis.top.plot()
ml.dis.top.plot(axes=ax, contour=True, pcolor=False)
```

```
[7]: <Axes: title={'center': 'model_top'}>
```



<Figure size 640x480 with 0 Axes>

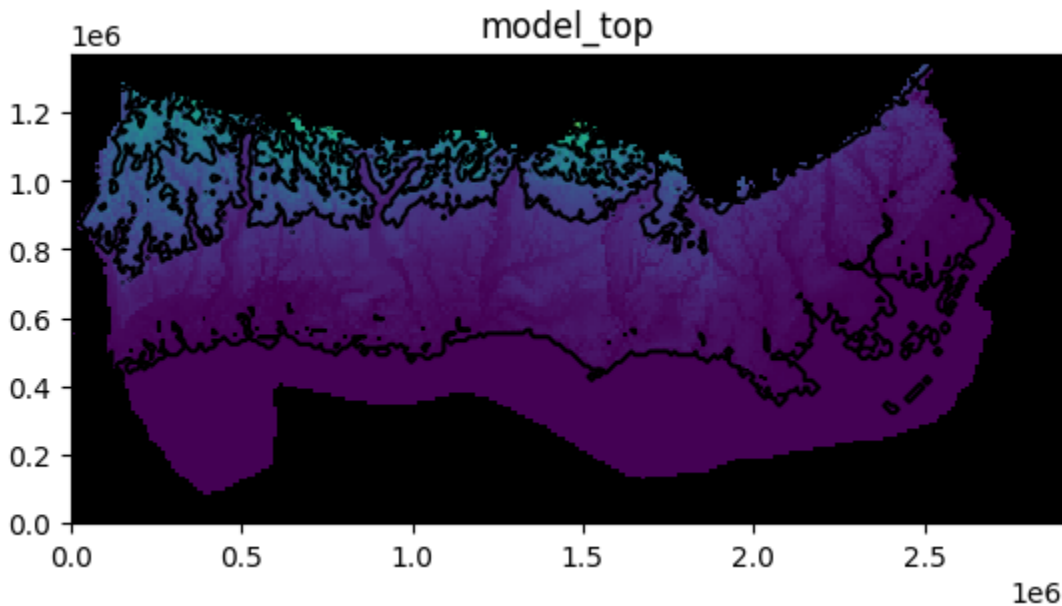
You will notice that we passed several keywords in the second line. There are a number of keywords that can be passed to the `.plot()` method to control plotting. Available keywords are:

1. `axes` - if you already have plot axes you can pass them to the method

2. `pcolor` - turns `pcolor` on if `pcolor=True` or off if `pcolor=False`, default is `pcolor=True`
3. `colorbar` - turns on colorbar if `colorbar=True` or off if `colorbar=False`, default is `colorbar=False` and is only used if `pcolor=True`
4. `inactive` - turns on a black inactive cell overlay if `inactive=True` or turns off the inactive cell overlay if `inactive=False`, default is `inactive=True`
5. `contour` - turns on contours if `contour=True` or off if `contour=False`, default is `contour=False`
6. `clabel` - turns on contour labels if `clabel=True` or off if `clabel=False`, default is `clabel=False` and is only used if `contour=True`
7. `grid` - turns on model grid if `grid=True` or off if `grid=False`, default is `grid=False`
8. `masked_values` - list with unique values to be excluded from the plot (for example, HNOFLO)
9. `mflay` - for three-dimensional data (for example layer bottoms or simulated heads) `mflay` can be used to plot data for a single layer - note `mflay` is zero-based
10. `kper` - for transient two-dimensional data (for example recharge package data) `kper` can be used to plot data for a single stress period - note `kper` is zero-based
11. `filename_base` - a base file name that will be used to automatically generate file names for two-dimensional, three-dimensional, and transient two-dimensional data, default is `filename_base=None`
12. `file_extension` - valid matplotlib file extension, default is `png` and is only used if `filename_base` is specified
13. `matplotlib.pyplot` keywords are also accepted

The previous code block is recreated in a single line using keywords in the code block below.

```
[8]: ml.dis.top.plot(contour=True)
[8]: <Axes: title={'center': 'model_top'}>
```



We can save the same image to a file.

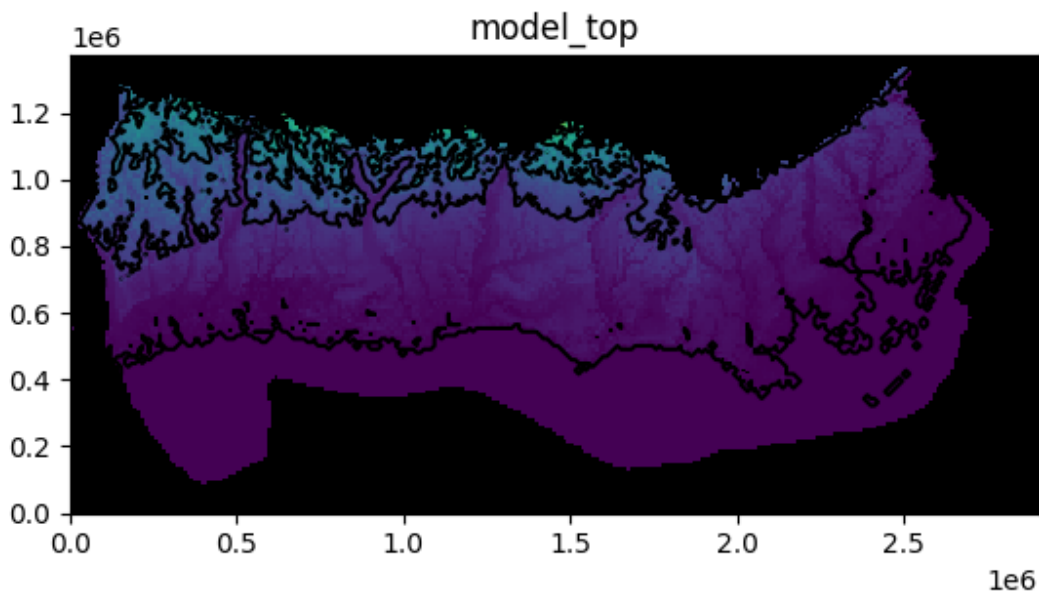

```
[9]: fname = os.path.join(modelpth, "secp")
ml.dis.top.plot(contour=True, filename_base=fname)

created...secp_model_top.png
```

The image file that was just created is shown below

```
[10]: fname = os.path.join(modelpth, "secp_model_top.png")
Image(filename=fname)
```

[10]:



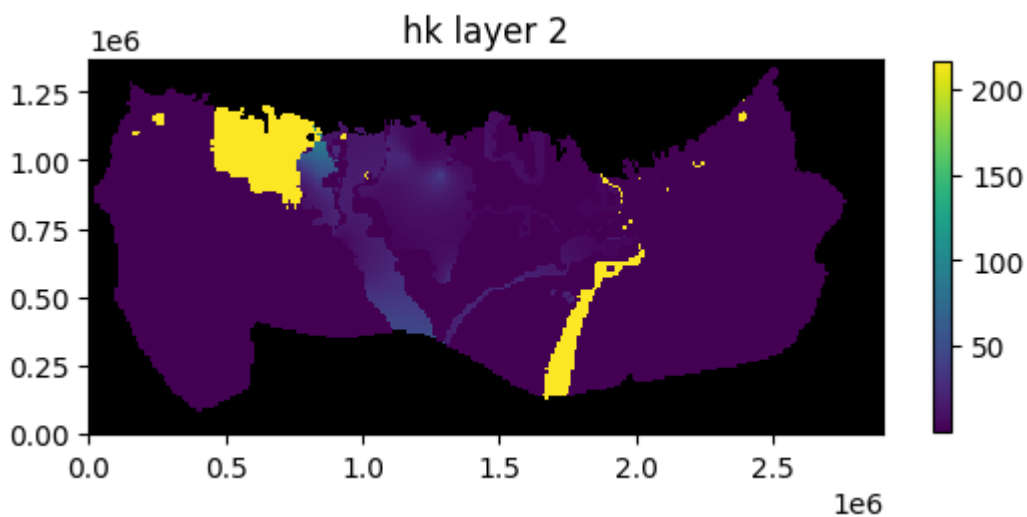
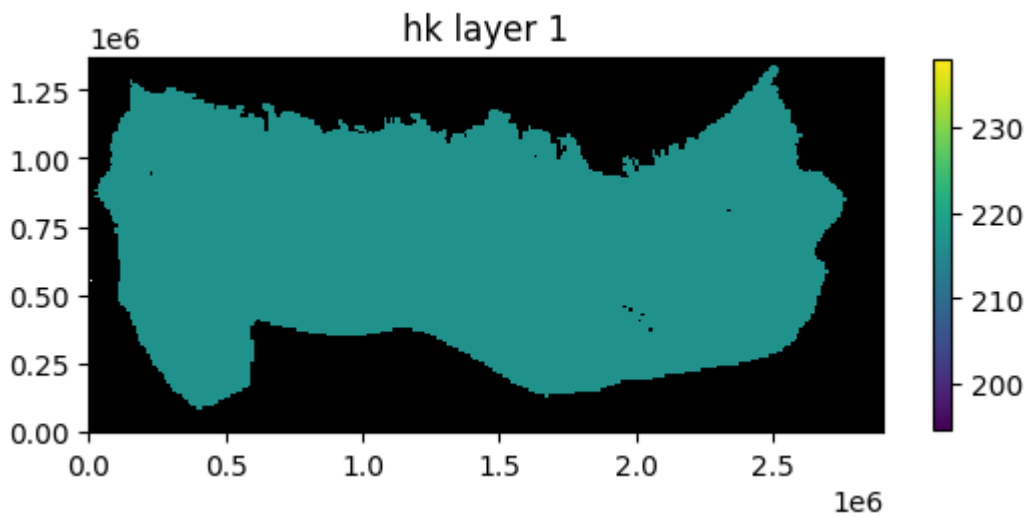
Plotting three-dimensional data

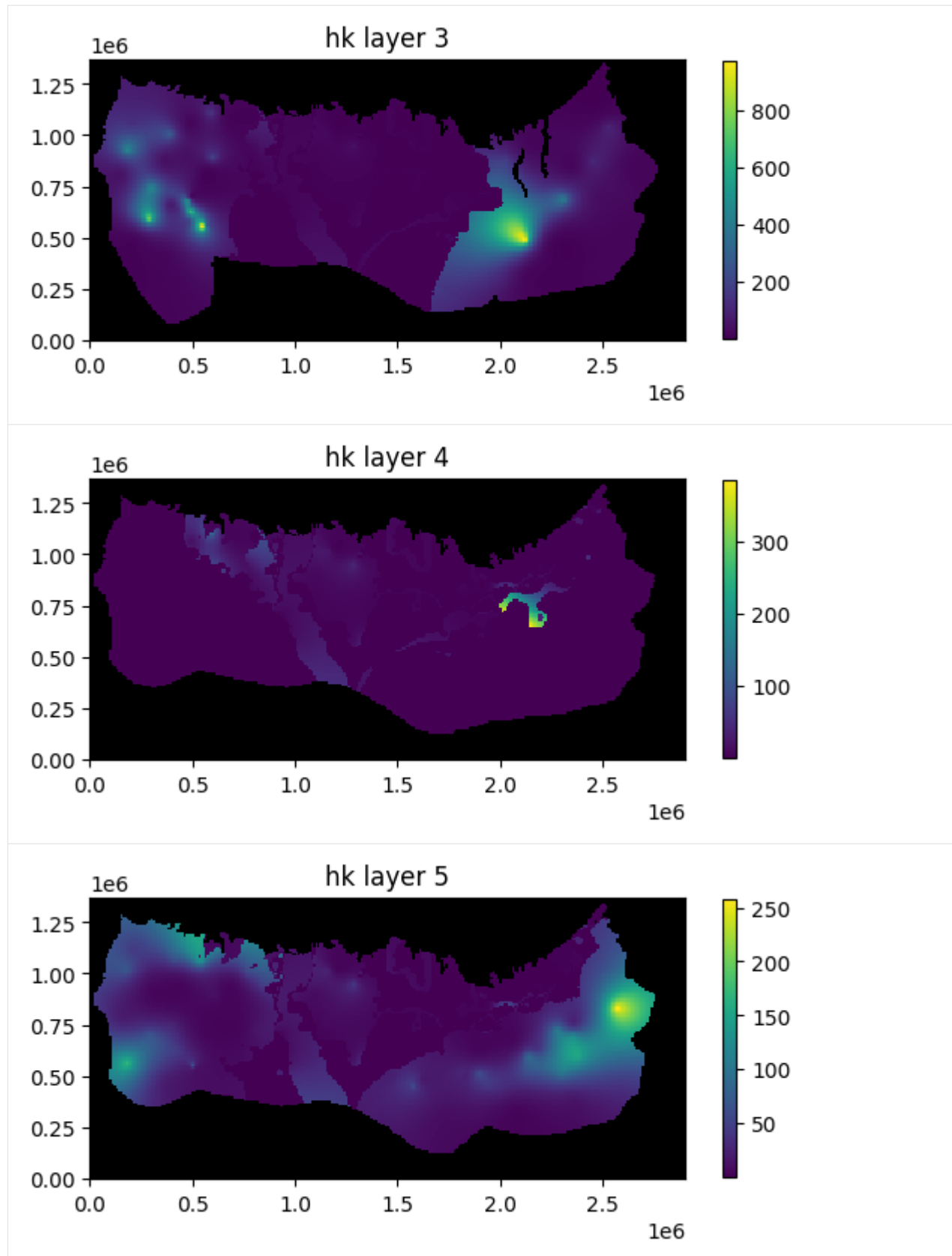
Three-dimensional data can be plotted using the `.plot()` method. User's do not actually need to know that the data are two- or three-dimensional. The `.plot()` method is attached to the two- and three-dimensional data objects so it knows how to process the model data. Examples of three-dimensional data are horizontal hydraulic conductivity (`hk`), layer bottoms (`botm`), specific yield (`sy`), *etc.*

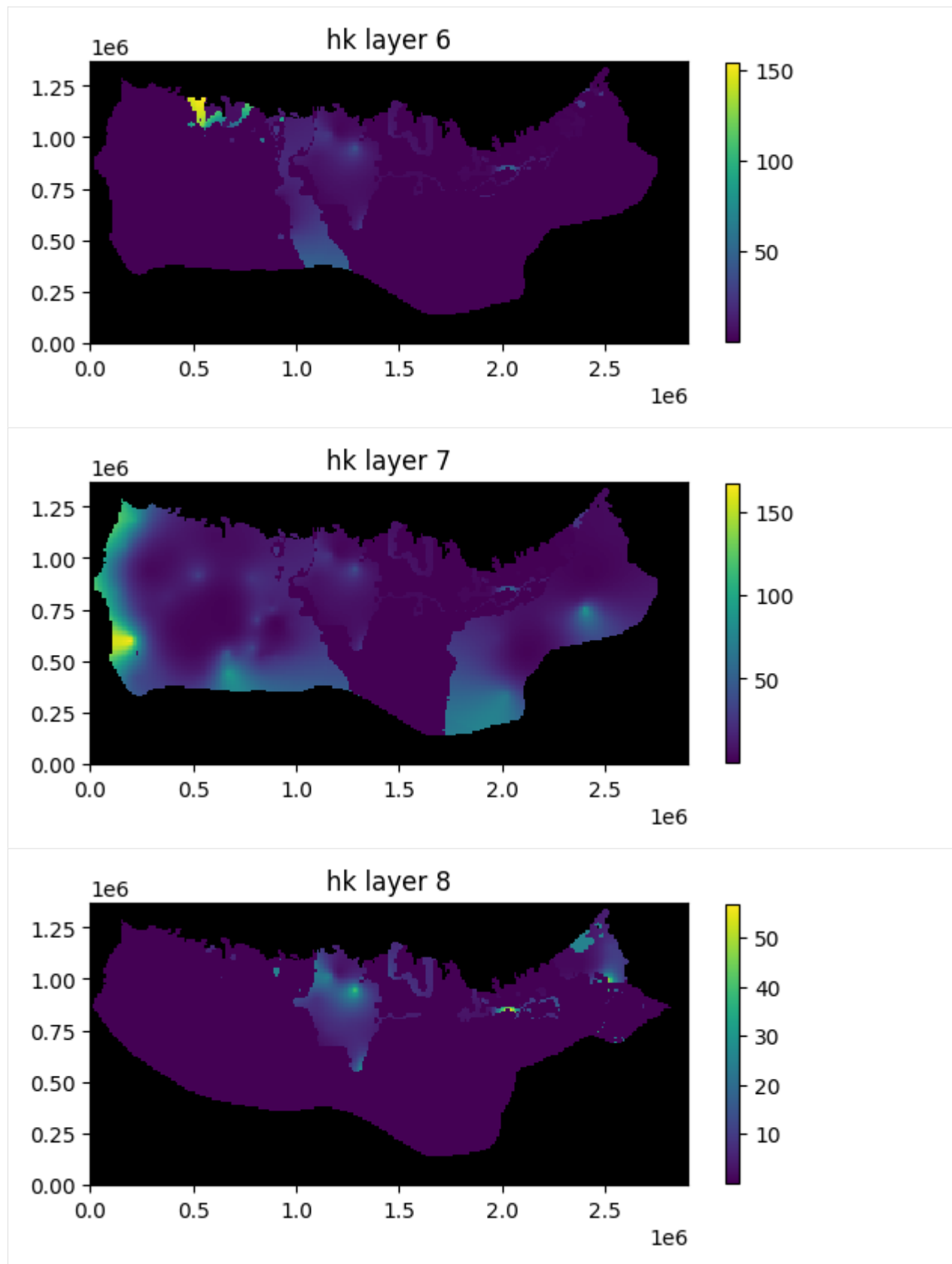
Here we plot the horizontal hydraulic conductivity for each layer. We are also masking areas where the horizontal hydraulic conductivity is zero and adding a color bar.

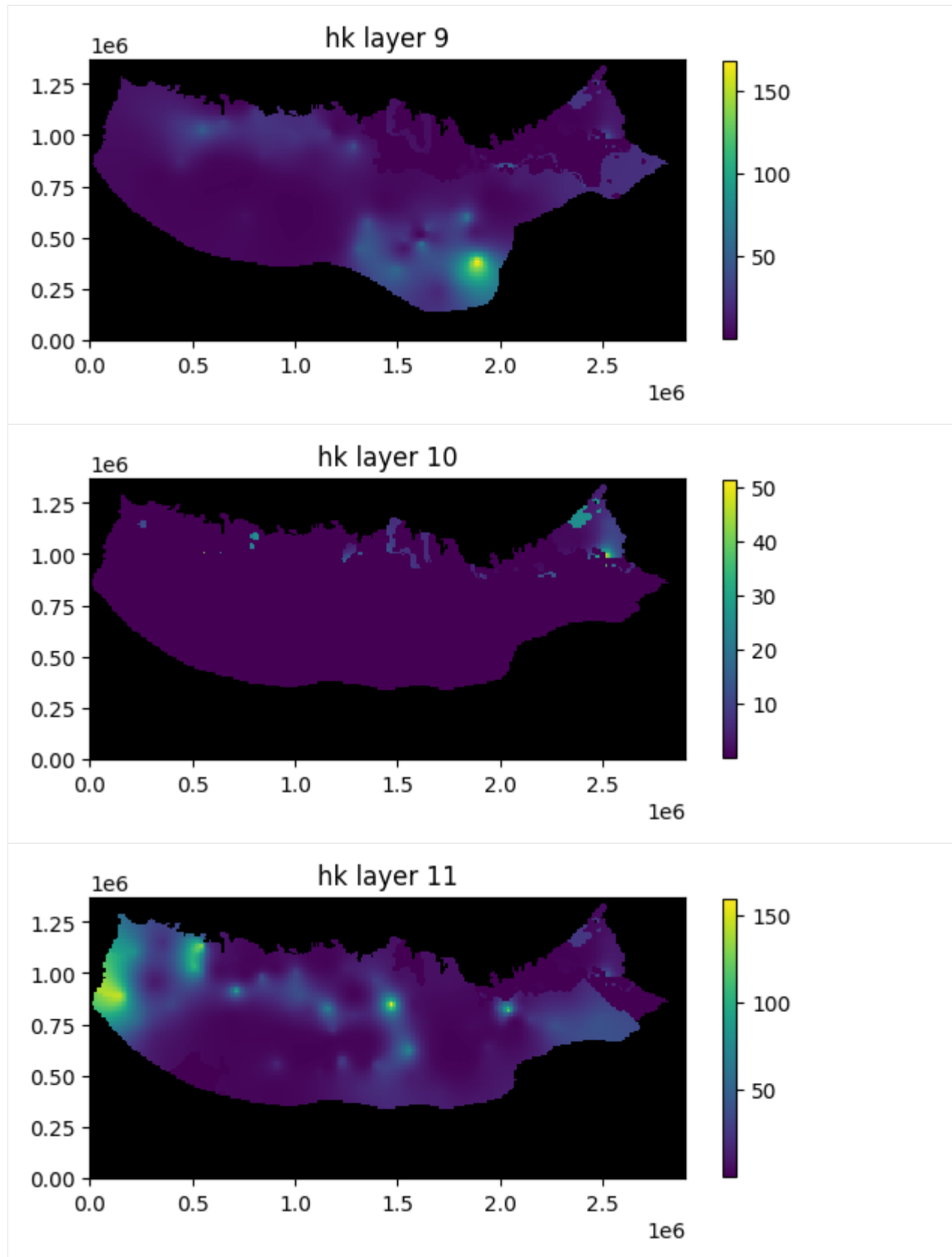
```
[11]: ml.lpf.hk.plot(masked_values=[0.0], colorbar=True)
```

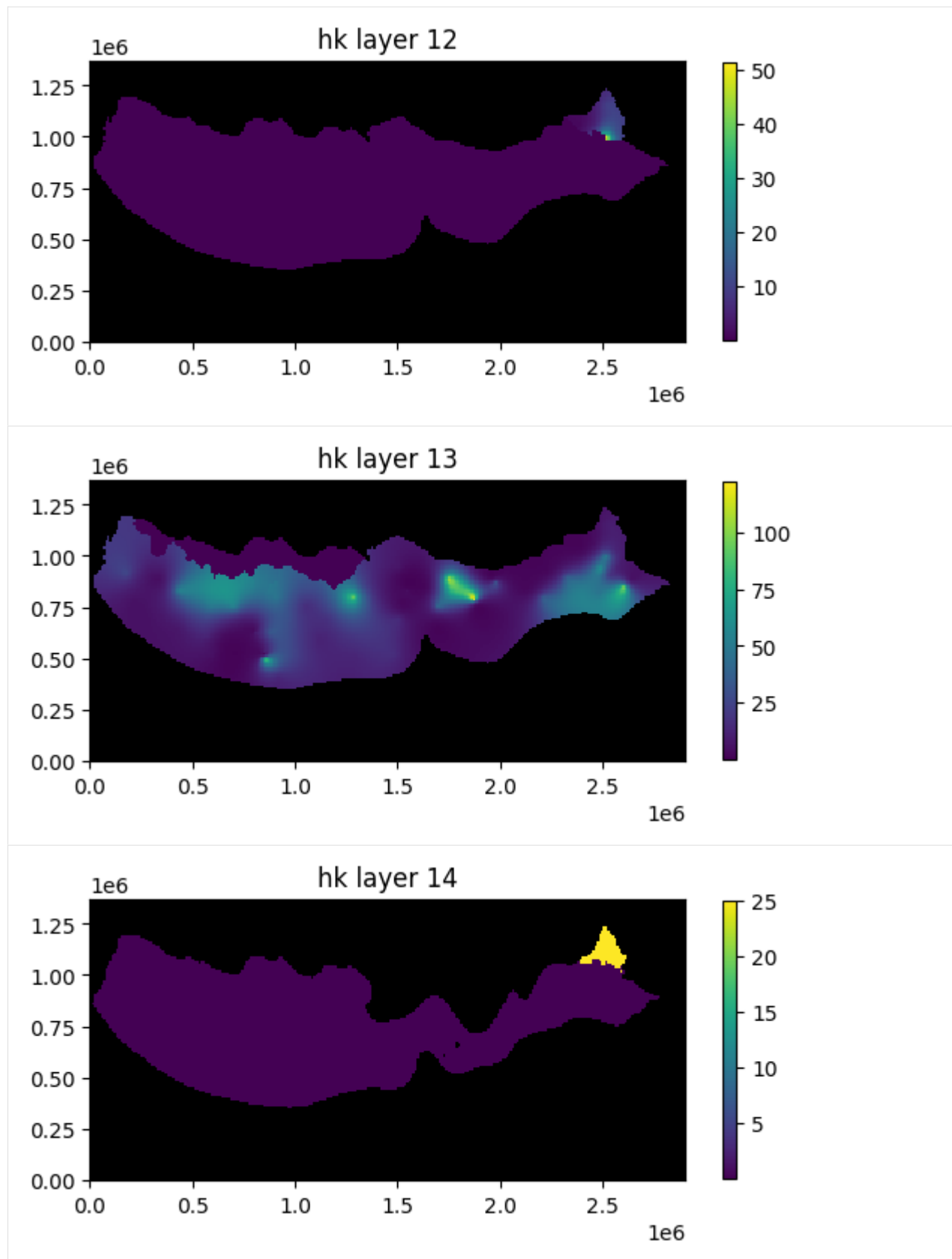
```
[11]: [<Axes: title={'center': 'hk layer 1'}>,  
<Axes: title={'center': 'hk layer 2'}>,  
<Axes: title={'center': 'hk layer 3'}>,  
<Axes: title={'center': 'hk layer 4'}>,  
<Axes: title={'center': 'hk layer 5'}>,  
<Axes: title={'center': 'hk layer 6'}>,  
<Axes: title={'center': 'hk layer 7'}>,  
<Axes: title={'center': 'hk layer 8'}>,  
<Axes: title={'center': 'hk layer 9'}>,  
<Axes: title={'center': 'hk layer 10'}>,  
<Axes: title={'center': 'hk layer 11'}>,  
<Axes: title={'center': 'hk layer 12'}>,  
<Axes: title={'center': 'hk layer 13'}>,  
<Axes: title={'center': 'hk layer 14'}>,  
<Axes: title={'center': 'hk layer 15'}>,  
<Axes: title={'center': 'hk layer 16'}>]
```

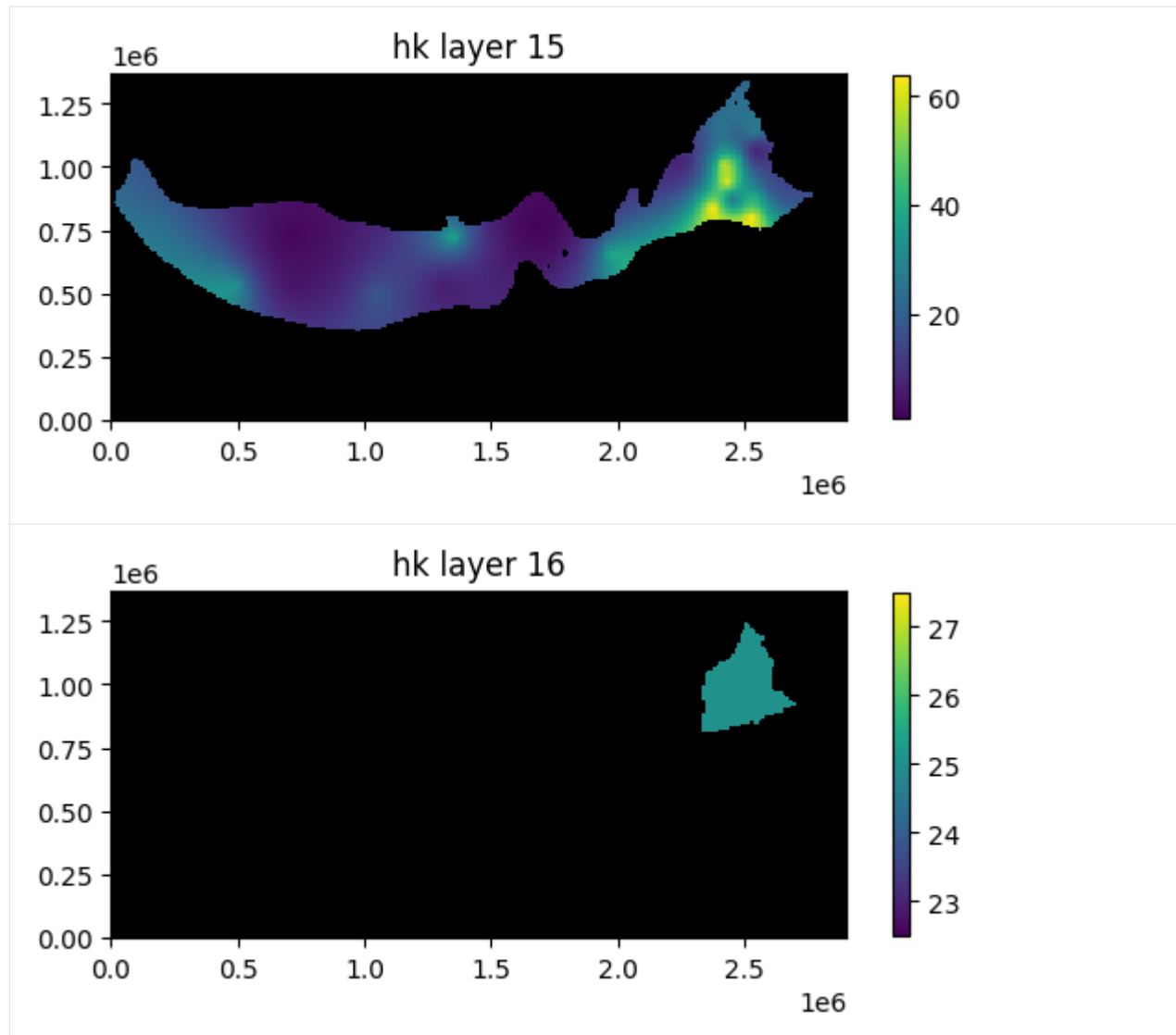












In addition to the plots of horizontal hydraulic conductivity you can see that the `.plot()` method returned a list containing 16 axis objects (one for each layer).

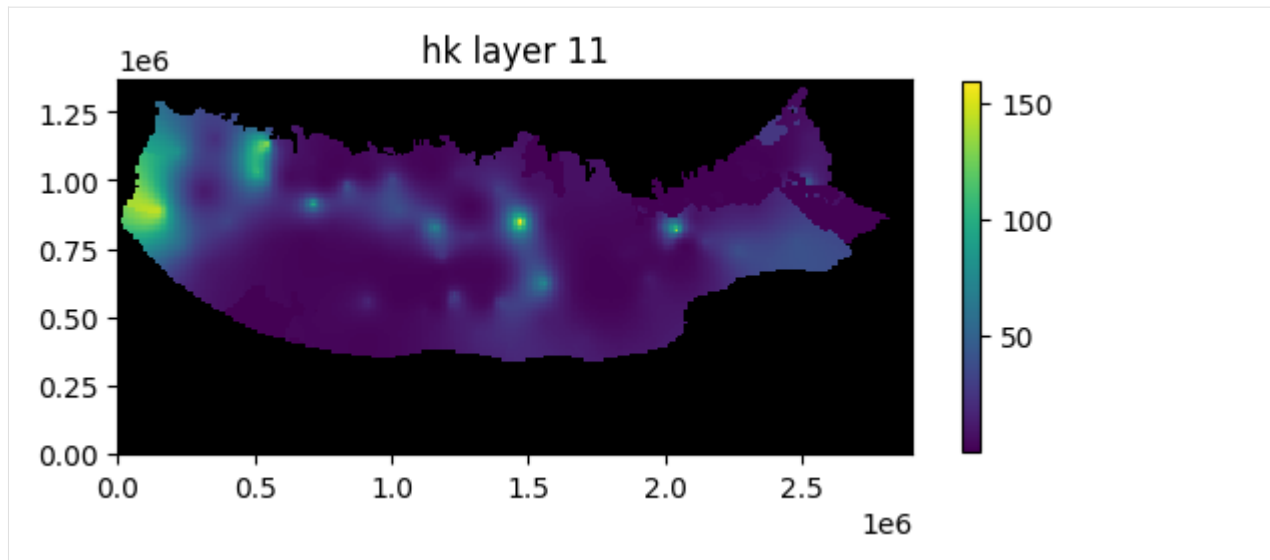
Plotting data for a single layer

If the `mflay=` keyword is provided to the `plot.()` method then data for an individual layer is plotted. Remember `mflay` is zero-based.

Here we plot the horizontal hydraulic conductivity for layer 11 (`mflay=10`).

```
[12]: ml.lpf.hk.plot(mflay=10, masked_values=[0.0], colorbar=True)
```

```
[12]: <Axes: title={'center': 'hk layer 11'}>
```



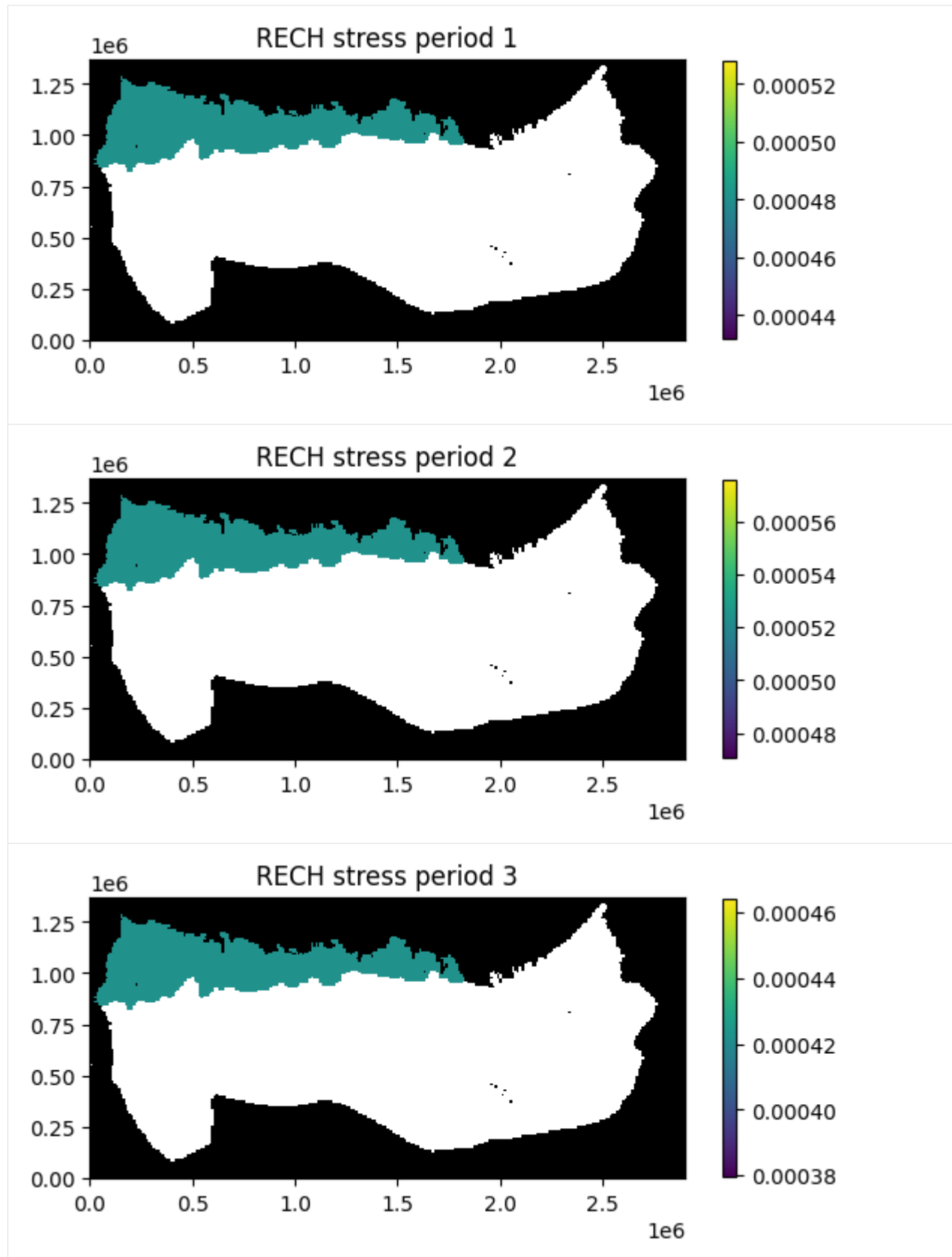
Plotting transient two-dimensional data

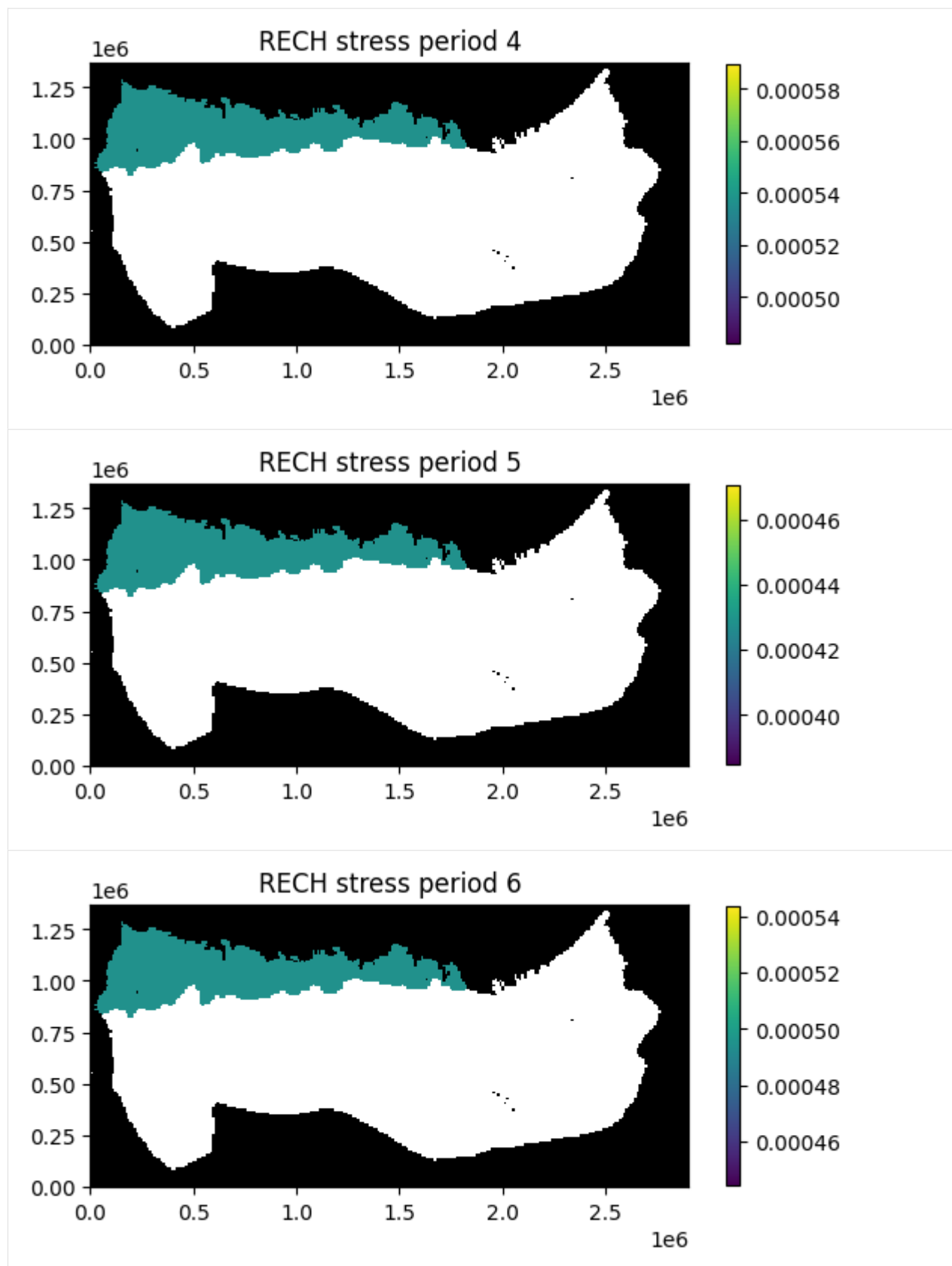
Transient two-dimensional data can be plotted using the `.plot()` method. User's do not actually need to know that the data are two- or three-dimensional. The `.plot()` method is attached to the two- and three-dimensional, and transient two-dimensional data objects so it knows how to process the model data. Examples of transient two-dimensional data are recharge rates (`rch.rech`) and evapotranspiration rates (`evt.evtr`).

Here we plot recharge rates for all six stress periods in the model. We are also masking areas where the recharge rate is zero and adding a color bar.

```
[13]: ml.rch.rech.plot(kper="all", masked_values=[0.0], colorbar=True)
```

```
[13]: [<Axes: title={'center': 'RECH stress period 1'}>,
<Axes: title={'center': 'RECH stress period 2'}>,
<Axes: title={'center': 'RECH stress period 3'}>,
<Axes: title={'center': 'RECH stress period 4'}>,
<Axes: title={'center': 'RECH stress period 5'}>,
<Axes: title={'center': 'RECH stress period 6'}>]
```



In addition to the plots of recharge rates you can see that the `.plot()` method returned a list containing 6 axis objects (one for each stress period).

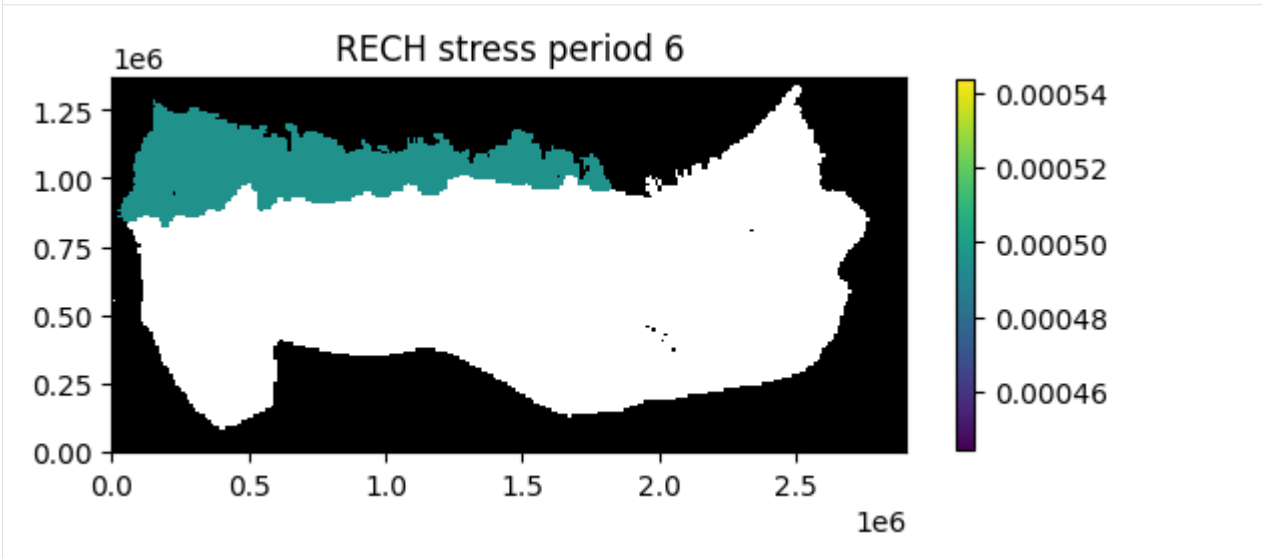
Plotting data for a single stress period

If the `kper=` keyword is provided to the `plot.` () method then data for an individual stress period is plotted. Remember `kper` is zero-based.

Here we plot the recharge rate for stress period 6 (`kper=5`).

```
[14]: ml.rch.rech.plot(kper=5, masked_values=[0.0], colorbar=True)
```

```
[14]: [<Axes: title={'center': 'RECH stress period 6'}>]
```



We can also save the image to a file by provided the `filename_base` keyword with an appropriate base file name.

```
[15]: fr = os.path.join(modelpth, "secp")
ml.rch.rech.plot(kper=5, masked_values=[0.0], colorbar=True, filename_base=fr)
```

```
created...secp_RECH_000006.png
```

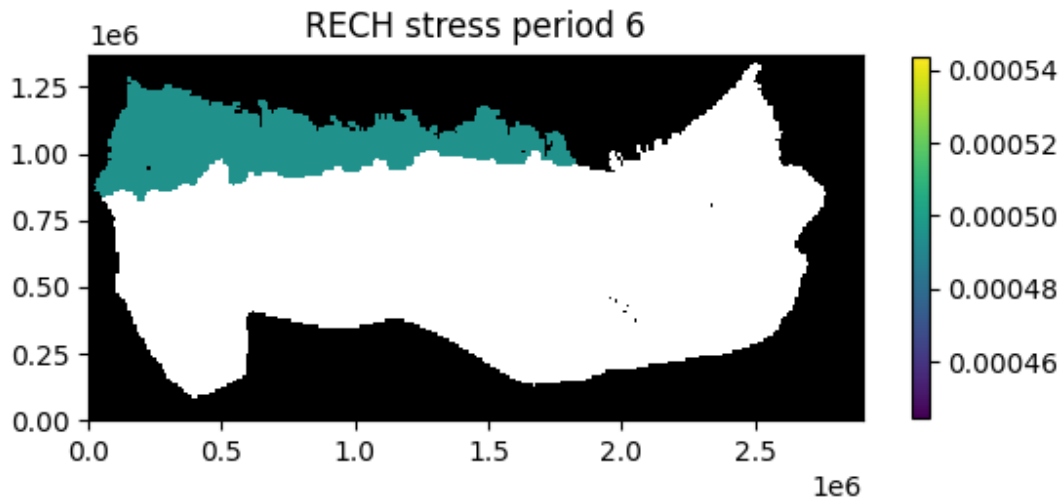
```
[15]: [None]
```

If the `kper` keyword is not provided images are saved for each stress period in the model.

The image file that was just created of recharge rates for stress period 6 is shown below.

```
[16]: fname = os.path.join(modelpth, "secp_RECH_000006.png")
Image(filename=fname)
```

[16]:



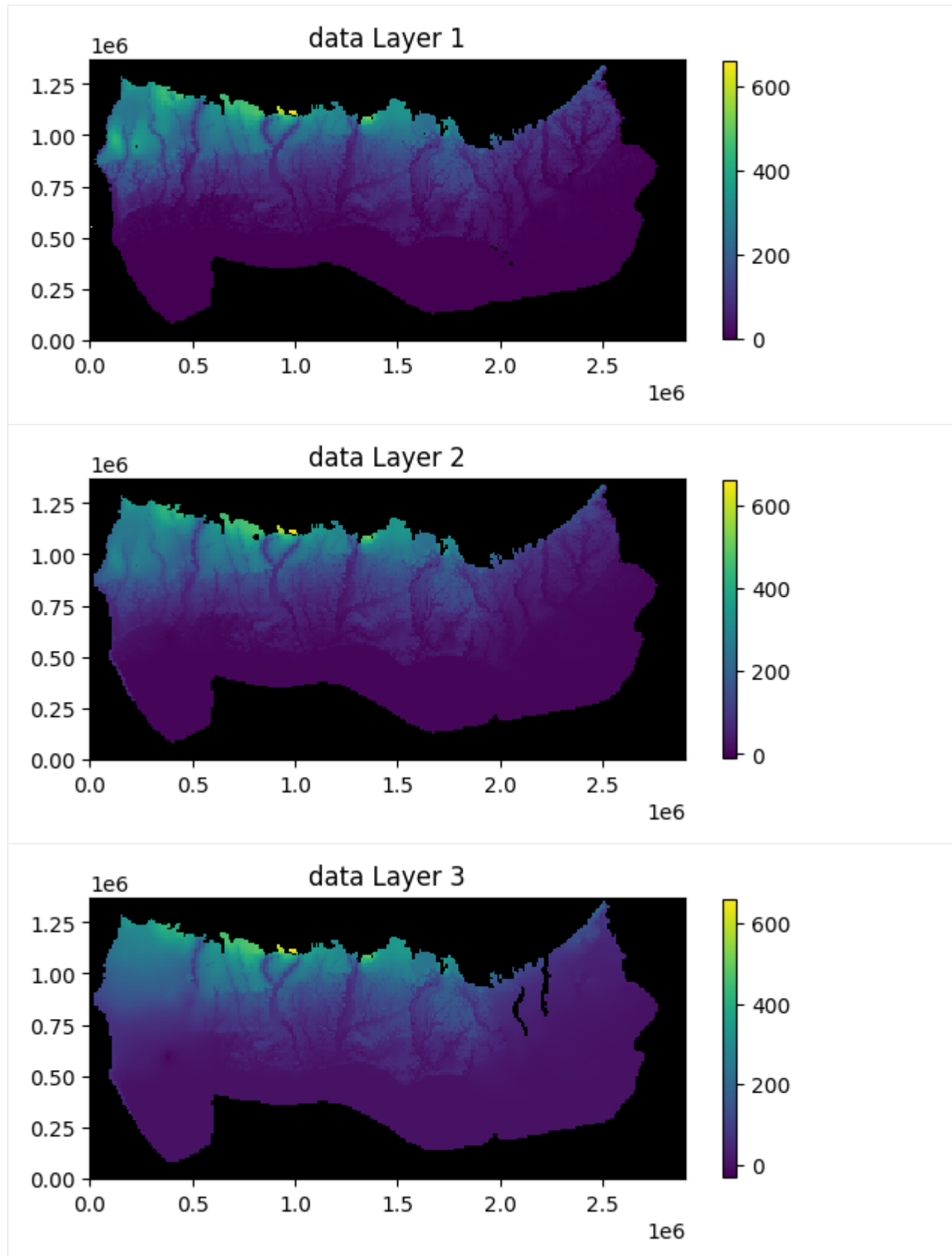
Plotting simulated model results

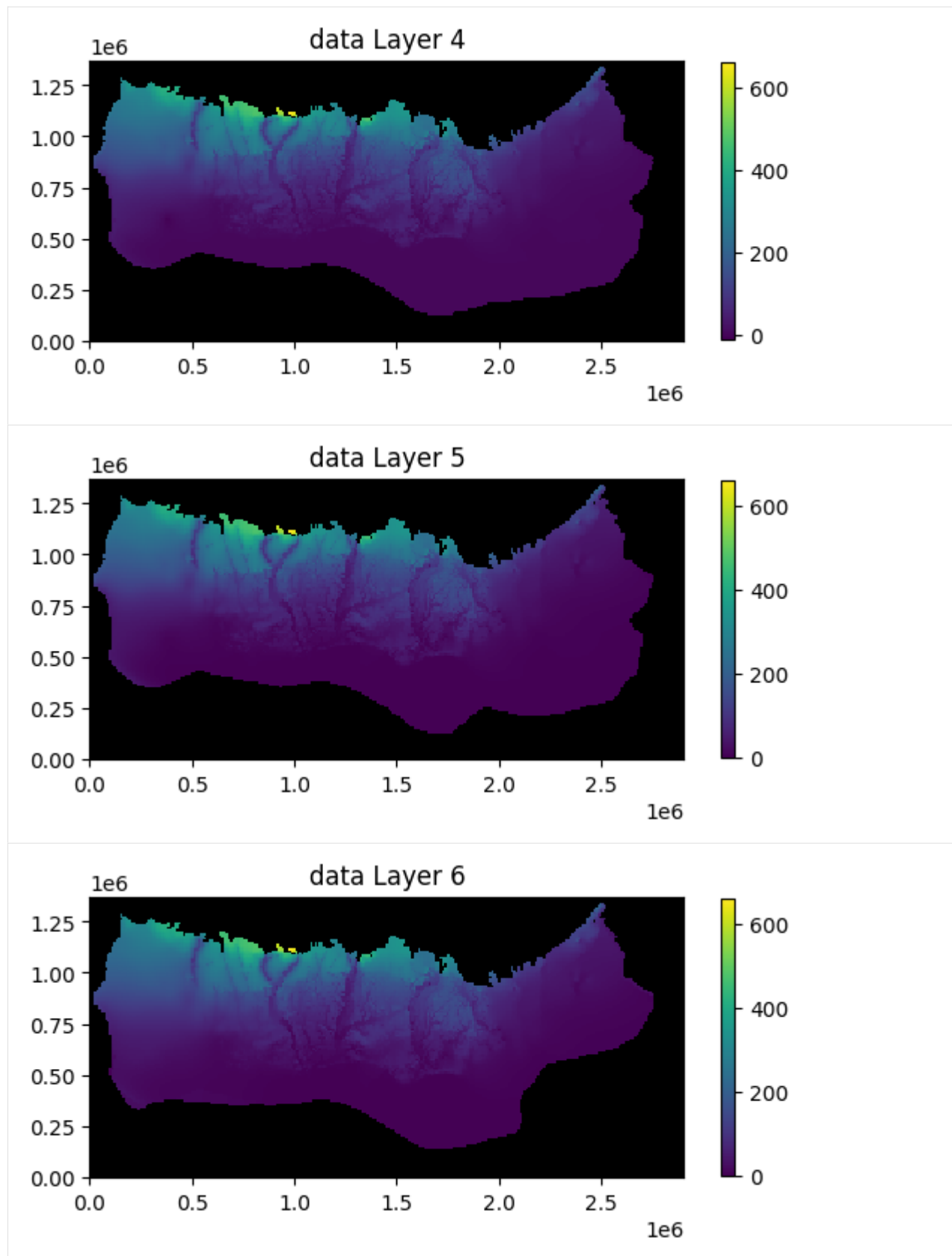
Simulated model results can be plotted using the `.plot()` method.

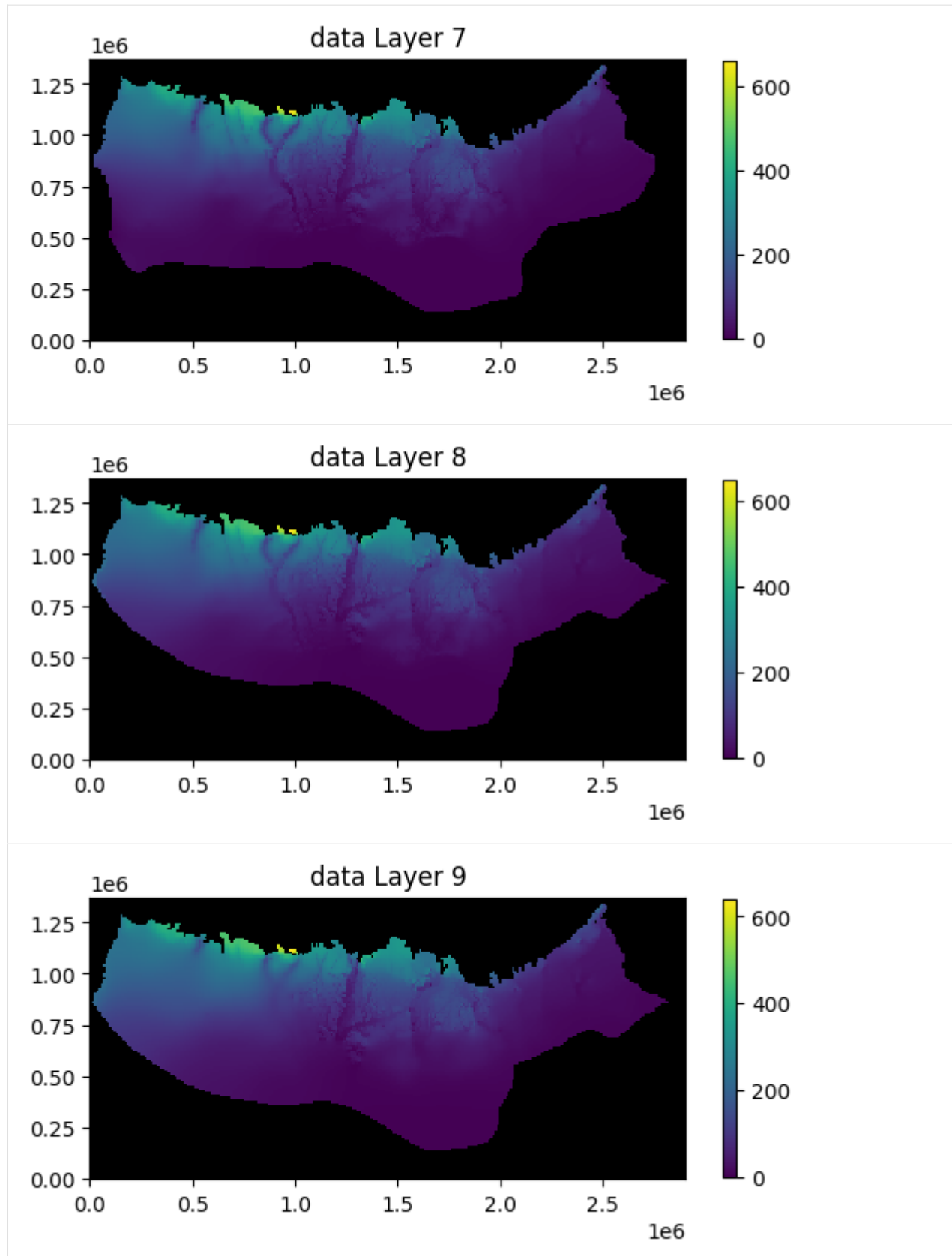
First we create an instance of the `HeadFile` class with the simulated head file (`secp.hds`) and extract the simulation times available in the binary head file using the `.get_times()` method. Here we plot last simulated heads in the binary heads file (`totim=times[-1]`). We are also masking cells having the `HDRY` (-9999.0) value and adding a color bar.

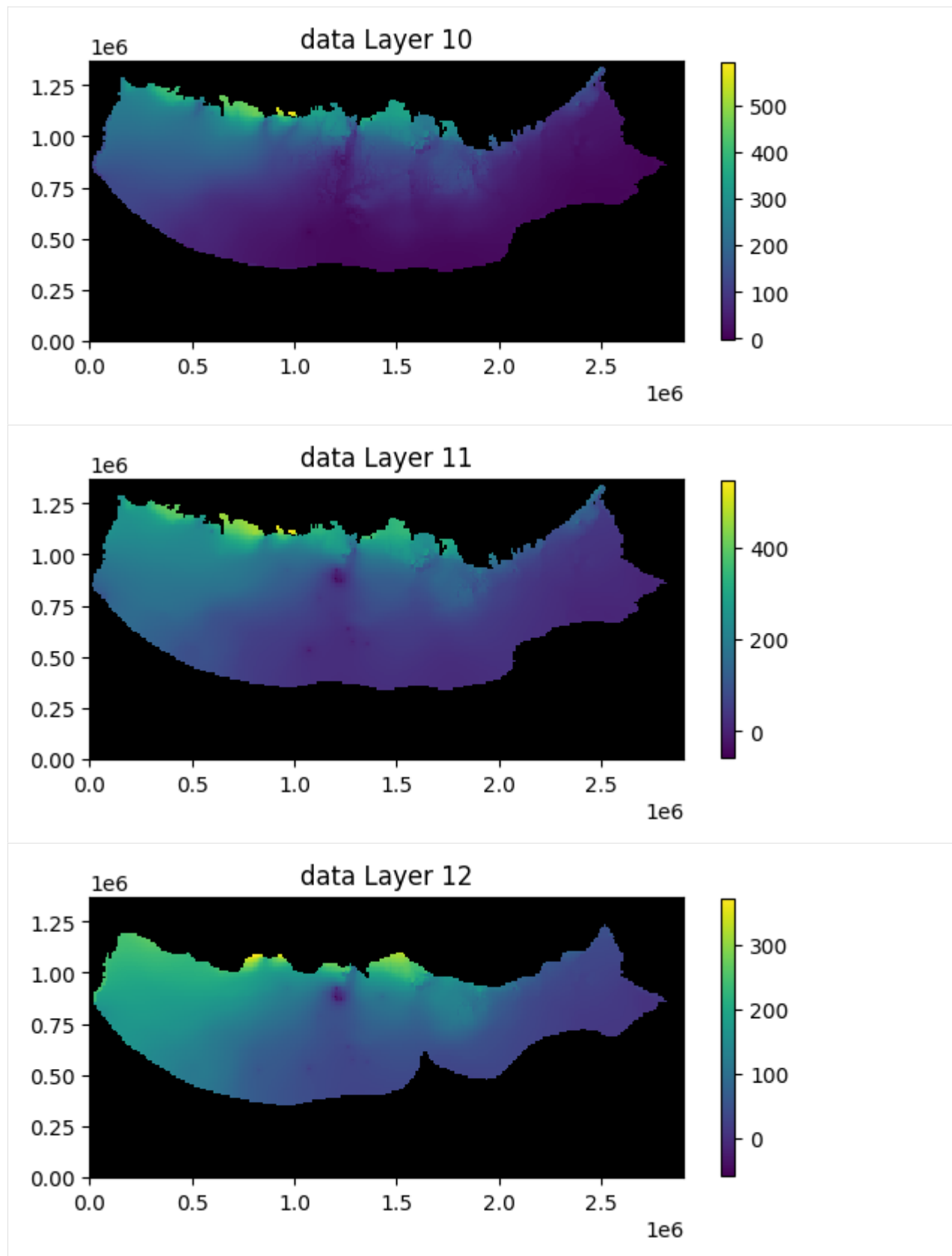
```
[17]: print(files)
      fname = os.path.join(modelpth, files[0])
      hdoobj = flopy.utils.HeadFile(fname, model=m1)
      times = hdoobj.get_times()
      head = hdoobj.plot(totim=times[-1], masked_values=[-9999.0], colorbar=True)

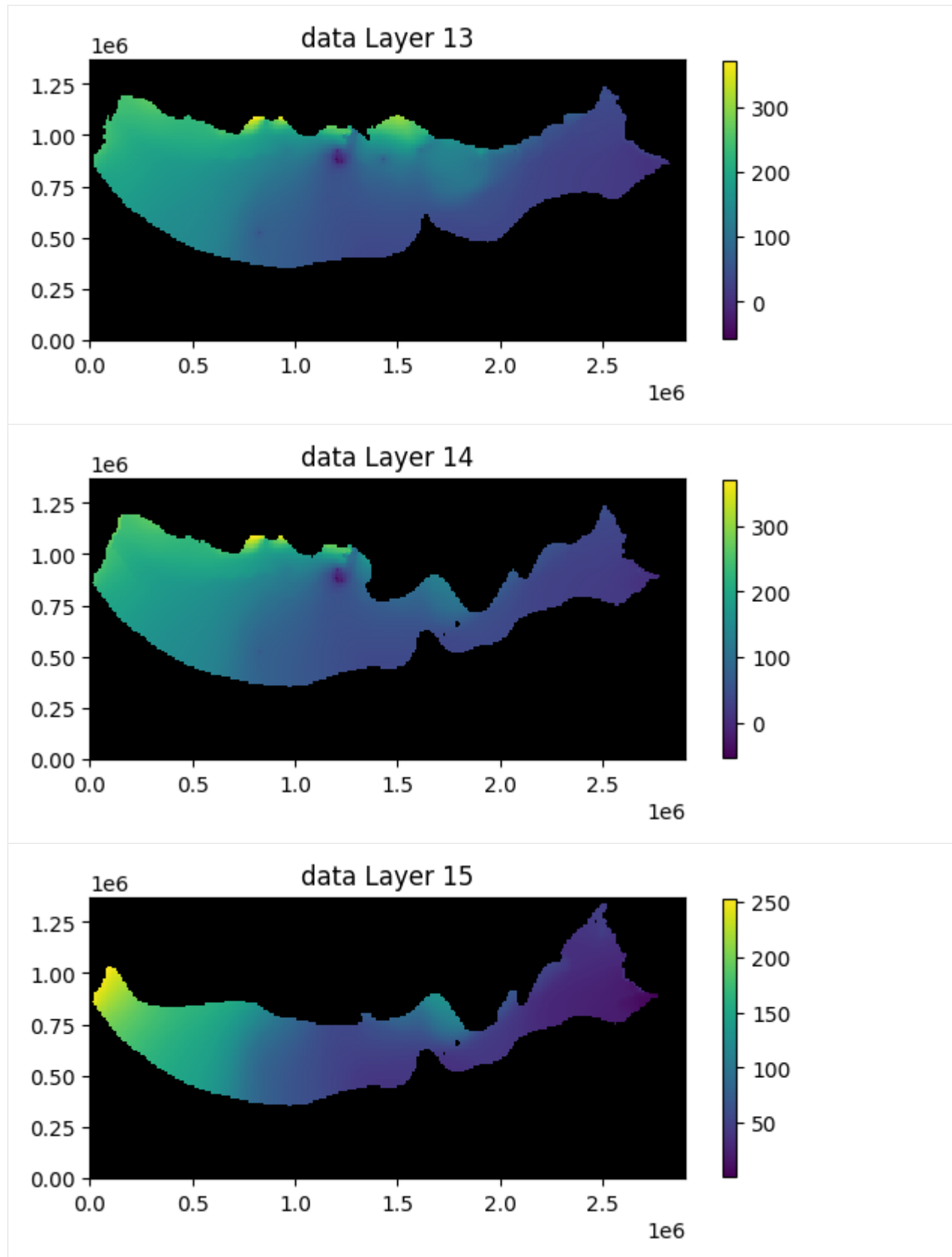
['secp.hds']
```

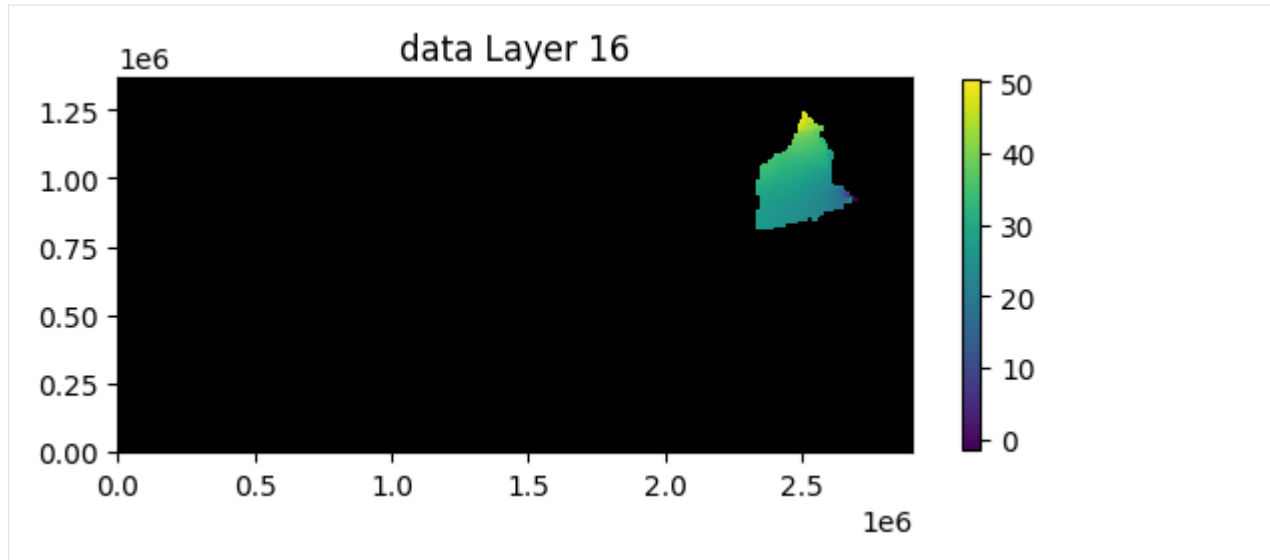










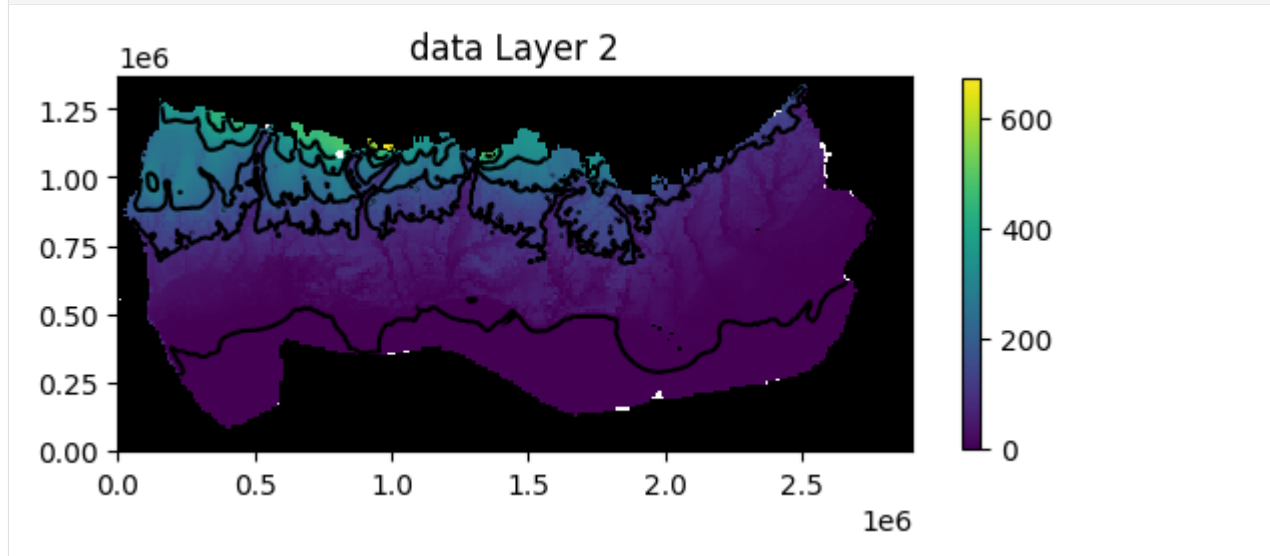


Plotting results for a single layer

If the `mflay=` keyword is provided to the `plot . ()` method then results for an individual layer can be plotted. Remember `mflay` is zero-based.

Here we plot the results for layer 2 (`mflay=1`) for stress period 2 (`totim=times[1]`). We also add black contour lines.

```
[18]: head = hdoobj.plot(
    totim=times[1],
    mflay=1,
    masked_values=[-9999.0],
    colorbar=True,
    contour=True,
    colors="black",
)
```



We can also save the plots of the head results for every layer (or a single layer) to a file by provided the `filename_base`

keyword with an appropriate base file name.

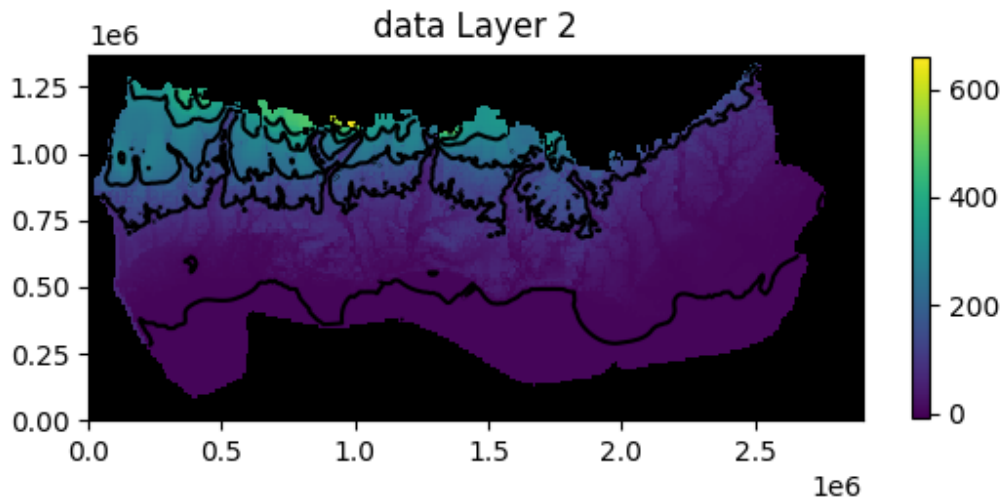
```
[19]: fh = os.path.join(modelpth, "secp_head")
head = hdoobj.plot(
    totim=times[-1],
    masked_values=[-9999.0],
    colorbar=True,
    contour=True,
    colors="black",
    filename_base=fh,
)
```

```
created...secp_head_Layer1.png
created...secp_head_Layer2.png
created...secp_head_Layer3.png
created...secp_head_Layer4.png
created...secp_head_Layer5.png
created...secp_head_Layer6.png
created...secp_head_Layer7.png
created...secp_head_Layer8.png
created...secp_head_Layer9.png
created...secp_head_Layer10.png
created...secp_head_Layer11.png
created...secp_head_Layer12.png
created...secp_head_Layer13.png
created...secp_head_Layer14.png
created...secp_head_Layer15.png
created...secp_head_Layer16.png
```

The image file that was just created of the simulated head for layer 2 for stress period 6 is shown below.

```
[20]: fname = os.path.join(modelpth, "secp_head_Layer2.png")
Image(filename=fname)
```

[20]:



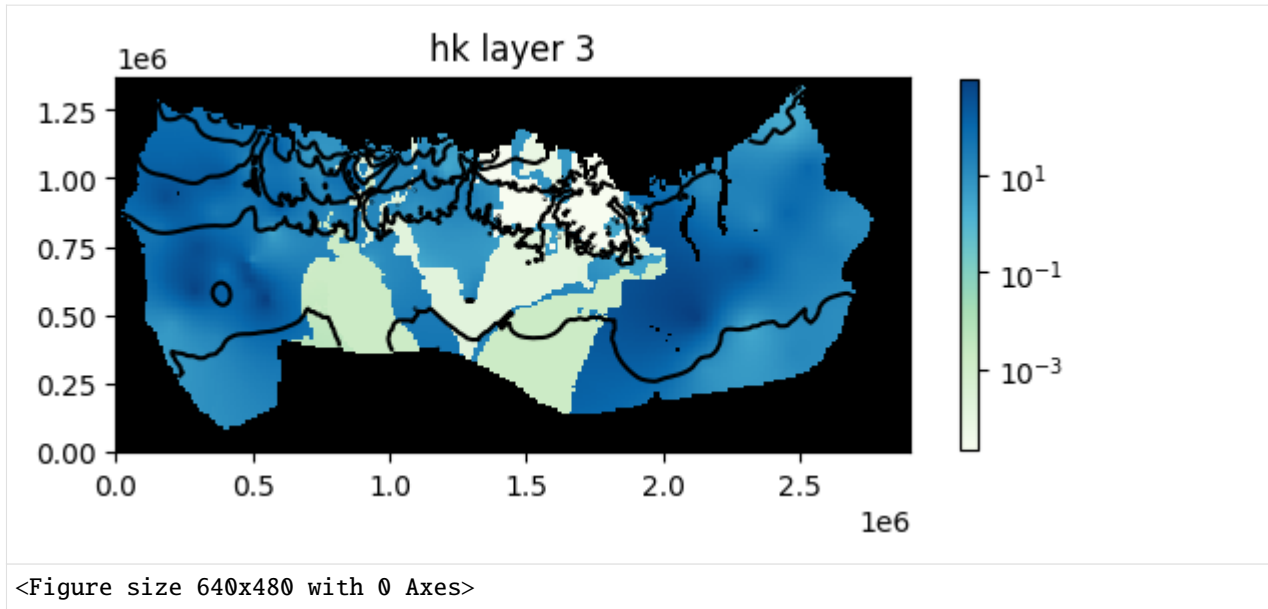
Passing other matplotlib.pyplot keywords to .plot() methods

We can also pass matplotlib.pyplot keywords to .plot() methods attached to the model input data arrays. For example you can pass a matplotlib colormap (cmap=) keyword to the .plot() method to plot contours of simulated heads over a color flood of hk. We can also use the norm=LogNorm() keyword to use a log color scale when plotting hydraulic conductivity.

Available matplotlib colormaps can be found at <https://matplotlib.org/stable/tutorials/colors/colormaps.html>

```
[21]: from matplotlib.colors import LogNorm

ax = ml.lpf.hk.plot(mflay=2, cmap="GnBu", norm=LogNorm(), colorbar=True)
t = hdoobj.plot(
    axes=ax,
    mflay=2,
    masked_values=[-9999.0],
    pcolor=False,
    contour=True,
    colors="black",
)
```



Plotting data for a package or a model

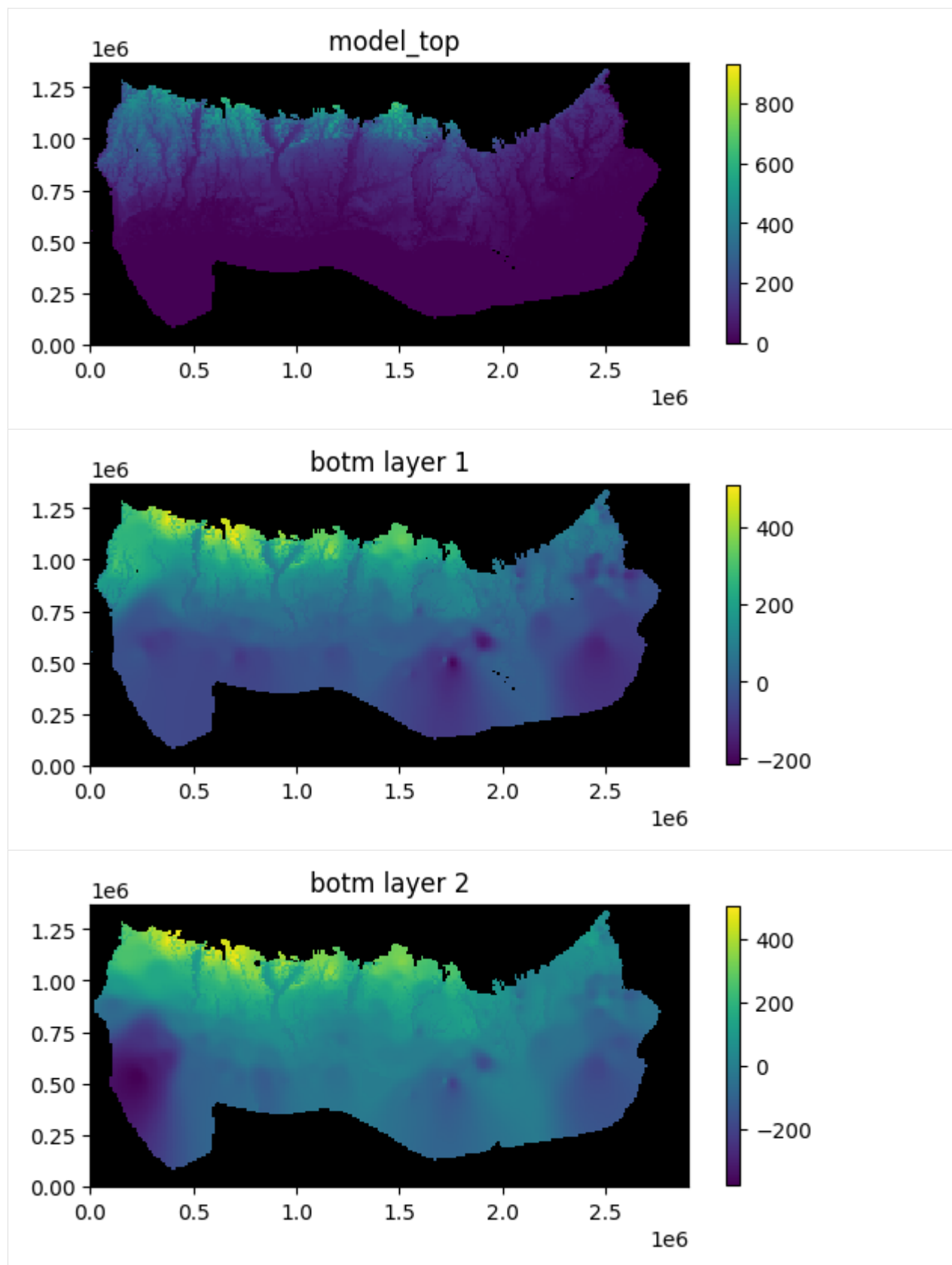
The input data for a model or an individual package can also be plotted using the `.plot()` method. The `.plot()` methods attached to a model or an individual package are meant to provide a method to quickly evaluate model or package input. As a result, there is limited ability to customize the plots. Example of using the `.plot()` method with a model or and individual packages is demonstrated below.

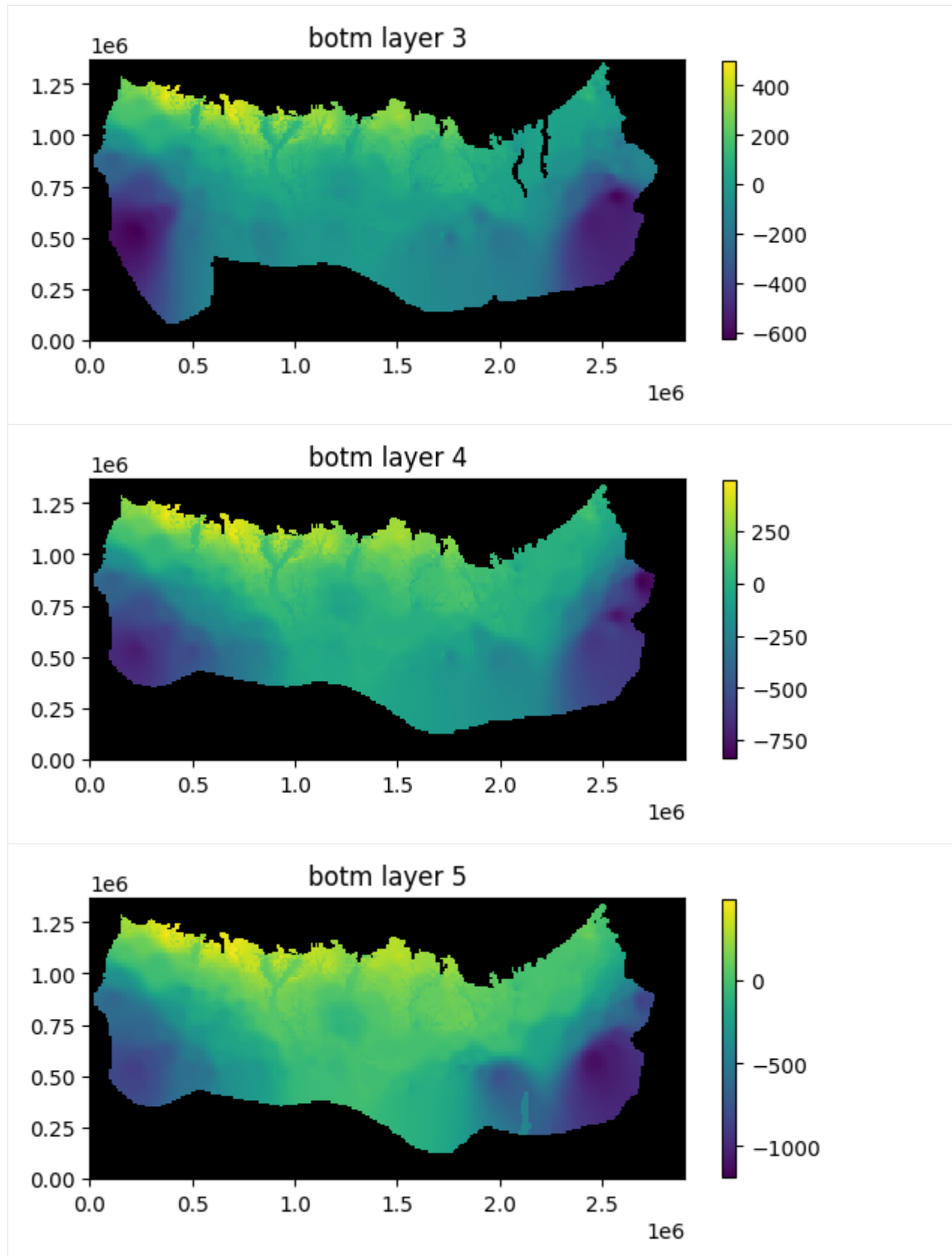
Plot all data for a package

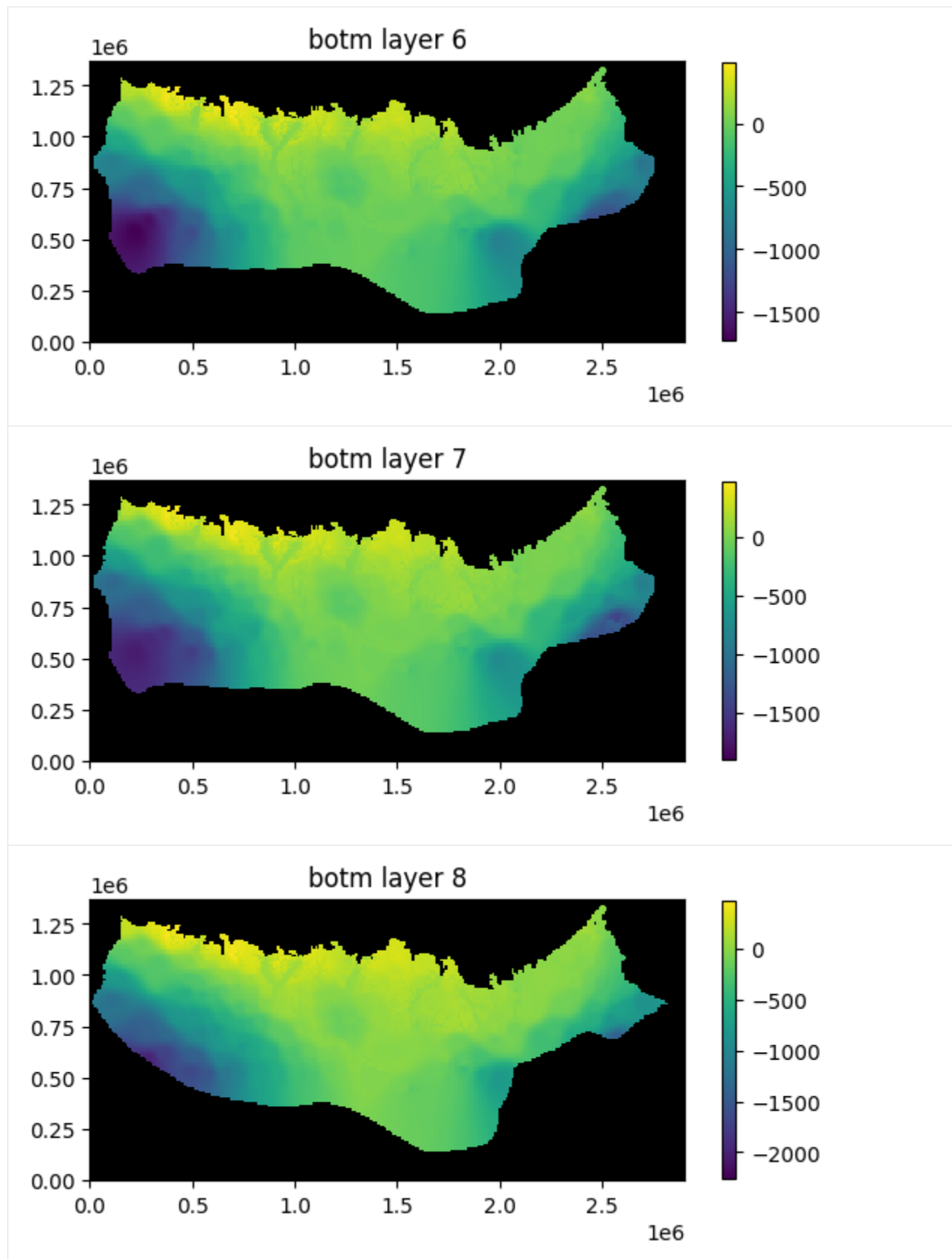
All input data for a package can be plotted using the `.plot()` method. Below all of the data for the `lpf` package is plotted.

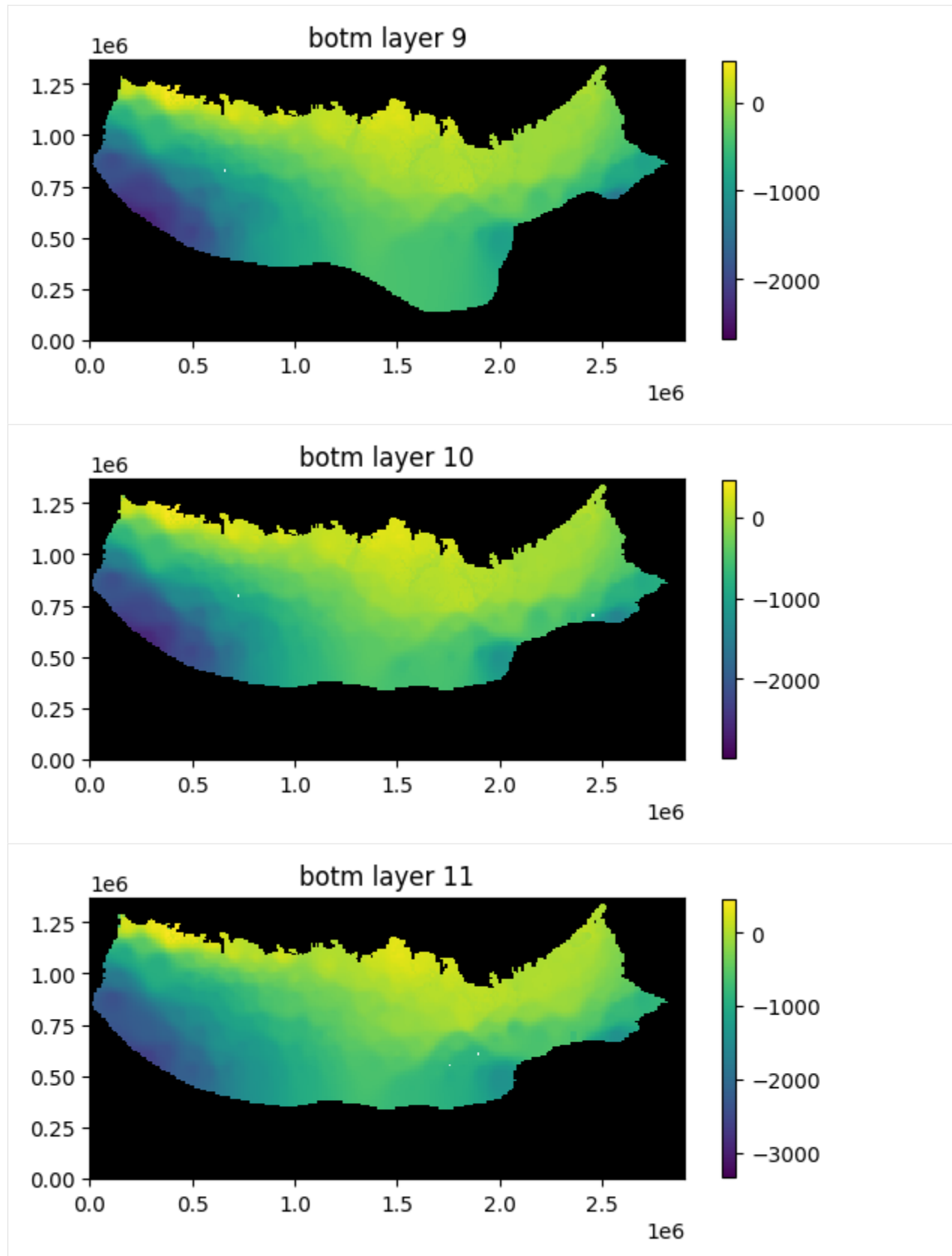
```
[22]: ml.dis.plot()
```

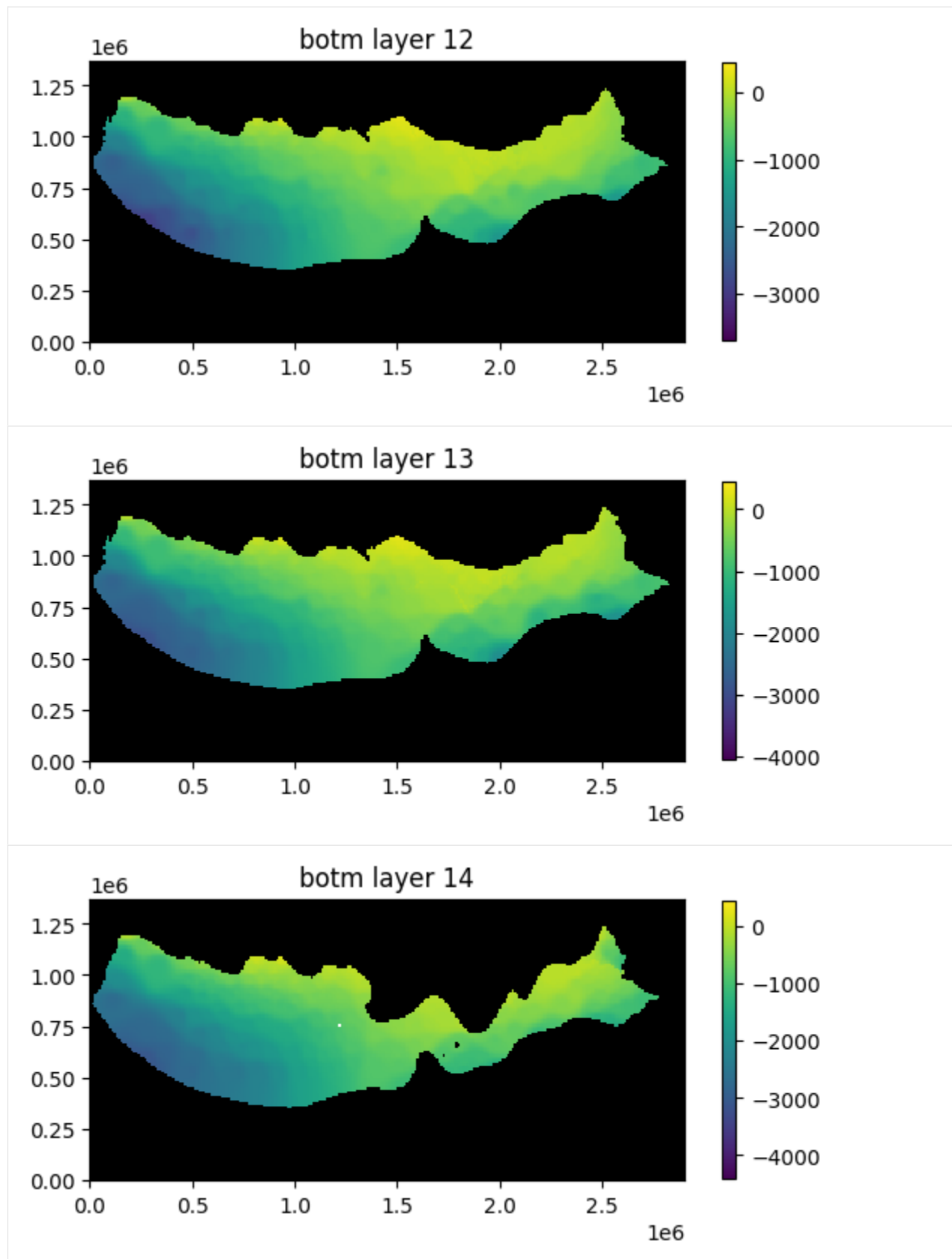
```
[22]: [<Axes: title={'center': ' model_top'}>,
<Axes: title={'center': 'botm layer 1'}>,
<Axes: title={'center': 'botm layer 2'}>,
<Axes: title={'center': 'botm layer 3'}>,
<Axes: title={'center': 'botm layer 4'}>,
<Axes: title={'center': 'botm layer 5'}>,
<Axes: title={'center': 'botm layer 6'}>,
<Axes: title={'center': 'botm layer 7'}>,
<Axes: title={'center': 'botm layer 8'}>,
<Axes: title={'center': 'botm layer 9'}>,
<Axes: title={'center': 'botm layer 10'}>,
<Axes: title={'center': 'botm layer 11'}>,
<Axes: title={'center': 'botm layer 12'}>,
<Axes: title={'center': 'botm layer 13'}>,
<Axes: title={'center': 'botm layer 14'}>,
<Axes: title={'center': 'botm layer 15'}>,
<Axes: title={'center': 'botm layer 16'}>]
```

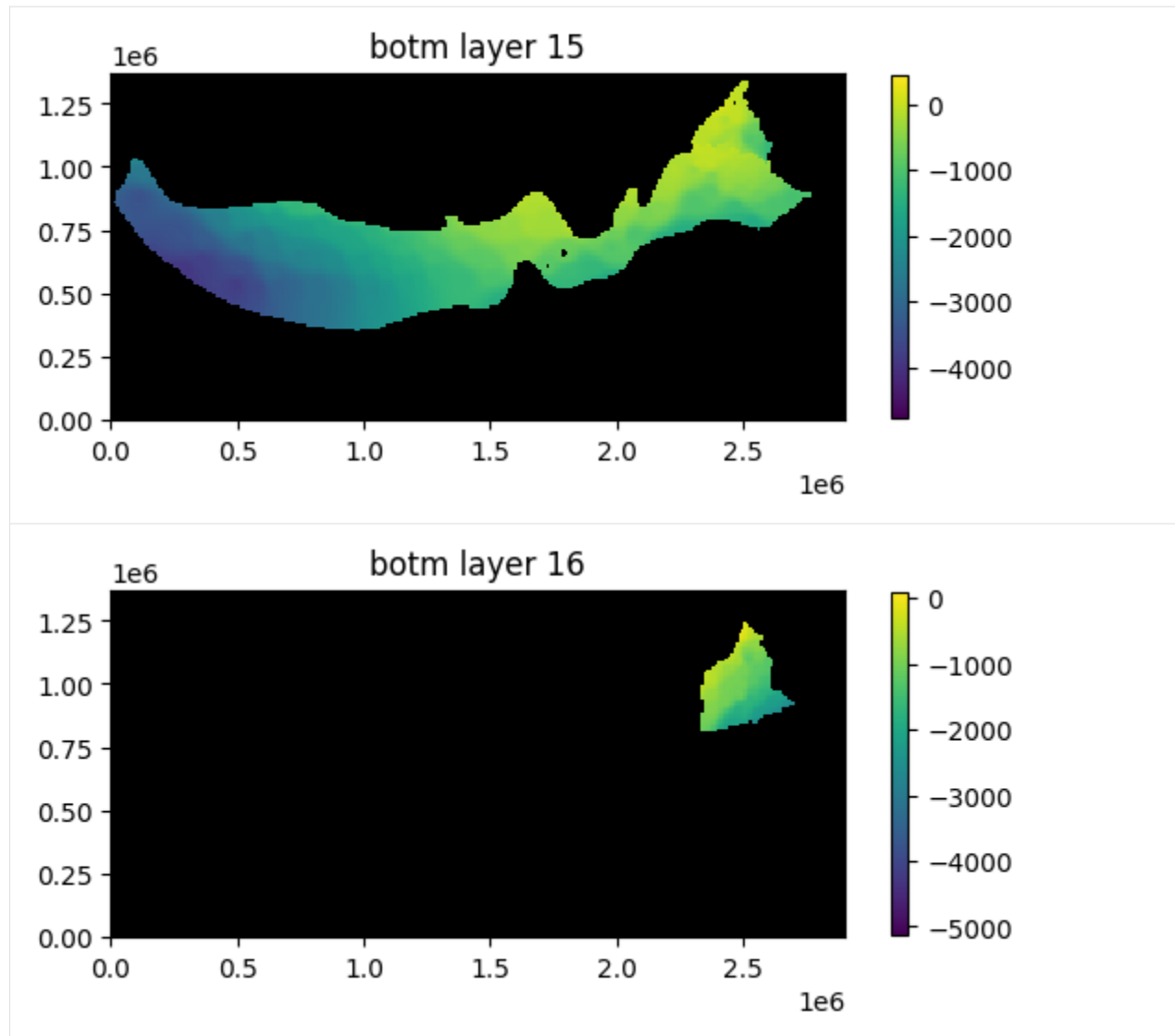








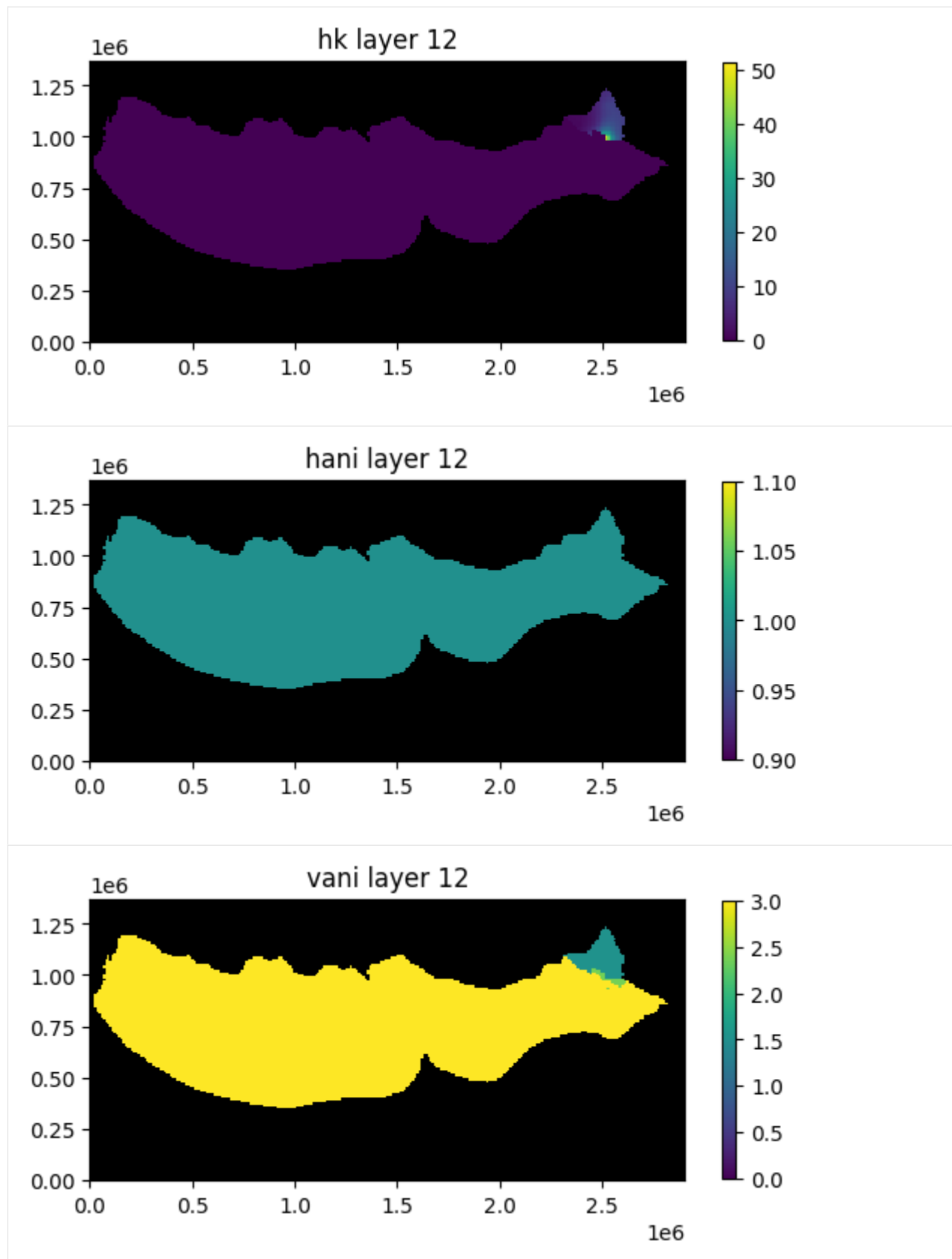


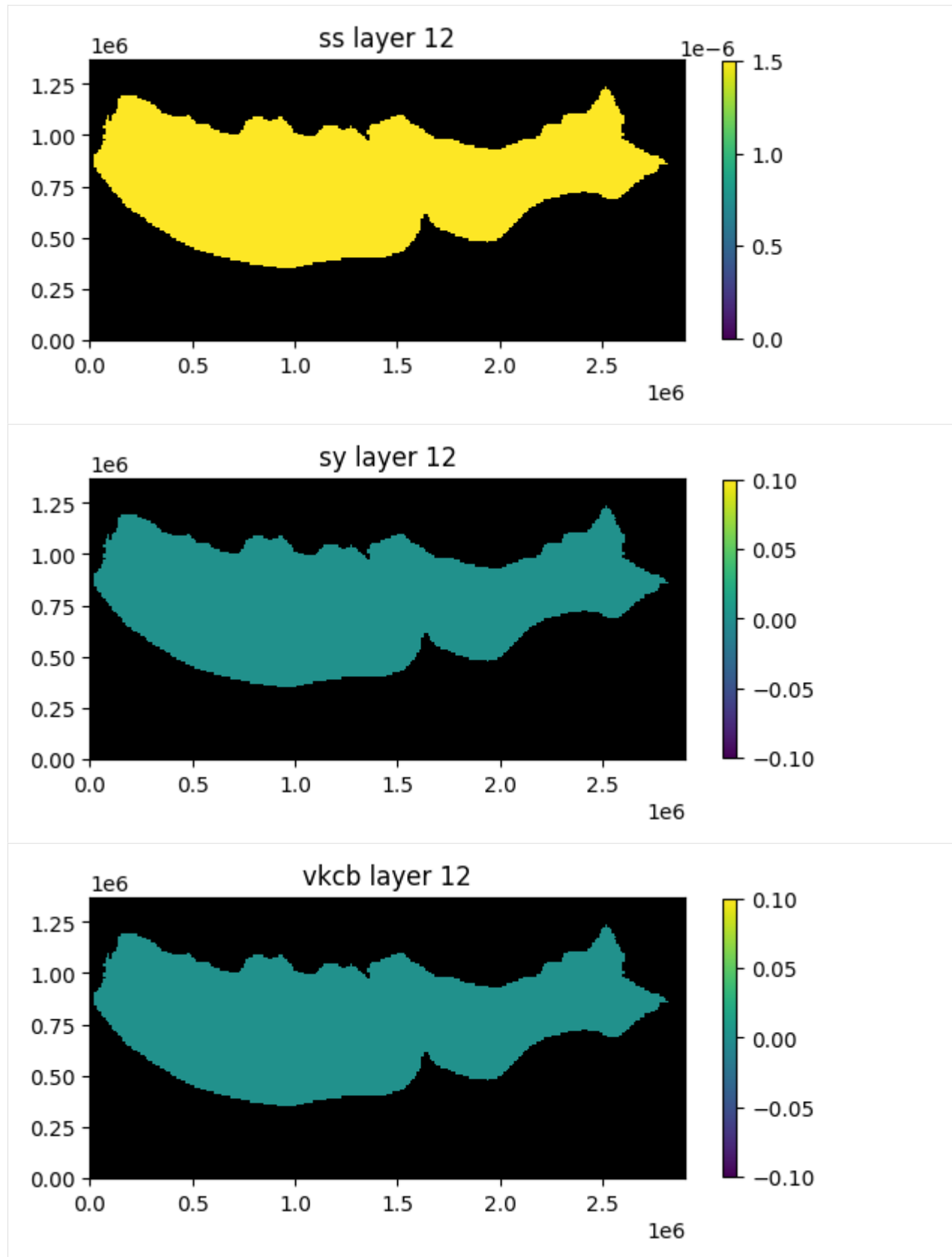


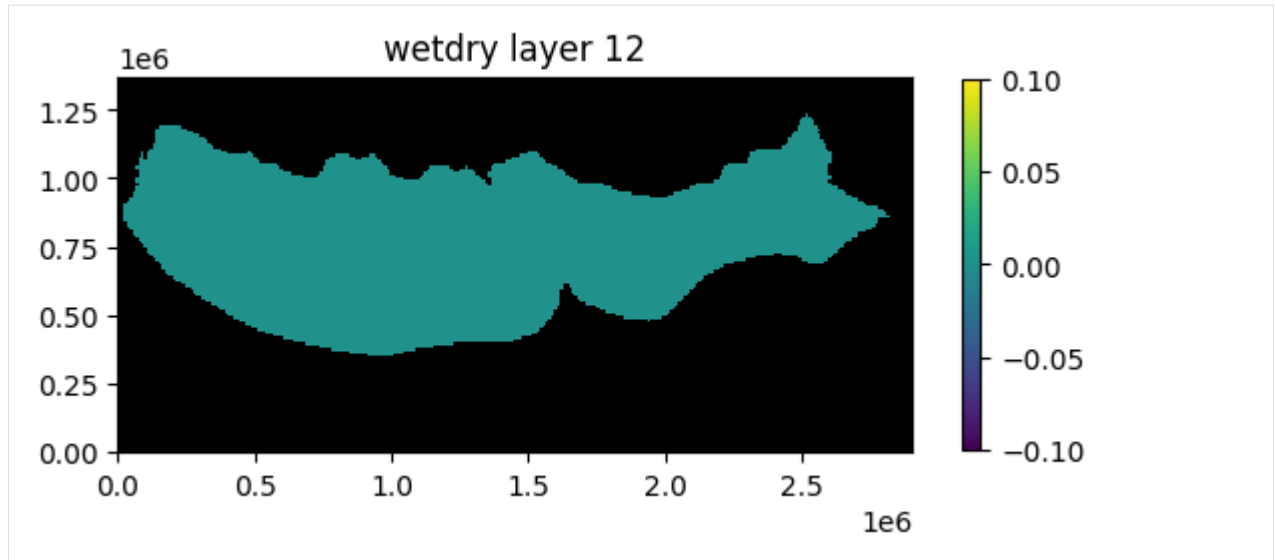
Plot package input data for a specified layer

Package input data for a specified layer can be plotted by passing the `mflay` keyword to the package `.plot()` method. Below `lpf` package input data for layer 12 (`mflay=11`) is plotted.

```
[23]: ml.lpf.plot(mflay=11)
[23]: [<Axes: title={'center': 'hk layer 12'}>,
<Axes: title={'center': 'hani layer 12'}>,
<Axes: title={'center': 'vani layer 12'}>,
<Axes: title={'center': 'ss layer 12'}>,
<Axes: title={'center': 'sy layer 12'}>,
<Axes: title={'center': 'vkcb layer 12'}>,
<Axes: title={'center': 'wetdry layer 12'}>]
```



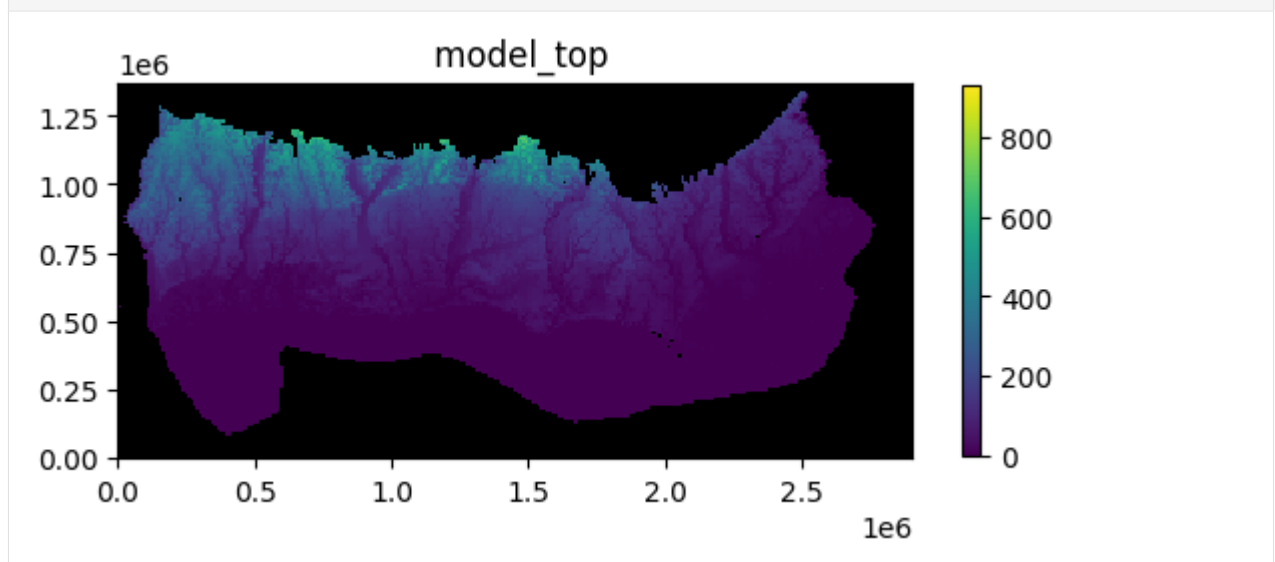


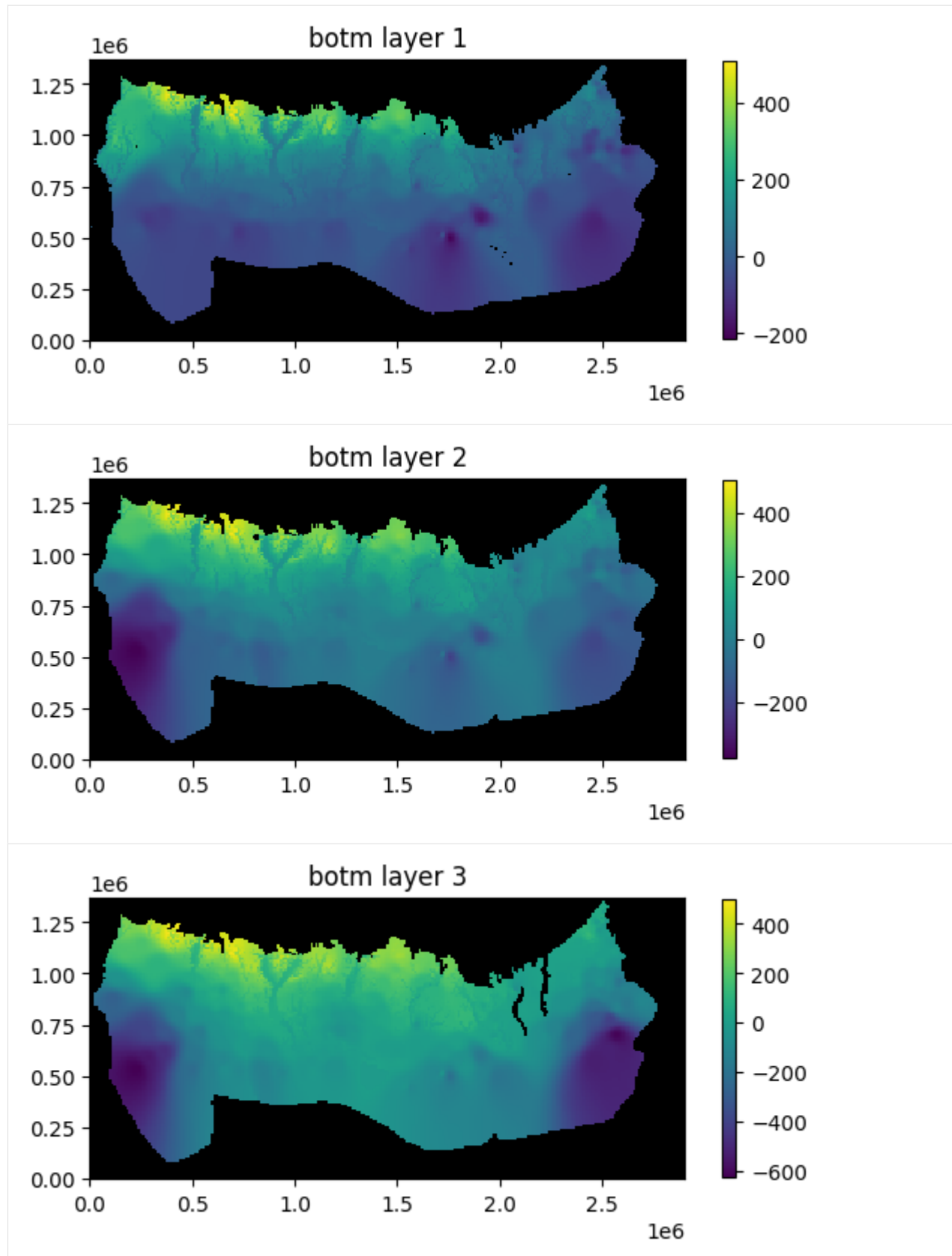


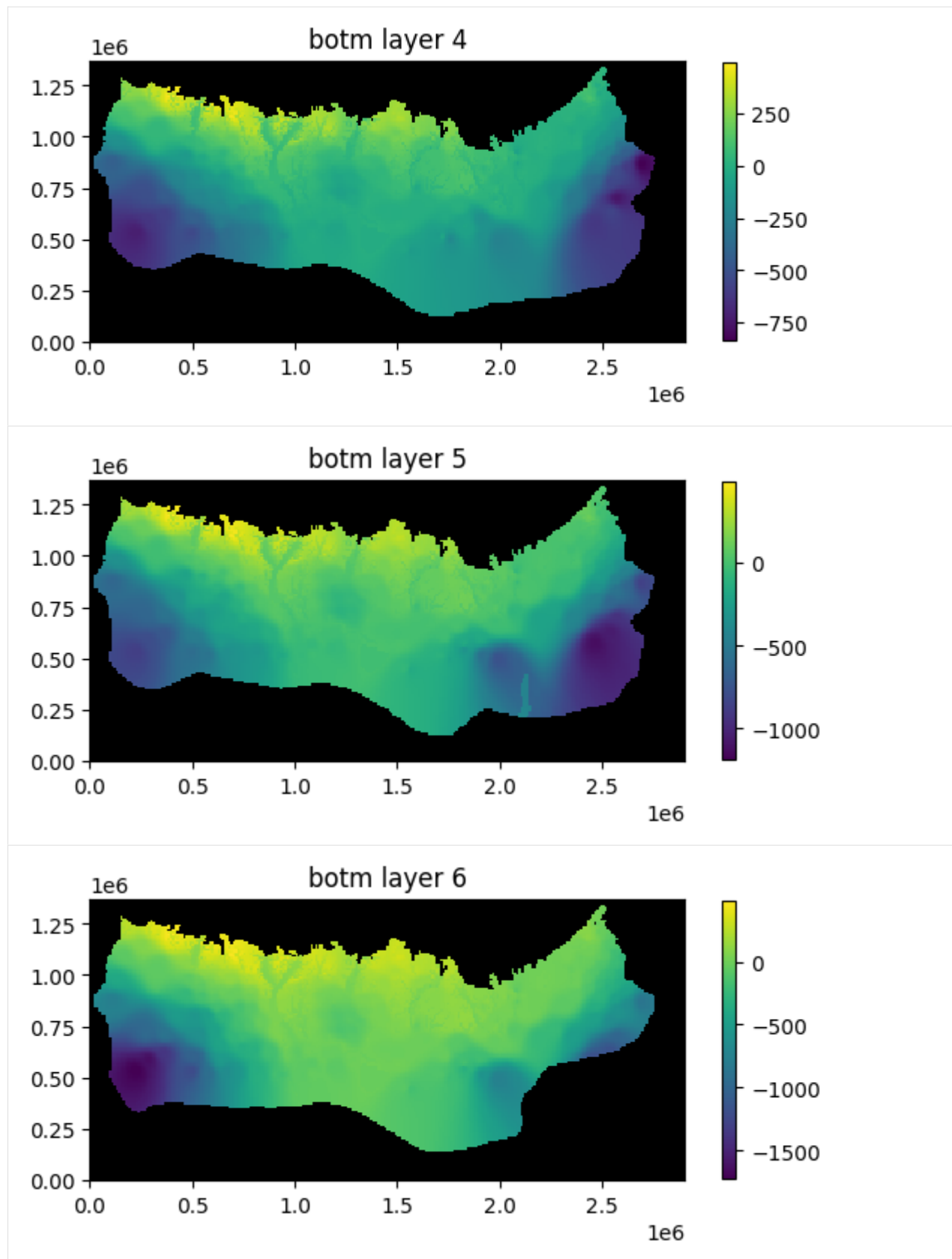
Plot all input data for a model

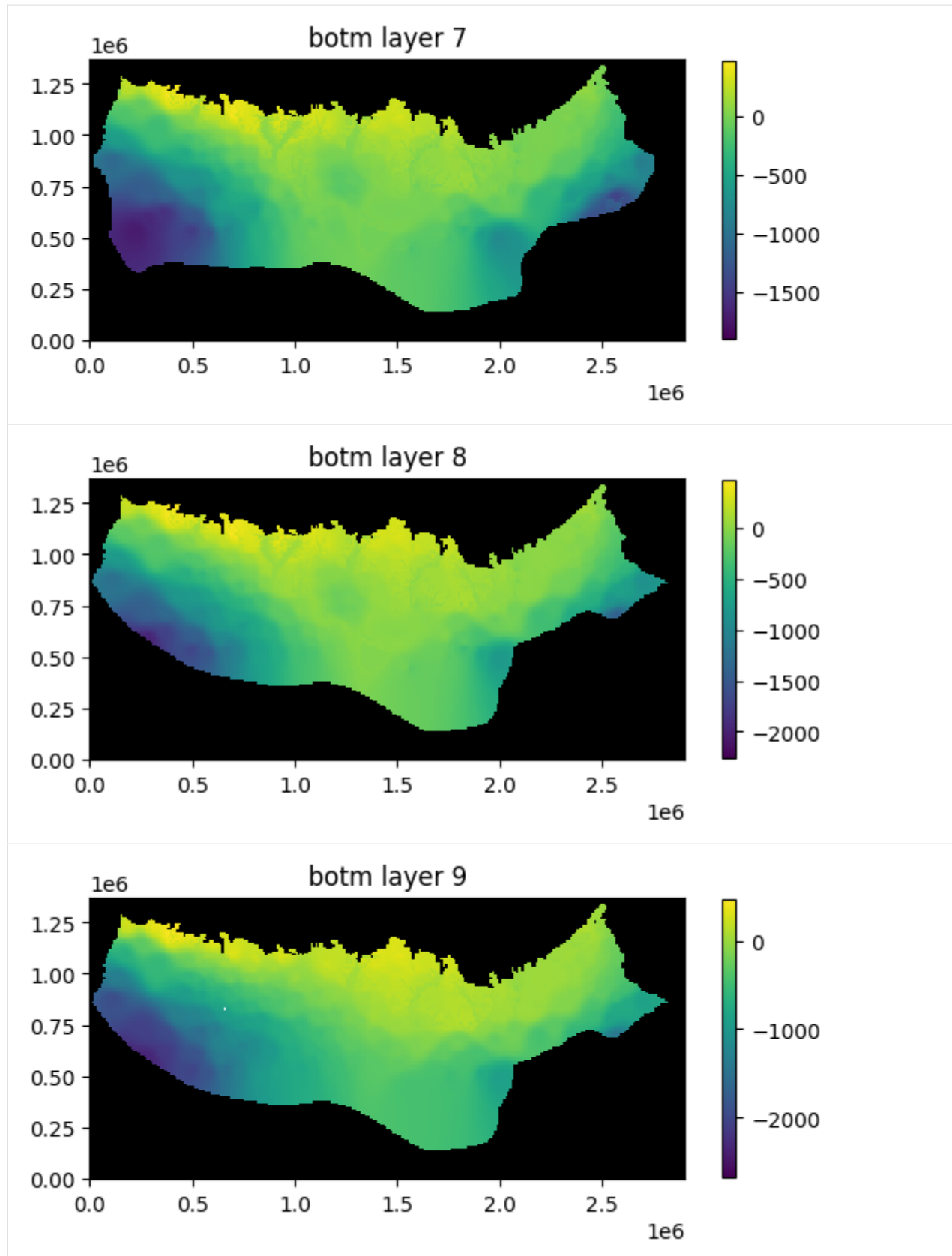
All of the input data for a model can be plotted using the `.plot()` method.

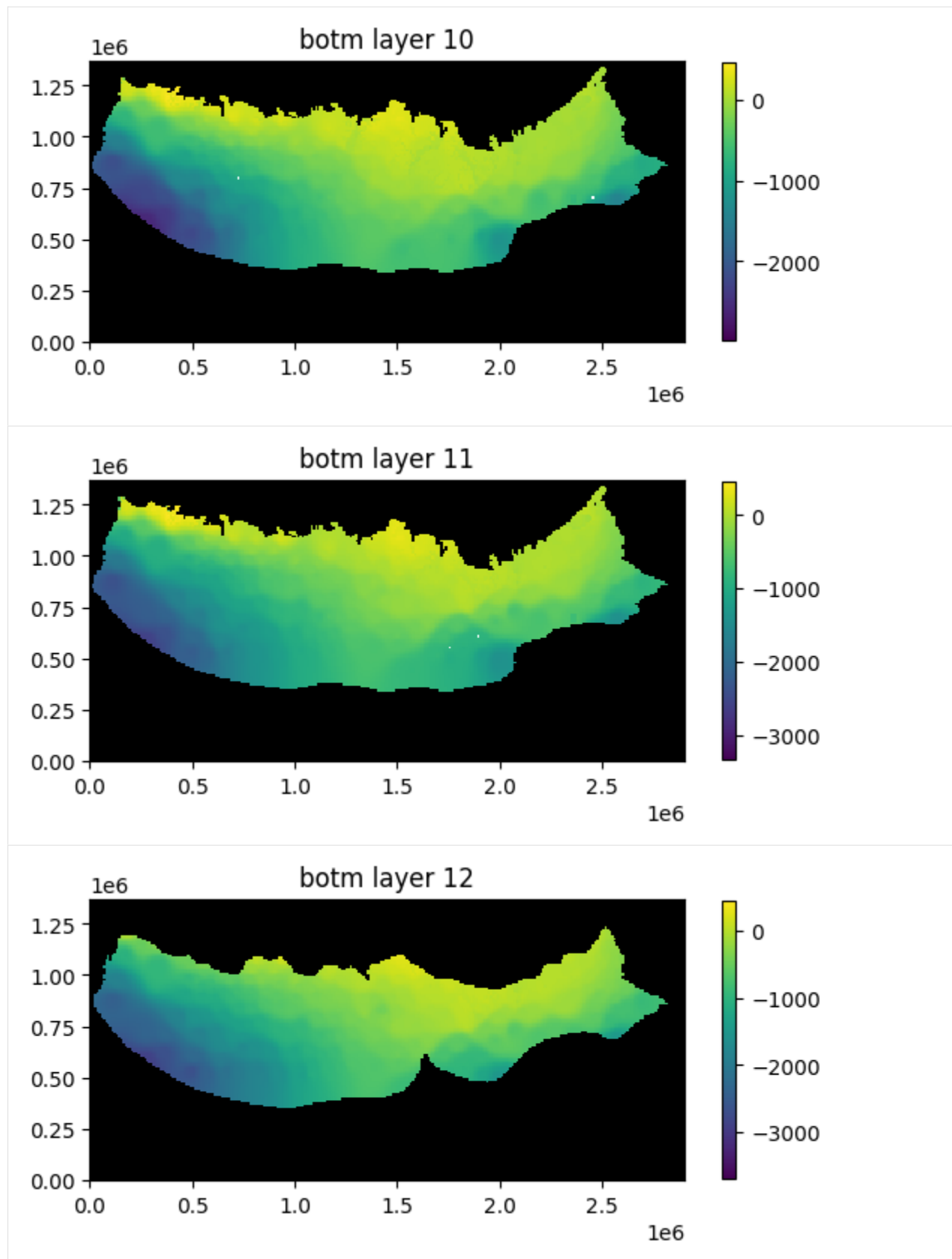
```
[24]: ap = ml.plot()
```

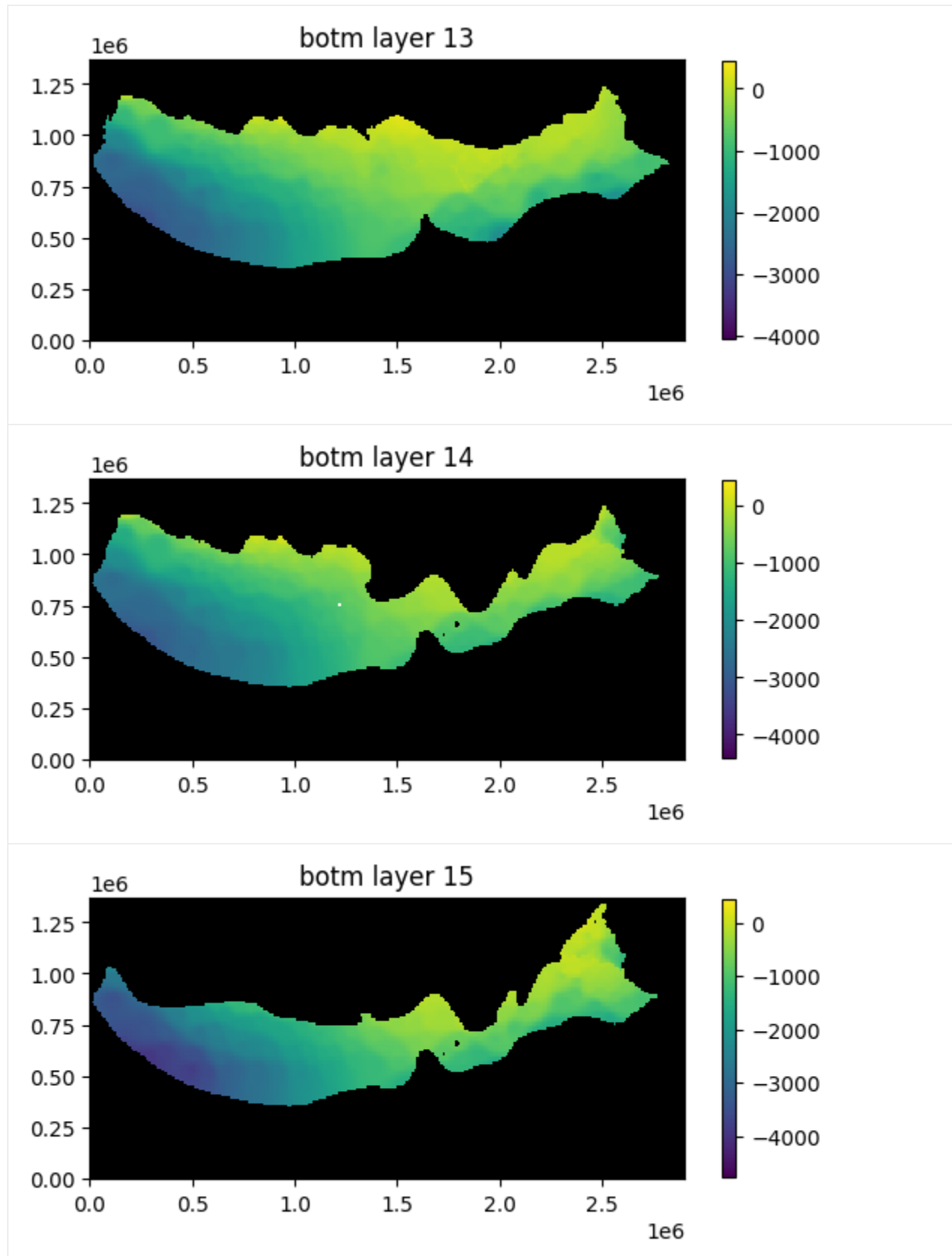


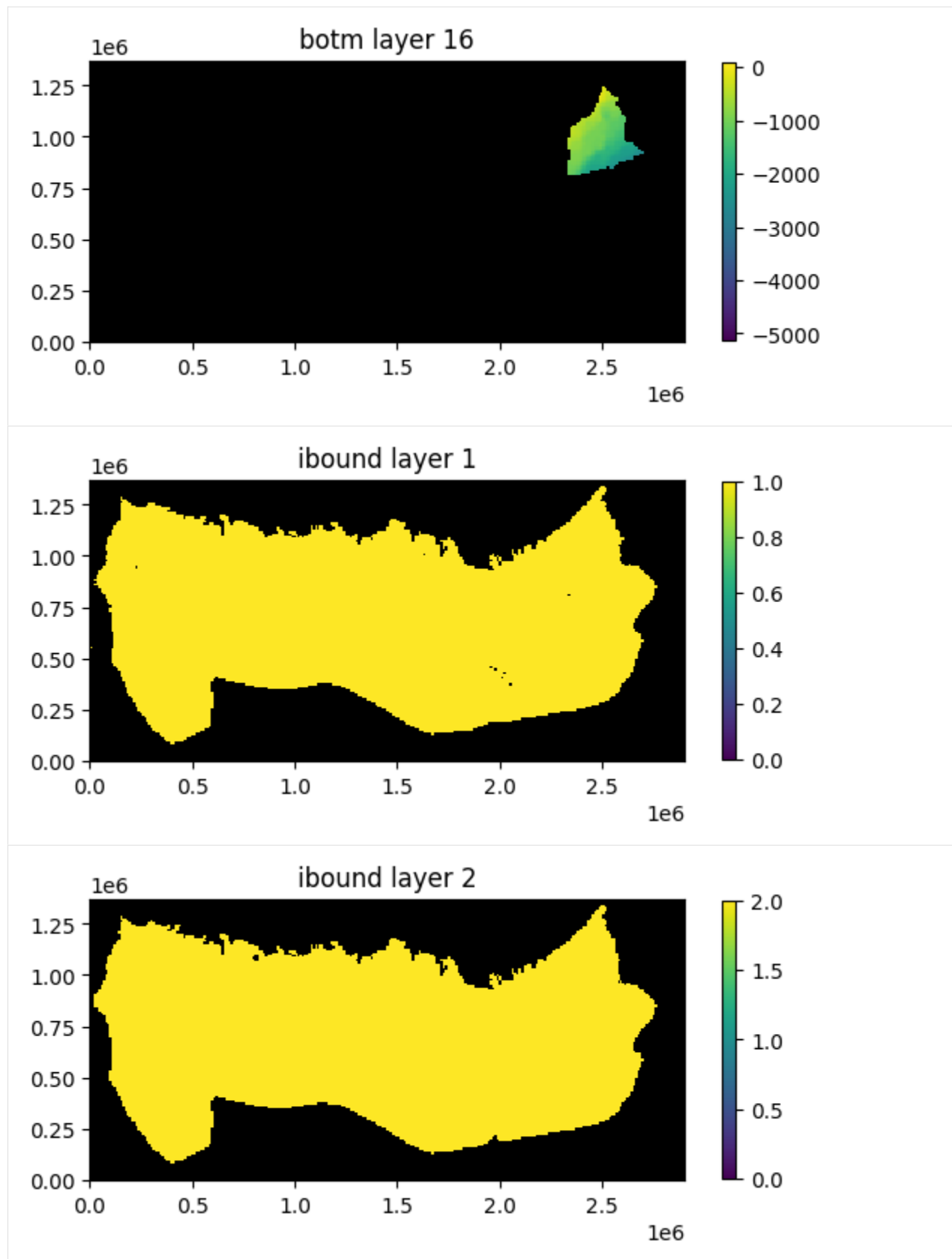


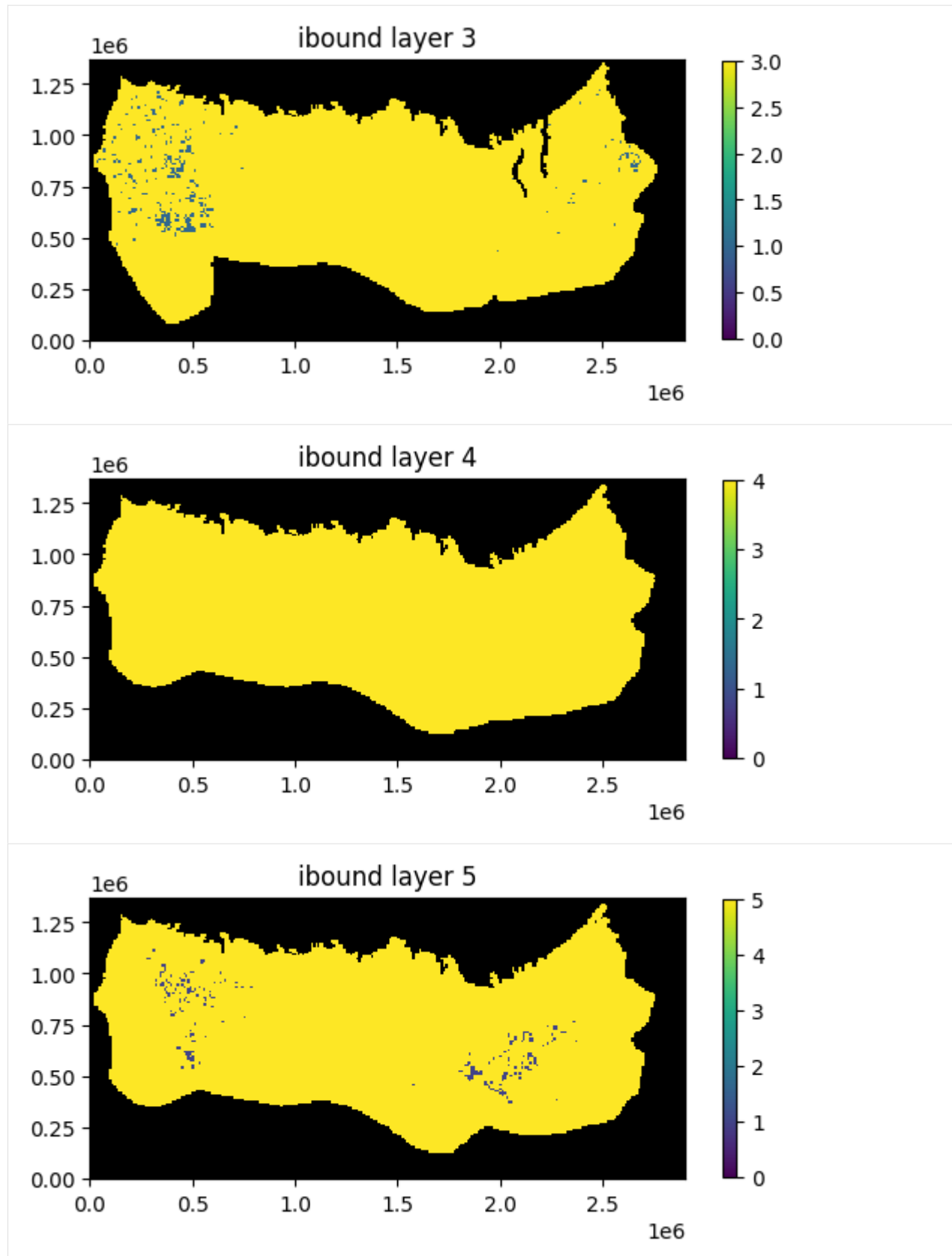


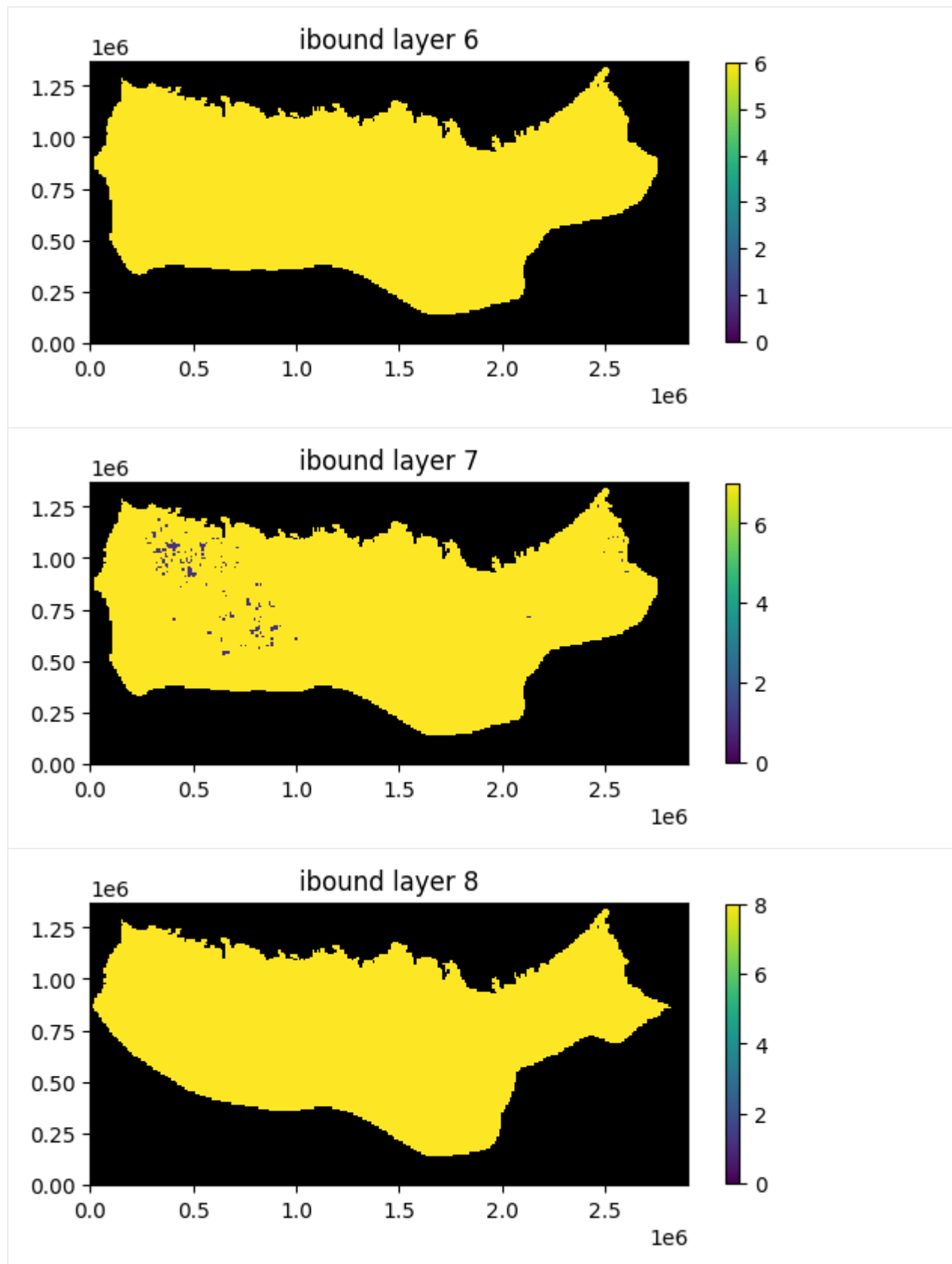


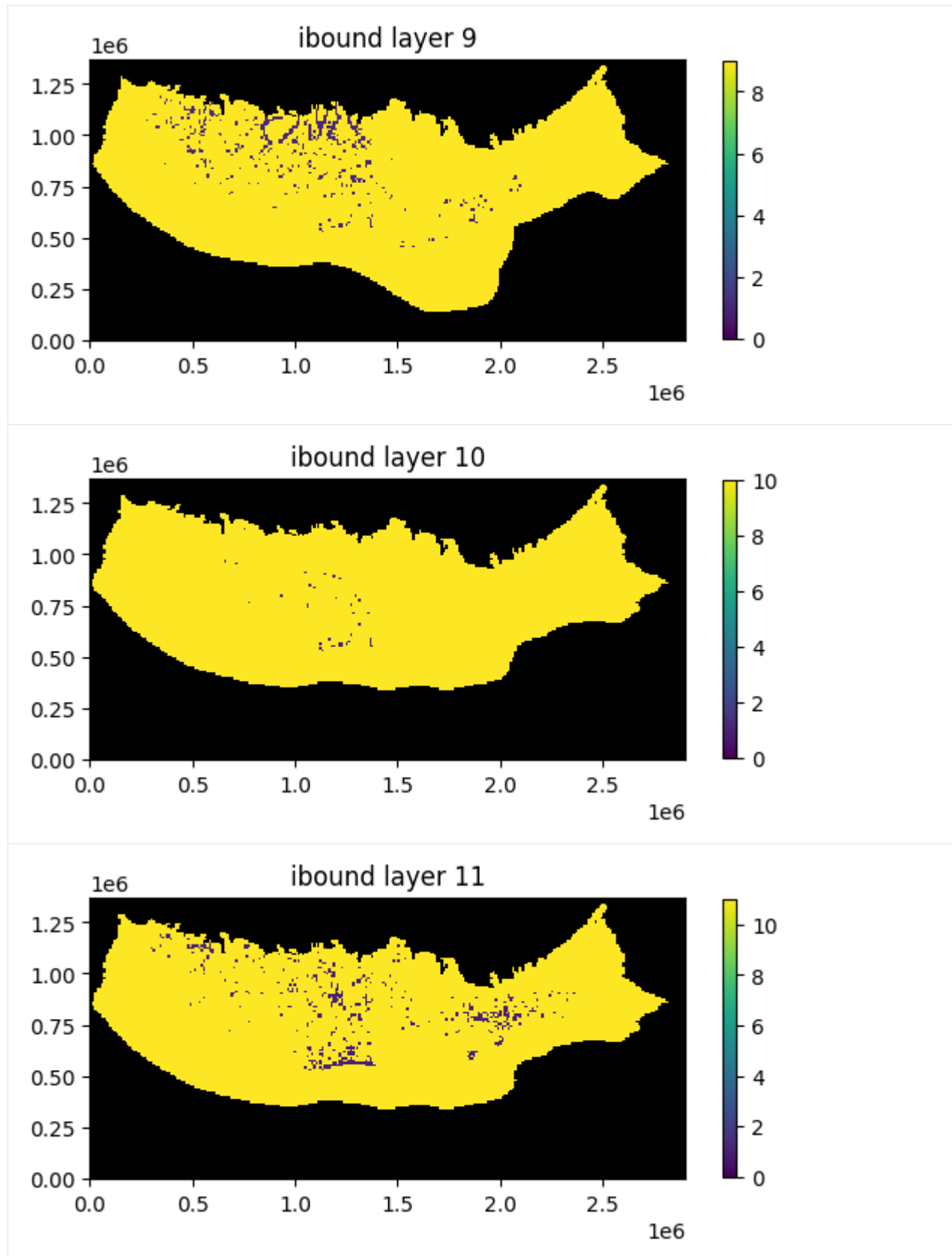


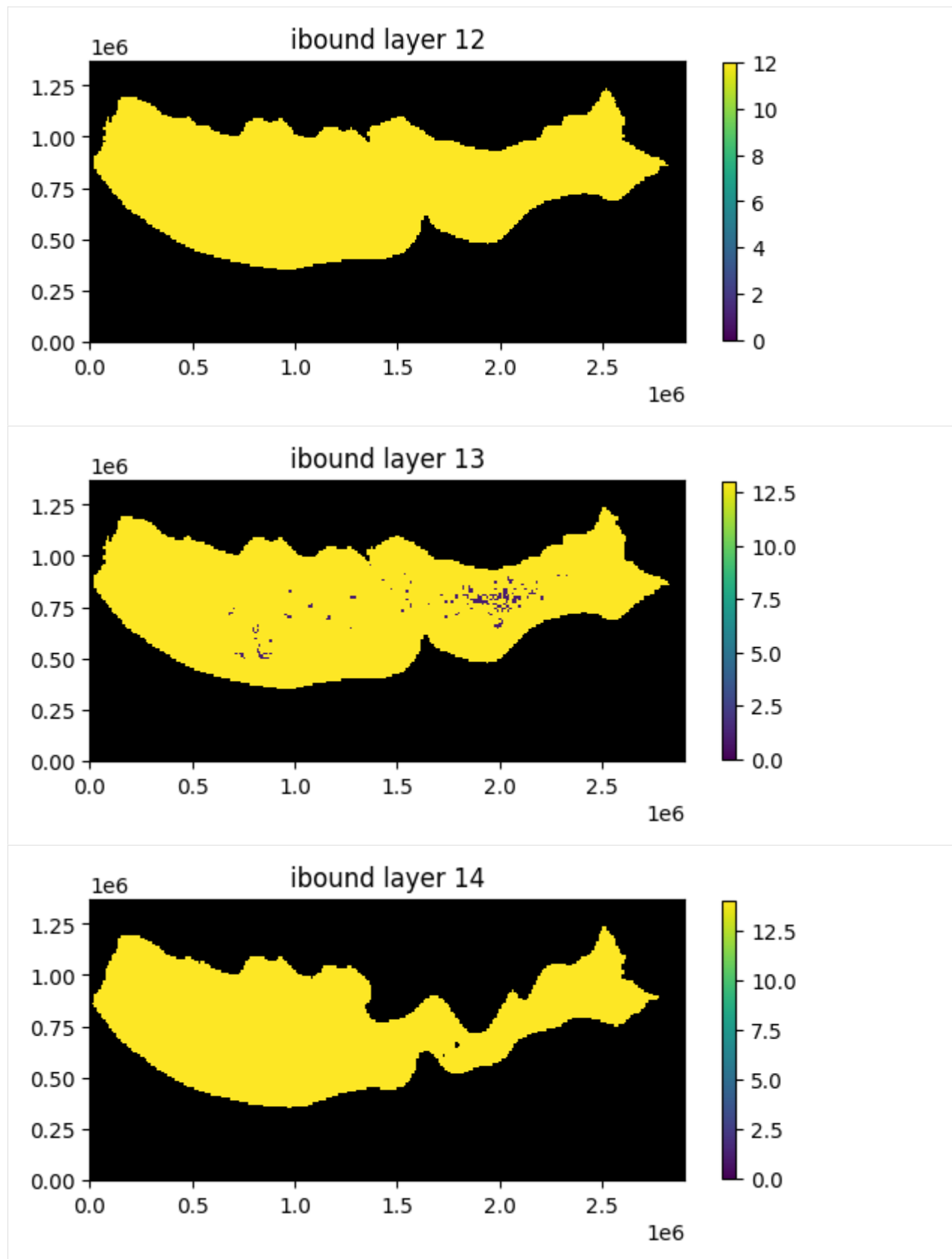


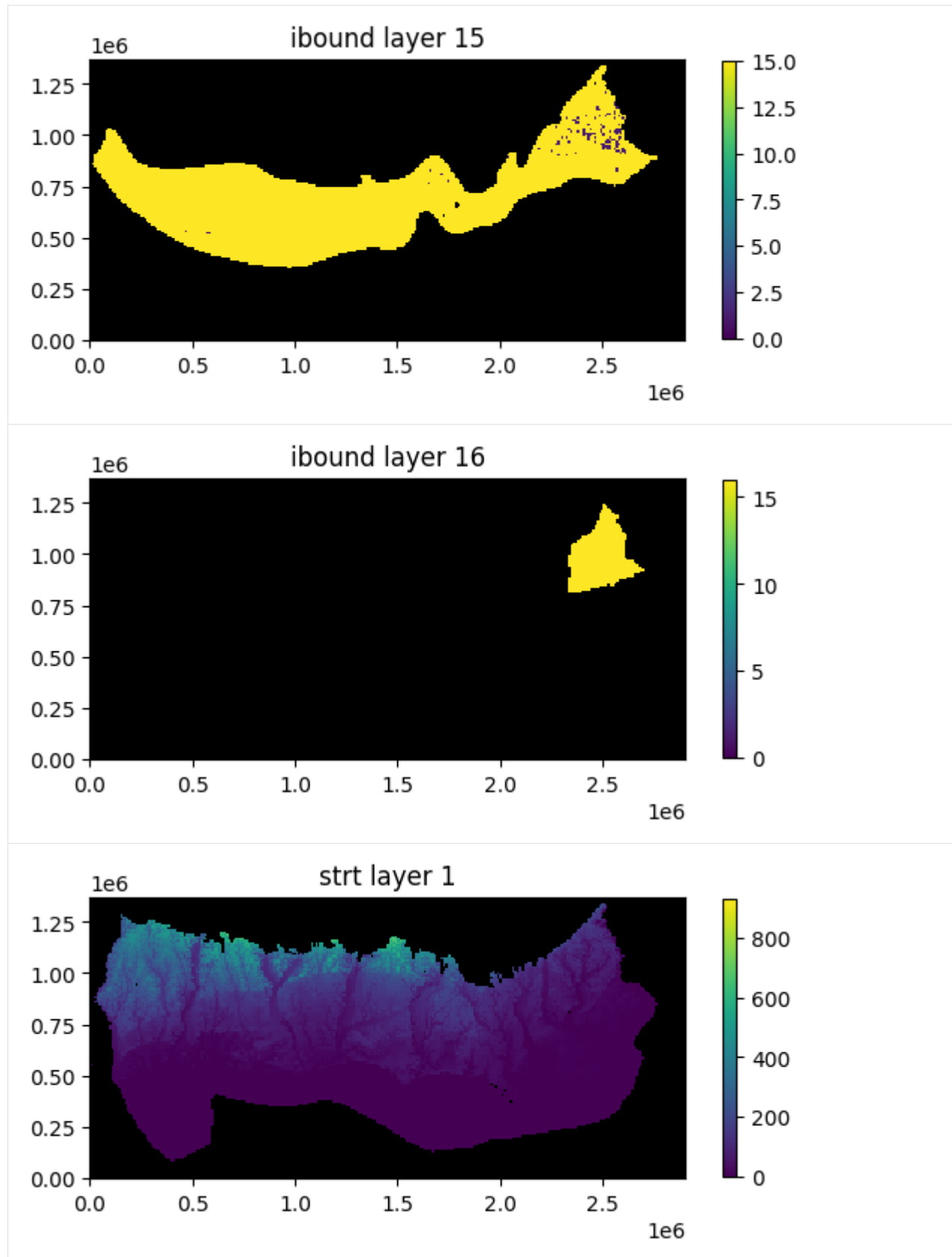


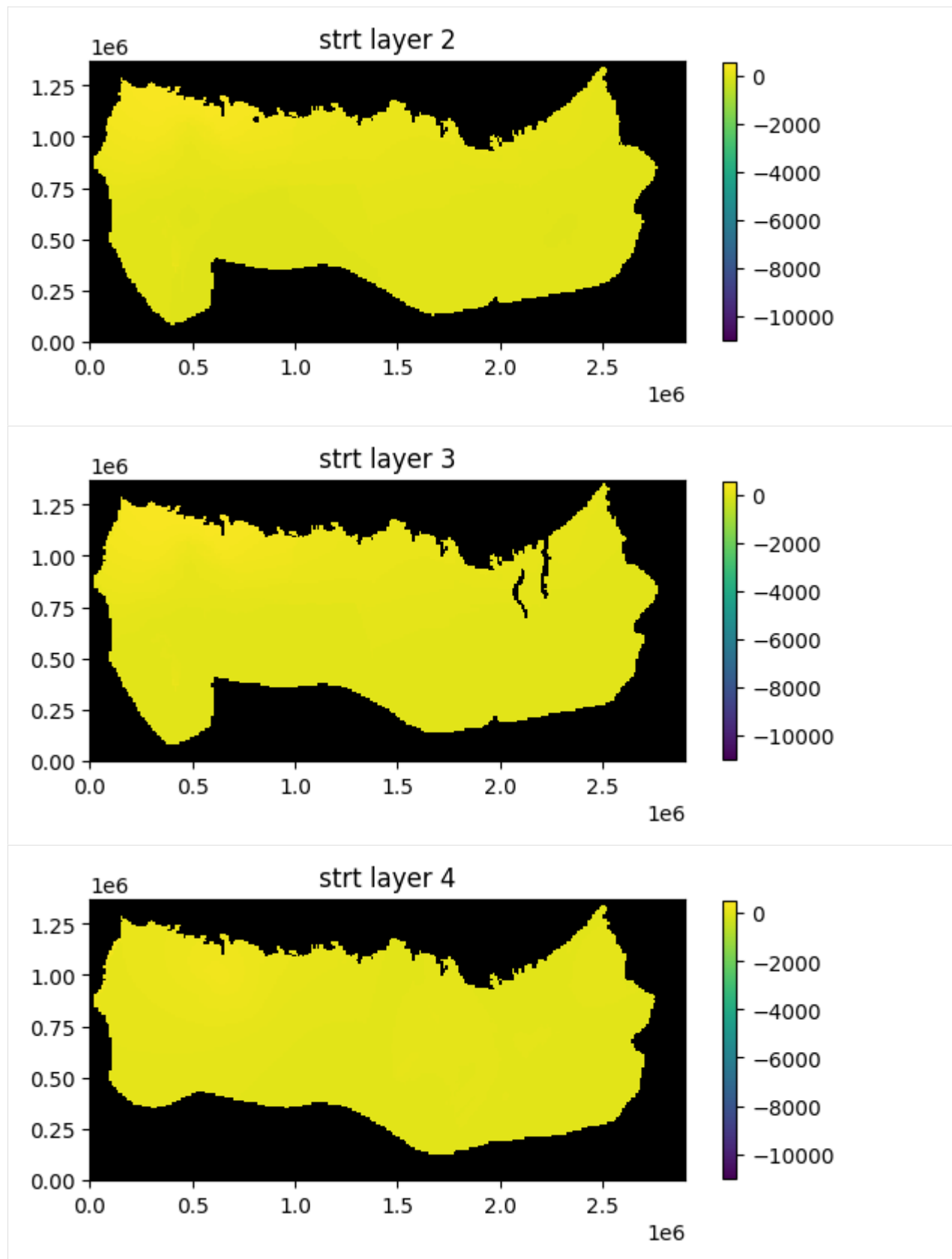


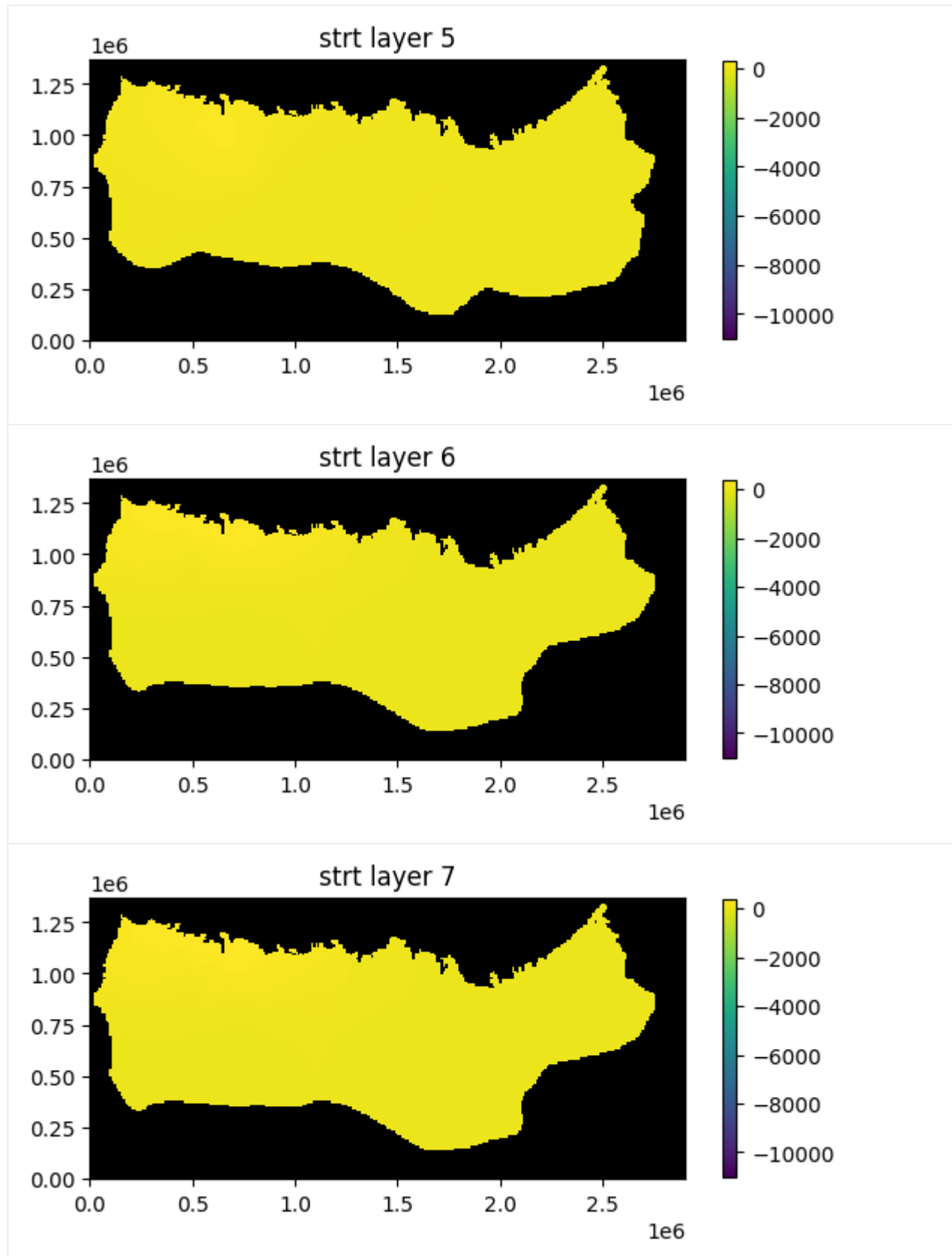


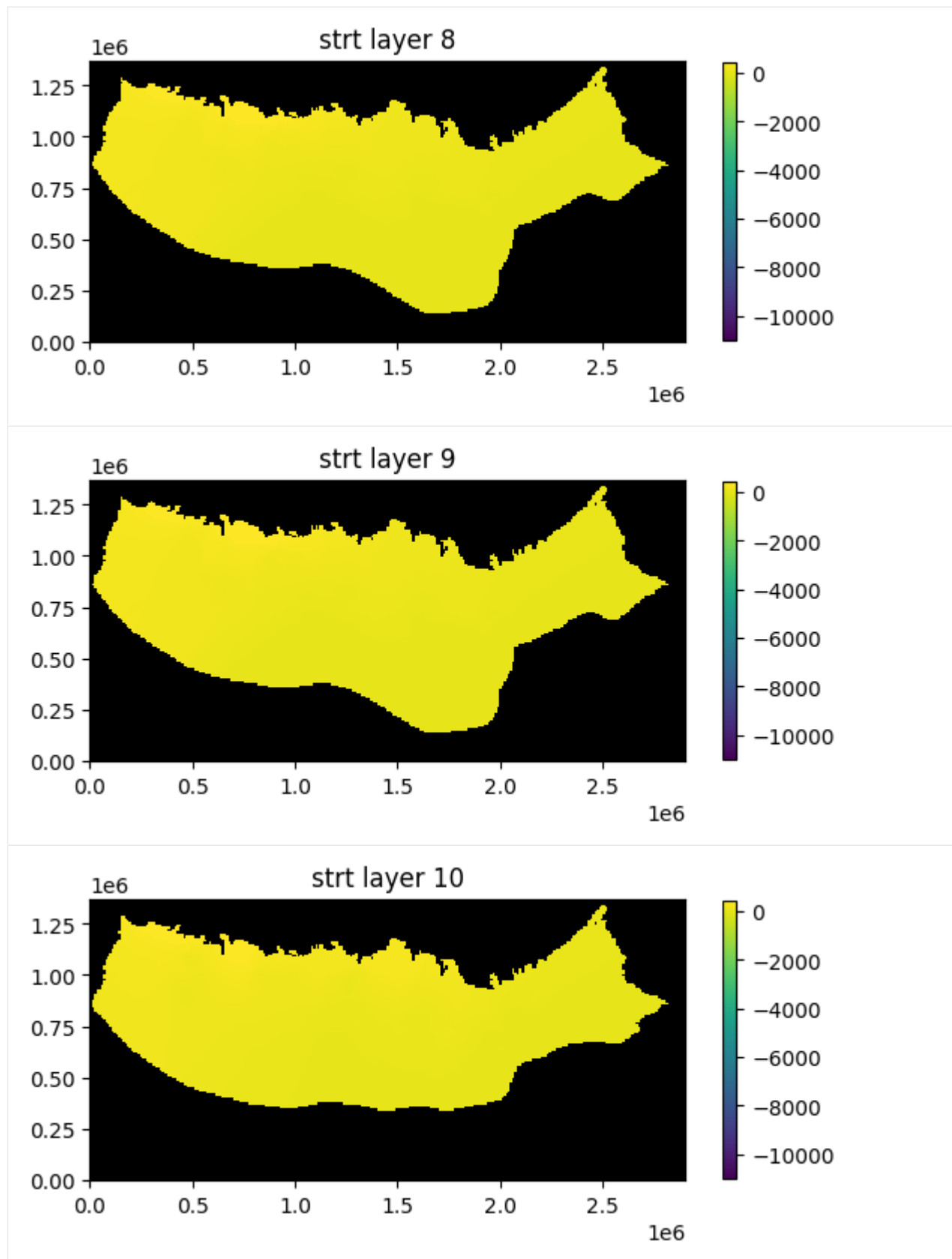


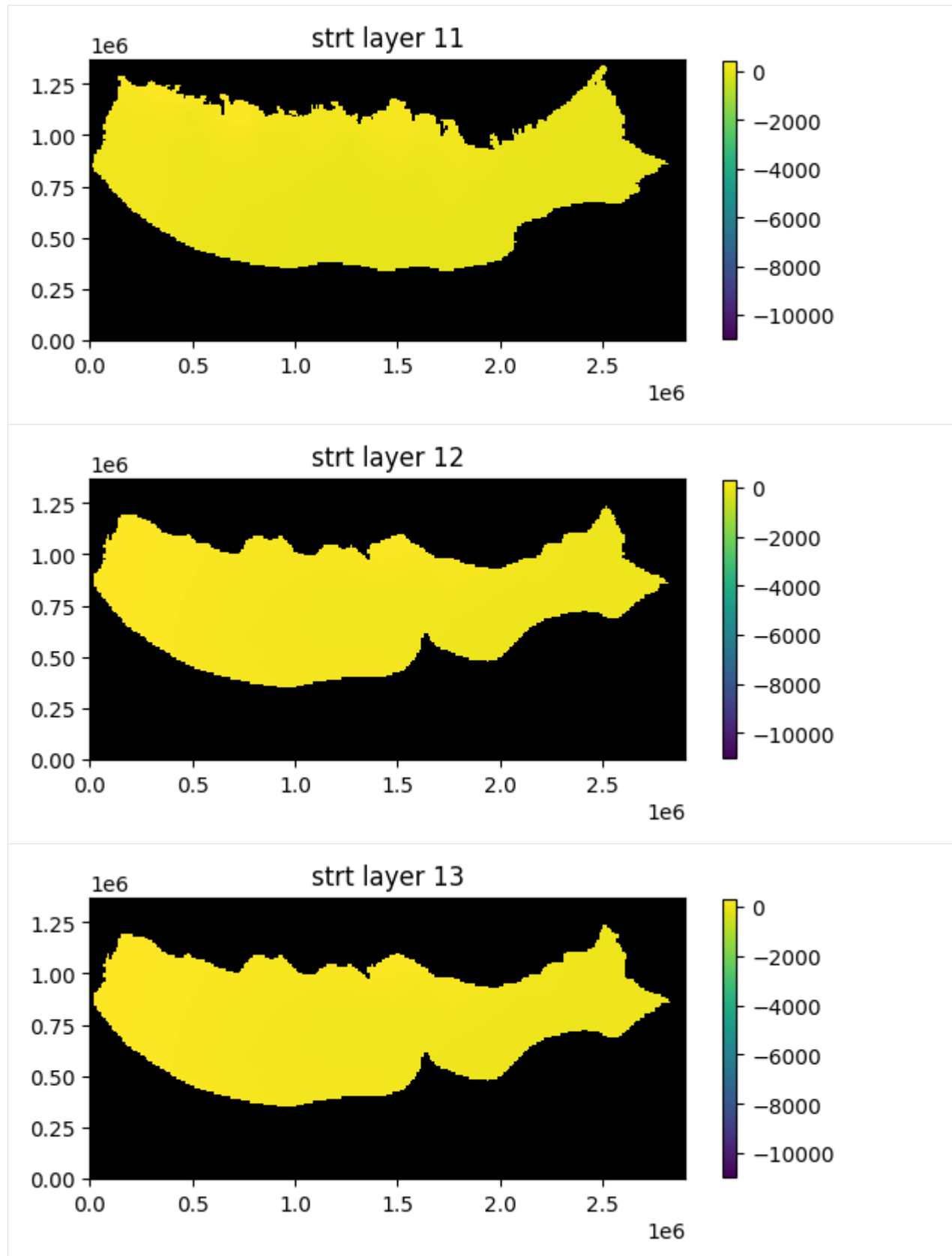


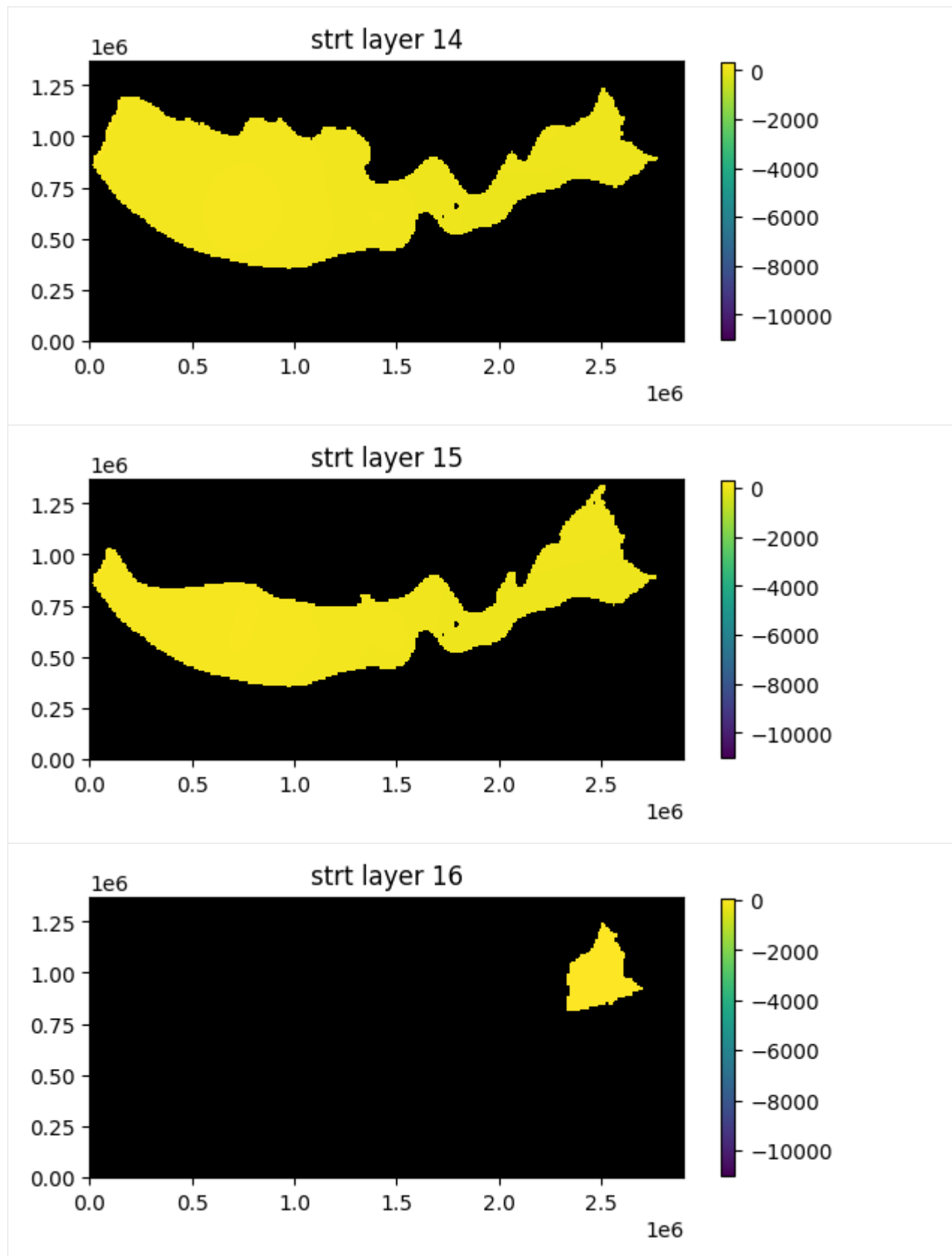


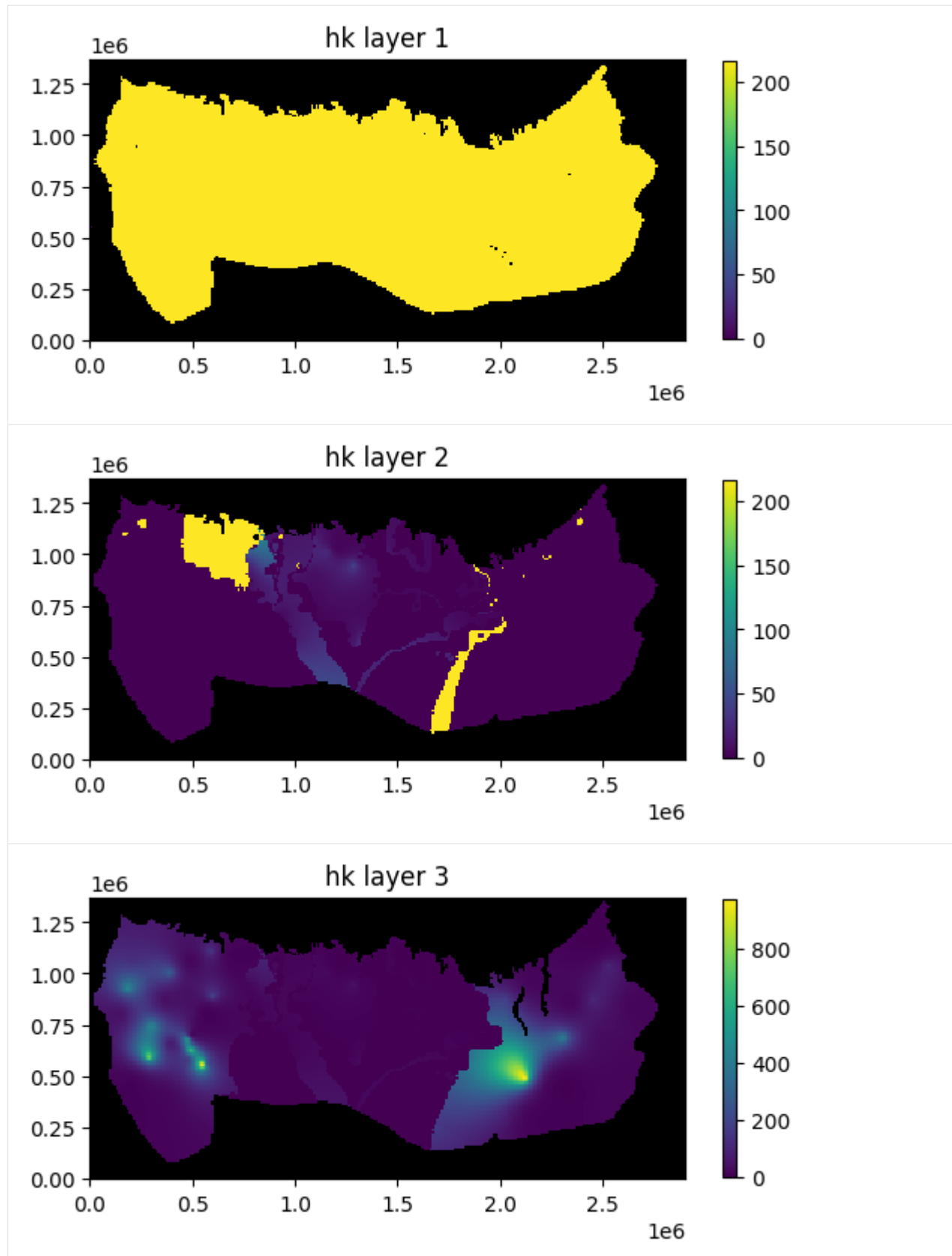


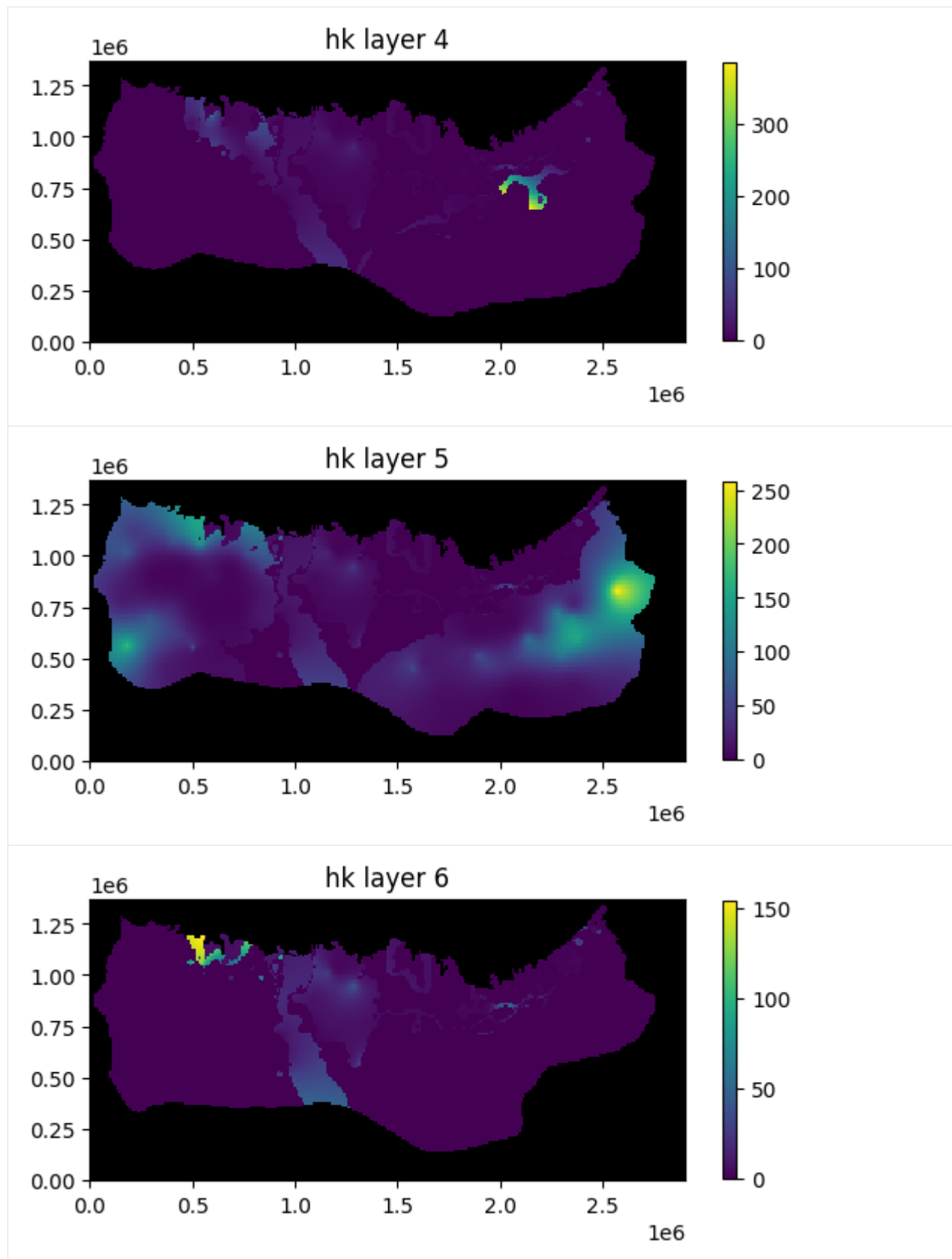


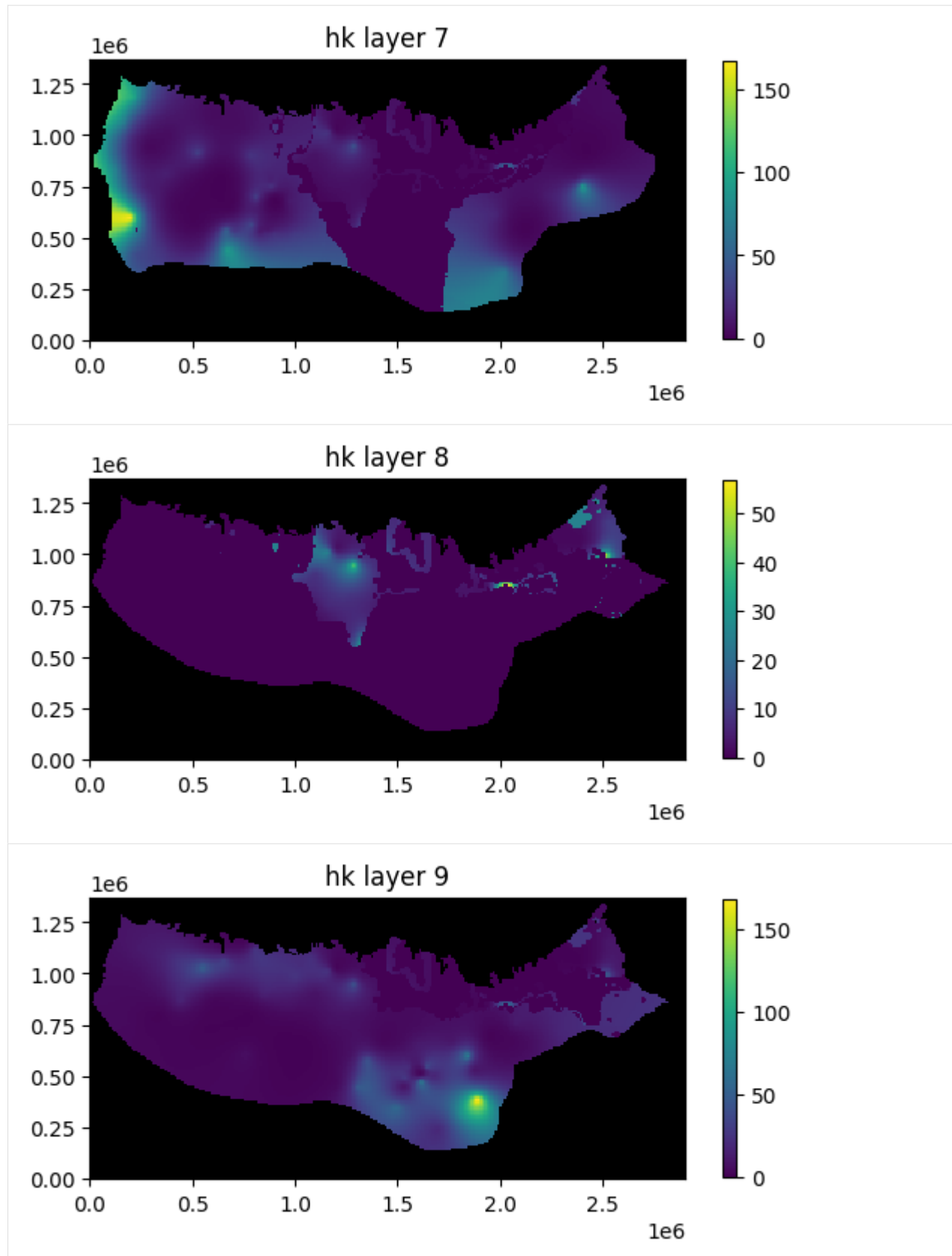


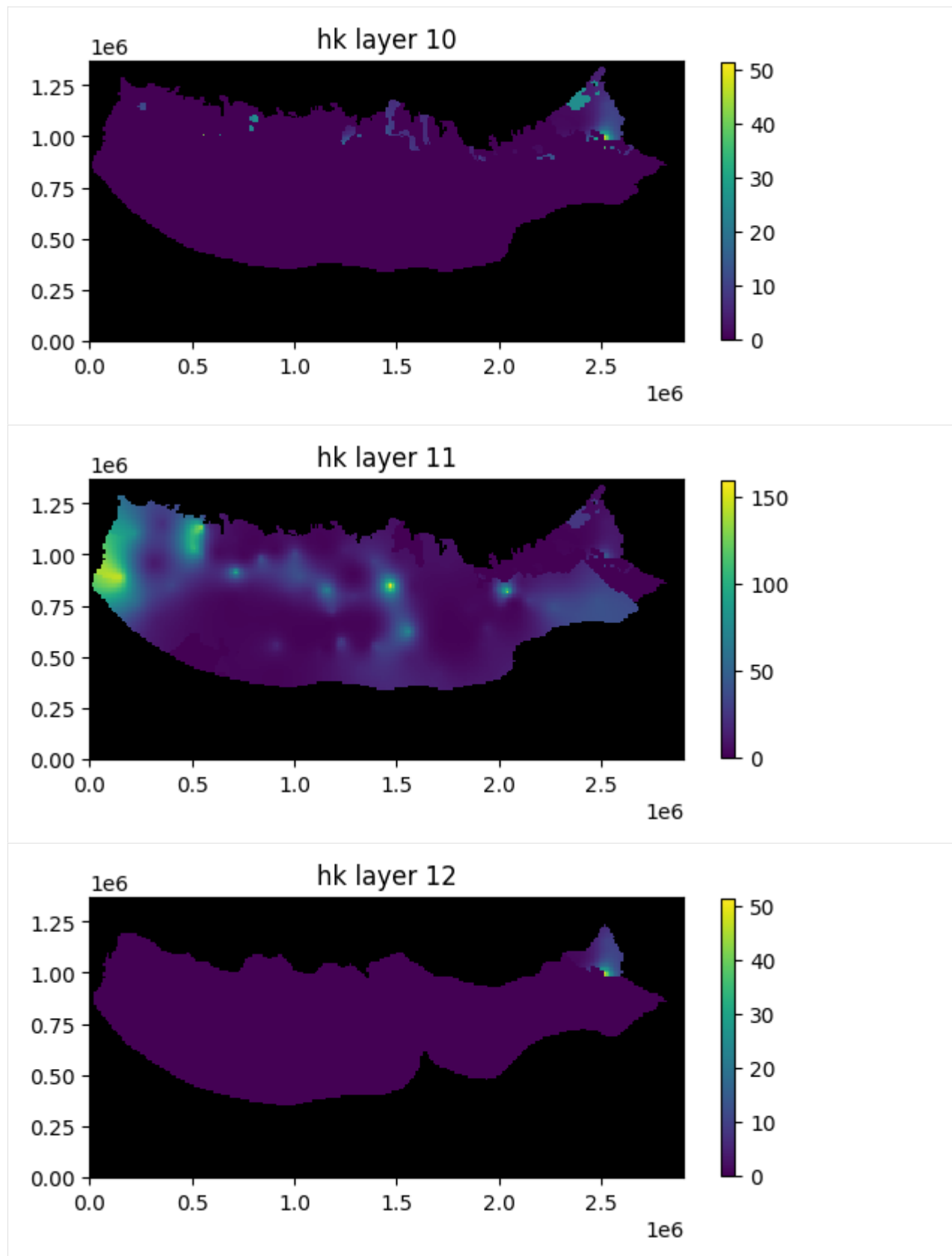


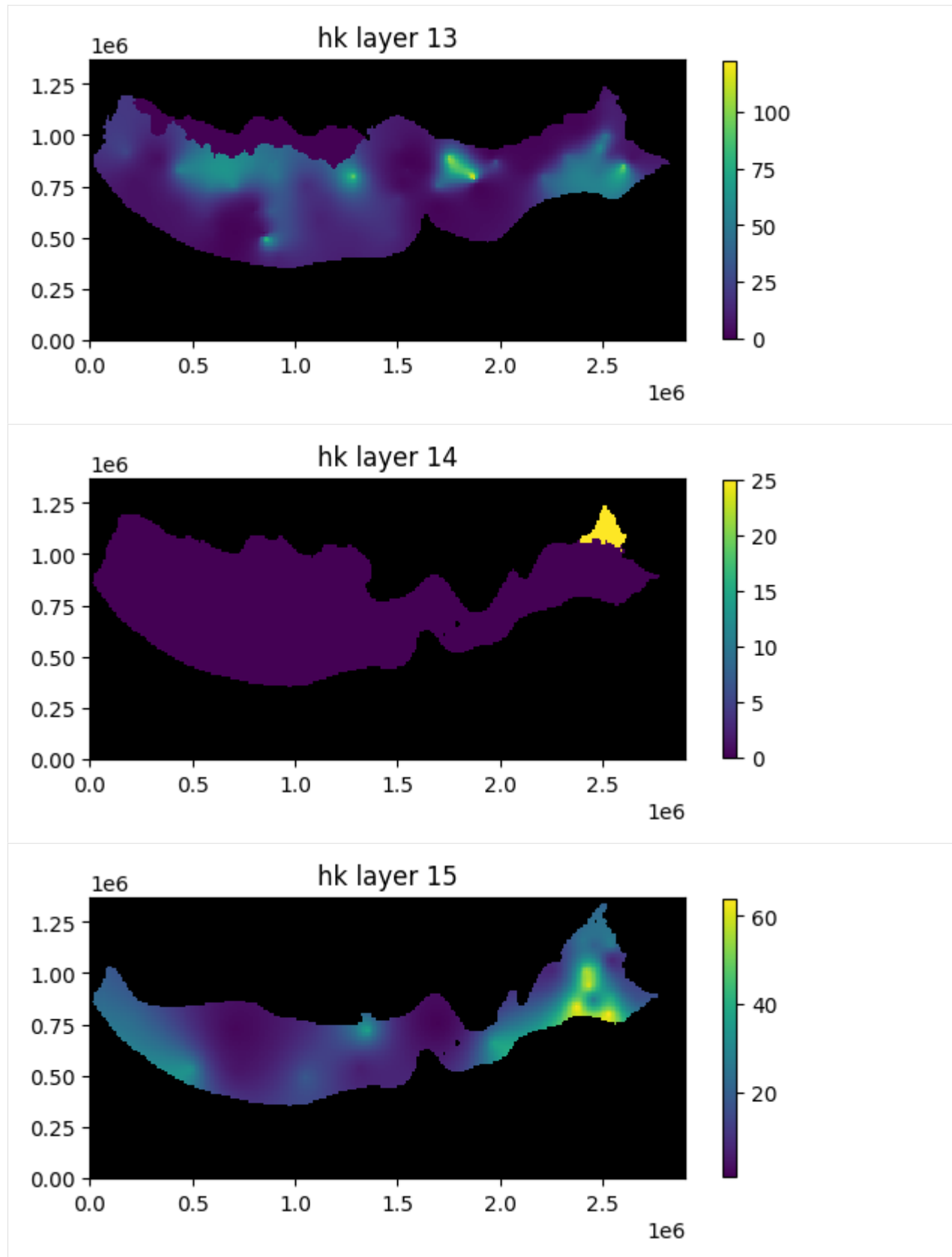


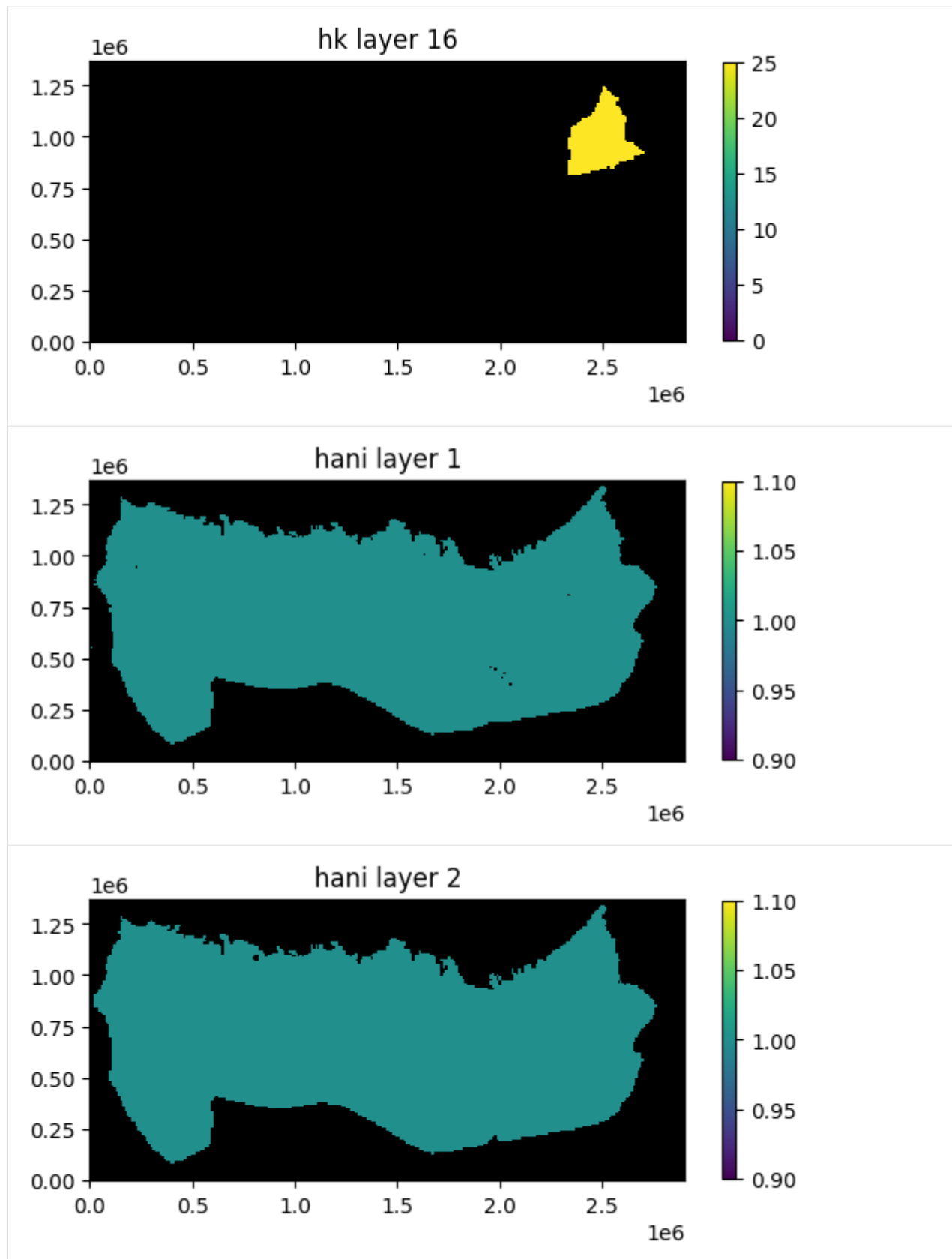


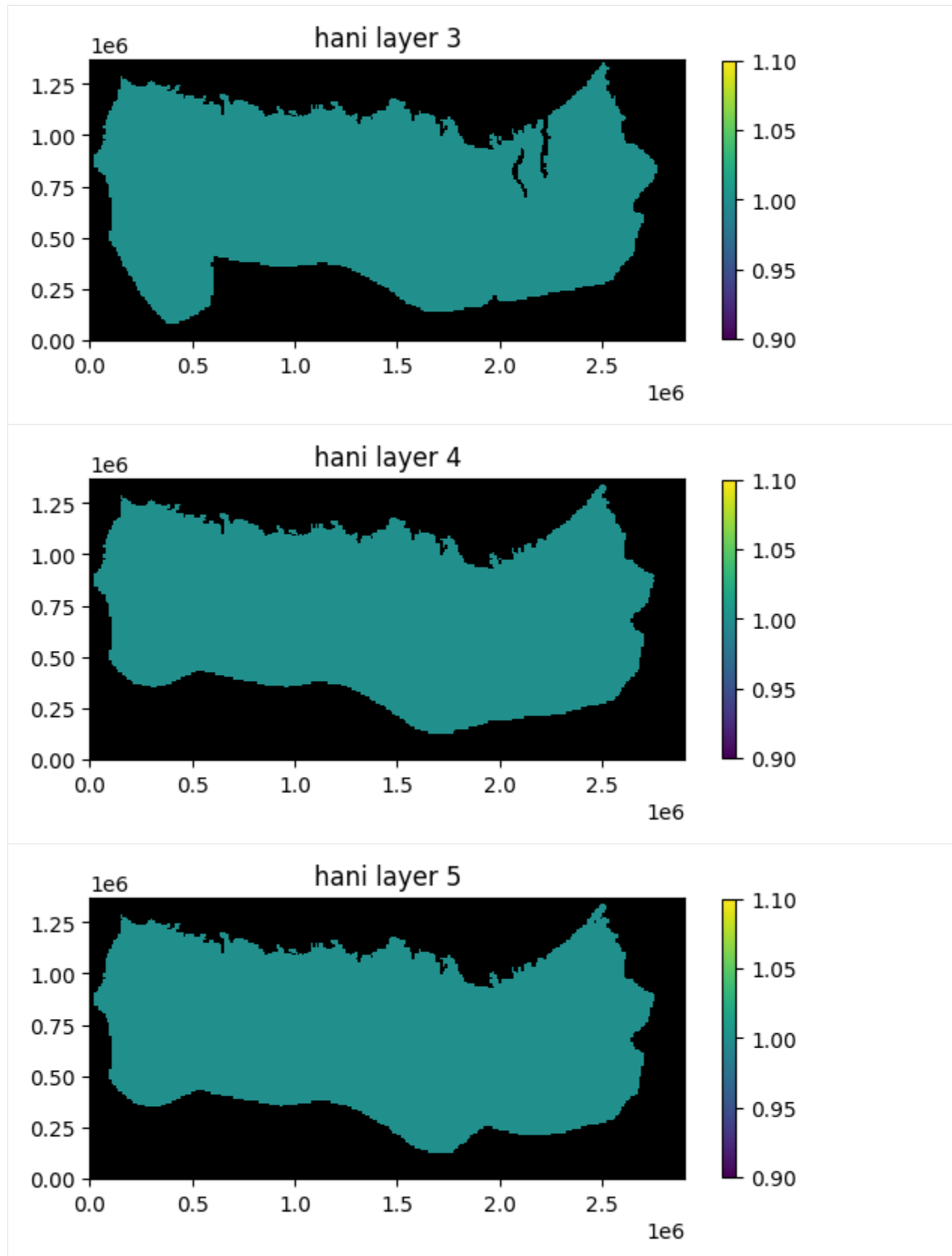


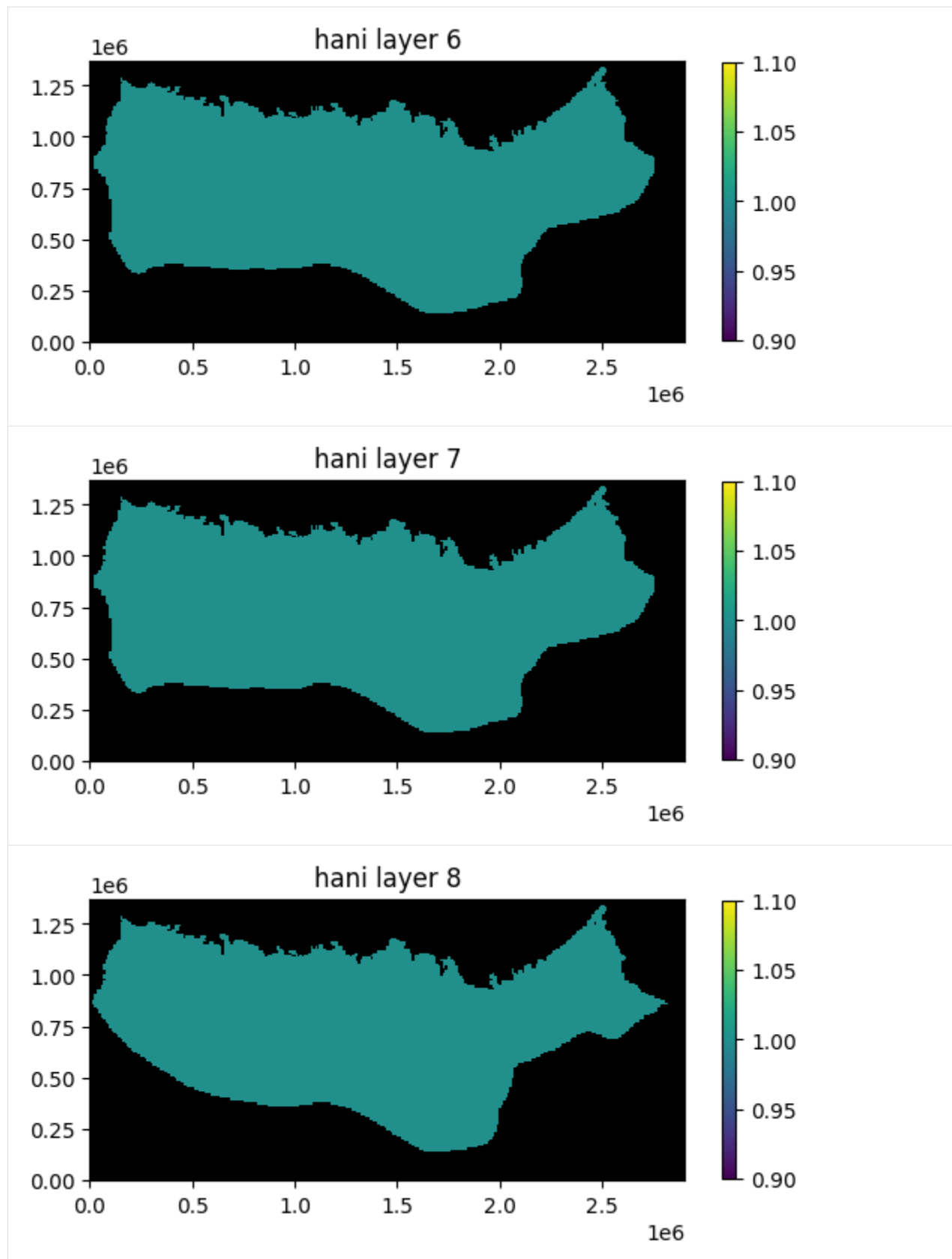


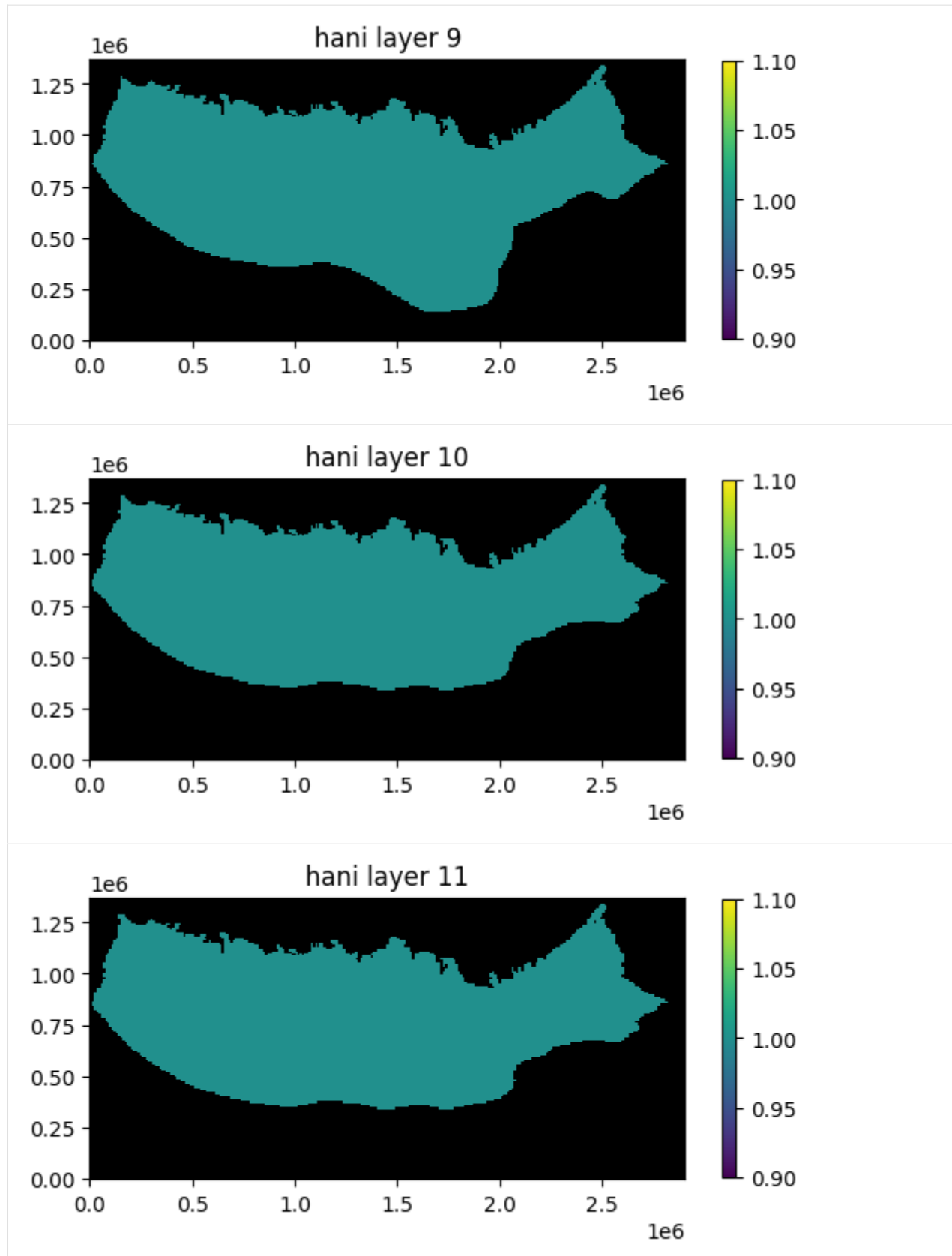


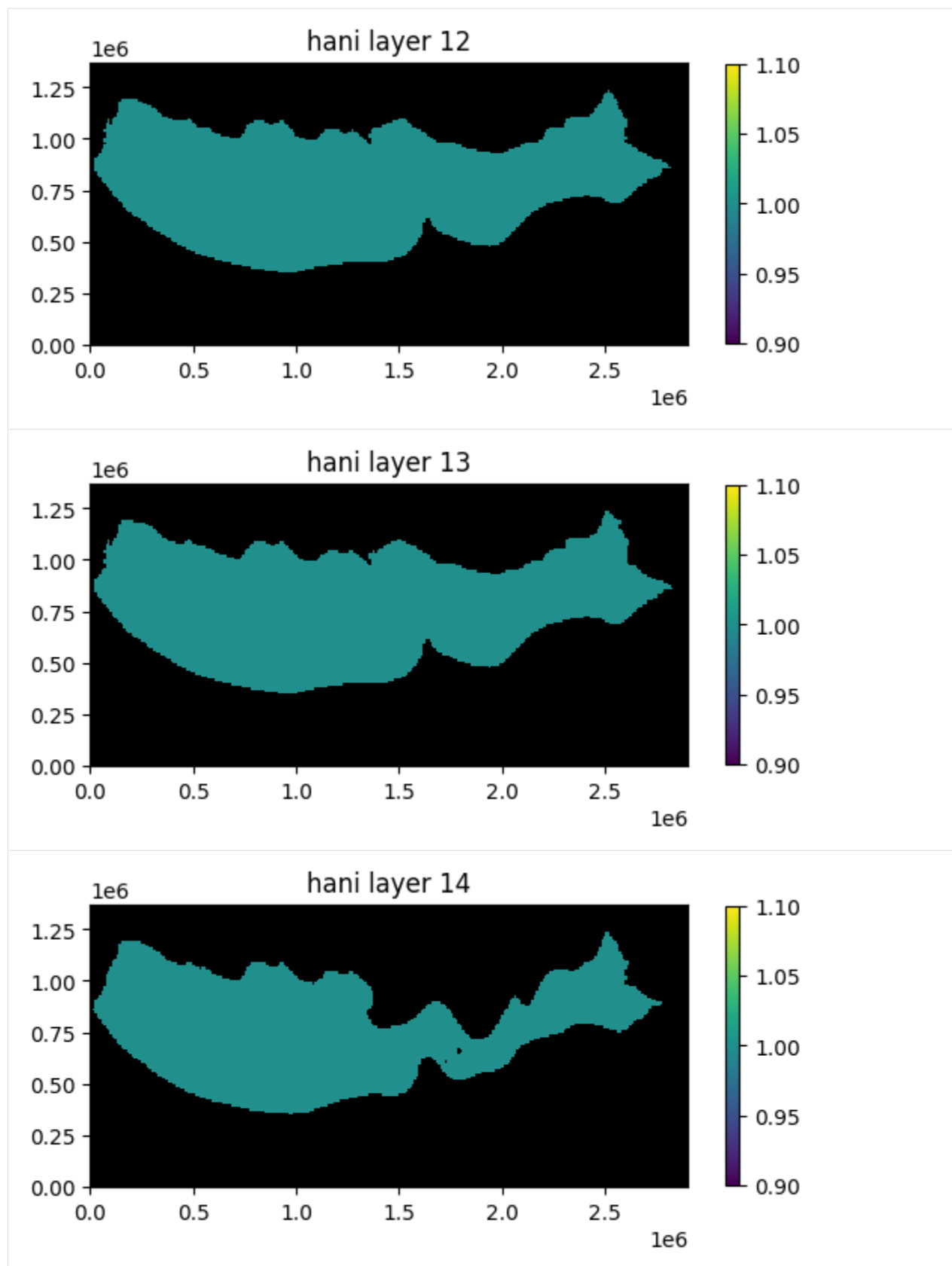


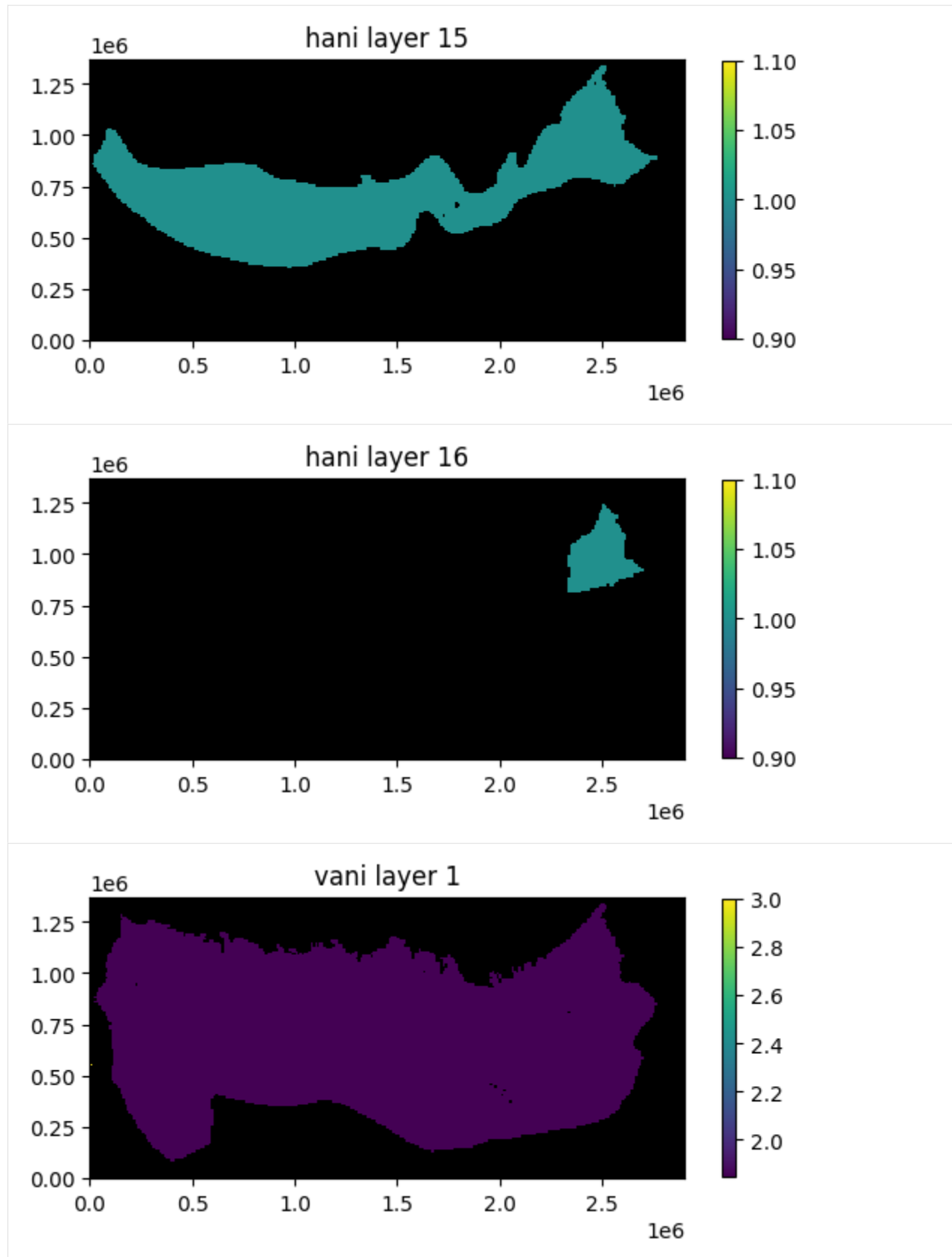


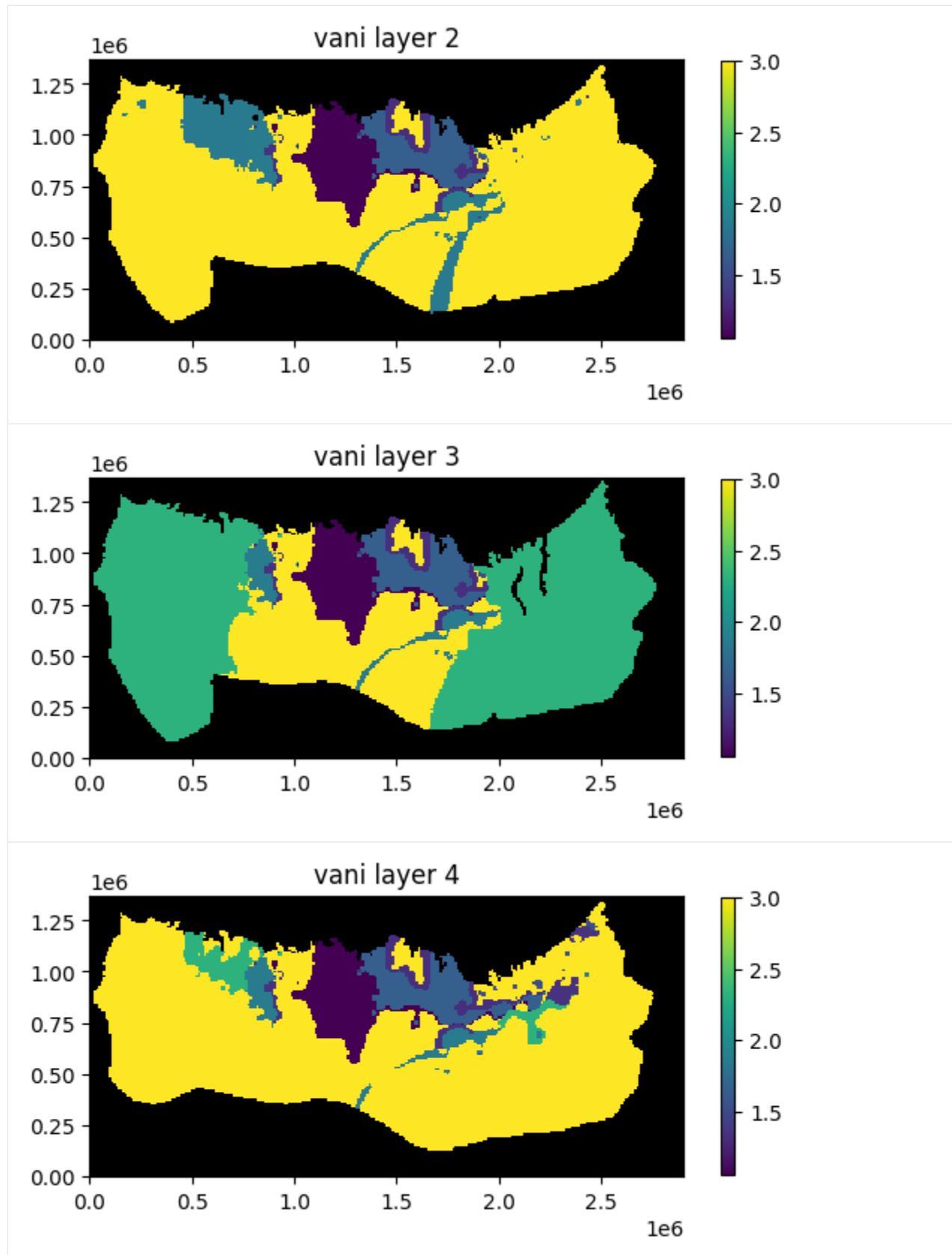


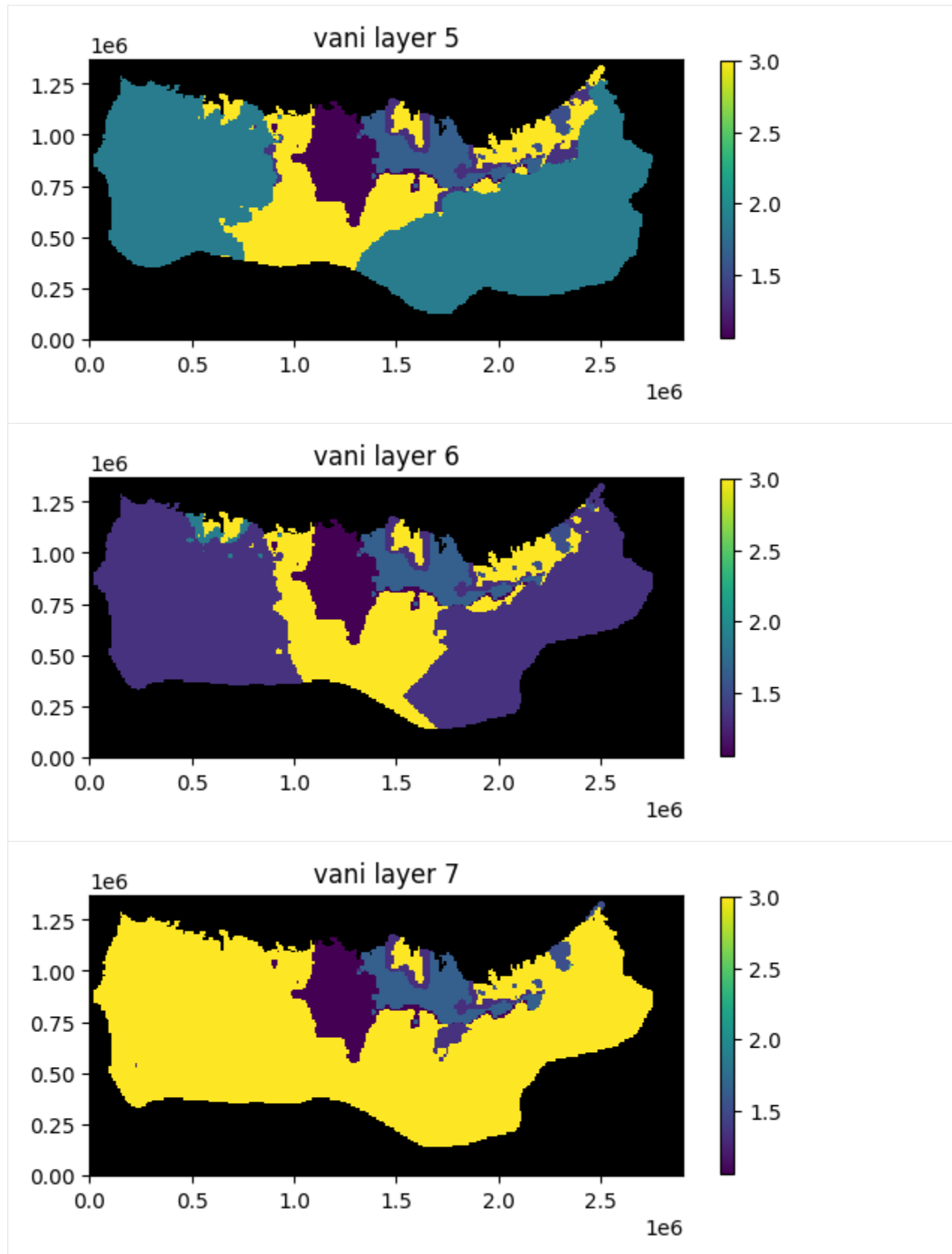


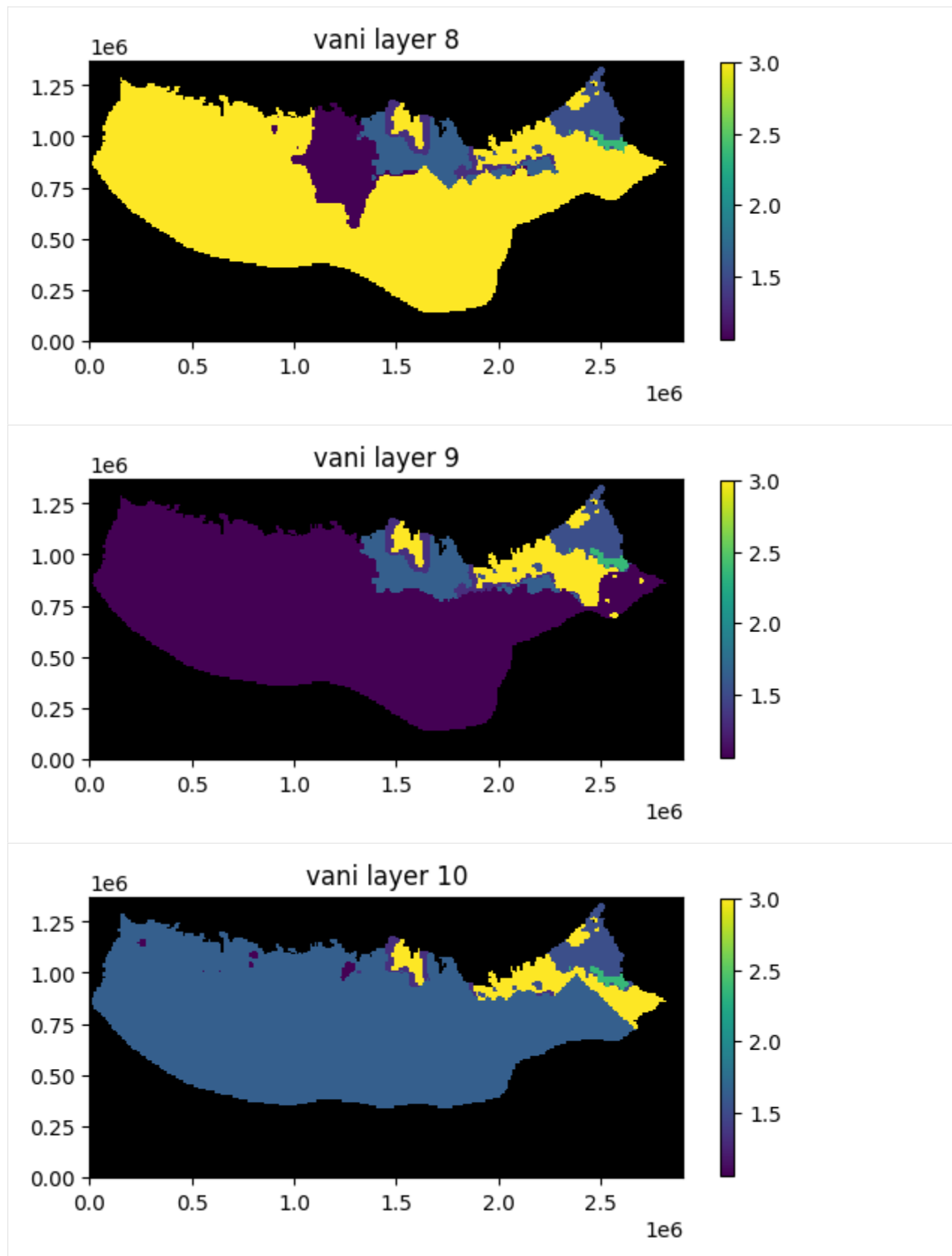


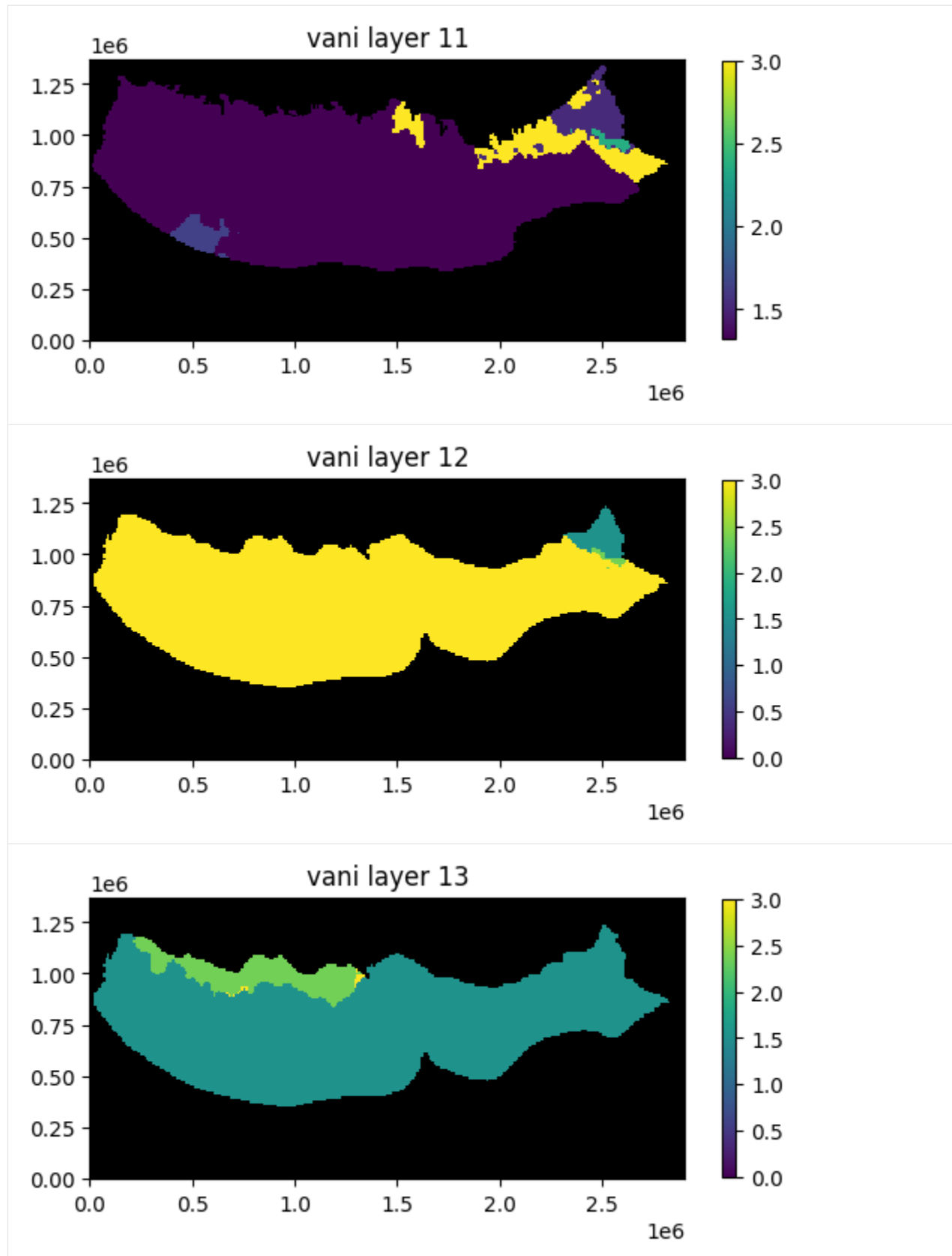


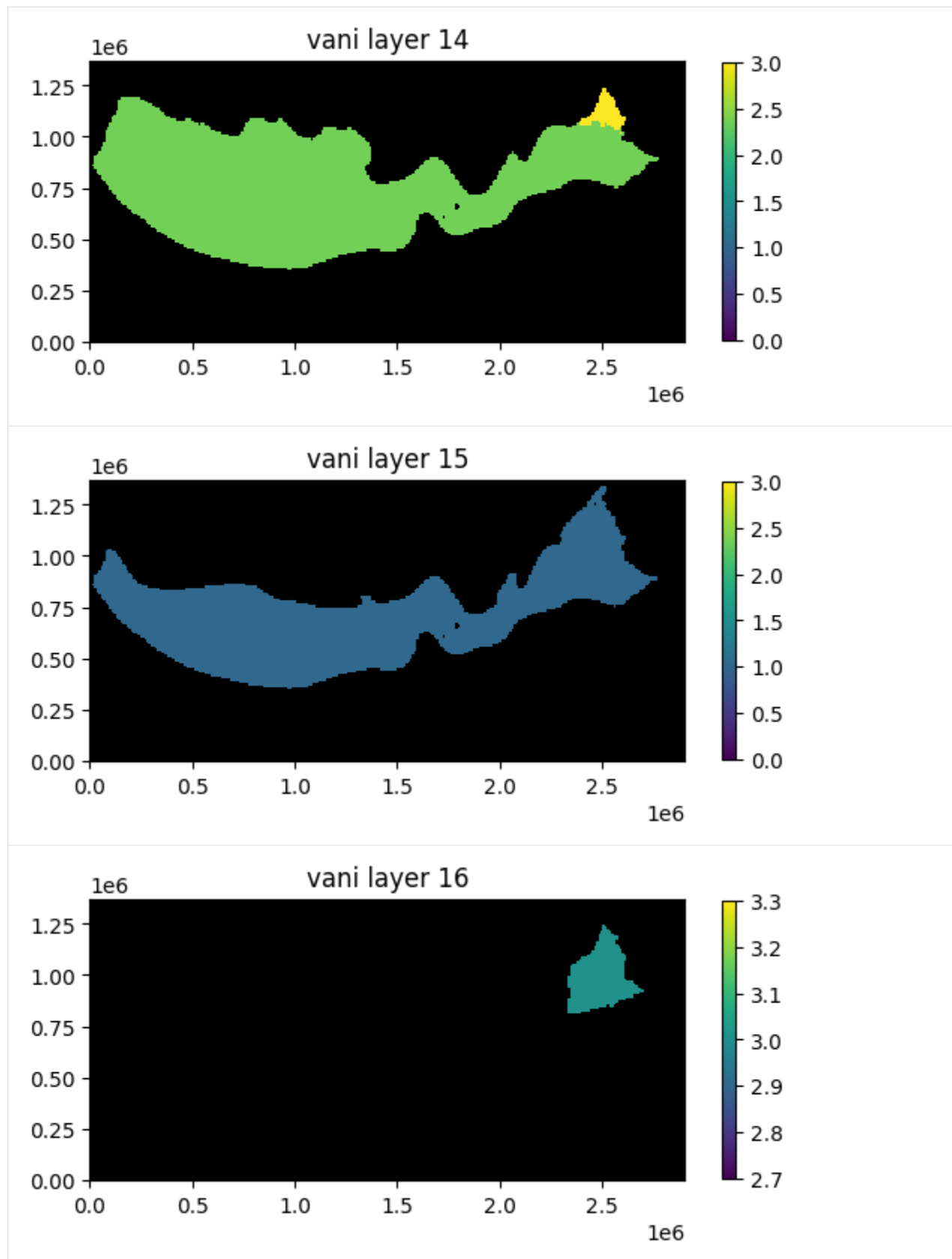


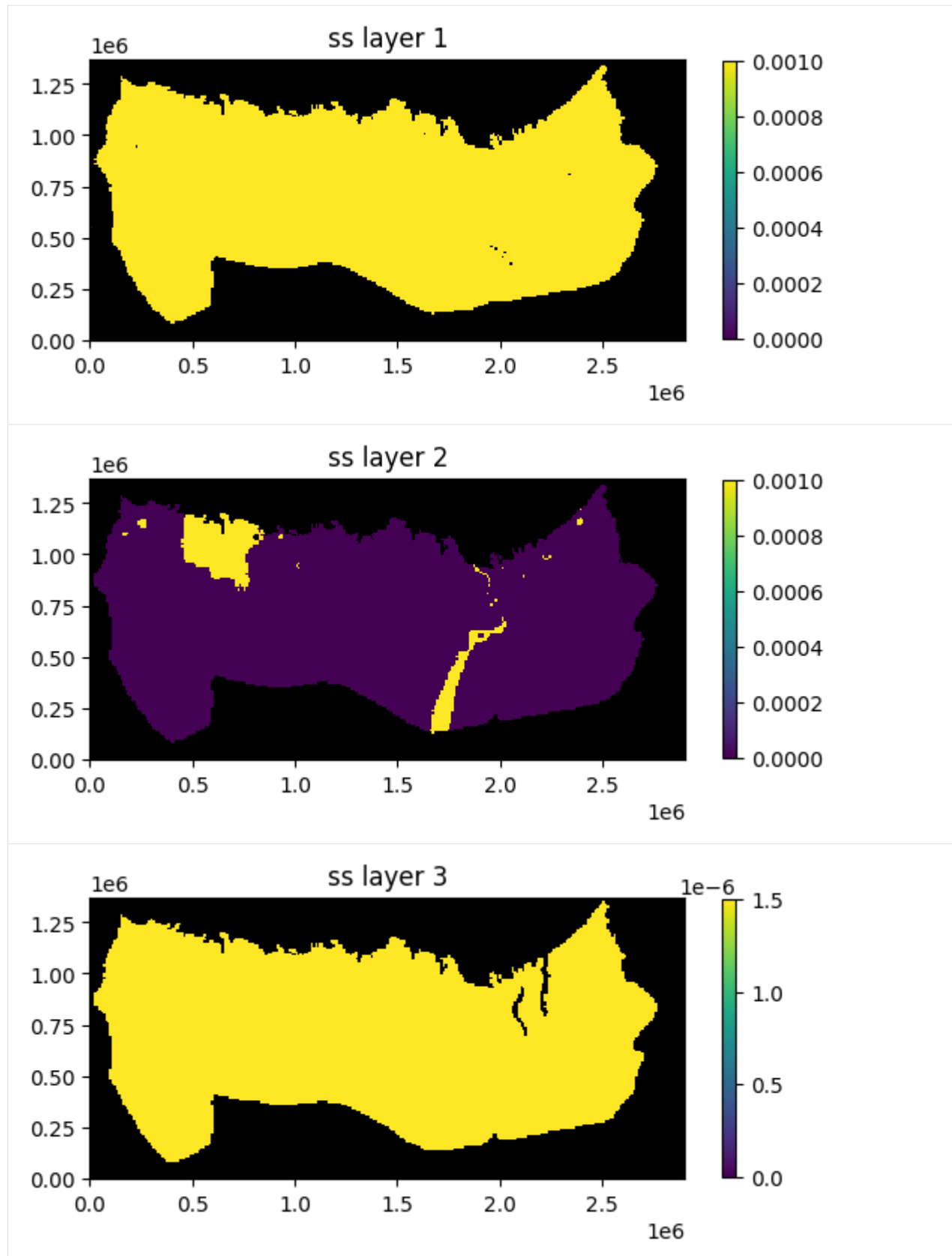


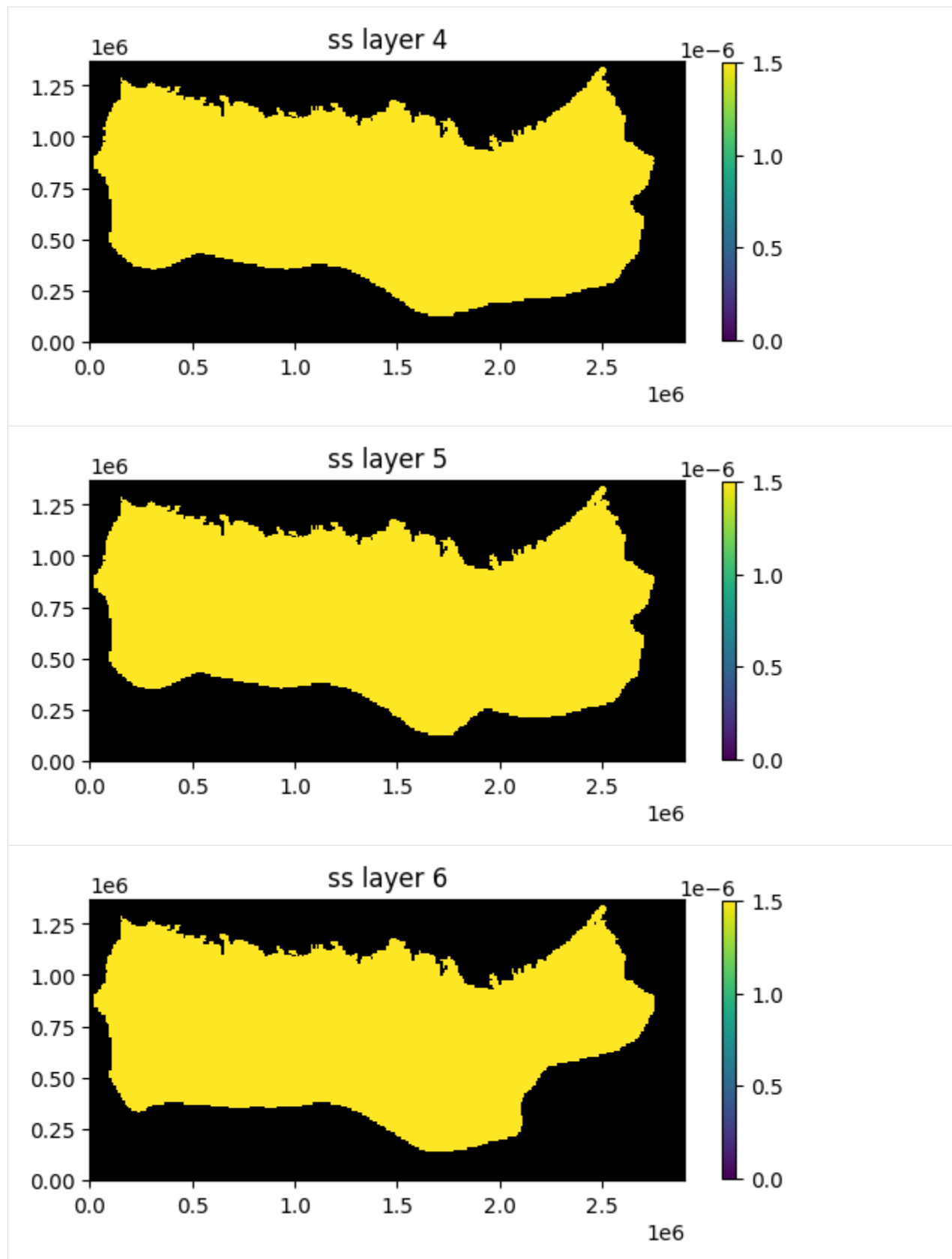


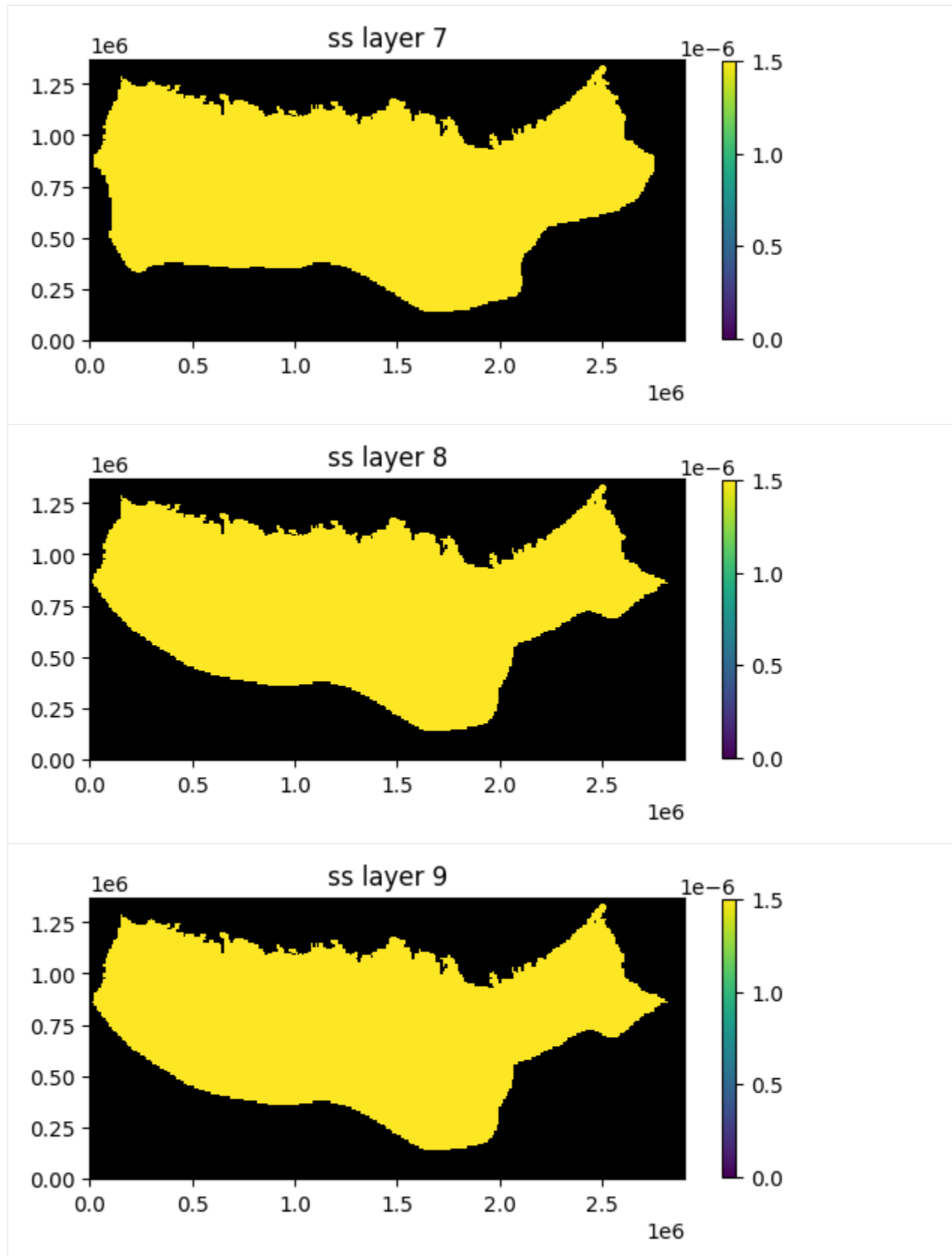


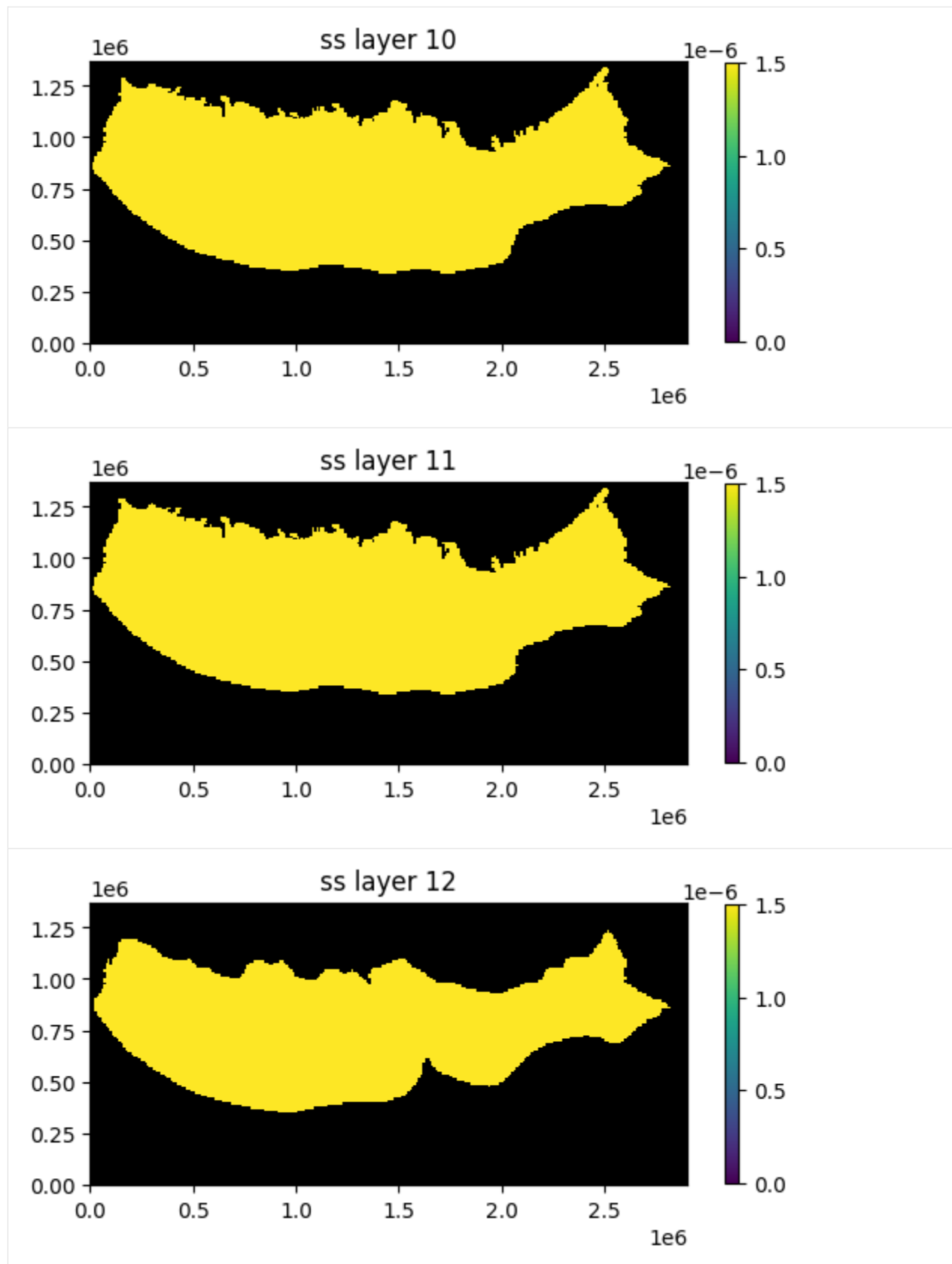


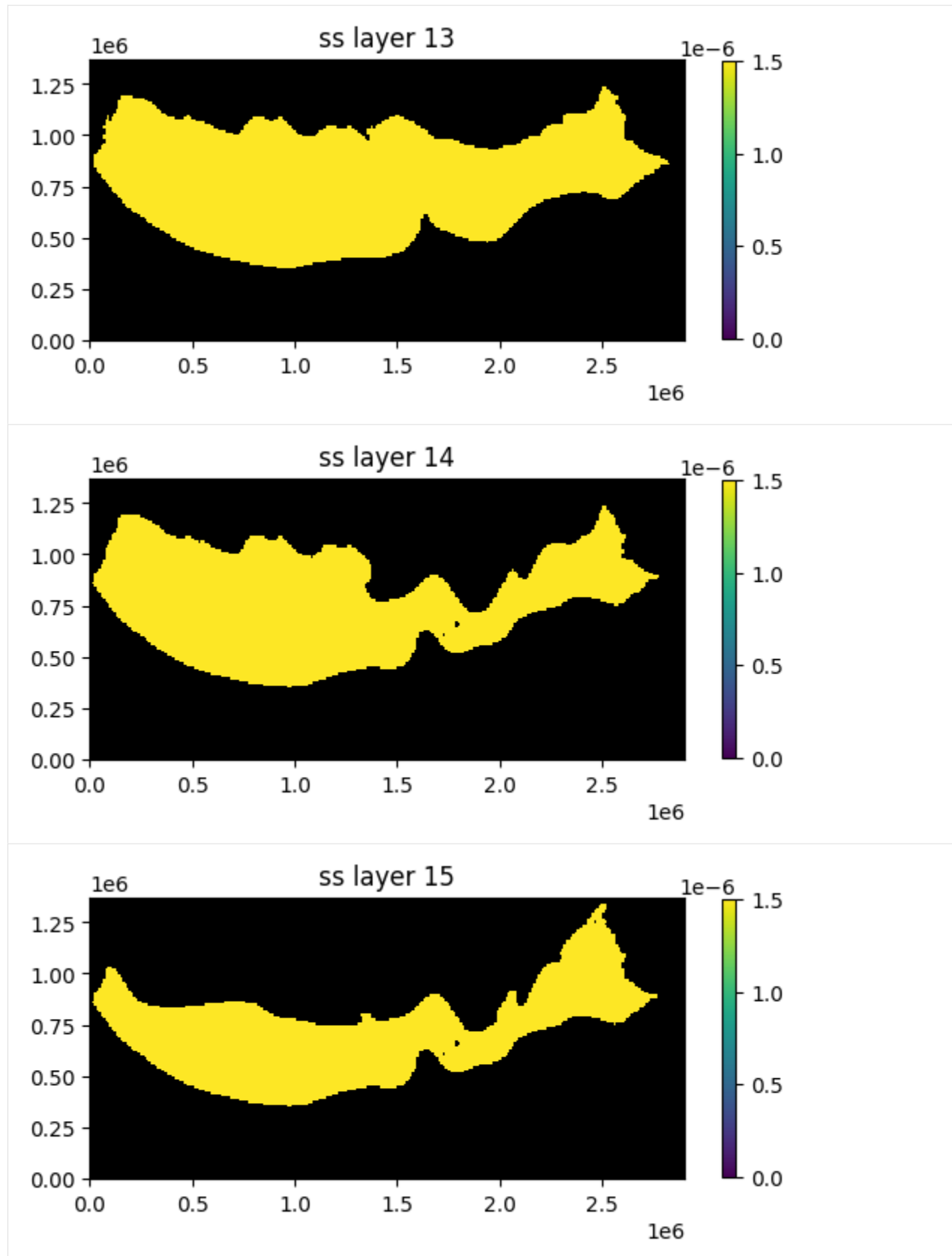


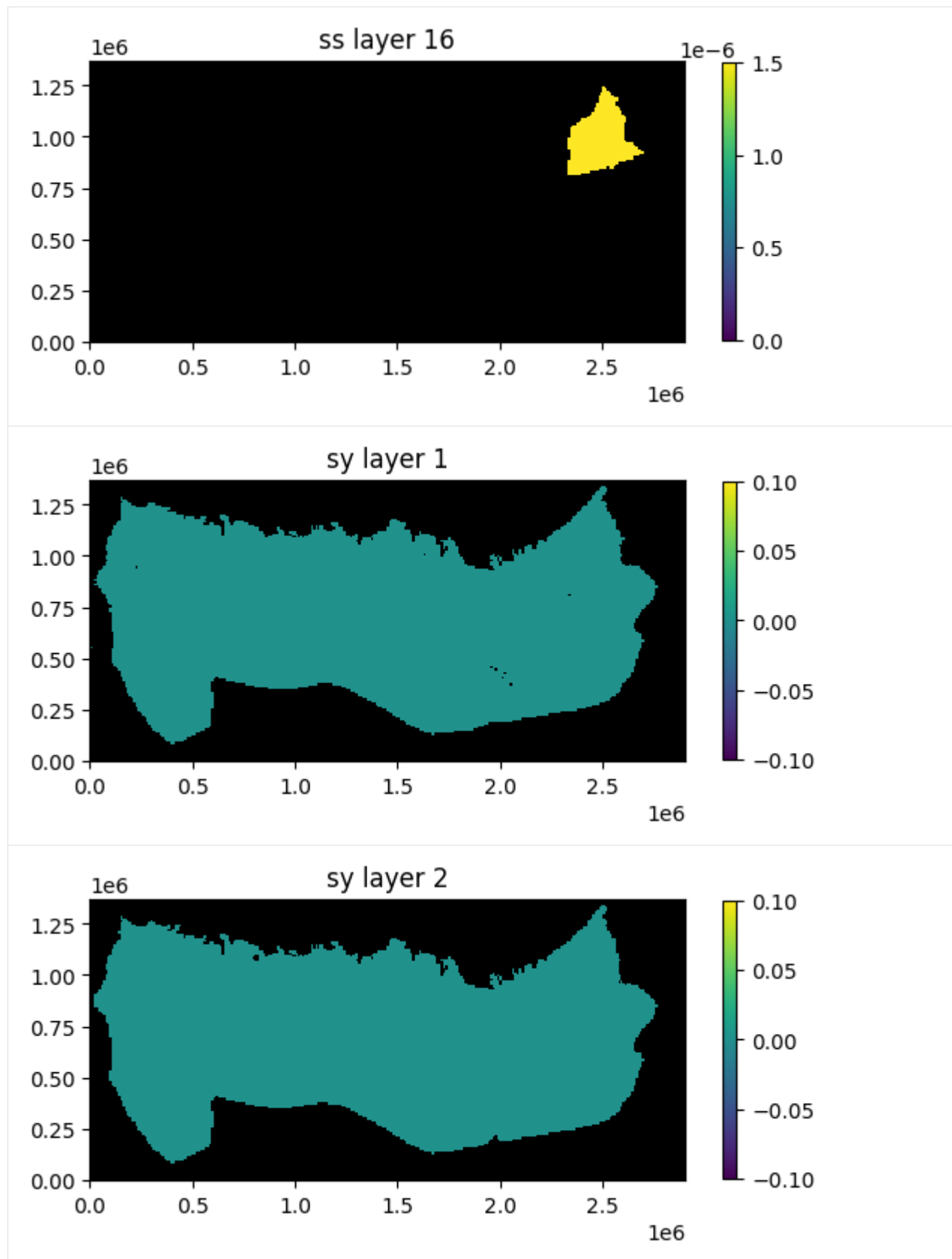


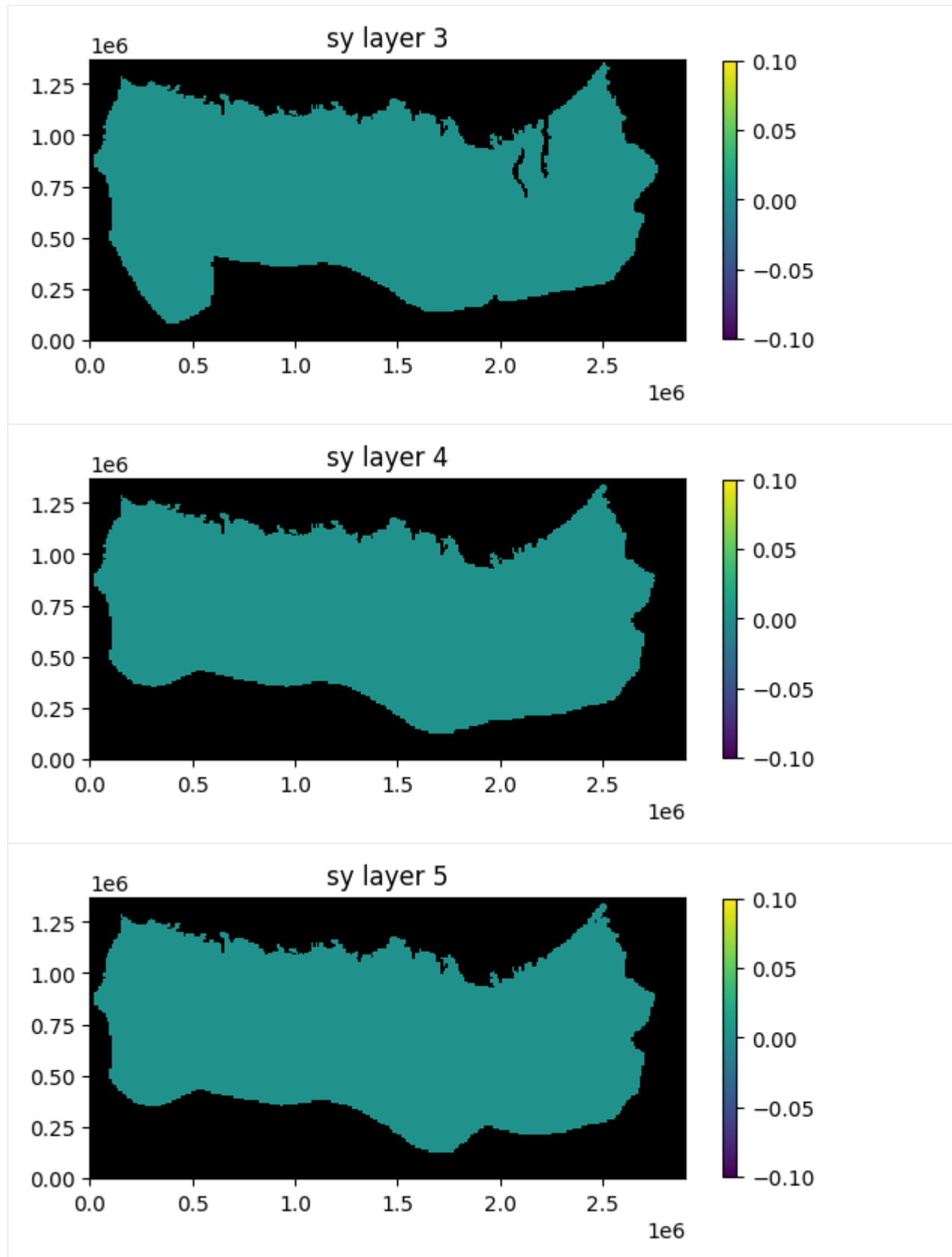


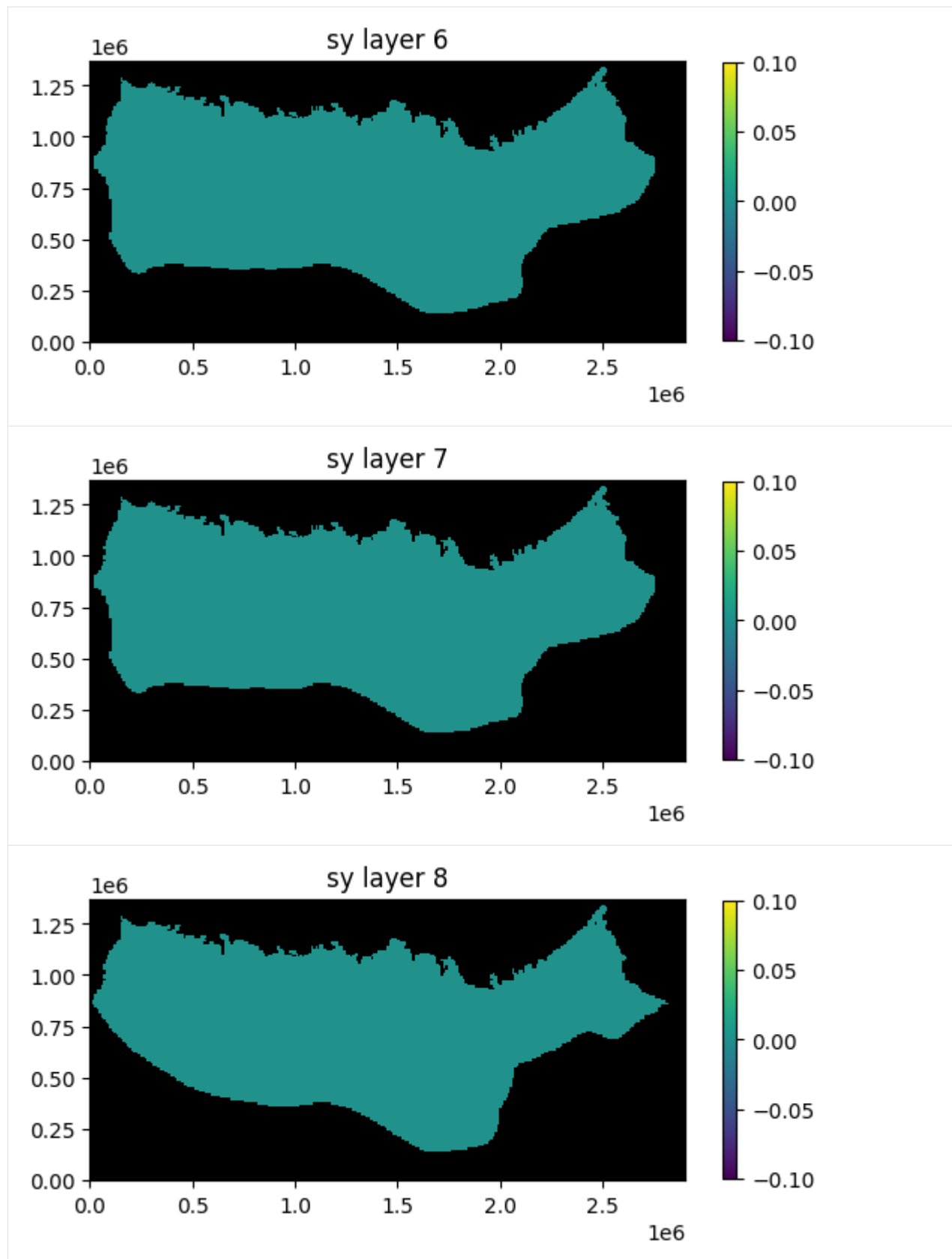


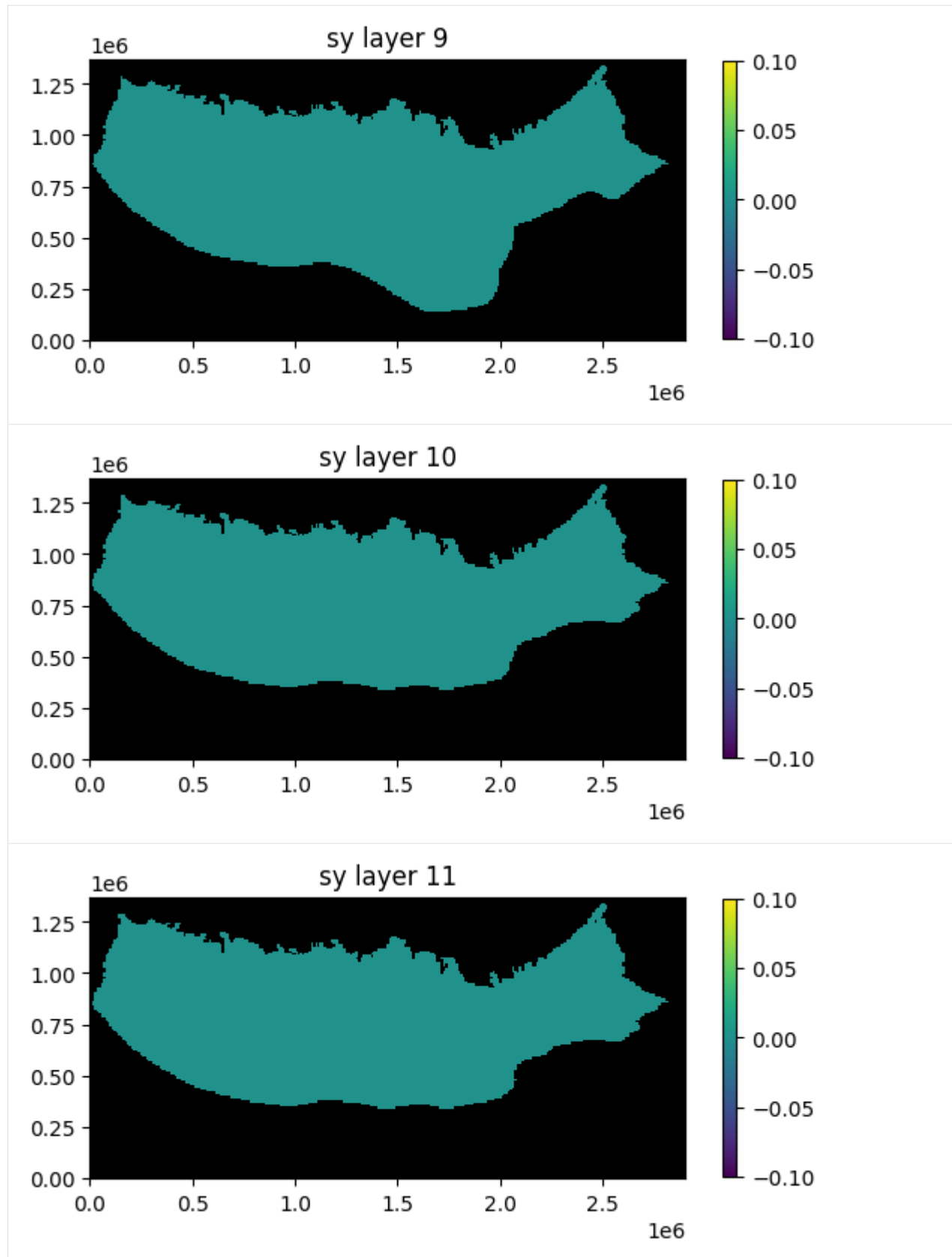


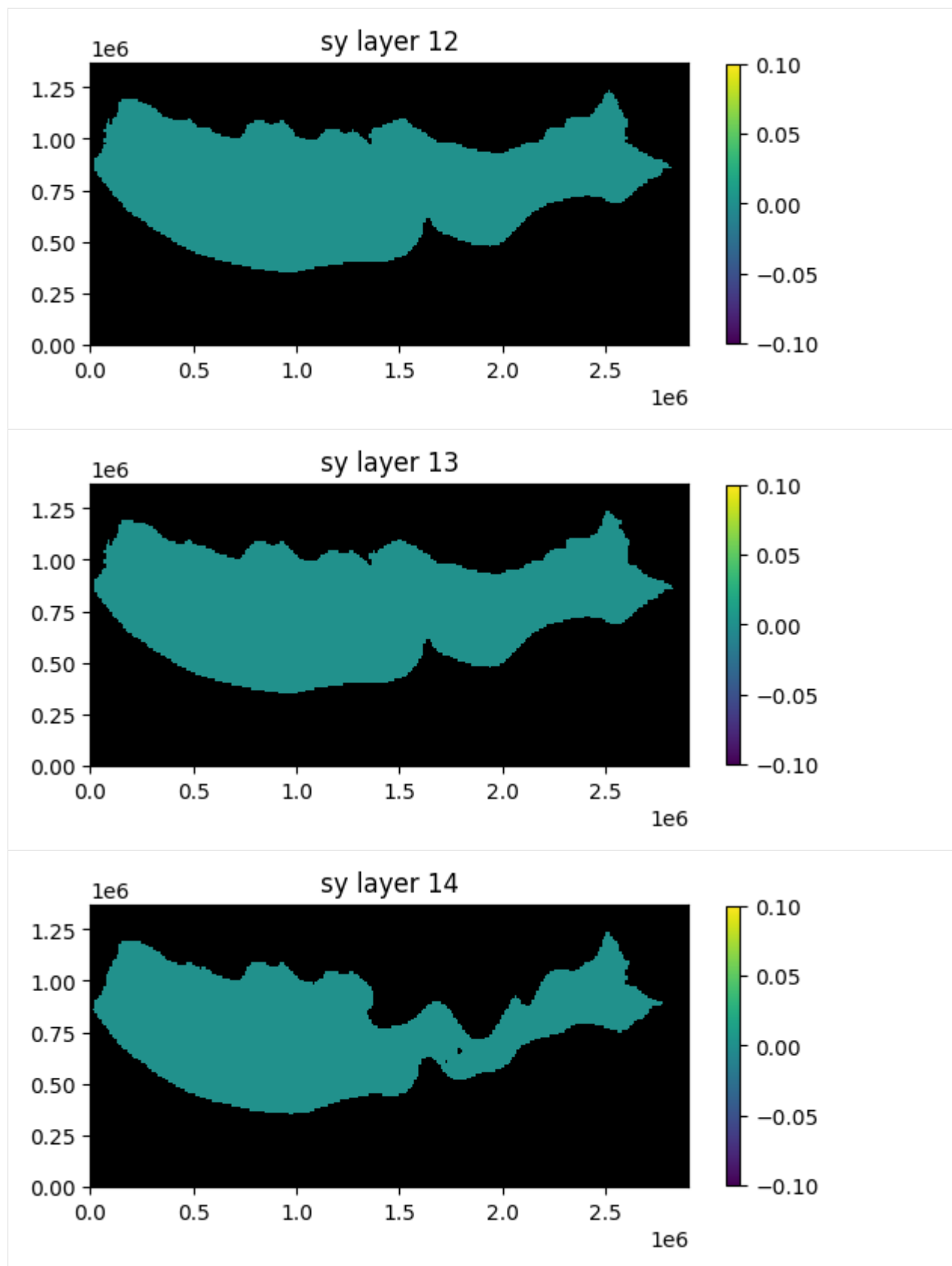


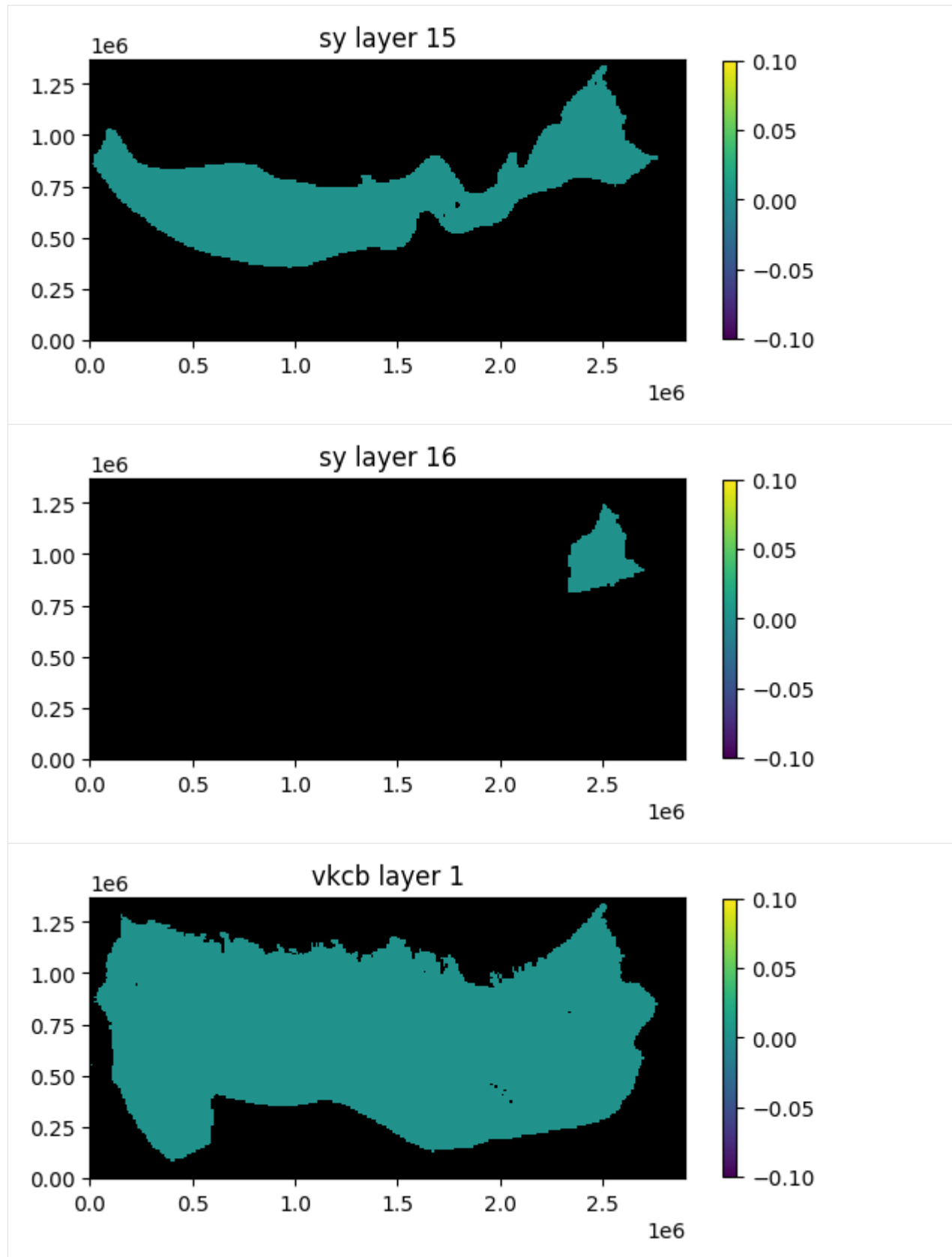


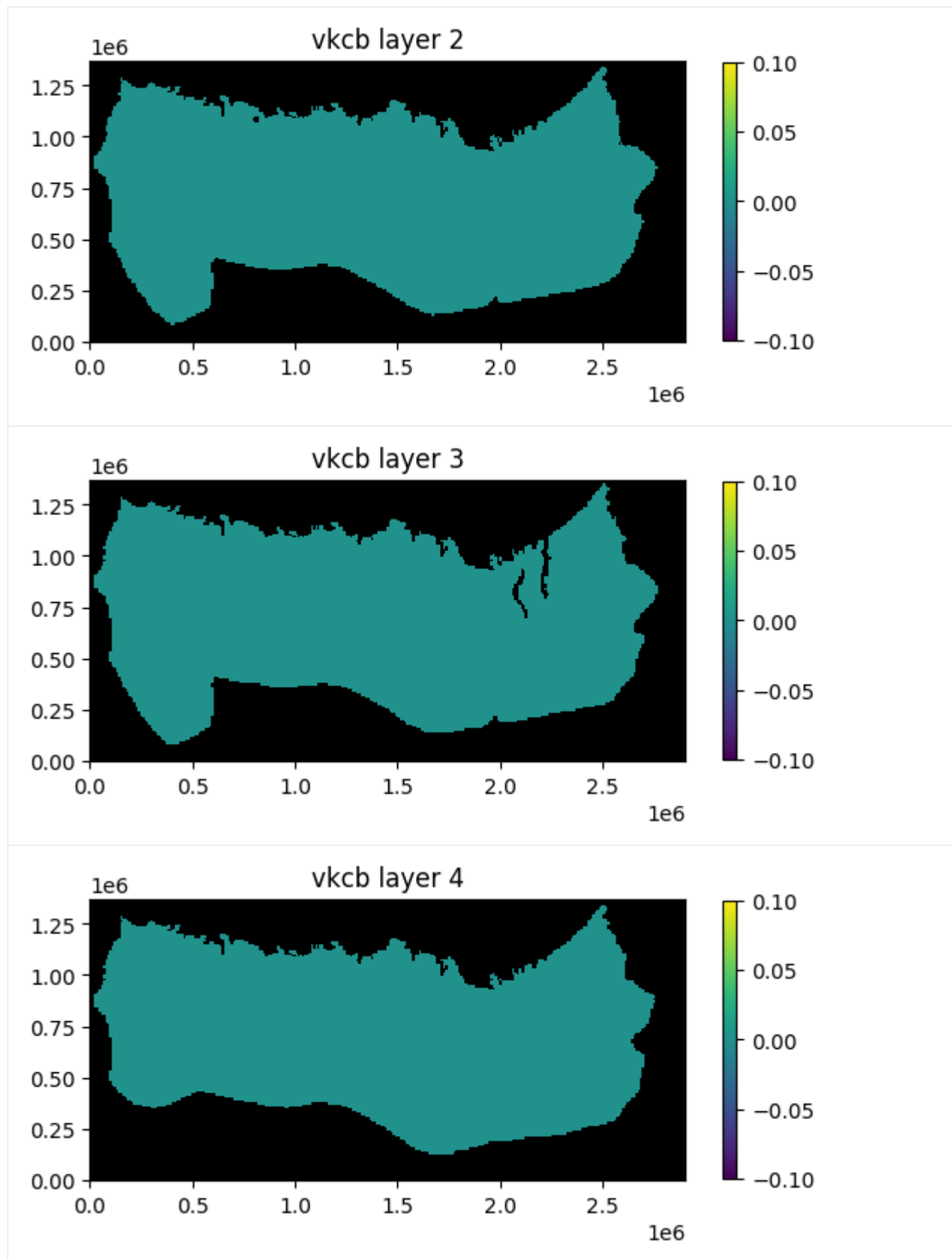


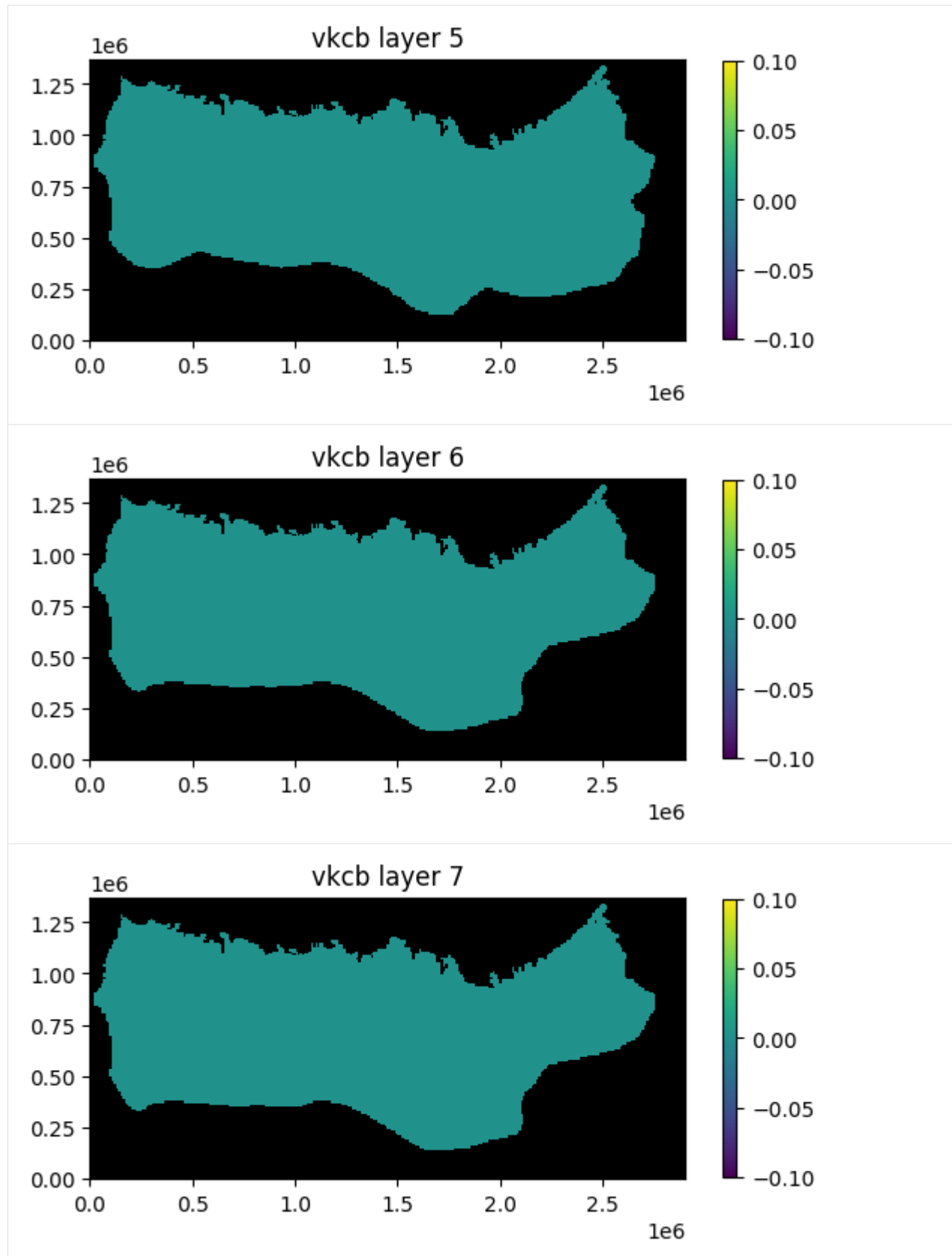


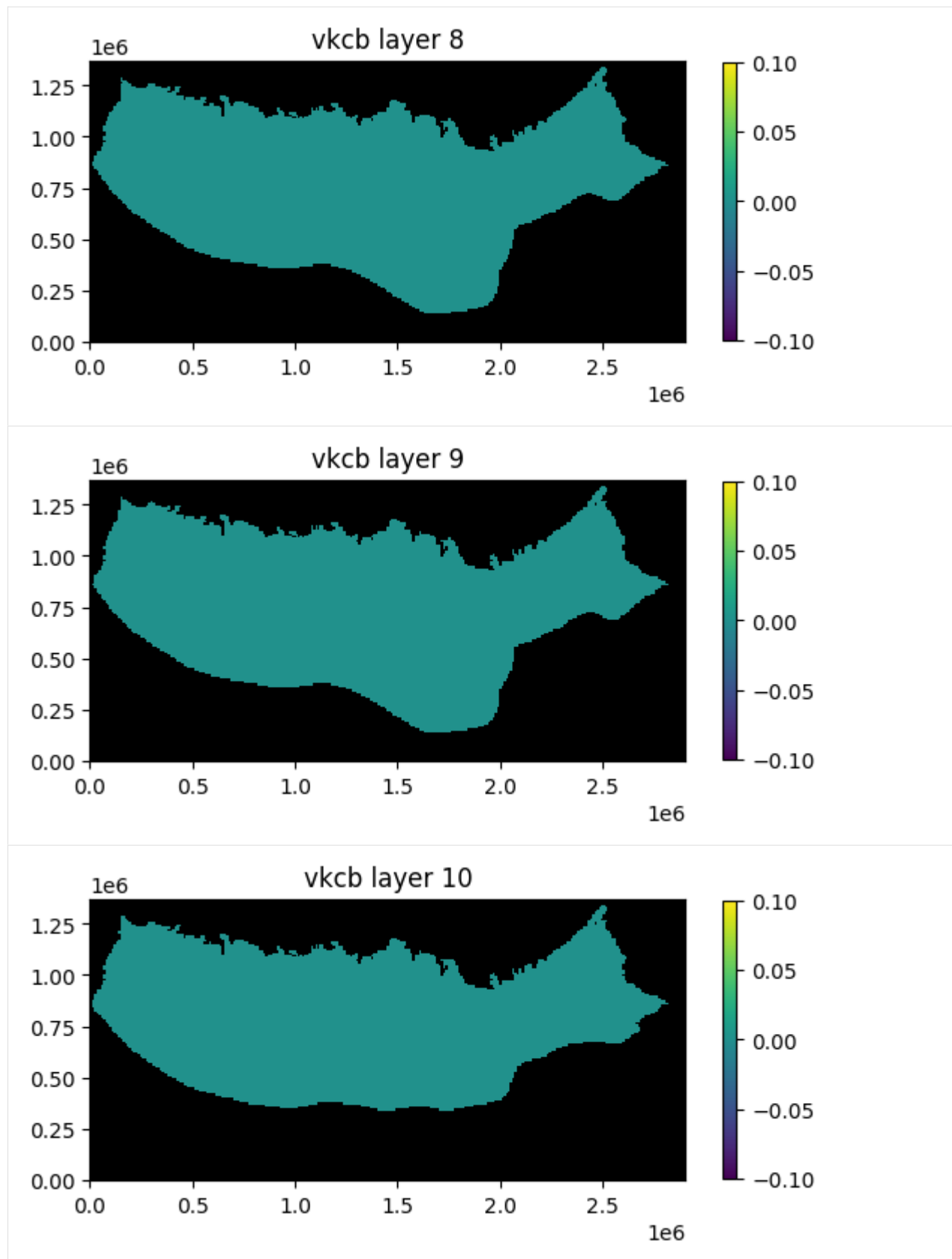


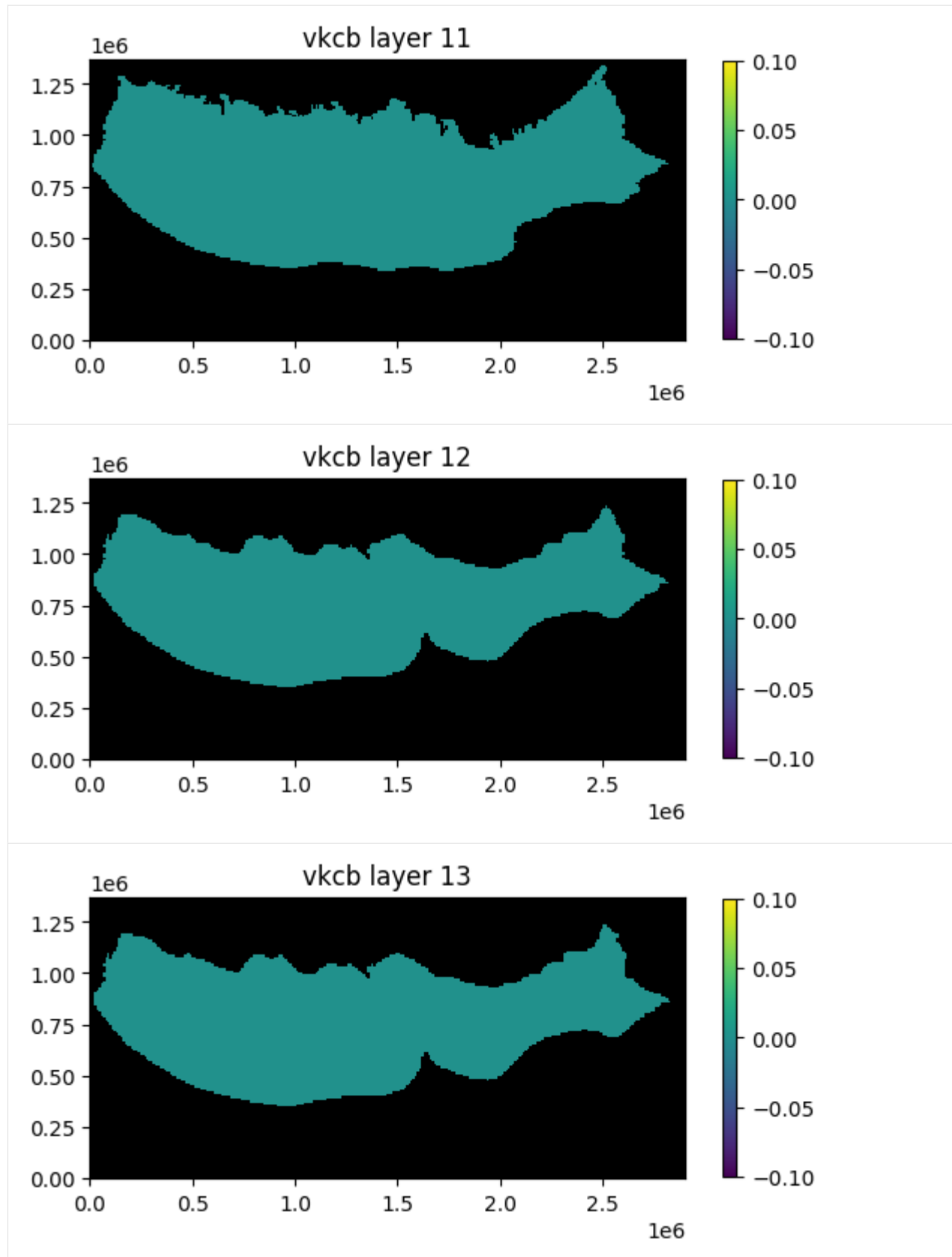


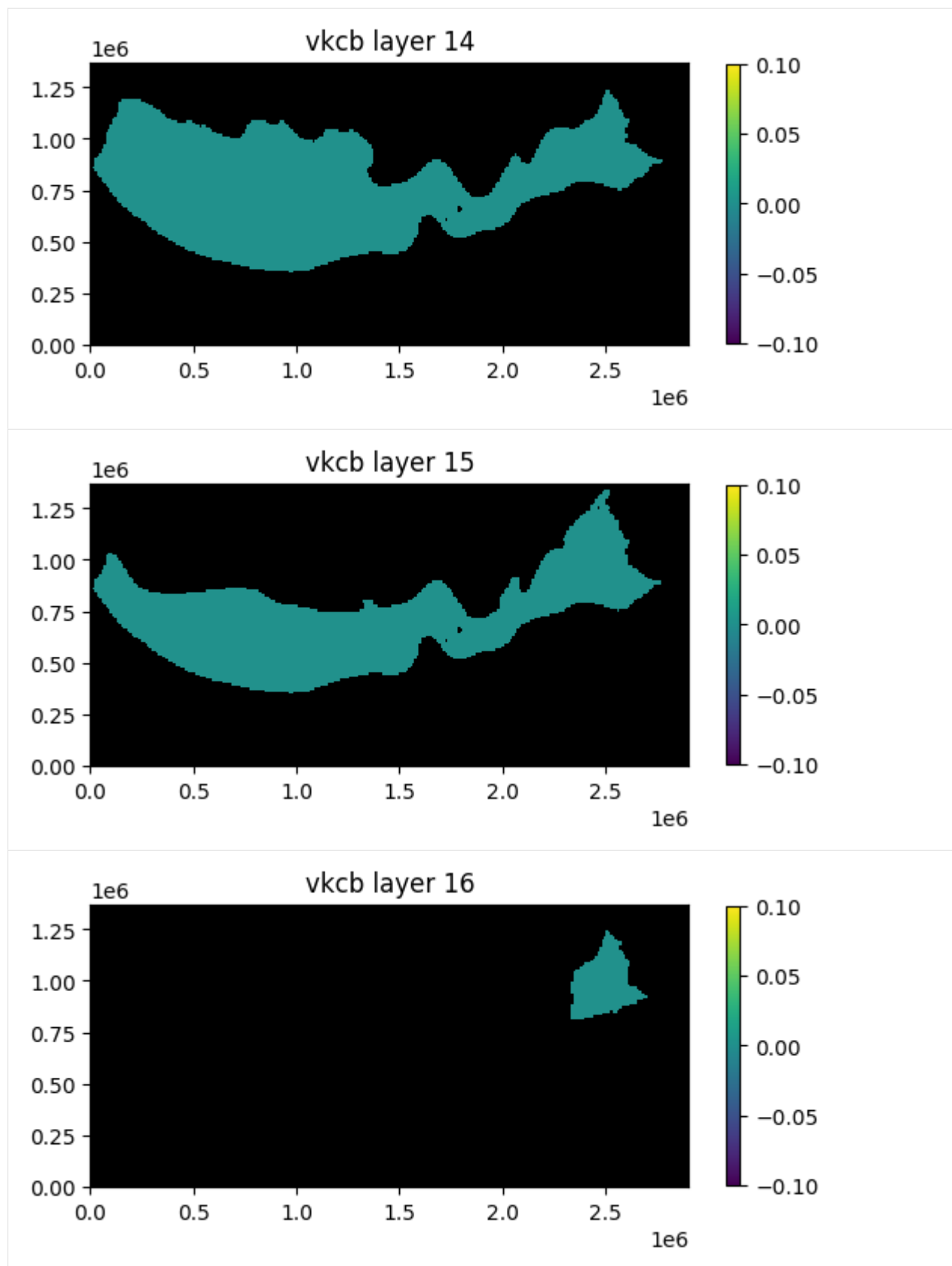


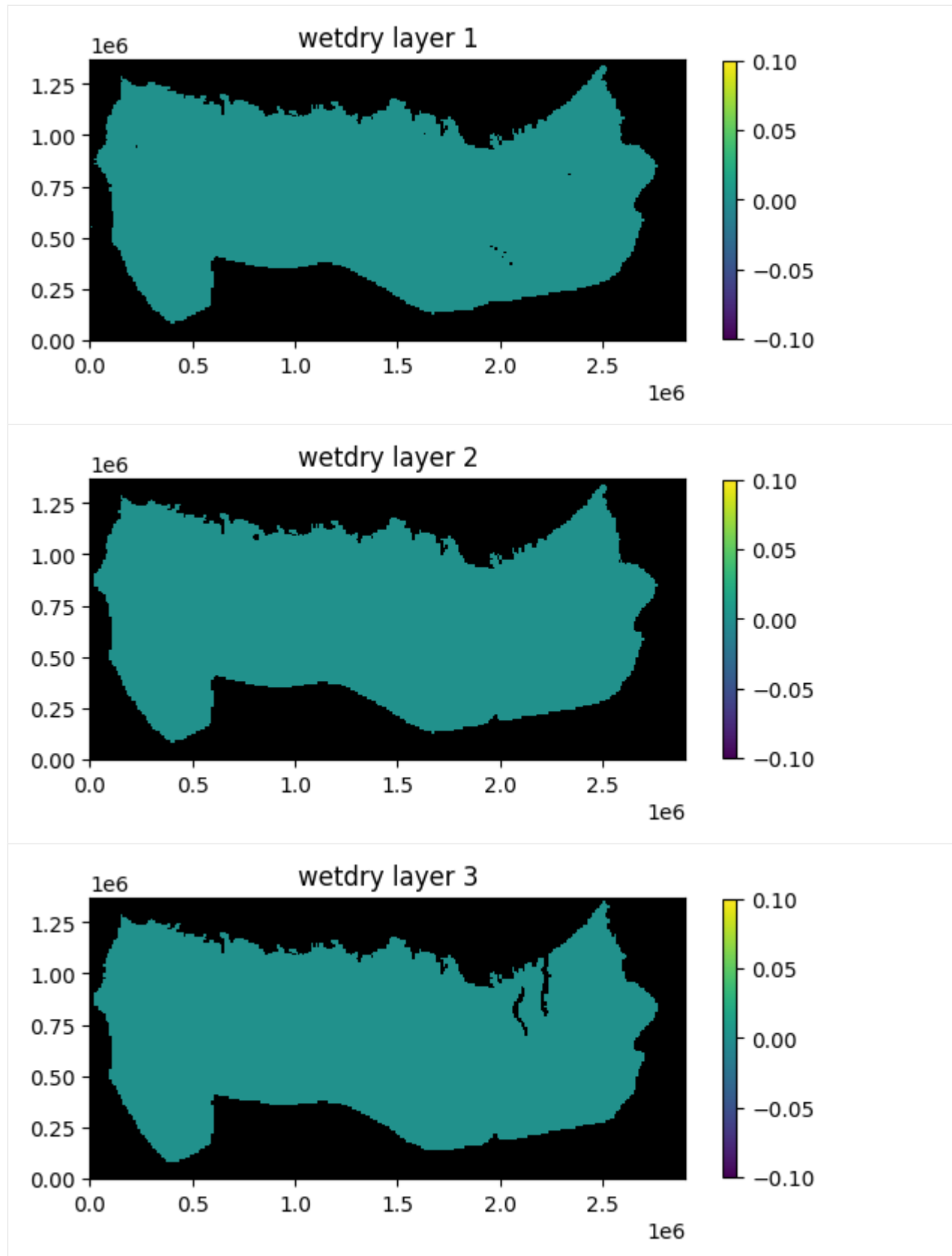


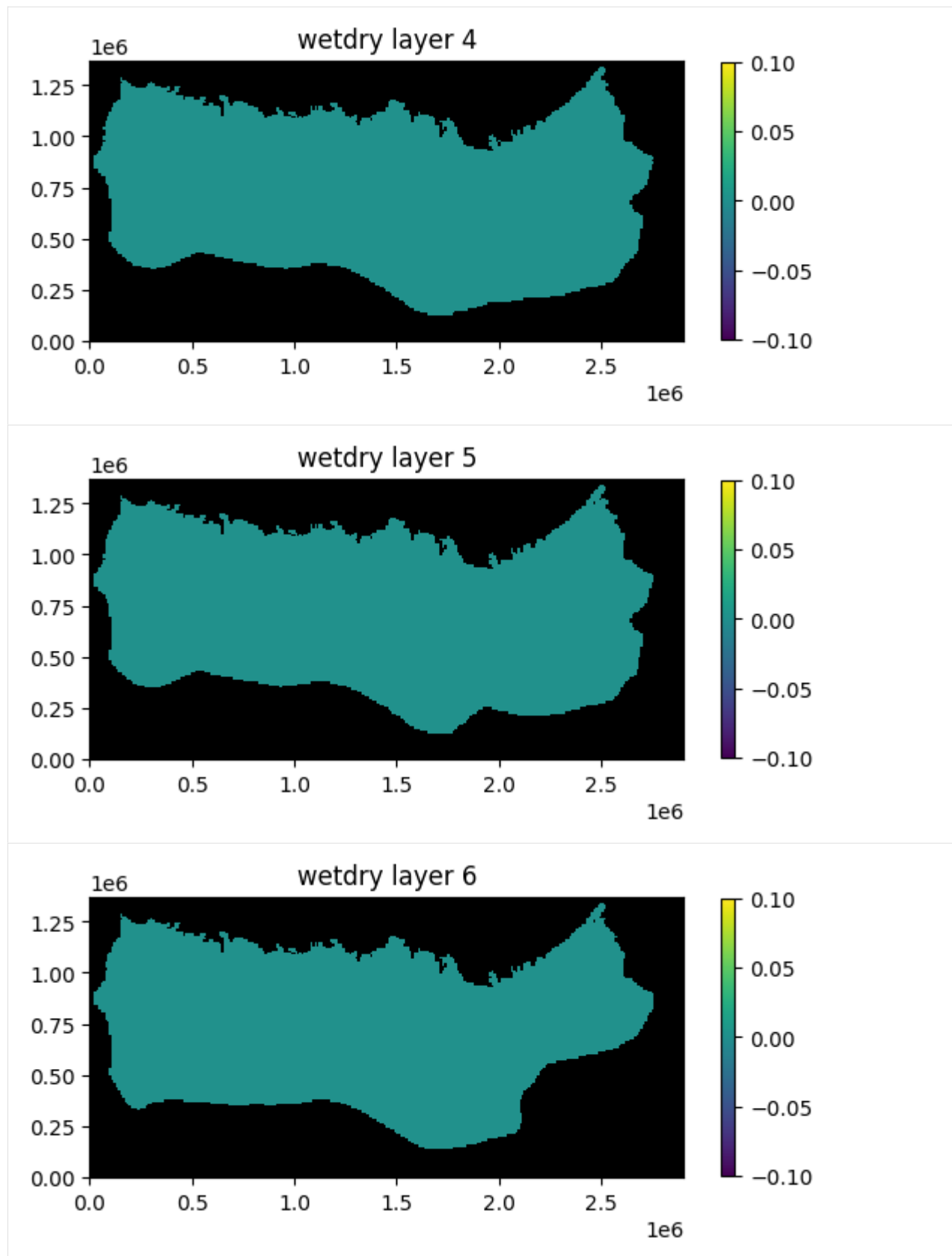


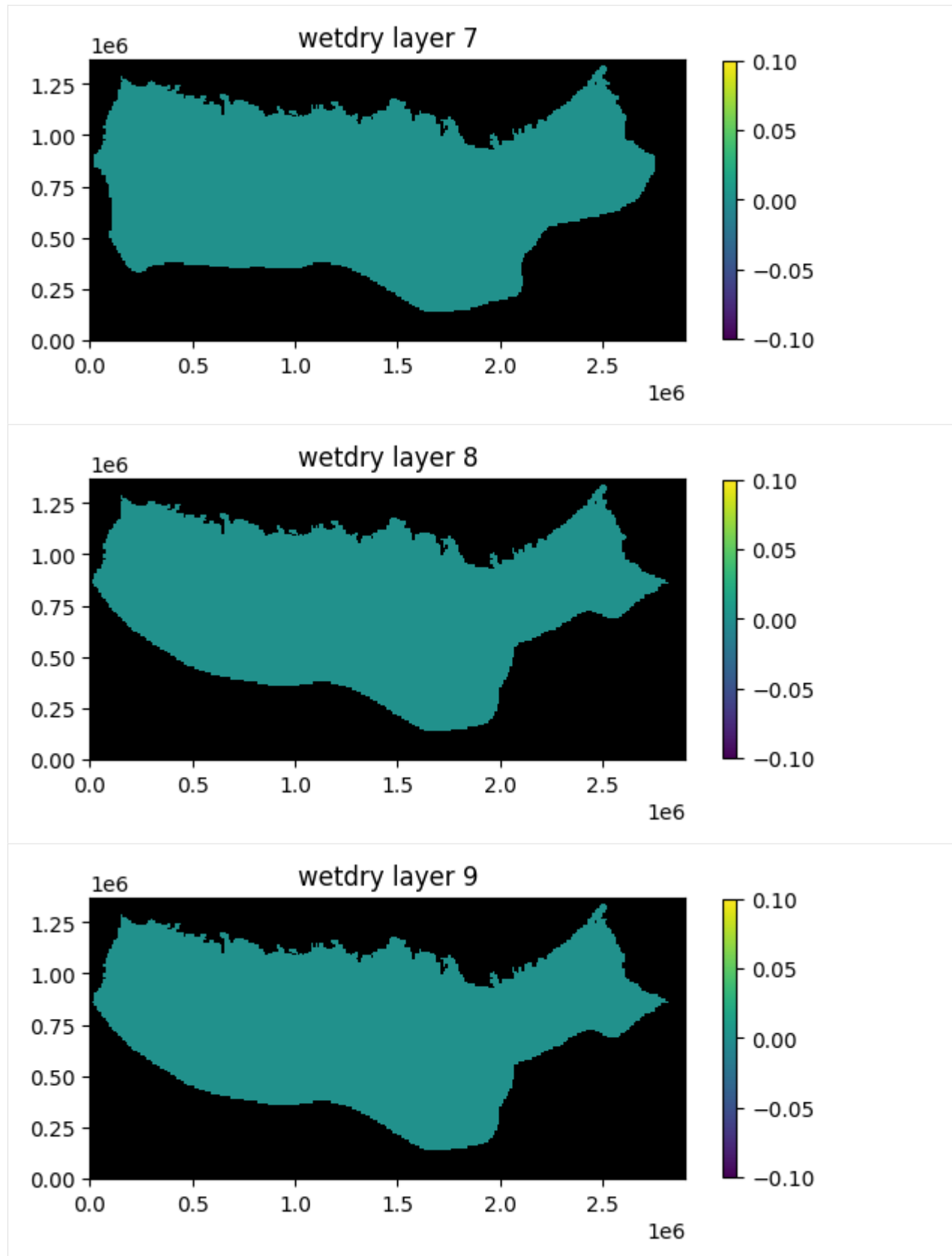


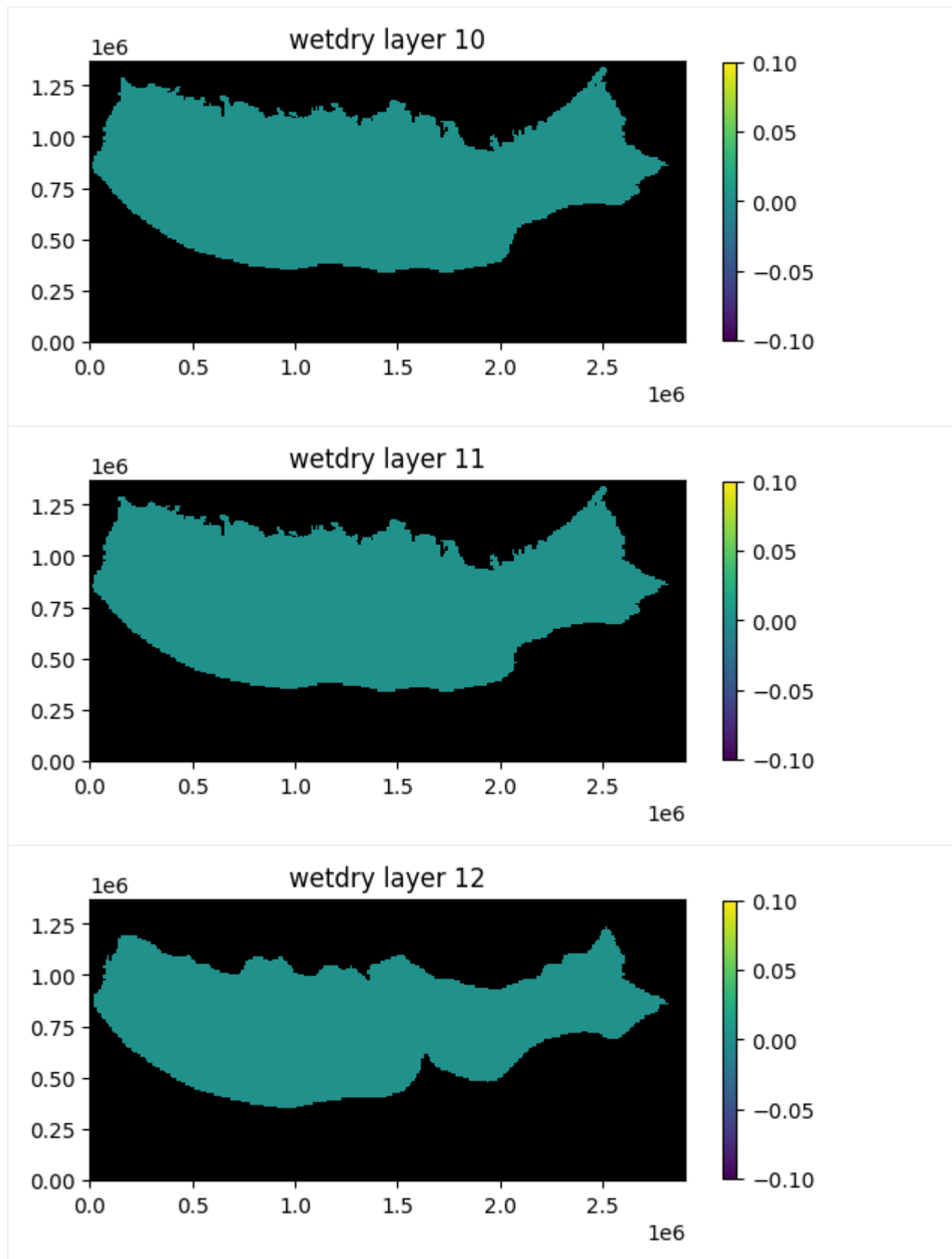


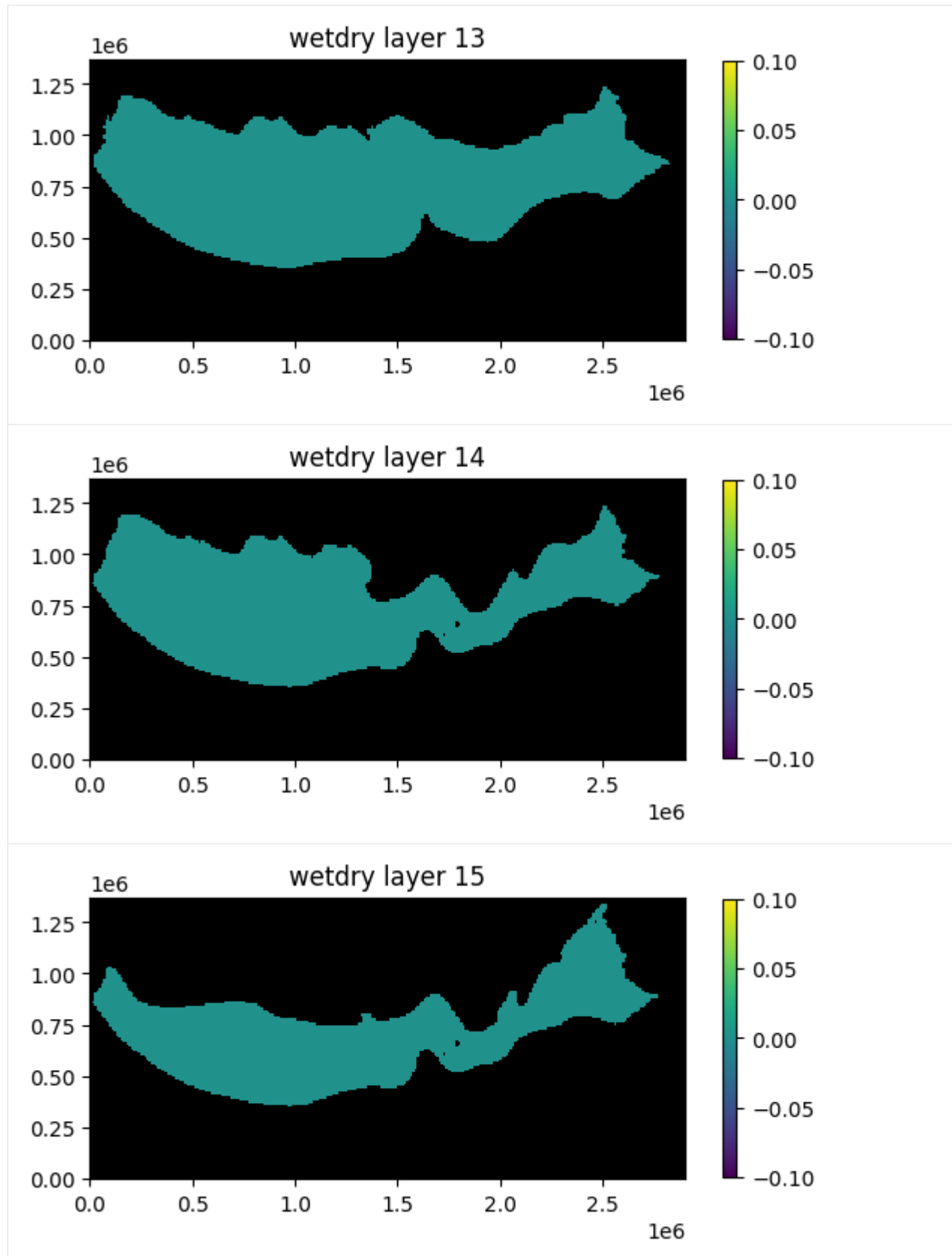


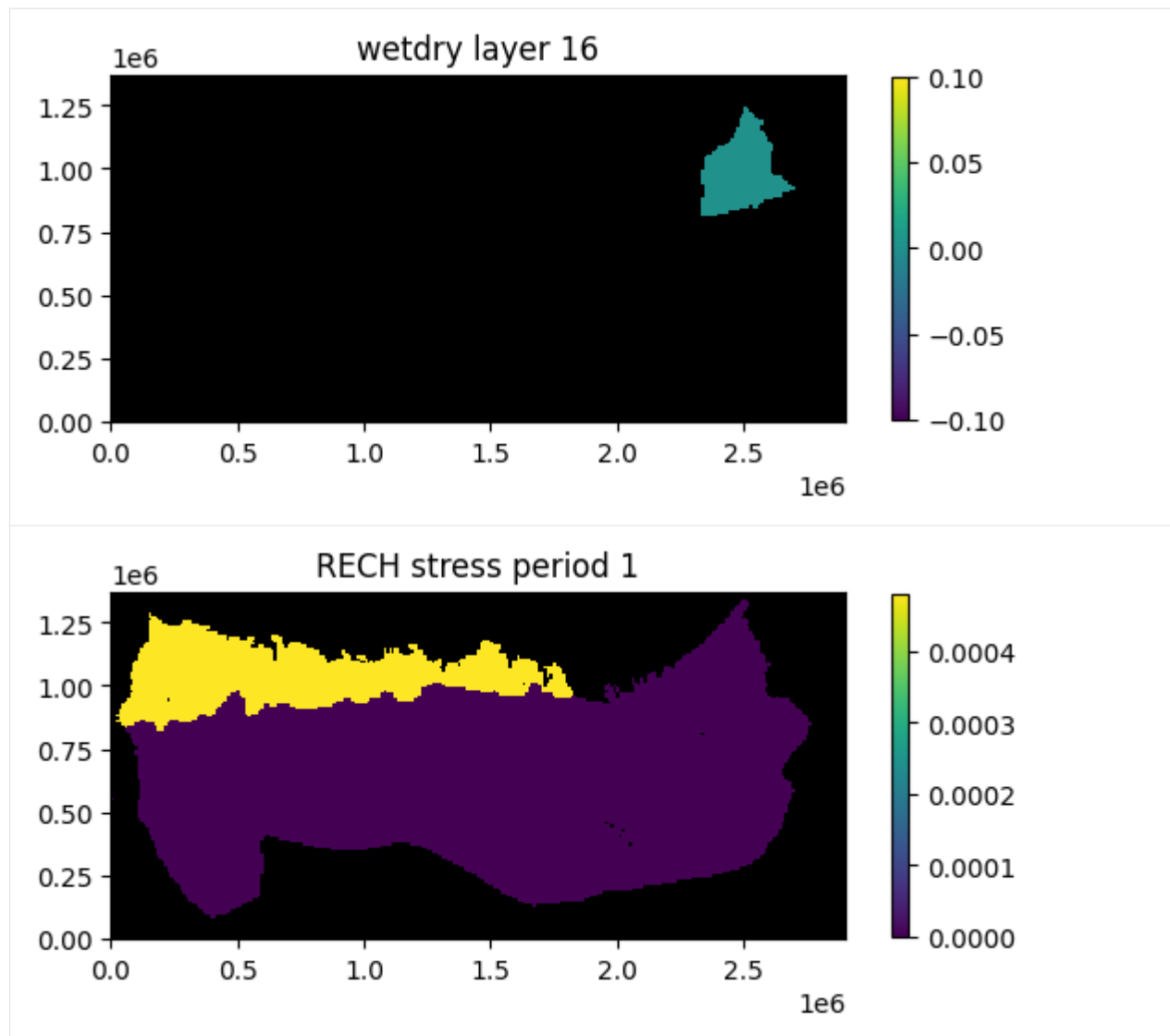


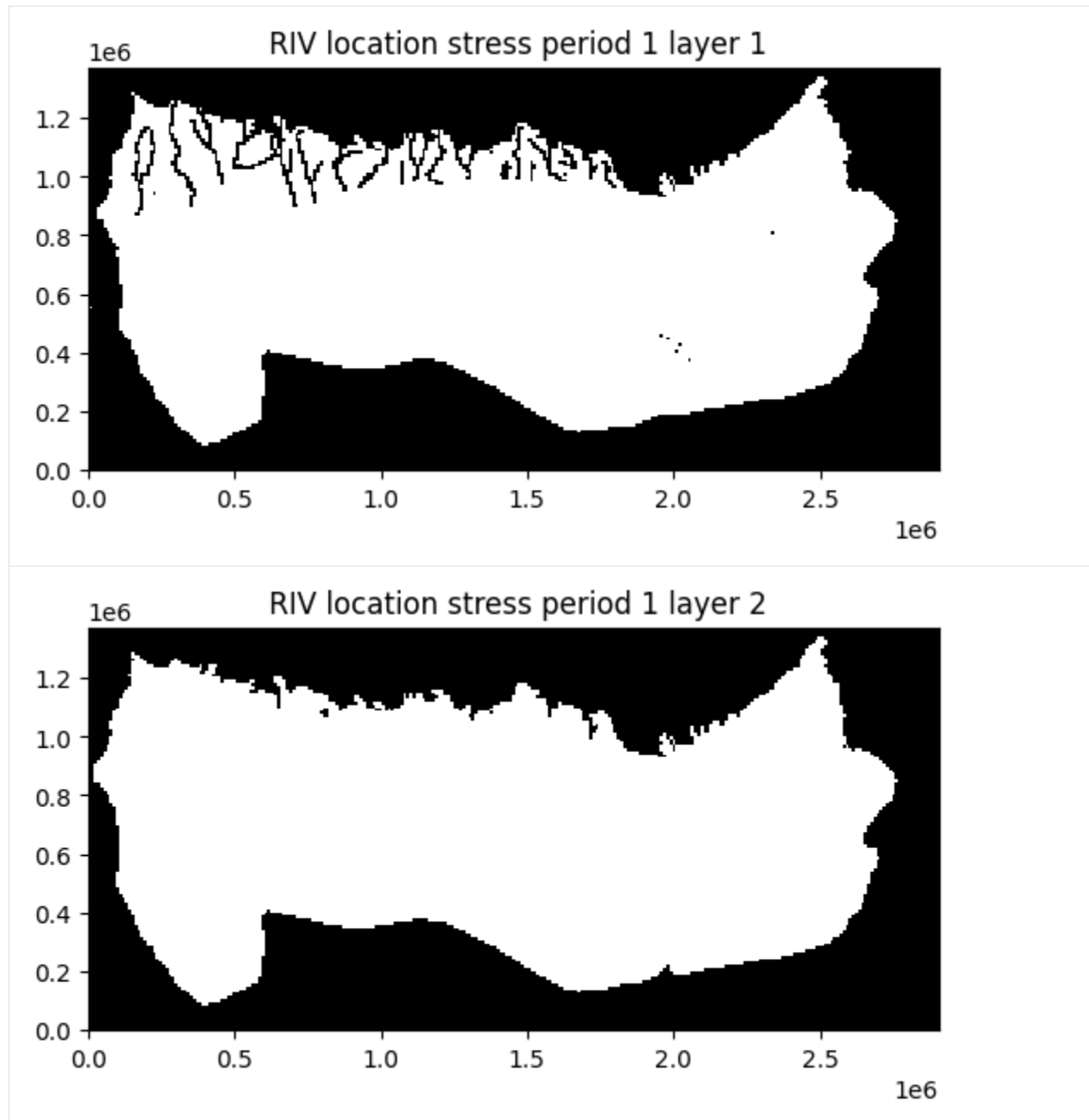


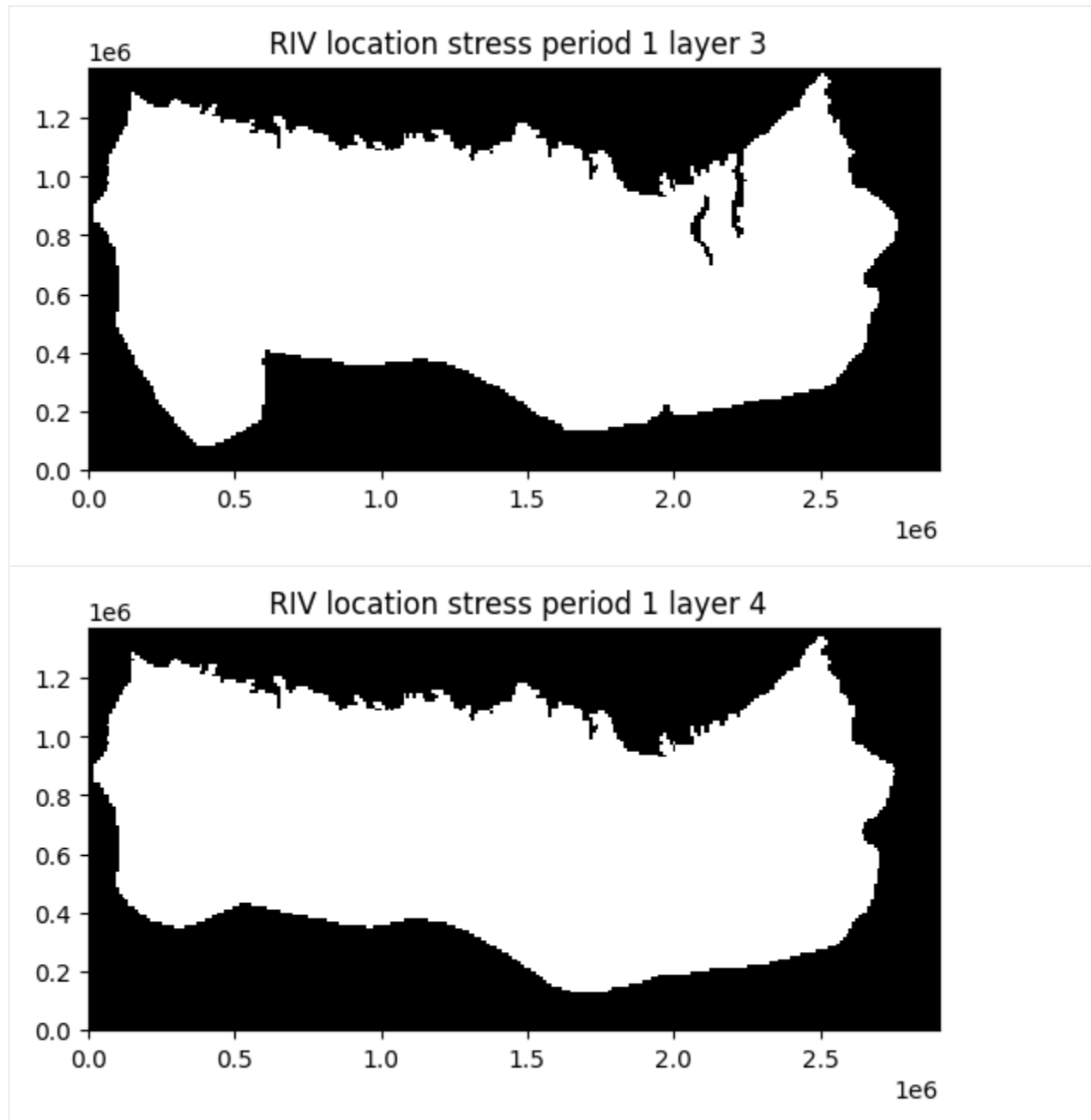


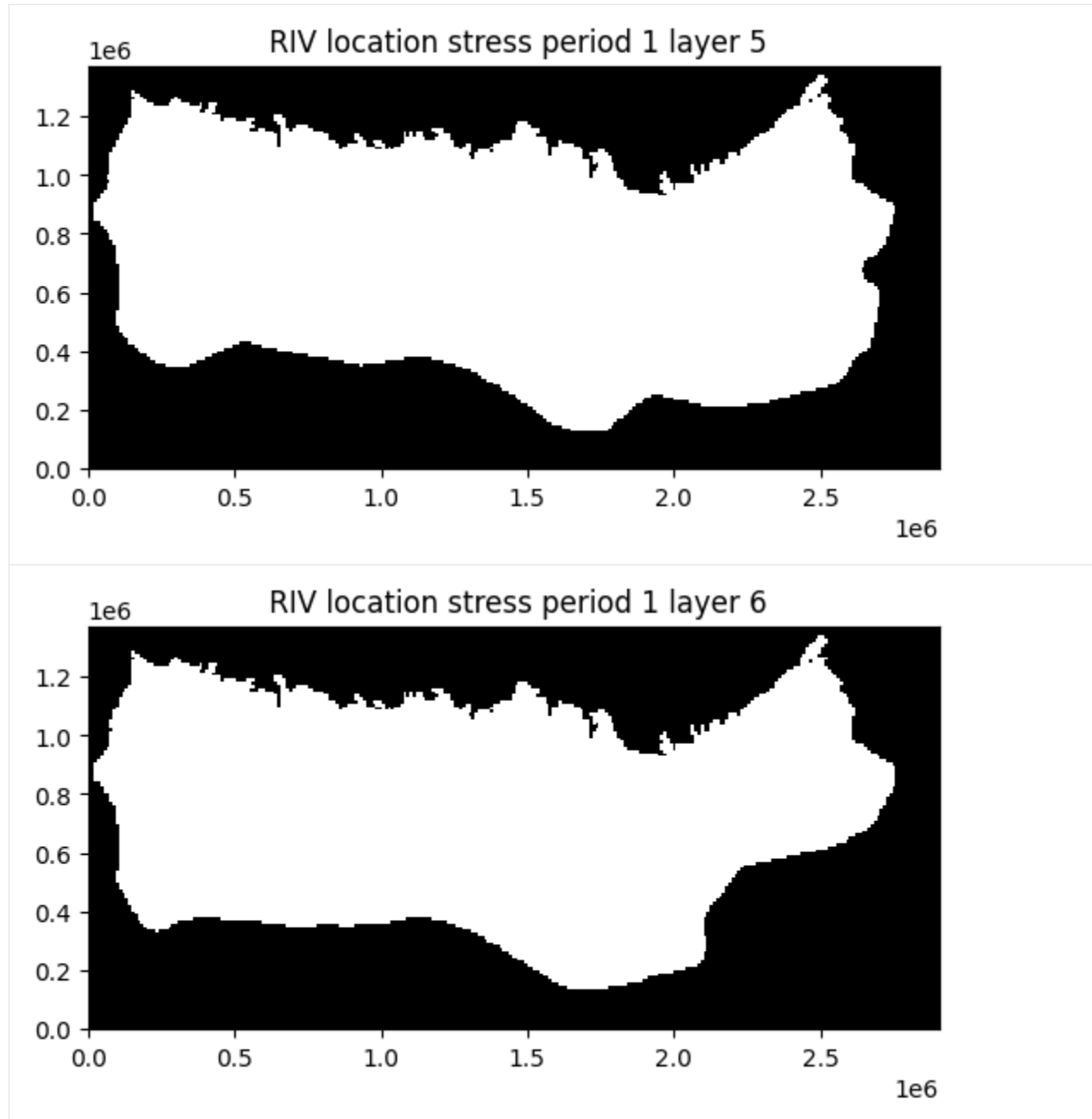


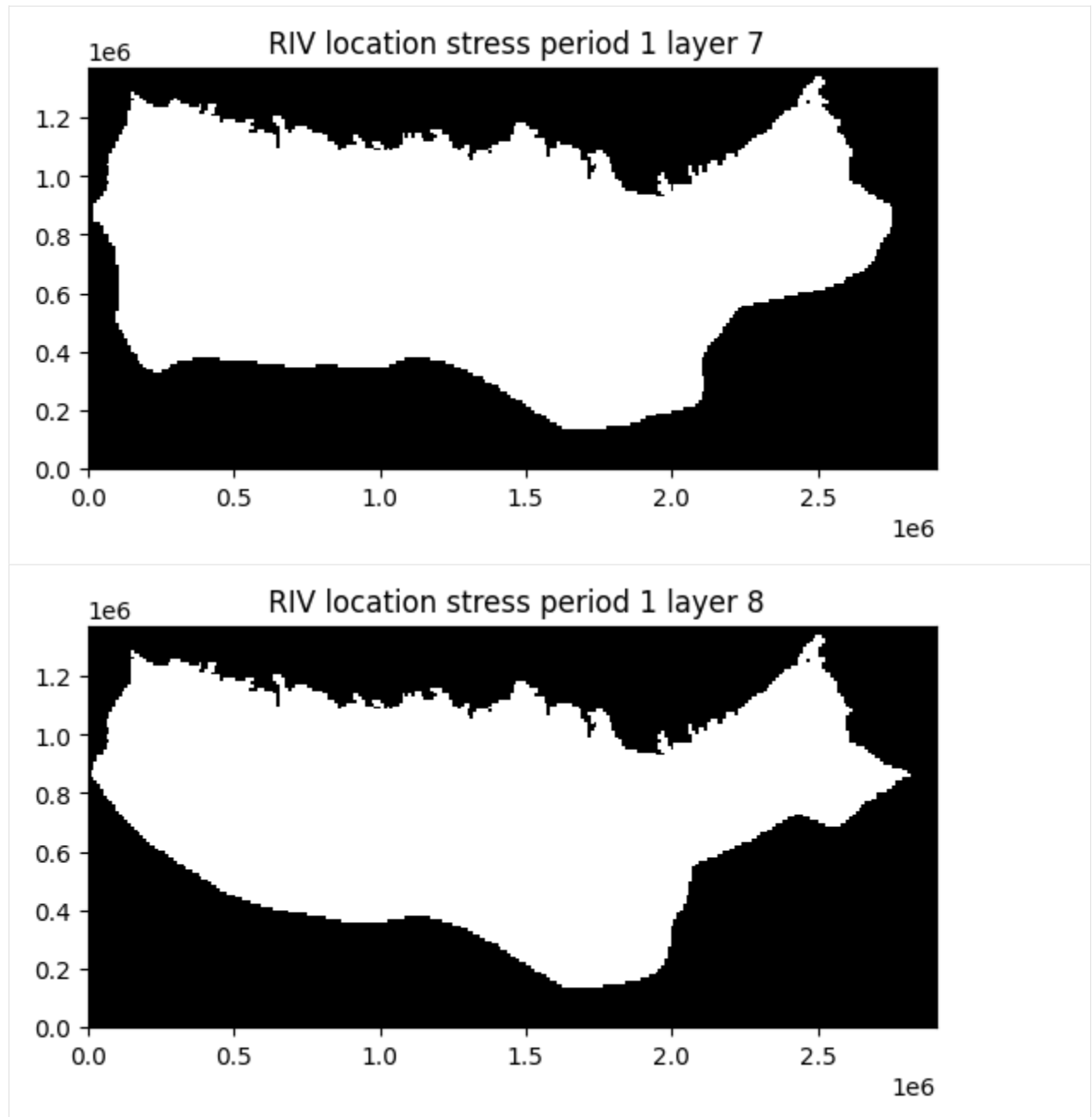


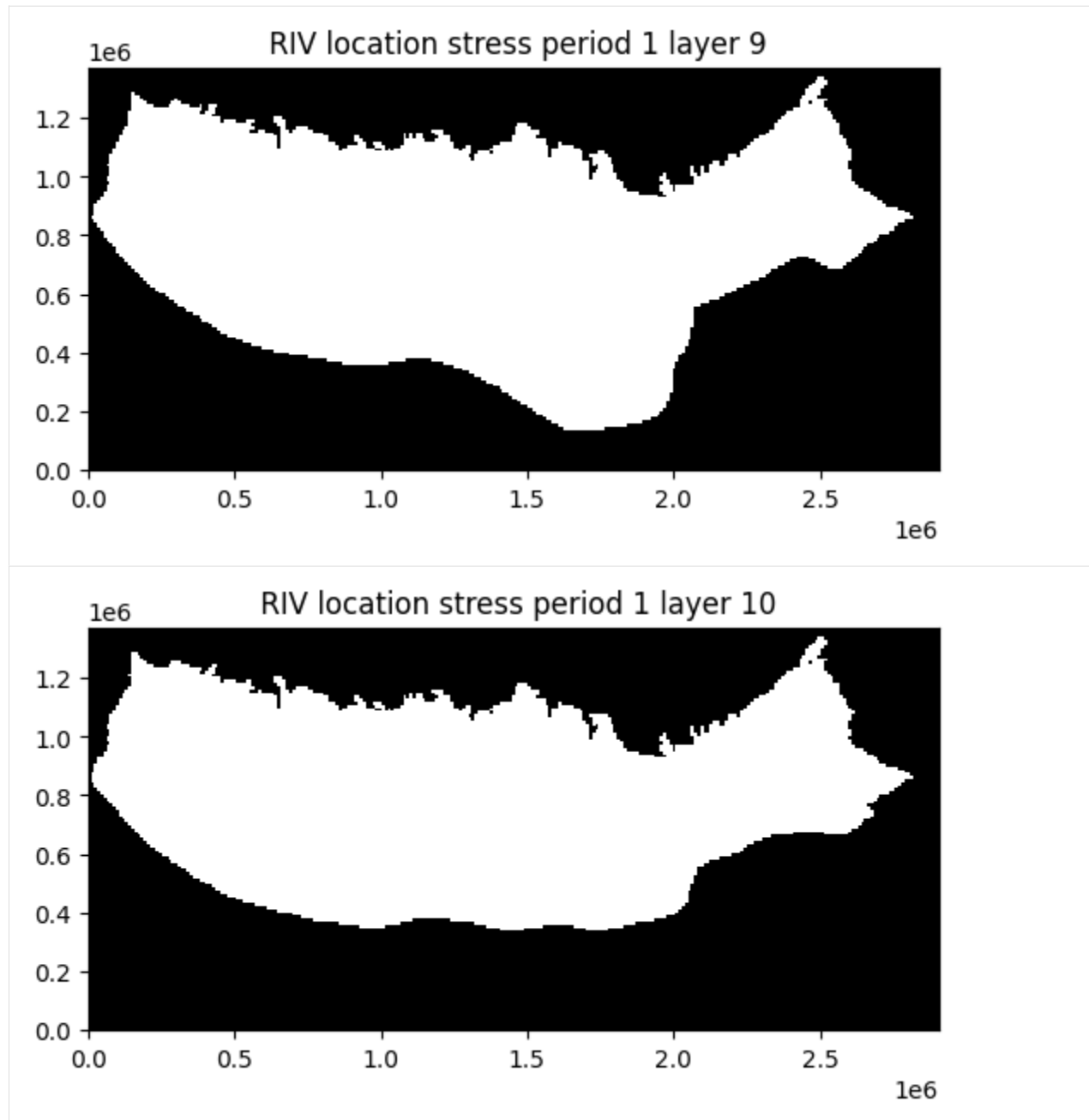


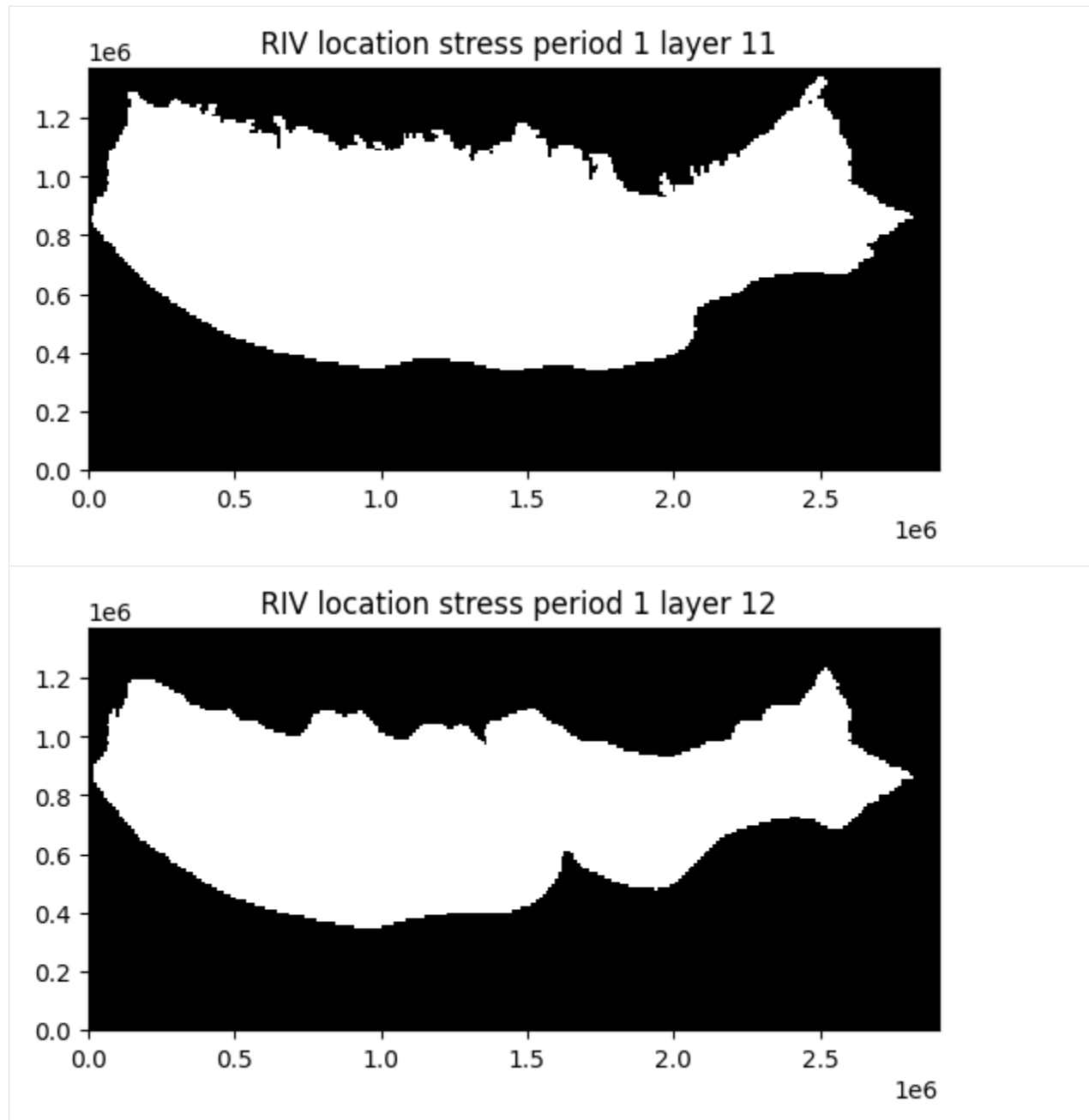


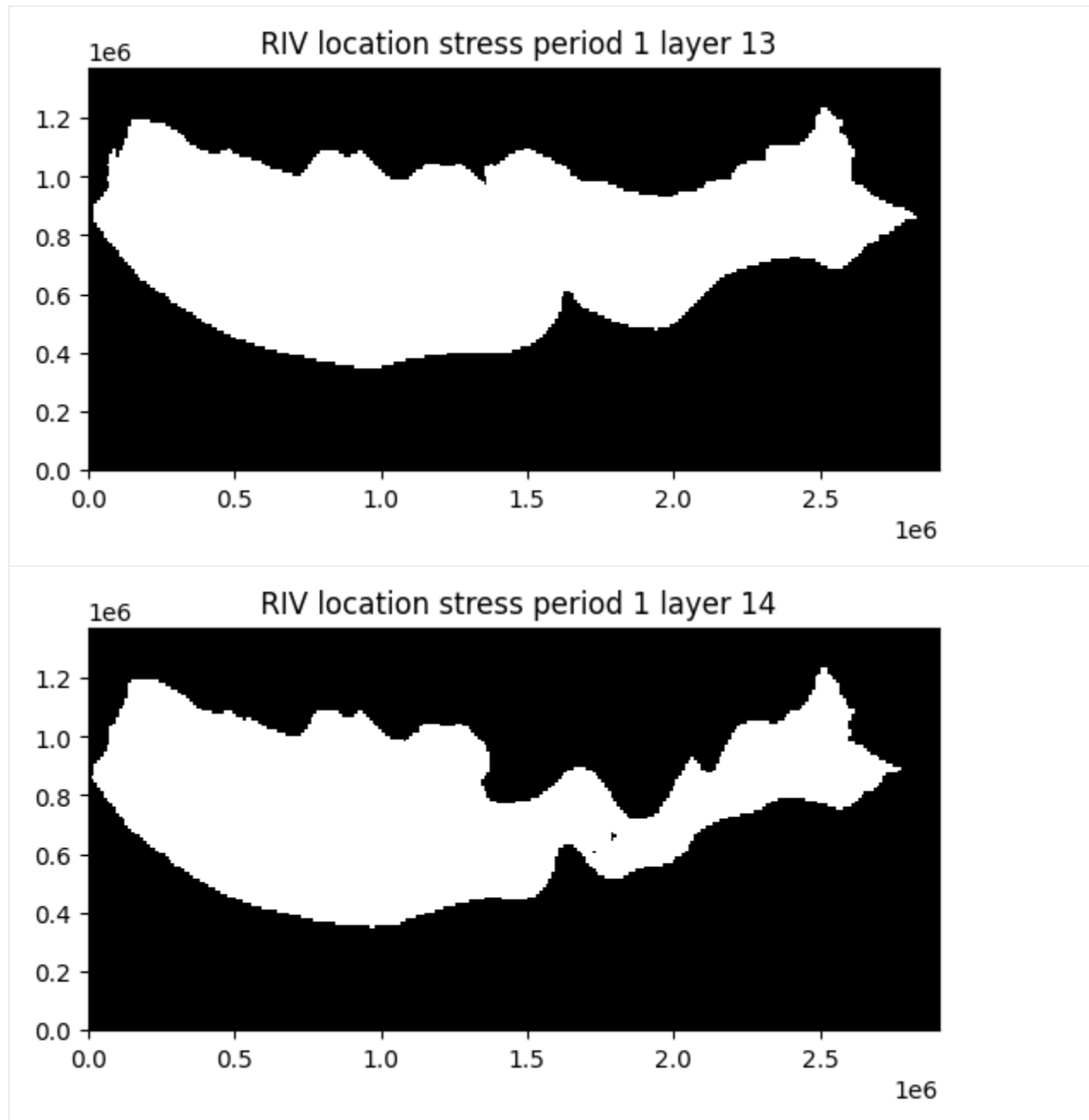


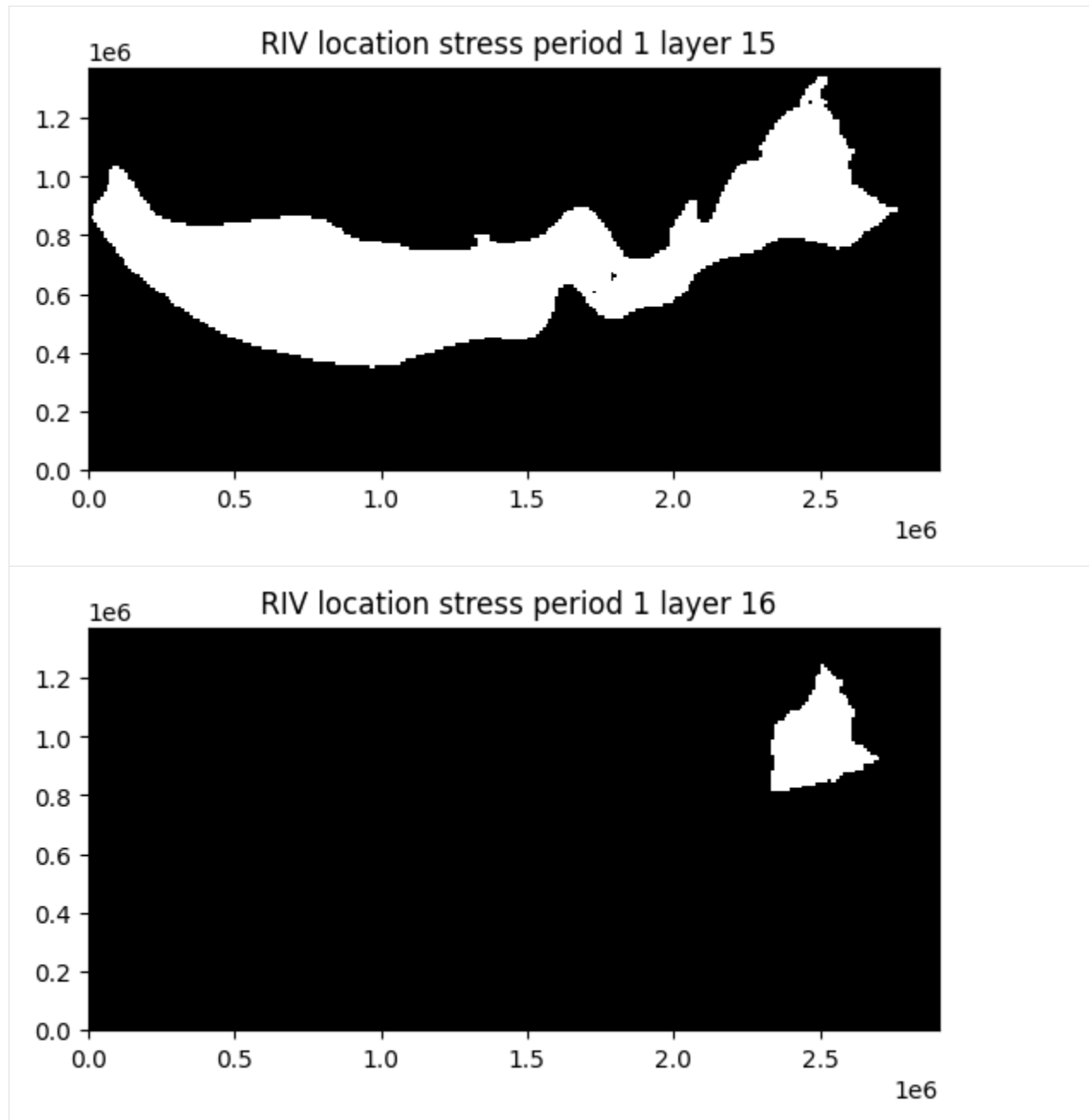


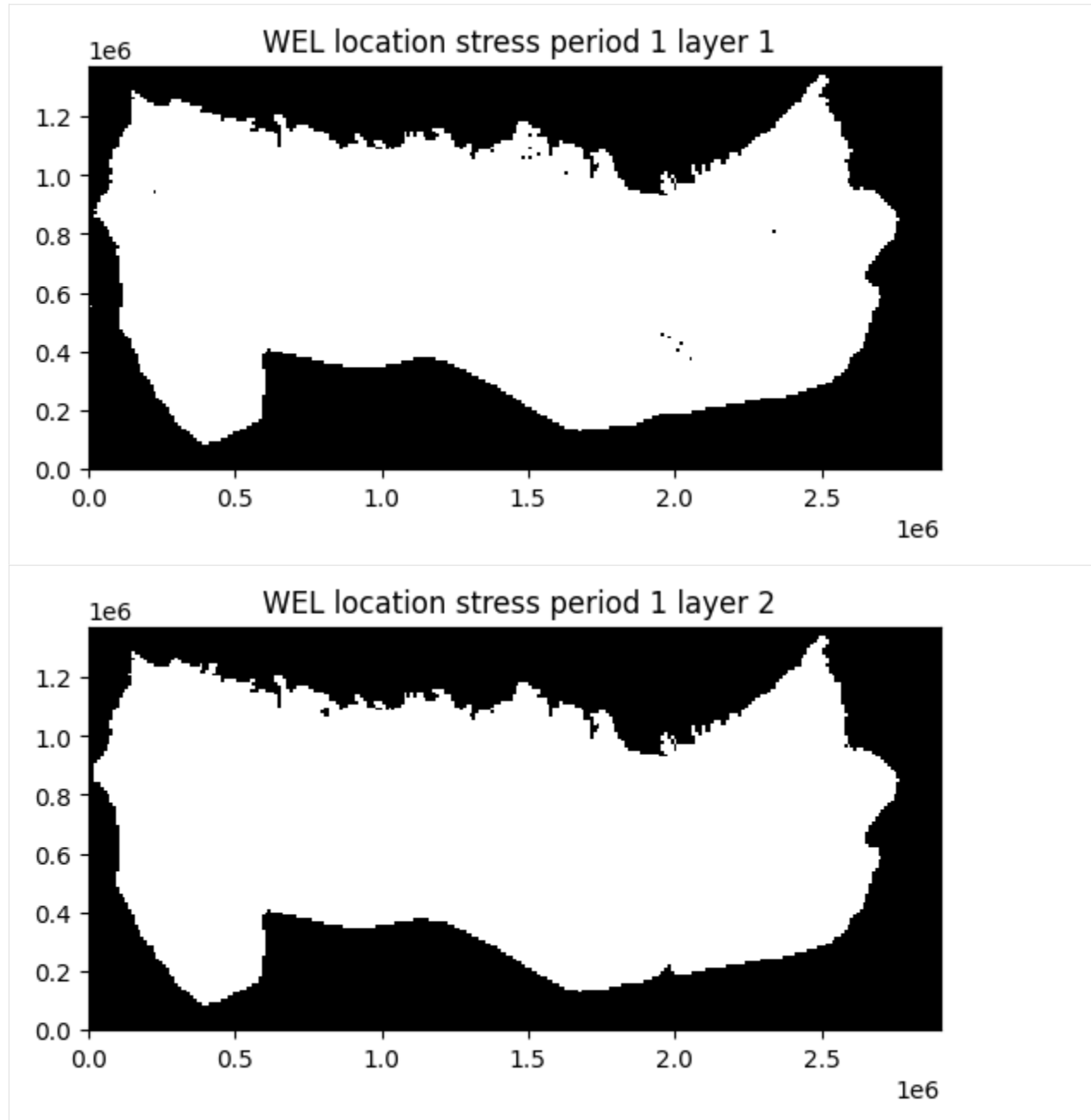


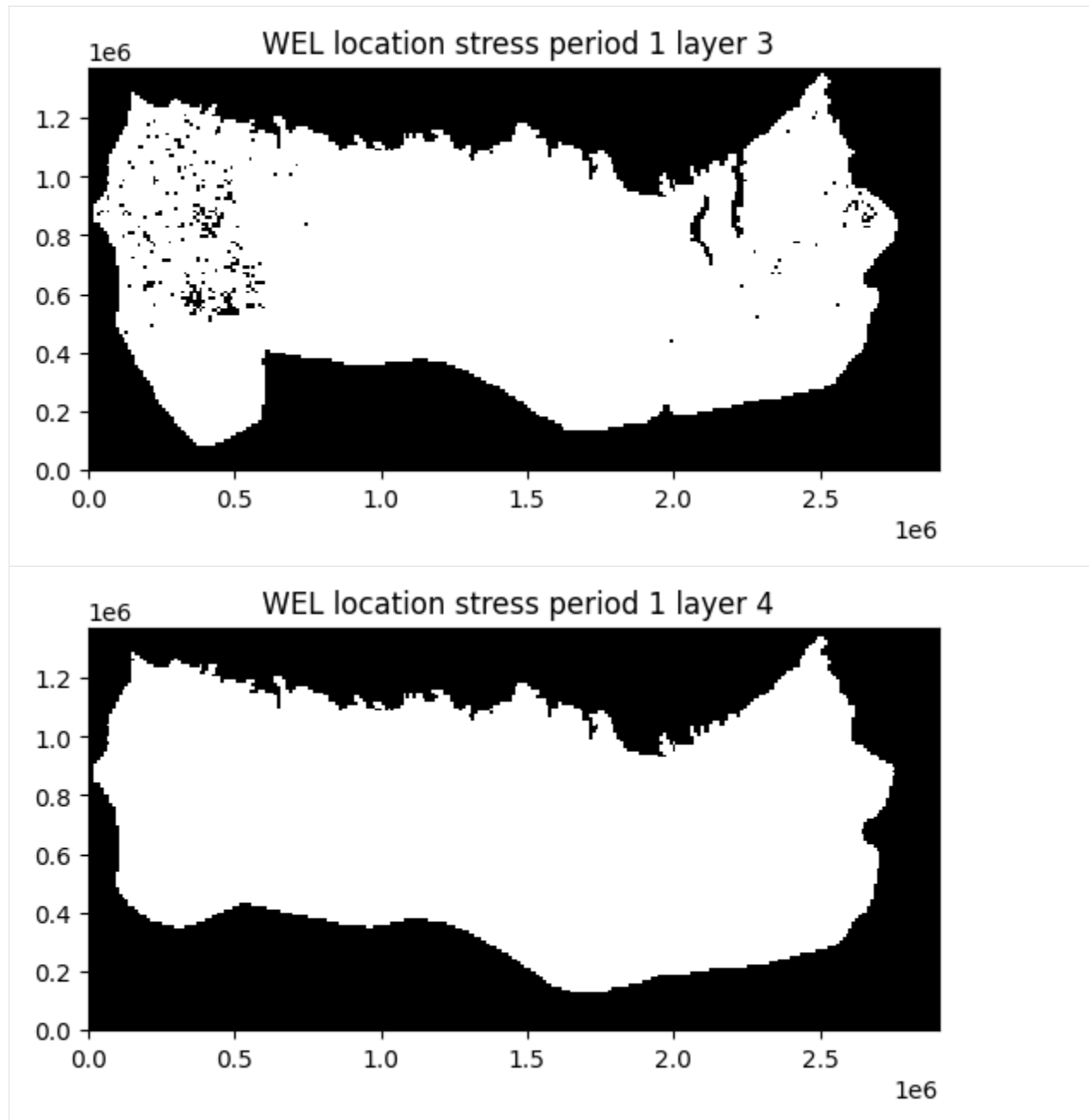


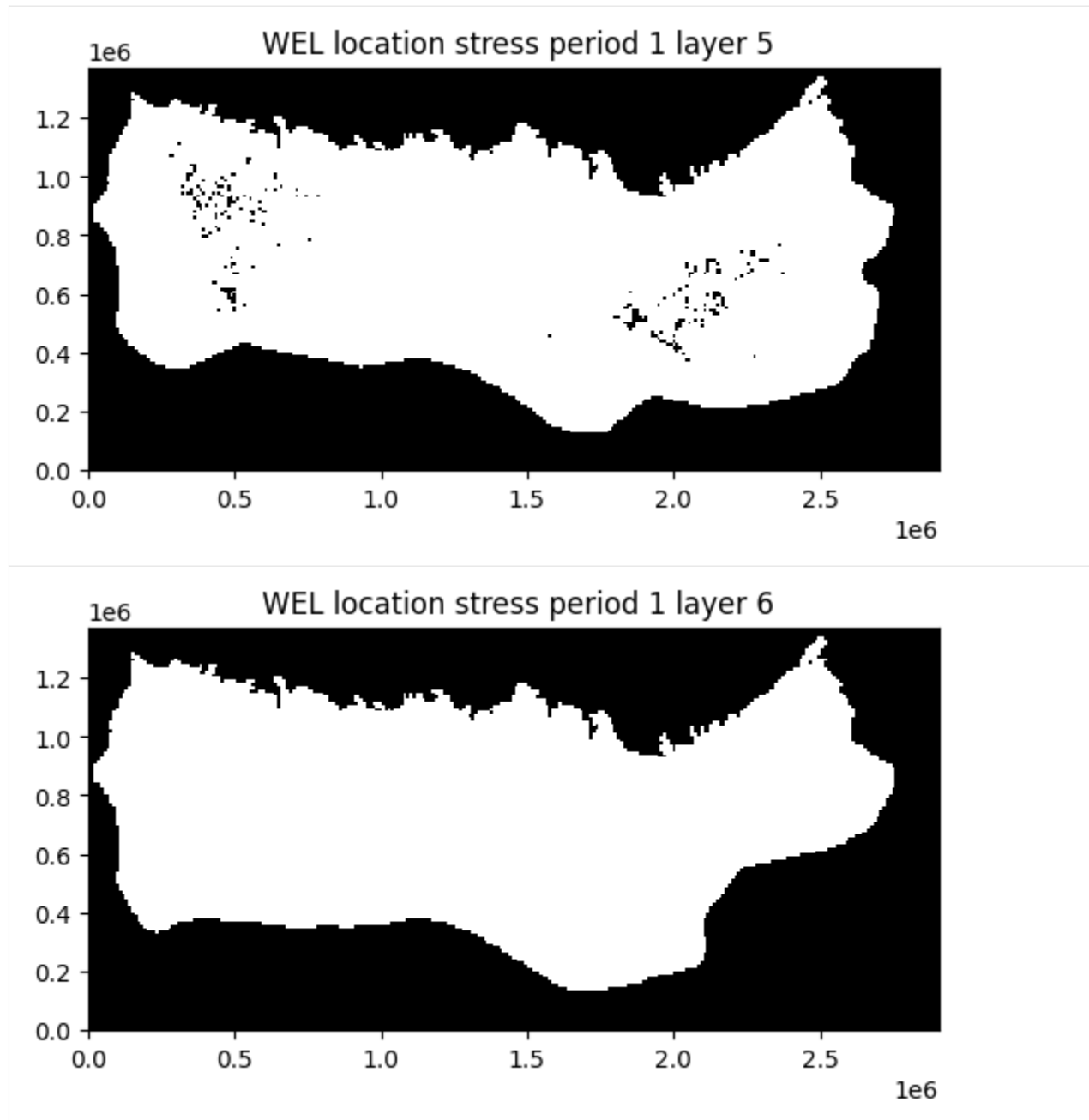


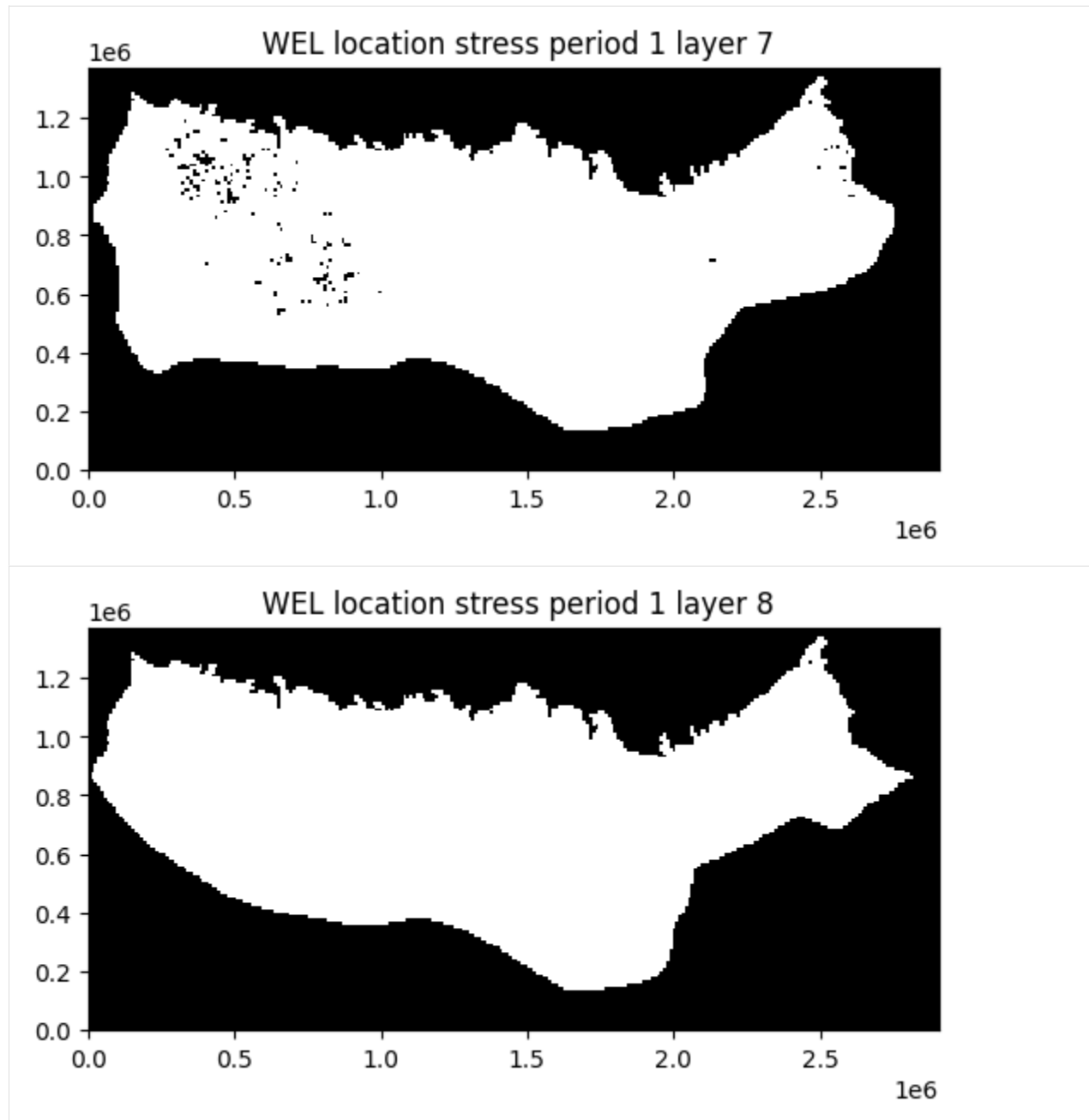


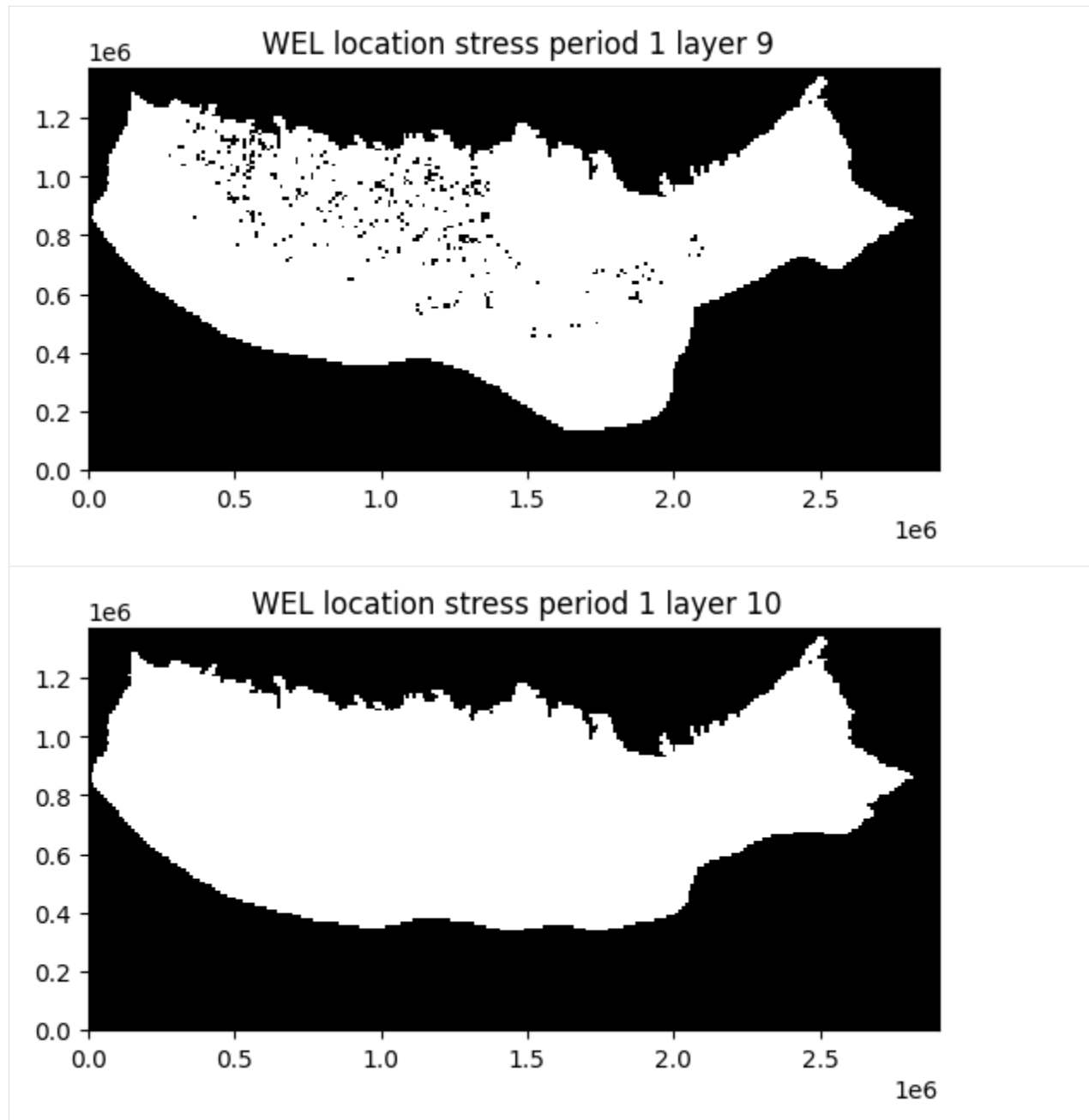


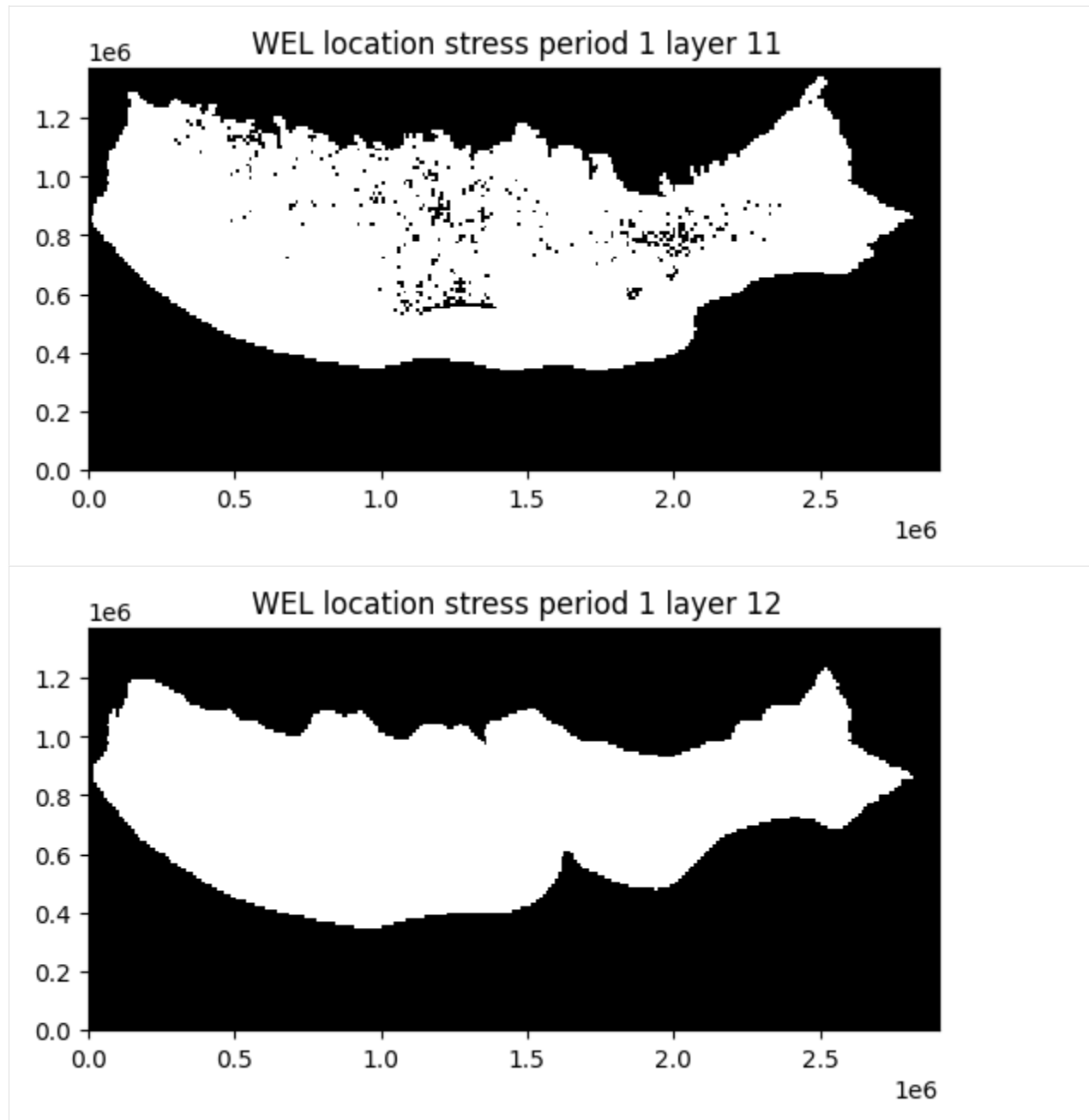


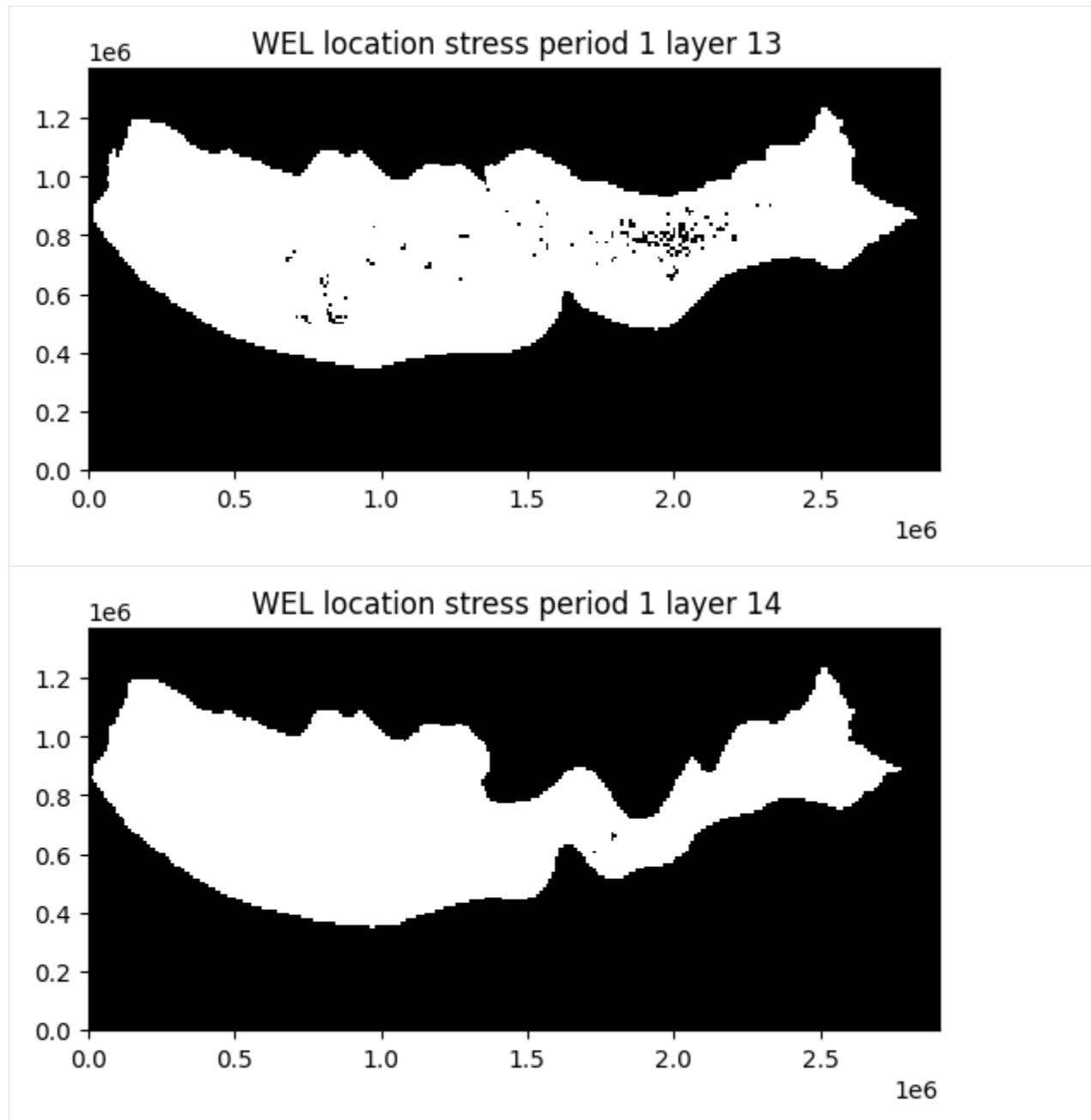


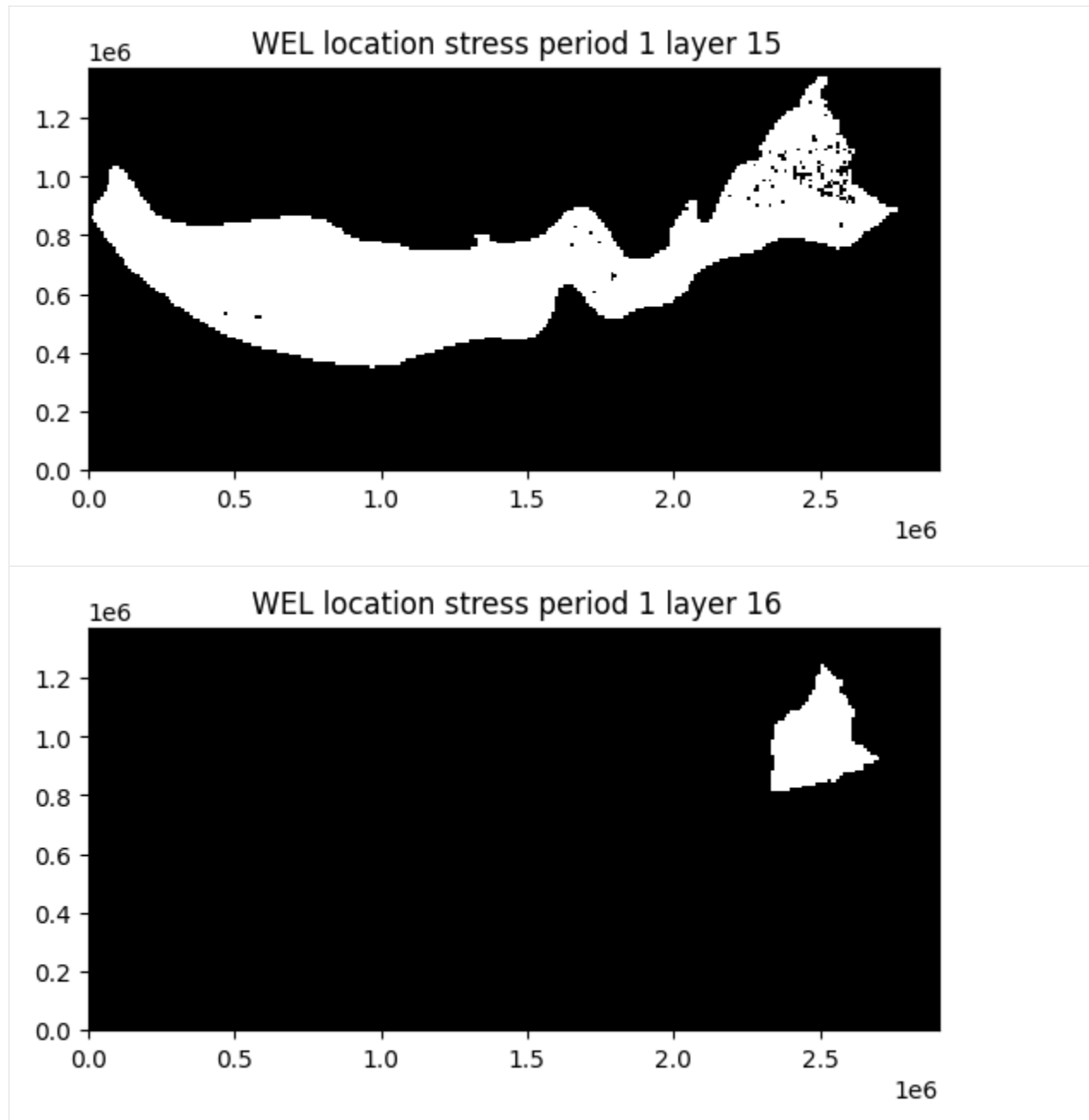


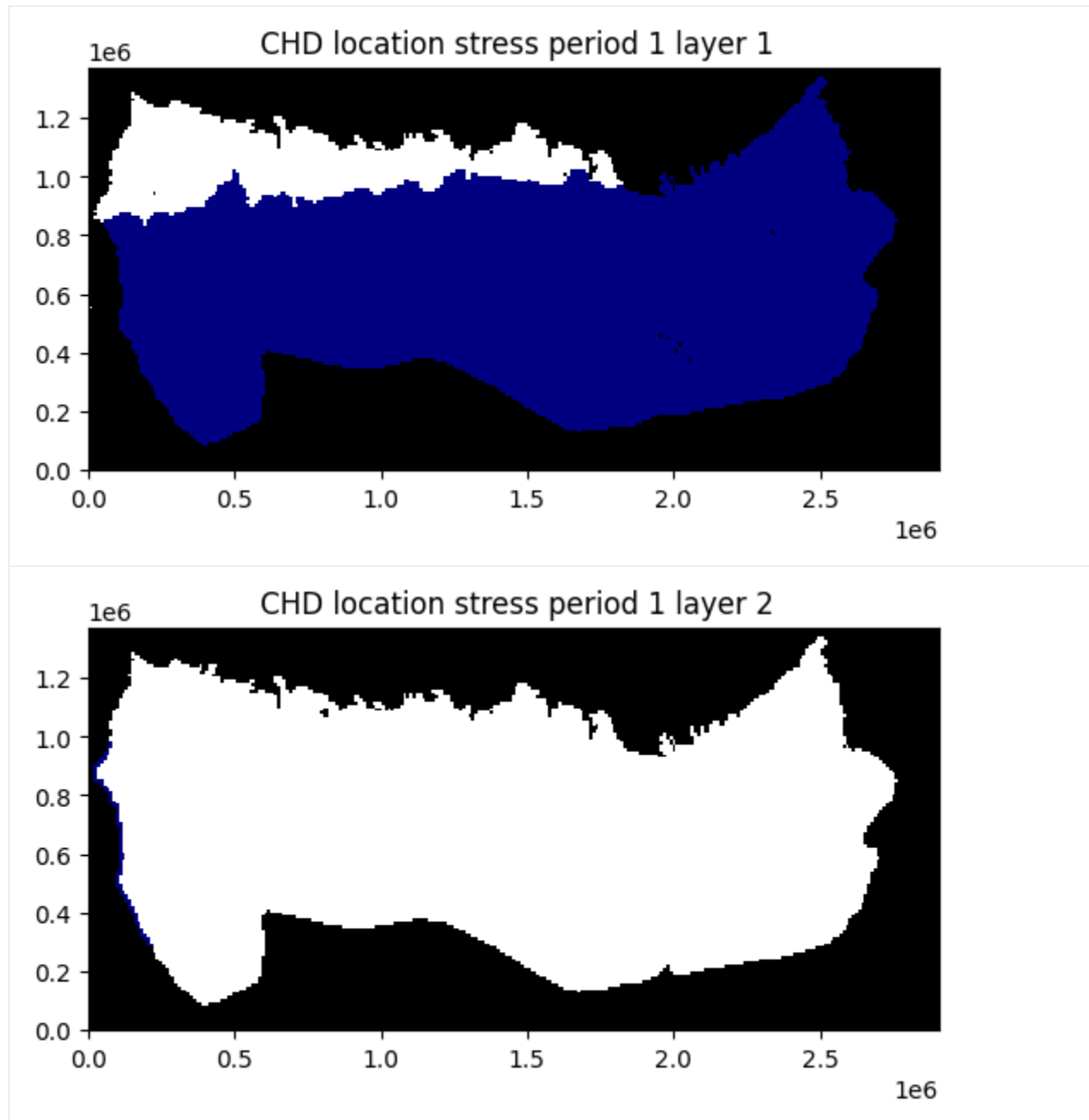


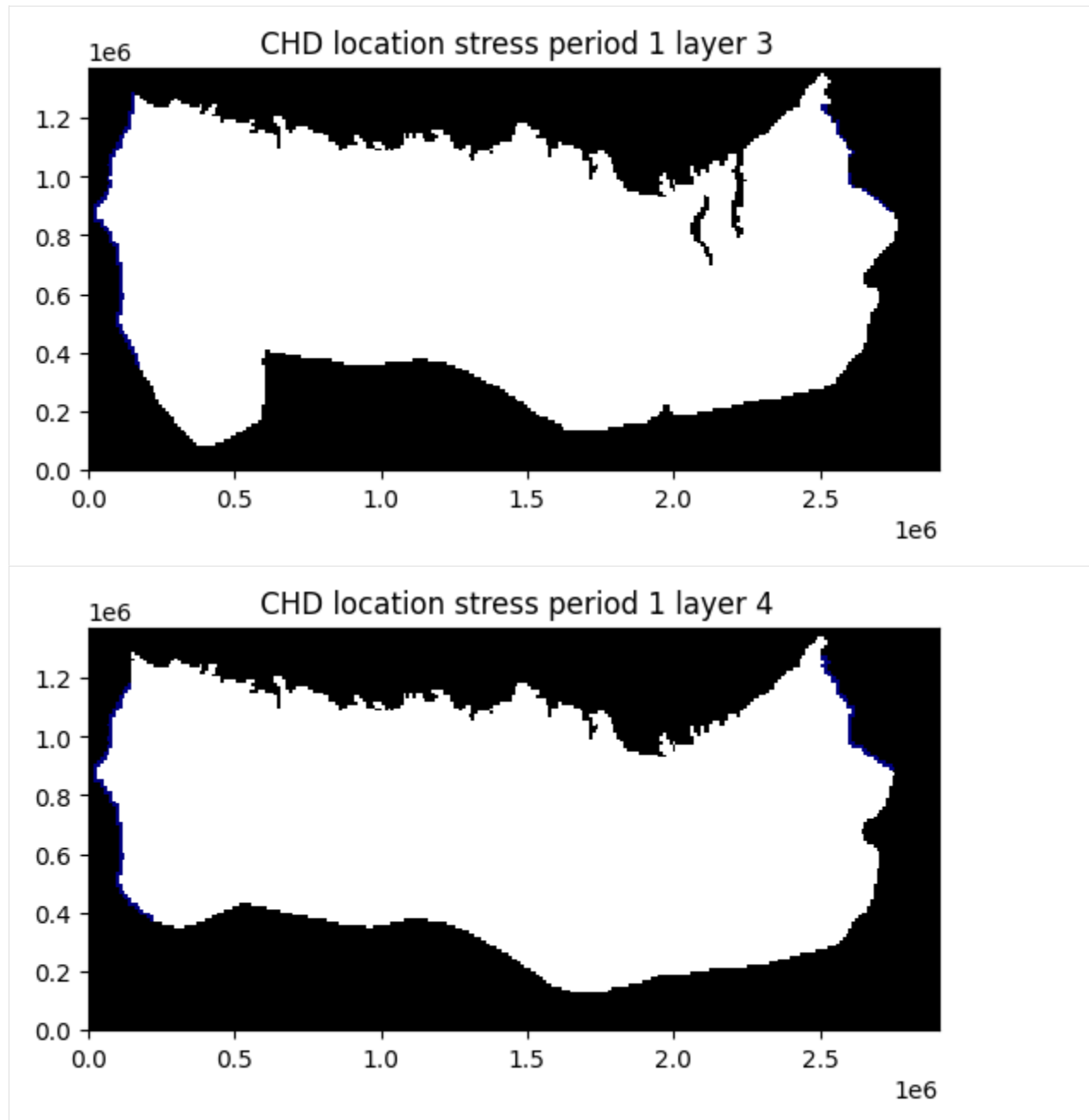


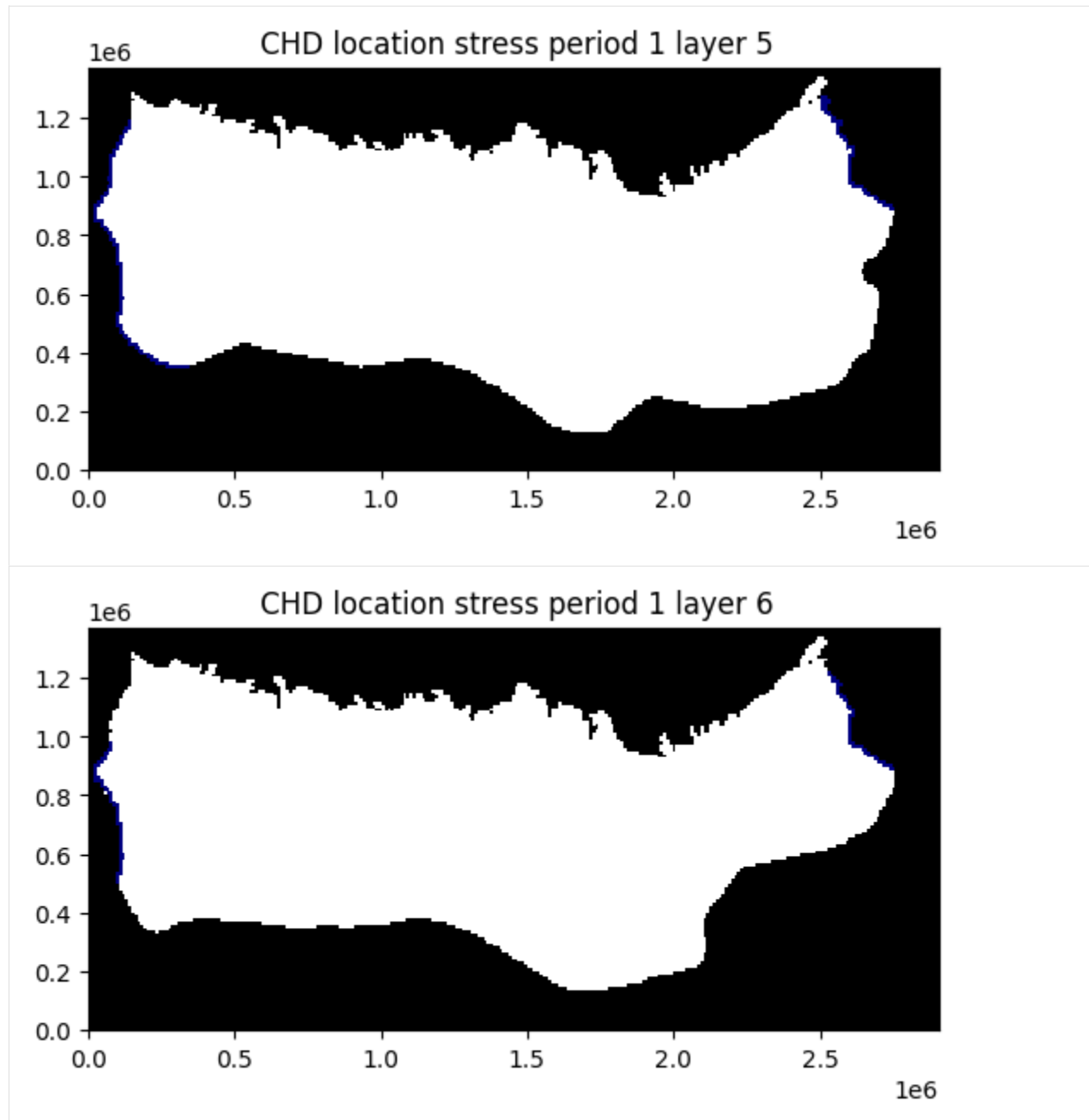


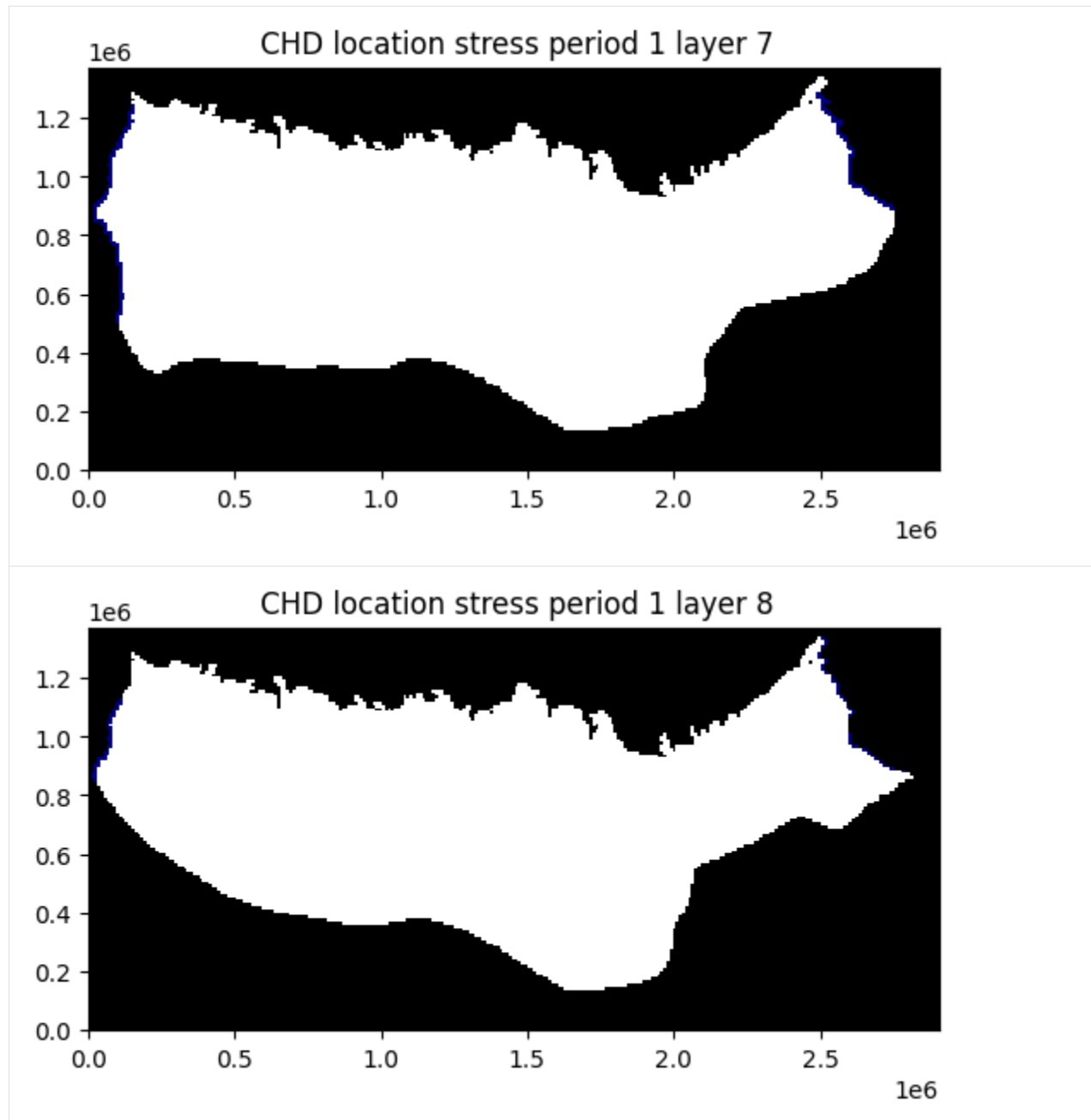


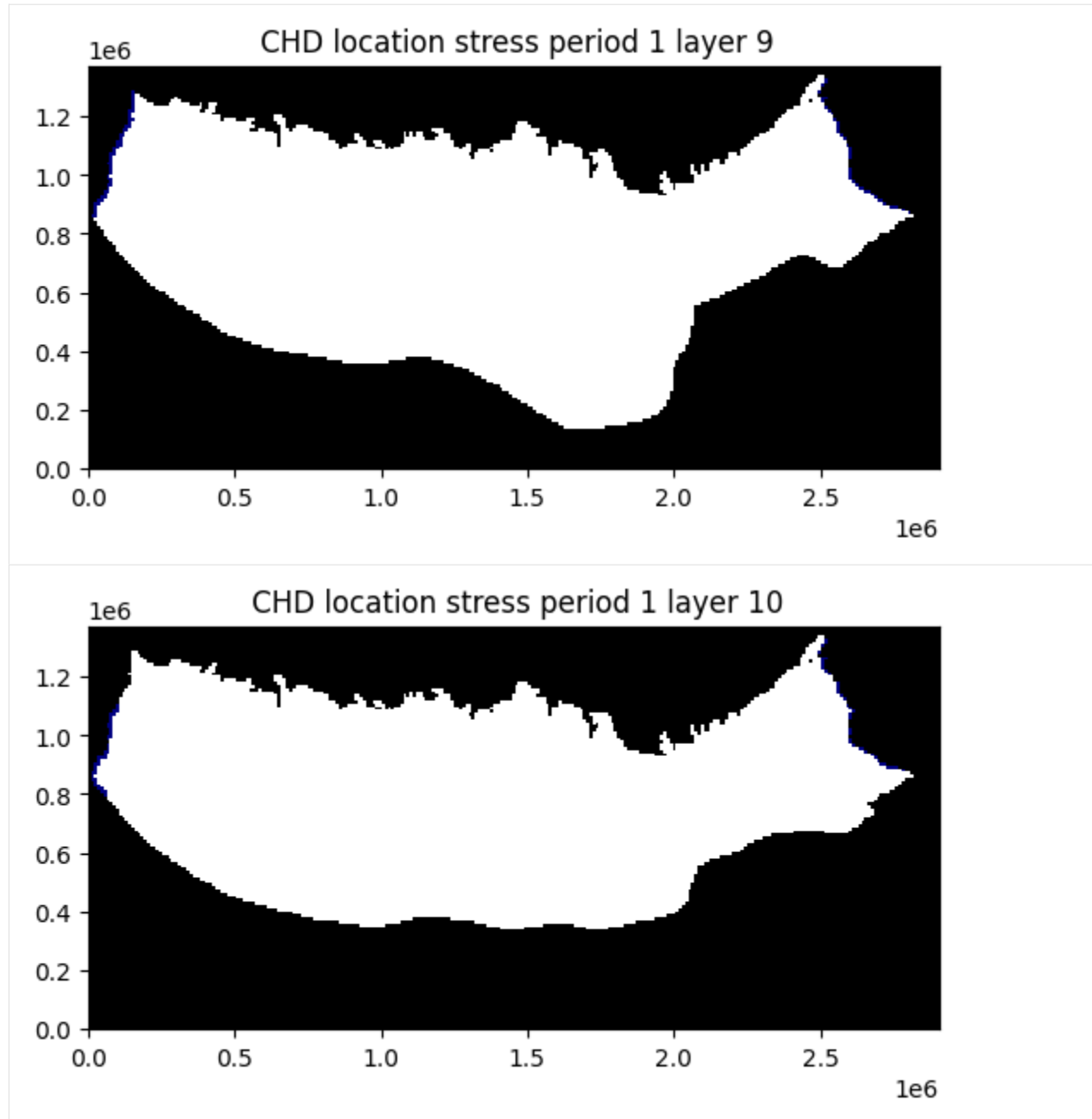


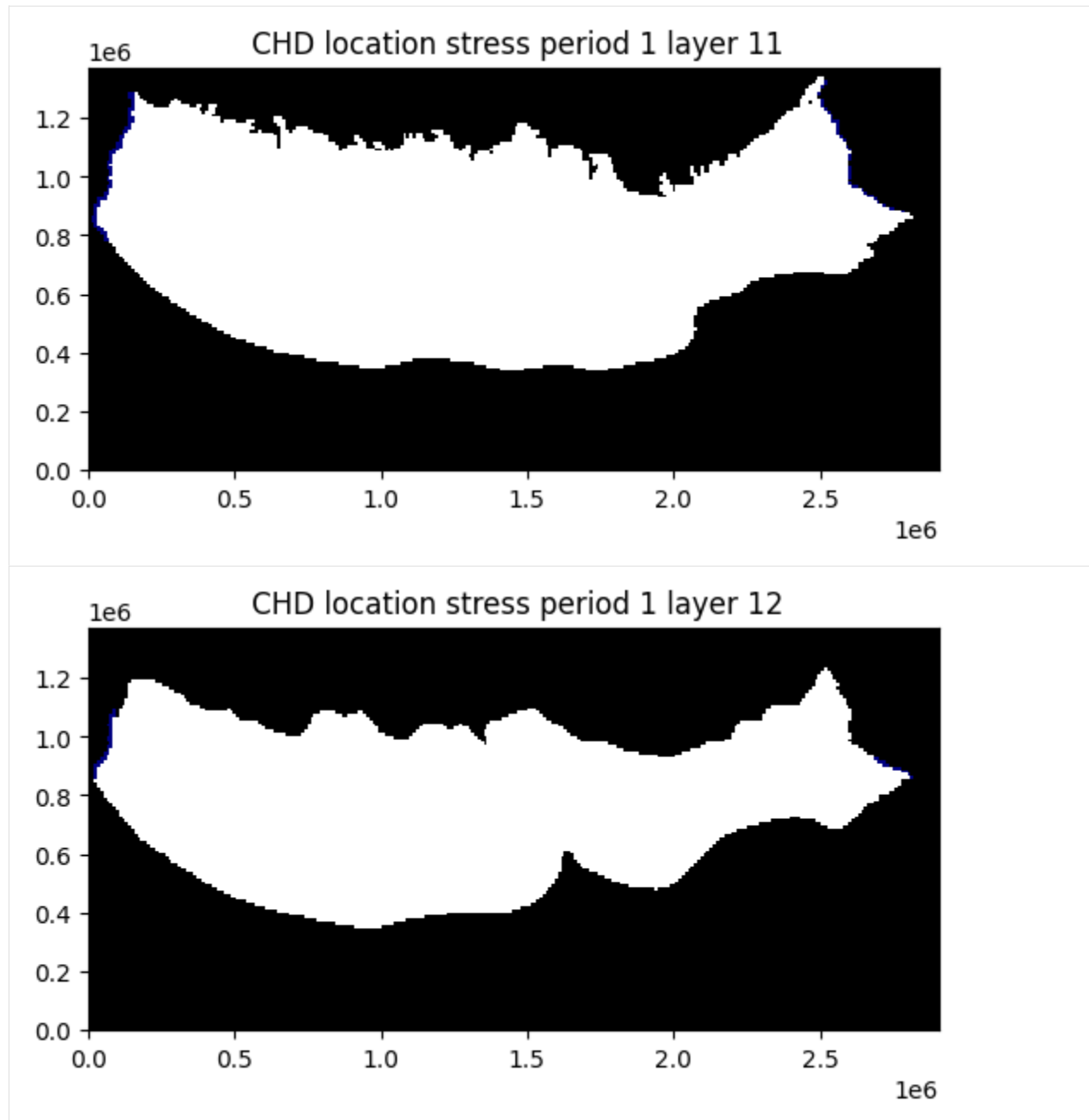


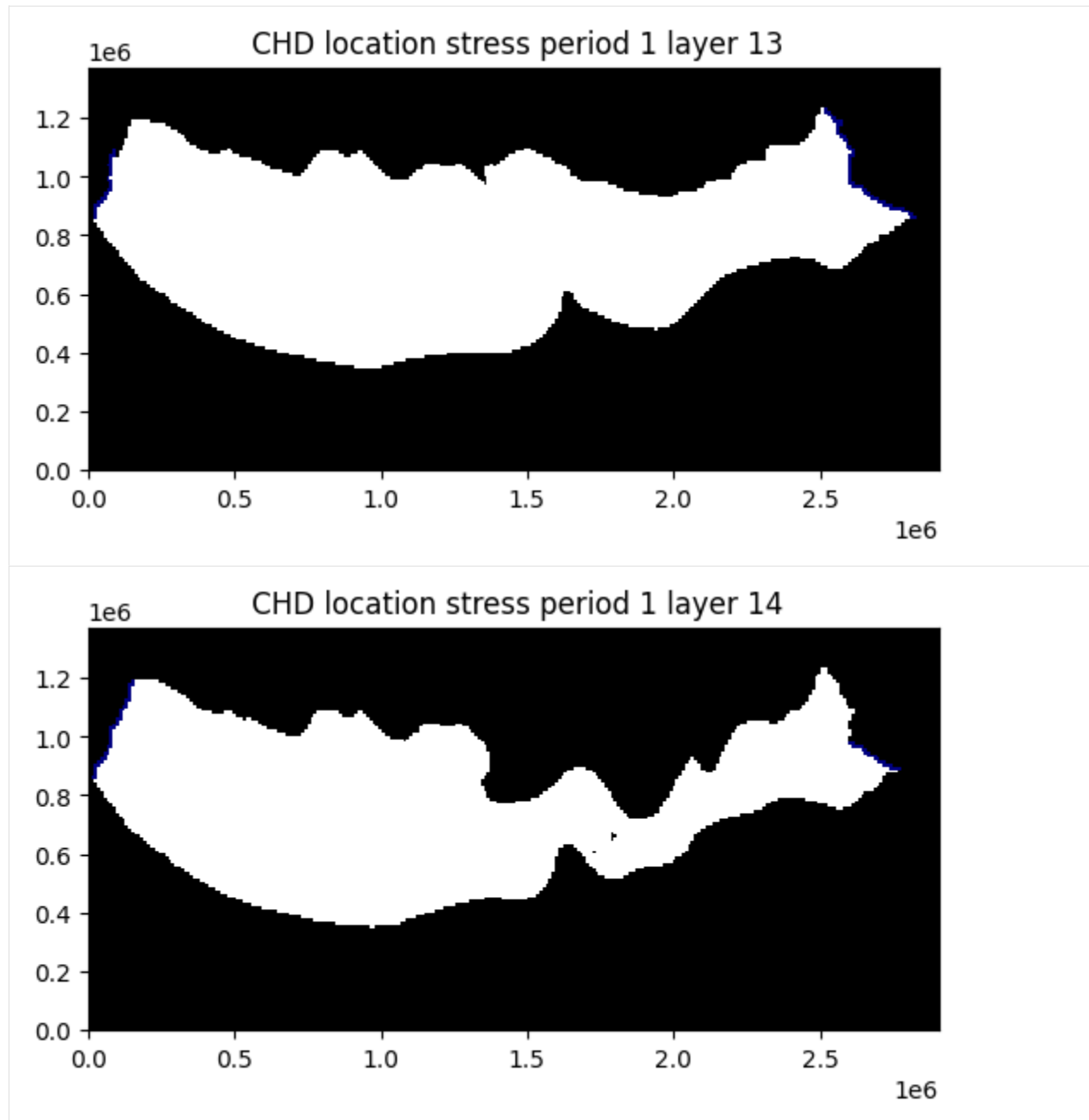


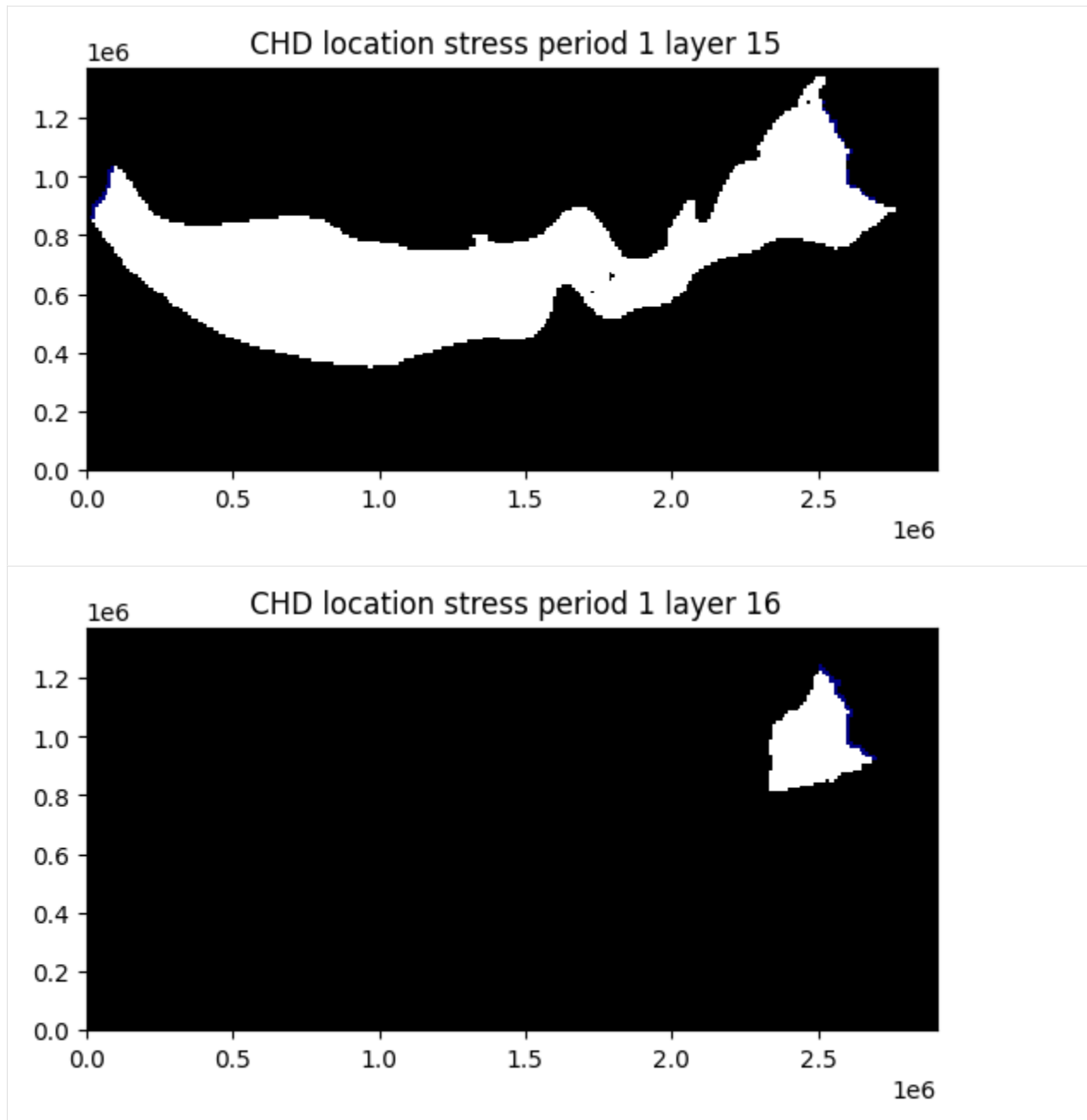








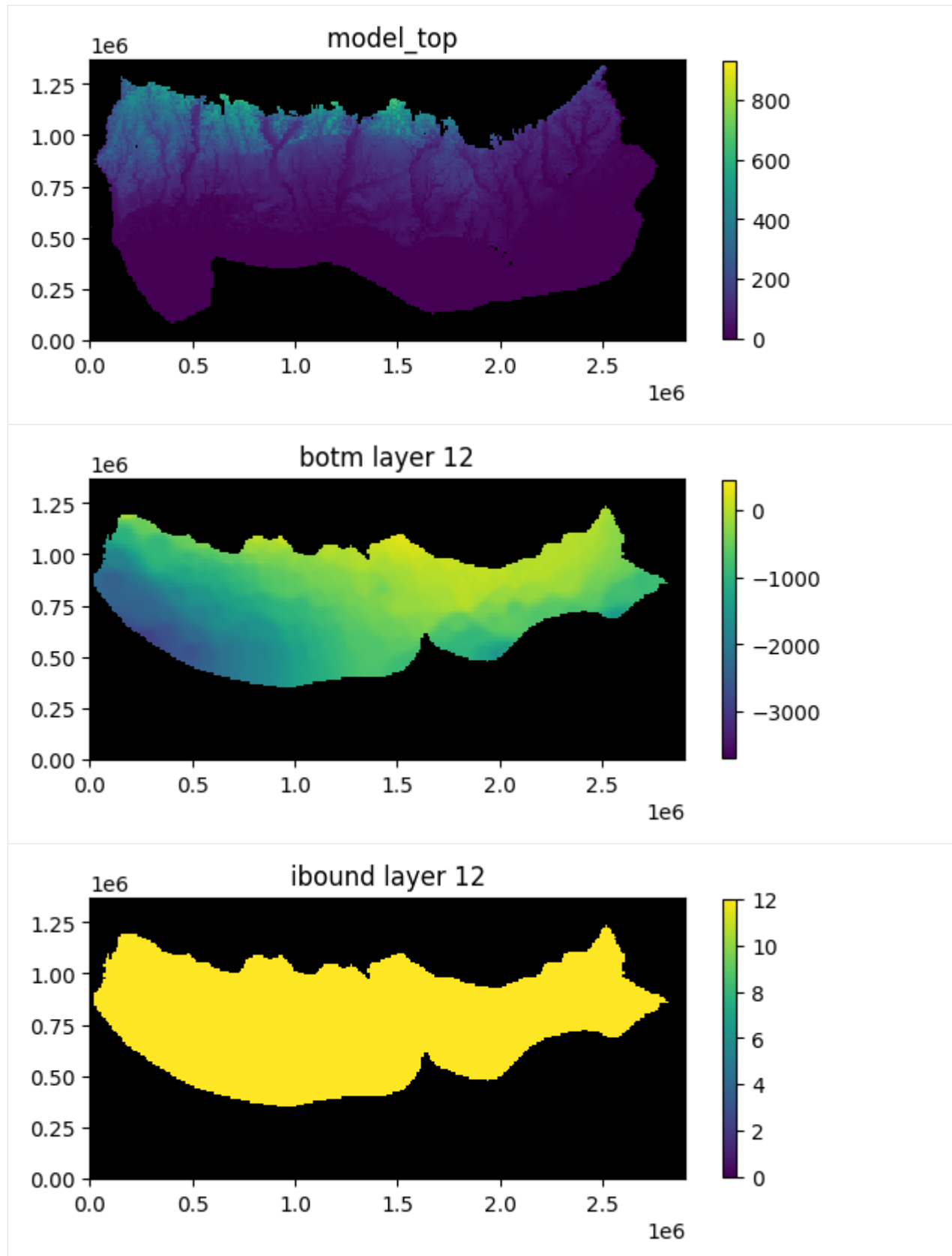


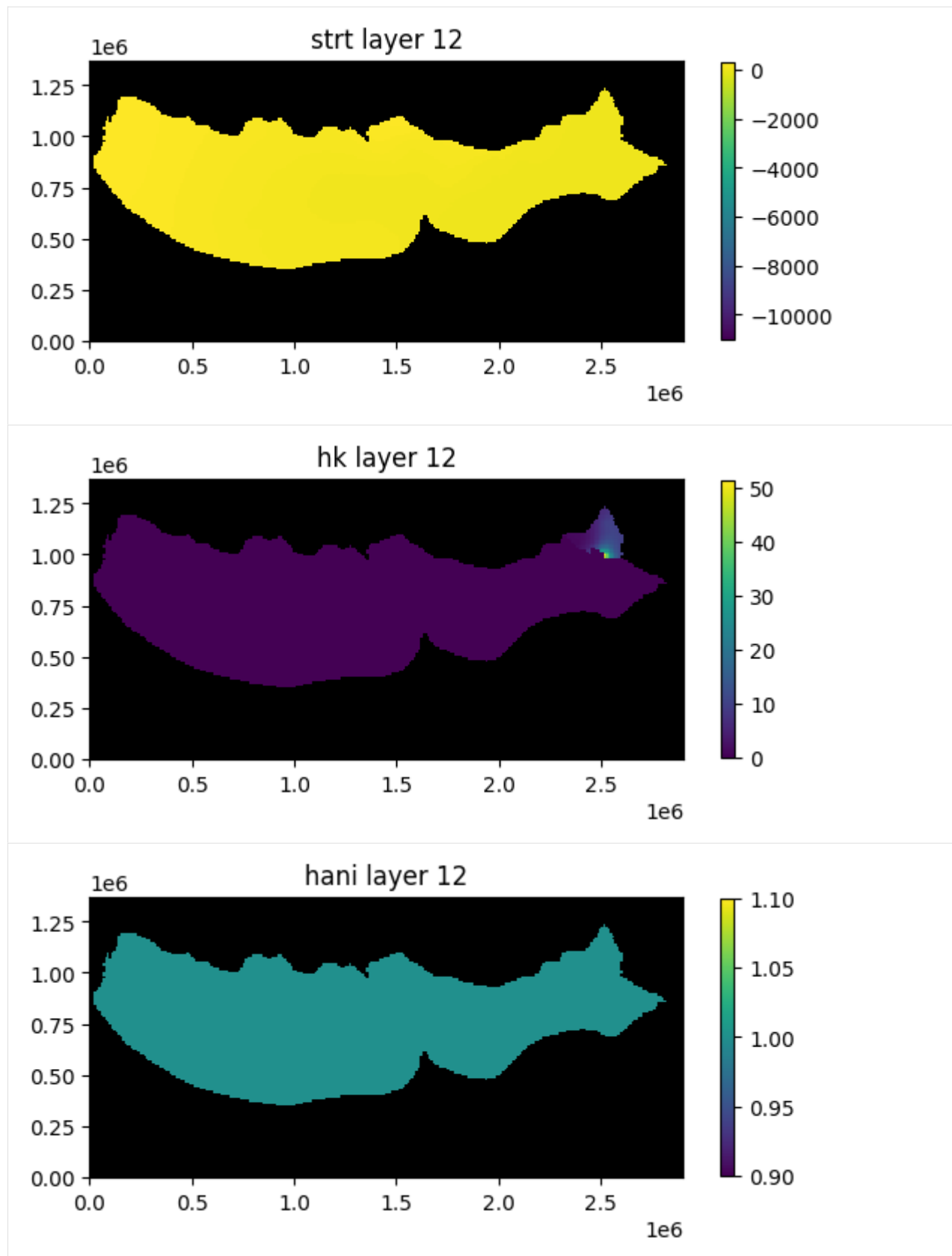


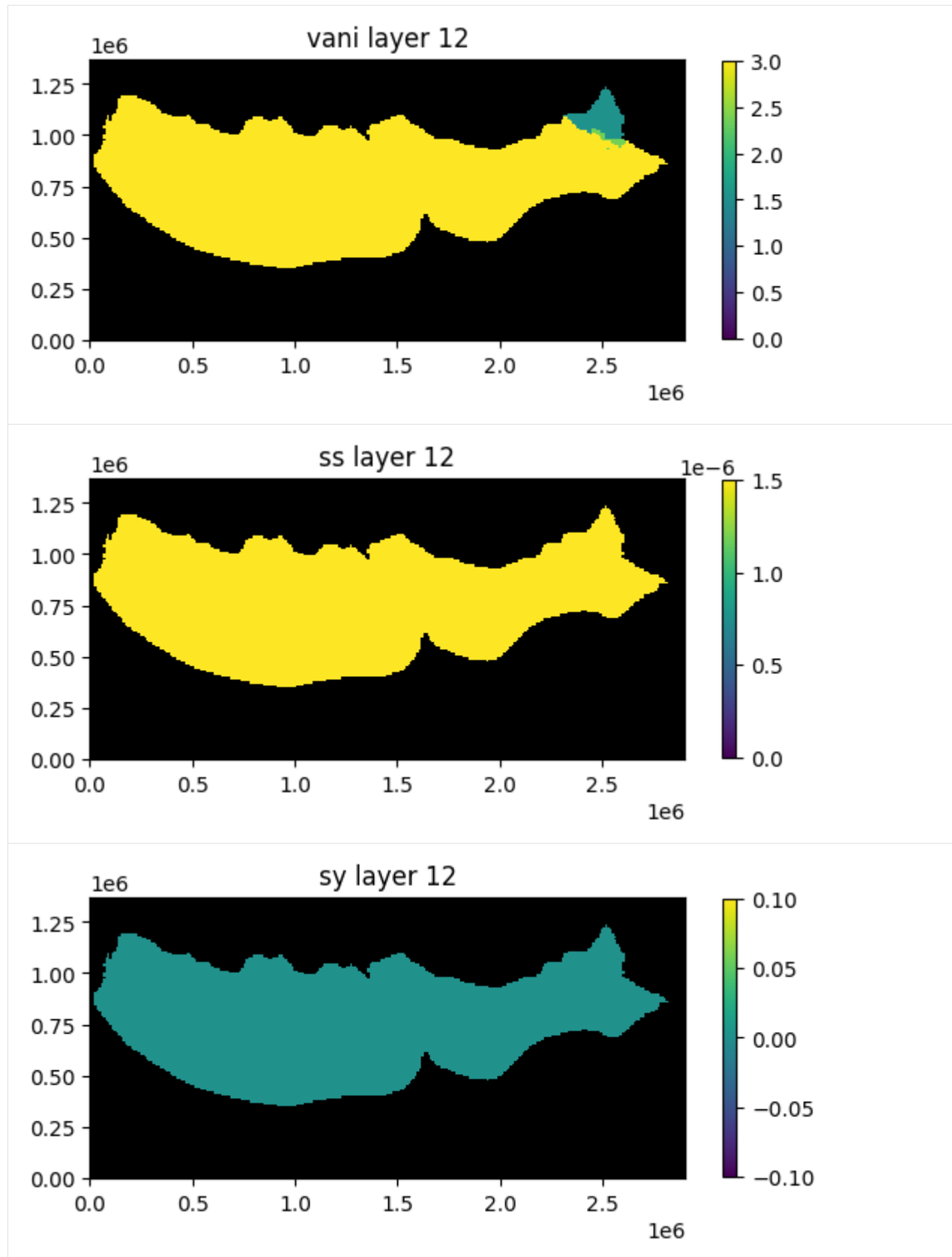
Plot model input data for a specified layer

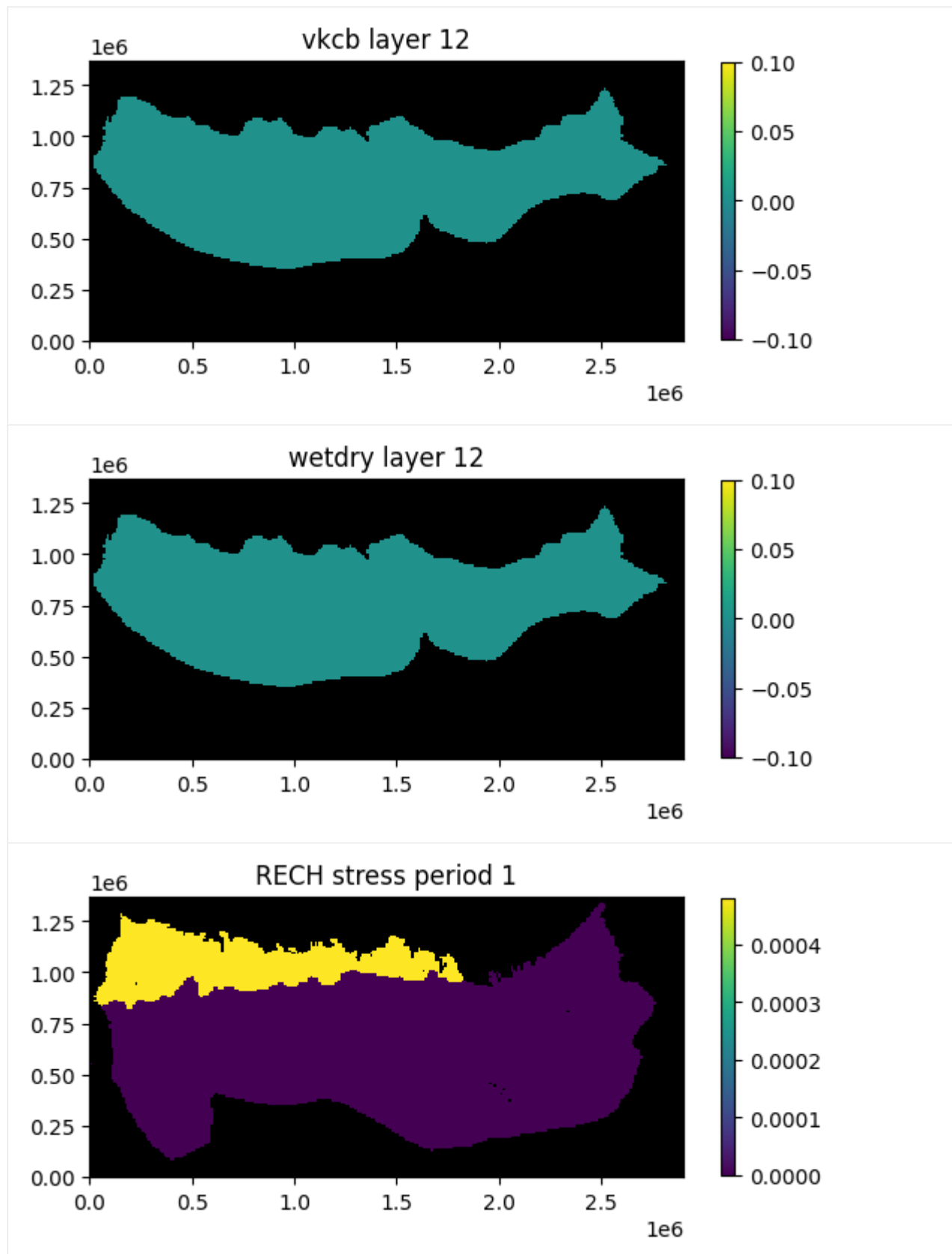
Model input data for a specified layer can be plotted by passing the `mflay` keyword to the package `.plot()` method. Below model input data for layer 12 (`mflay=11`) is plotted.

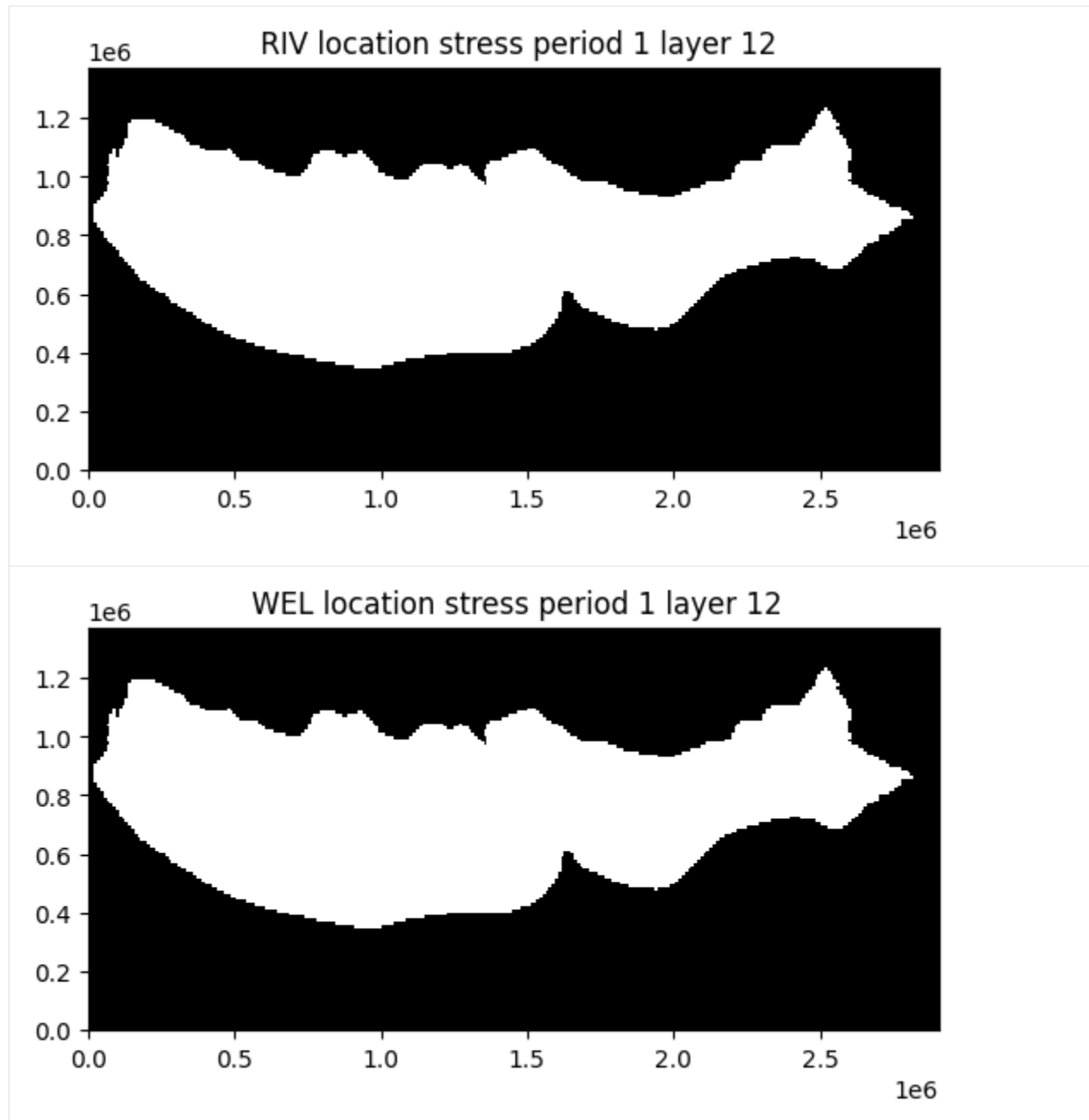
```
[25]: ap = ml.plot(mflay=11)
```

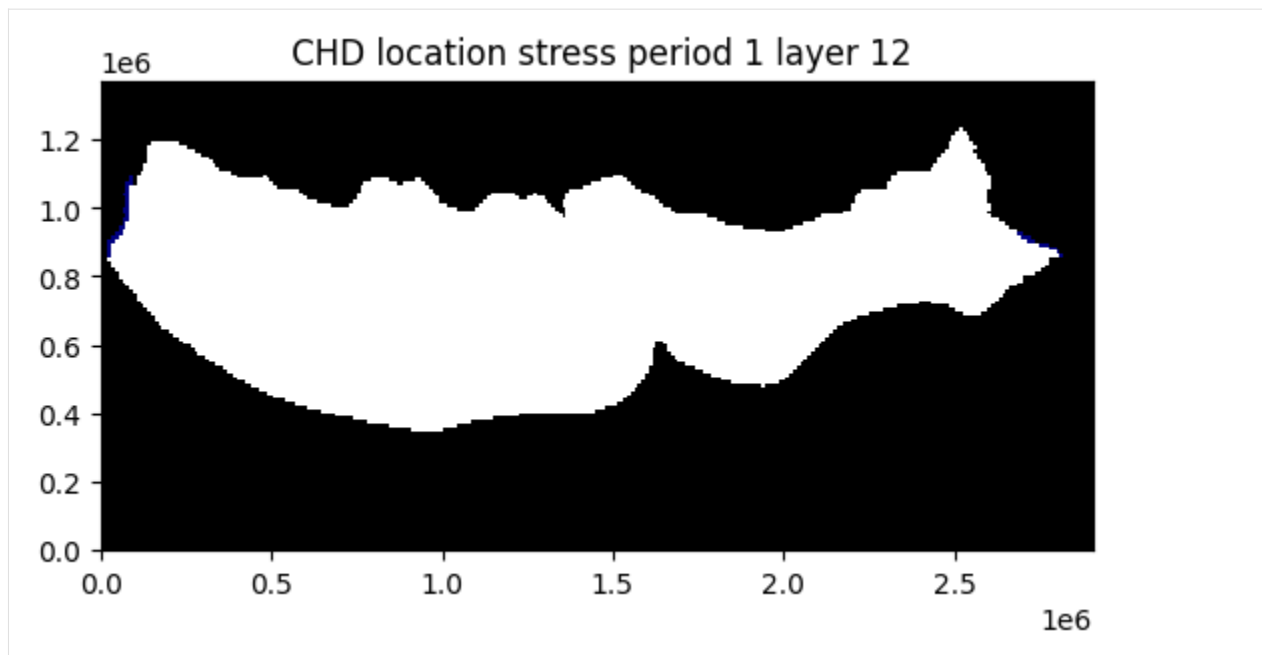












This notebook demonstrates some of the simple plotting functionality available with flopy. Although not described here, the plotting functionality tries to be general by passing keyword arguments passed to the `plot()` and `plot_data()` methods down into the `matplotlib.pyplot` routines that do the actual plotting. For those looking to customize these plots, it may be necessary to search for the available keywords. The `plot()` method return the `matplotlib.pyplot` axis objects that are created (or passed). These axes objects can be used to plot additional data (except when plots are saved as image files).

Hope this gets you started!

```
[26]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.2.3 Making Cross Sections of Your Model

This notebook demonstrates the cross sectional mapping capabilities of FloPy. It demonstrates these capabilities by loading and running existing models and then showing how the `PlotCrossSection` object and its methods can be used to make nice plots of the model grid, boundary conditions, model results, shape files, etc.

Mapping is demonstrated for MODFLOW-2005 and MODFLOW-6 models in this notebook

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
```

(continues on next page)

(continued from previous page)

```

sys.path.append(os.path.join("..", "common"))
import notebook_utils

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3

```

```

[2]: # Set names of the MODFLOW exes
# assumes that the executable is in users path statement
v2005 = "mf2005"
exe_name_2005 = "mf2005"
vmf6 = "mf6"
exe_name_mf6 = "mf6"

# Set the paths
prj_root = notebook_utils.get_project_root_path()
loadpth = str(prj_root / "examples" / "data" / "freyberg")
tempdir = TemporaryDirectory()
modelpth = tempdir.name

```

Load and Run an Existing MODFLOW-2005 Model

A model called the “Freyberg Model” is located in the loadpth folder. In the following code block, we load that model, then change into a new workspace (modelpth) where we recreate and run the model. For this to work properly, the MODFLOW-2005 executable (mf2005) must be in the path. We verify that it worked correctly by checking for the presence of freyberg.hds and freyberg.cbc.

```

[3]: ml = flopy.modflow.Modflow.load(
    "freyberg.nam", model_ws=loadpth, exe_name=exe_name_2005, version=v2005
)
ml.change_model_ws(new_pth=str(modelpth))
ml.write_input()
success, buff = ml.run_model(silent=True, report=True)
if success:
    for line in buff:

```

(continues on next page)

(continued from previous page)

```

        print(line)
    else:
        raise ValueError("Something bad happened.")

files = ["freyberg.hds", "freyberg.cbc"]
for f in files:
    if os.path.isfile(os.path.join(str(modelpth), f)):
        msg = "Output file located: {}".format(f)
        print(msg)
    else:
        errmsg = "Error. Output file cannot be found: {}".format(f)
        print(errmsg)

```

```

                                MODFLOW-2005
    U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
                                Version 1.12.00 2/3/2017

Using NAME file: freyberg.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:58

Solving:  Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:58
Elapsed run time:  0.008 Seconds

    Normal termination of simulation
Output file located: freyberg.hds
Output file located: freyberg.cbc

```

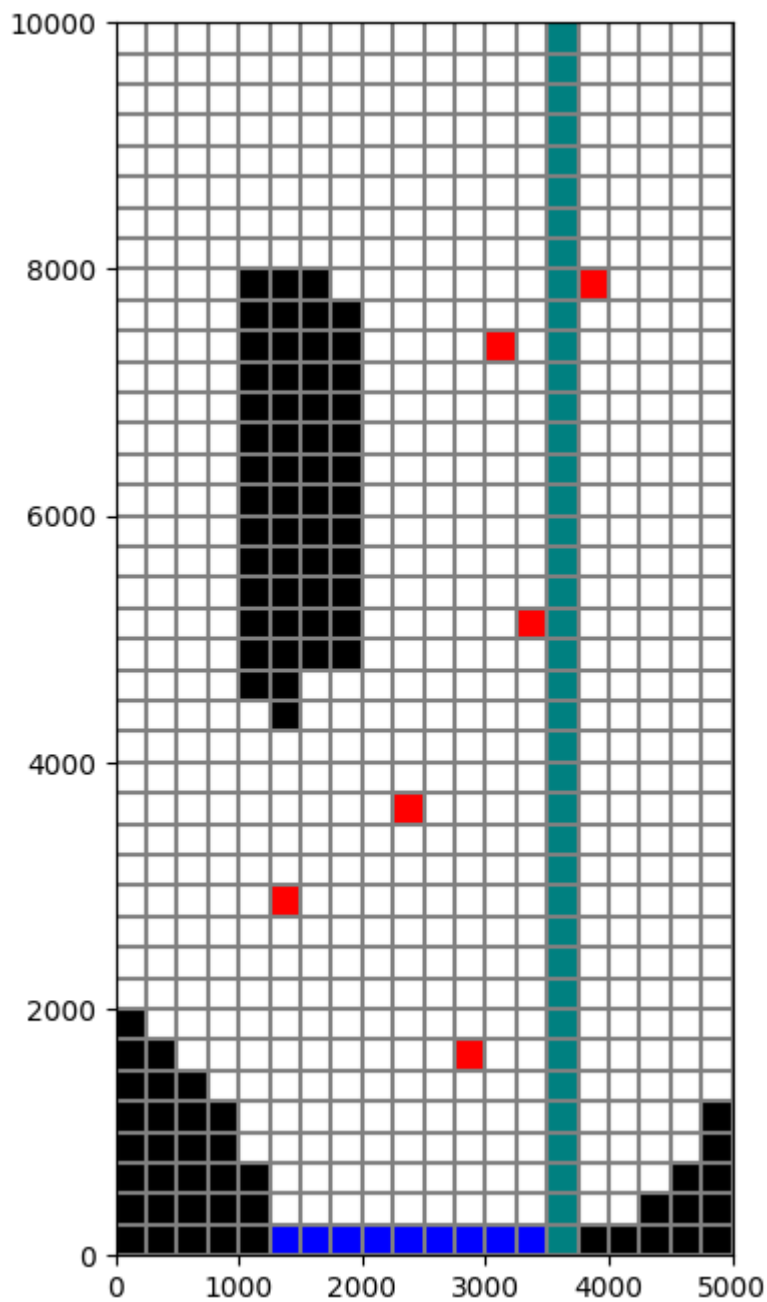
Creating a Cross-Section of the Model Grid

Now that we have a model, we can use the FloPy plotting utilities to make cross-sections. We'll start by making a Map to show the model grid and basic boundary conditions. Then we'll begin making a cross section using the PlotCrossSection class and the plot_grid() method of that class.

```

[4]: # let's take a look at our grid before making a cross section
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mapview = flopy.plot.PlotMapView(model=ml)
ibound = mapview.plot_ibound()
wel = mapview.plot_bc("WEL")
riv = mapview.plot_bc("RIV")
linecollection = mapview.plot_grid()

```



Next we will make a cross-section of the model grid at column 6.

```
[5]: # First step is to set up the plot
fig = plt.figure(figsize=(15, 5))
ax = fig.add_subplot(1, 1, 1)

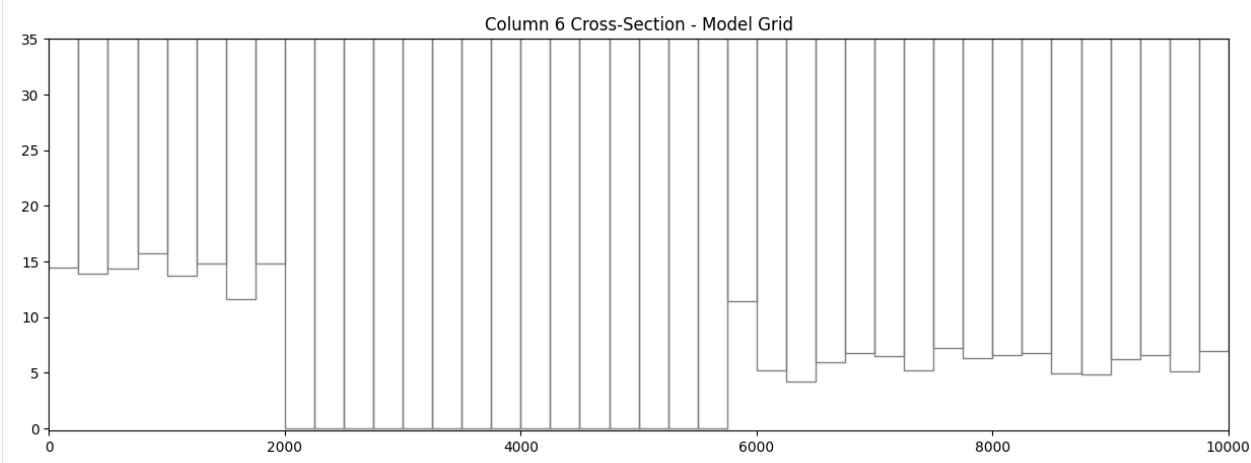
# Next we create an instance of the PlotCrossSection class
xsct = flopy.plot.PlotCrossSection(model=m1, line={"Column": 5})

# Then we can use the plot_grid() method to draw the grid
```

(continues on next page)

(continued from previous page)

```
# The return value for this function is a matplotlib LineCollection object,
# which could be manipulated (or used) later if necessary.
linecollection = xsect.plot_grid()
t = ax.set_title("Column 6 Cross-Section - Model Grid")
```

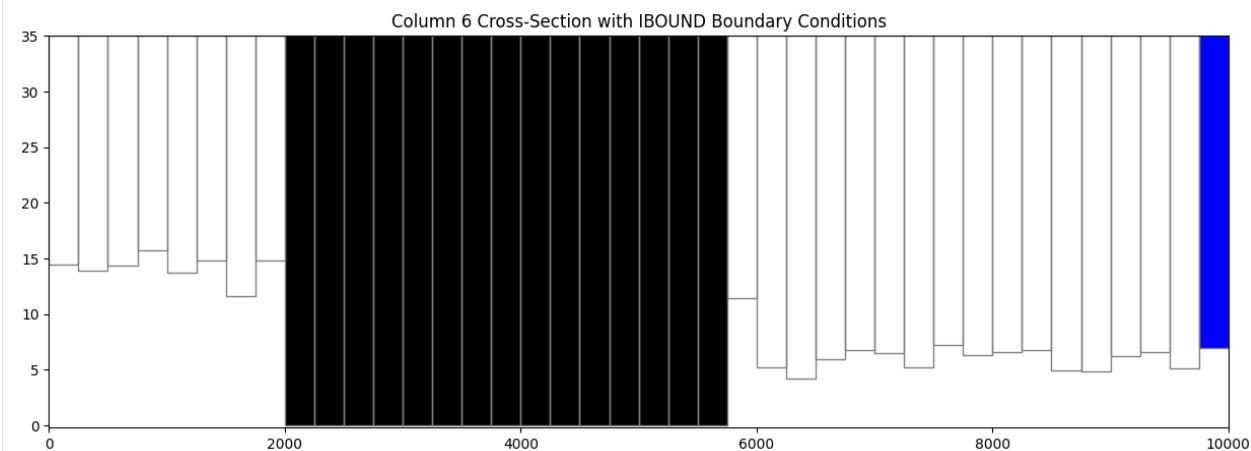


Plotting Ibound

The `plot_ibound()` method can be used to plot the boundary conditions contained in the `ibound` array, which is part of the MODFLOW Basic Package. The `plot_ibound()` method returns a matplotlib `PatchCollection` object (`matplotlib.collections.PatchCollection`). If you are familiar with the matplotlib collections, then this may be important to you, but if not, then don't worry about the return objects of these plotting function.

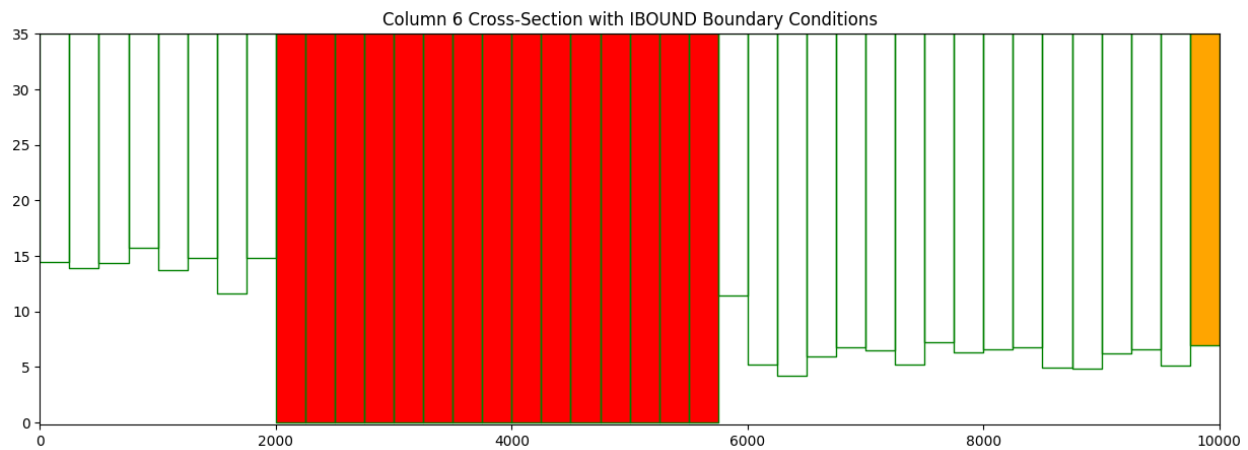
```
[6]: fig = plt.figure(figsize=(15, 5))
ax = fig.add_subplot(1, 1, 1)

xsect = flopy.plot.PlotCrossSection(model=m1, line={"Column": 5})
patches = xsect.plot_ibound()
linecollection = xsect.plot_grid()
t = ax.set_title("Column 6 Cross-Section with IBOUND Boundary Conditions")
```



```
[7]: # Or we could change the colors!
fig = plt.figure(figsize=(15, 5))
ax = fig.add_subplot(1, 1, 1)

xsect = flopy.plot.PlotCrossSection(model=m1, line={"Column": 5})
patches = xsect.plot_ibound(color_noflow="red", color_ch="orange")
linecollection = xsect.plot_grid(color="green")
t = ax.set_title("Column 6 Cross-Section with IBOUND Boundary Conditions")
```



Plotting Boundary Conditions

The `plot_bc()` method can be used to plot boundary conditions on a cross section. It is setup to use the following dictionary to assign colors, however, these colors can be changed in the method call.

```
bc_color_dict = {'default': 'black', 'WEL': 'red', 'DRN': 'yellow',
                 'RIV': 'green', 'GHB': 'cyan', 'CHD': 'navy'}
```

Just like the `plot_bc()` method for `PlotMapView`, the default boundary condition colors can be changed in the method call.

Here, we plot the location of well cells in column 6.

```
[8]: fig = plt.figure(figsize=(15, 5))
ax = fig.add_subplot(1, 1, 1)

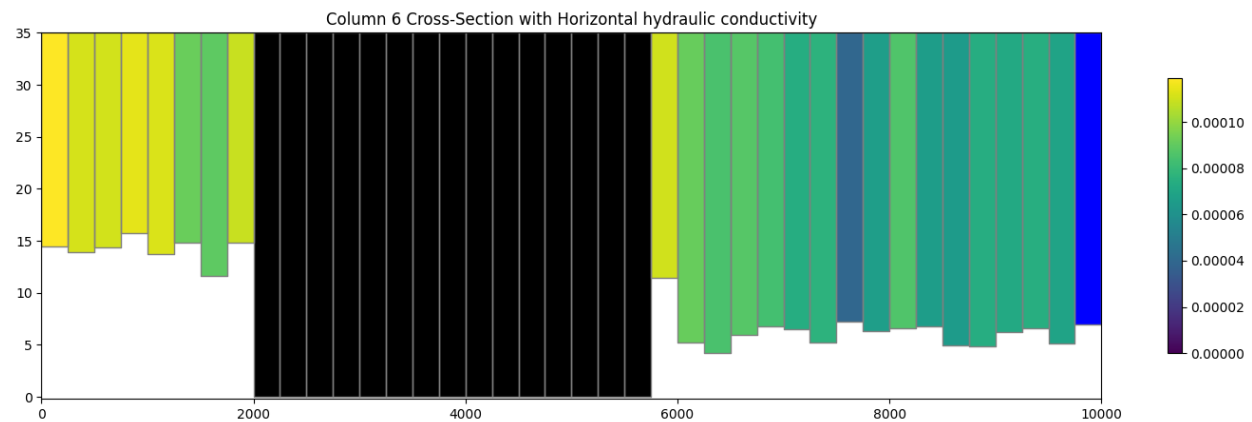
xsect = flopy.plot.PlotCrossSection(model=m1, line={"Column": 5})
patches = xsect.plot_bc("WEL", color="pink")
patches = xsect.plot_ibound()
linecollection = xsect.plot_grid()
t = ax.set_title("Column 6 Cross-Section with Boundary Conditions")
```


(continued from previous page)

```

xsect = flopy.plot.PlotCrossSection(model=m1, line={"Column": 5})
csa = xsect.plot_array(a)
patches = xsect.plot_ibound()
linecollection = xsect.plot_grid()
t = ax.set_title(
    "Column 6 Cross-Section with Horizontal hydraulic conductivity"
)
cb = plt.colorbar(csa, shrink=0.75)

```



Contouring an Array

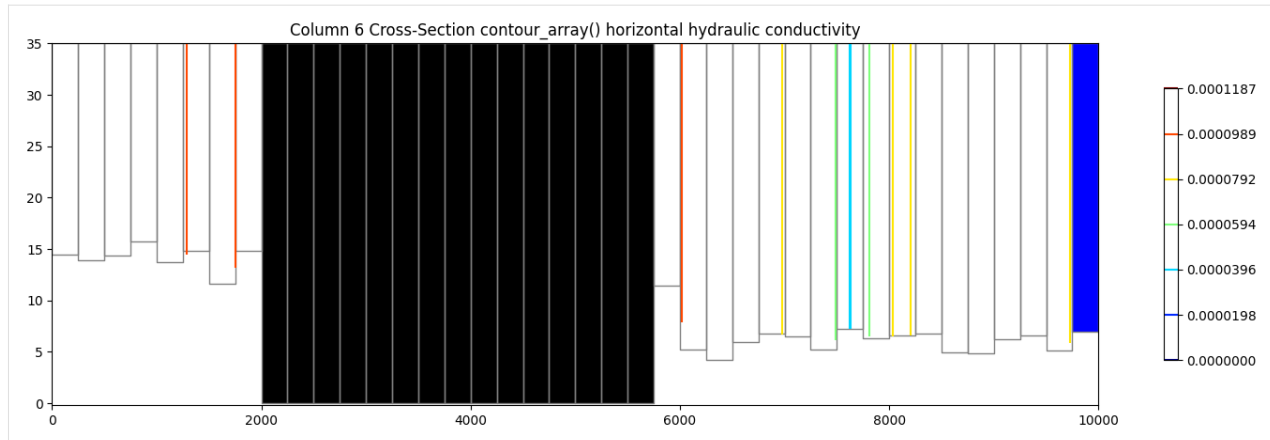
PlotCrossSection also has a `contour_array()` method. It also accepts a 3D array for structured grids.

```

[11]: # plot the horizontal hydraulic conductivities
a = m1.lpf.hk.array

fig = plt.figure(figsize=(18, 5))
ax = fig.add_subplot(1, 1, 1)
xsect = flopy.plot.PlotCrossSection(model=m1, line={"Column": 5})
contour_set = xsect.contour_array(a, masked_values=[0], cmap="jet")
patches = xsect.plot_ibound()
linecollection = xsect.plot_grid(color="grey")
t = ax.set_title(
    "Column 6 Cross-Section contour_array() horizontal hydraulic conductivity"
)
cb = plt.colorbar(contour_set, shrink=0.75)

```



Plotting Heads

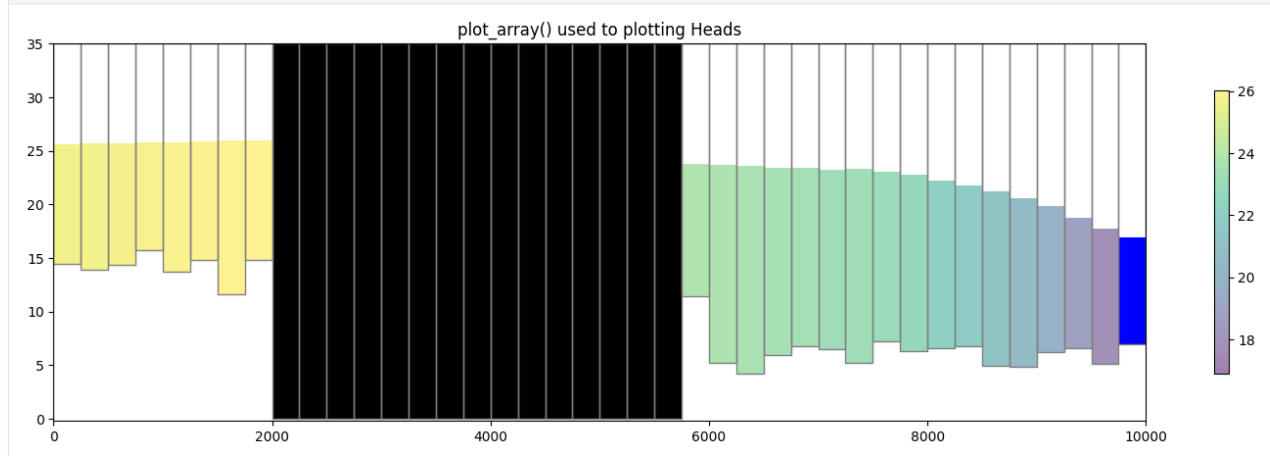
We can easily plot results from the simulation by extracting heads using `flopy.utils.HeadFile`.

The head can be passed into the `plot_array()` and `contour_array()` using the `head=` keyword argument to fix the top of the colored patch and contour lines at the top of the water table in each cell, respectively.

```
[12]: fname = os.path.join(str(modelpth), "freyberg.hds")
      hdojb = flopy.utils.HeadFile(fname)
      head = hdojb.get_data()

      fig = plt.figure(figsize=(18, 5))

      ax = fig.add_subplot(1, 1, 1)
      ax.set_title("plot_array() used to plotting Heads")
      xsect = flopy.plot.PlotCrossSection(model=m1, line={"Column": 5})
      pc = xsect.plot_array(head, head=head, alpha=0.5)
      patches = xsect.plot_ibound(head=head)
      linecollection = xsect.plot_grid()
      cb = plt.colorbar(pc, shrink=0.75)
```



```
[13]: # contour array on top of heads
      levels = np.arange(17, 26, 1)
```

(continues on next page)

(continued from previous page)

```

fig = plt.figure(figsize=(18, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_title("contour_array() and plot_array() of head values")

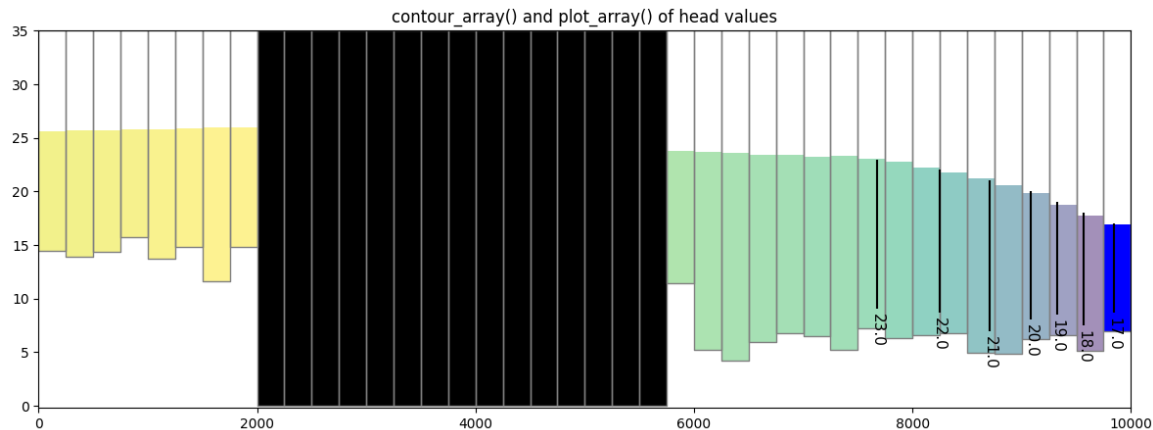
# instantiate the PlotCrossSection object
xsct = flopy.plot.PlotCrossSection(model=m1, line={"Column": 5})

# plot the head array and model grid
pc = xsct.plot_array(head, masked_values=[999.0], head=head, alpha=0.5)
patches = xsct.plot_ibound(head=head)
linecollection = xsct.plot_grid()

# do black contour lines of the head array
contour_set = xsct.contour_array(head, head=head, levels=levels, colors="k")
plt.clabel(contour_set, fmt="%.1f", colors="k", fontsize=11)

cb = plt.colorbar(pc, shrink=0.75)

```



Plotting a surface on the cross section

The `plot_surface()` method allows the user to plot a surface along the cross section. Here is a short example using head data.

```

[14]: levels = np.arange(10, 30, 0.5)

fig = plt.figure(figsize=(18, 5))
xsct = flopy.plot.PlotCrossSection(model=m1, line={"Column": 5})

# contour array and plot ibound
ct = xsct.contour_array(
    head, masked_values=[999.0], head=head, levels=levels, linewidths=2.5
)
pc = xsct.plot_ibound(head=head)

# plot the surface and model grid
wt = xsct.plot_surface(head, color="blue", lw=2.5)

```

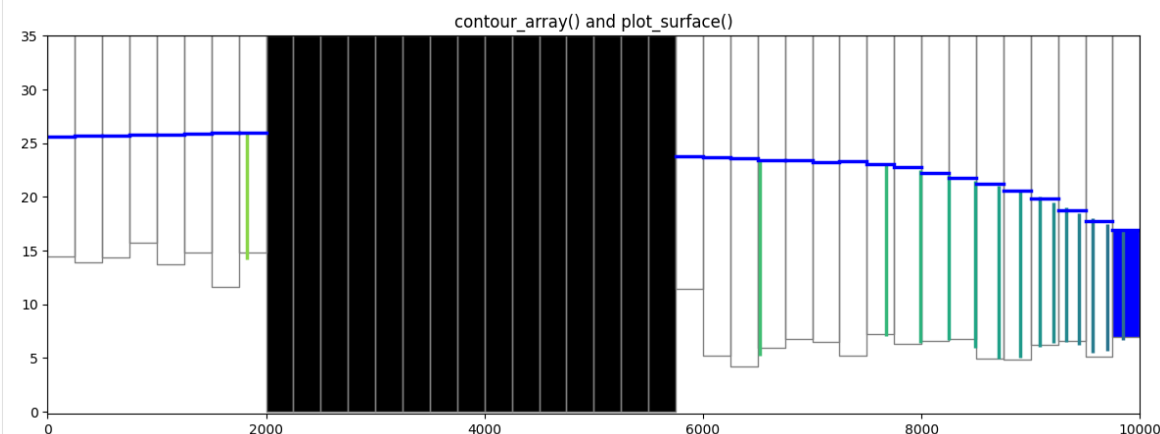
(continues on next page)

(continued from previous page)

```
linecollection = xsect.plot_grid()
```

```
plt.title("contour_array() and plot_surface()")
```

```
cb = plt.colorbar(ct, shrink=0.75)
```



Plotting discharge vectors

PlotCrossSection has a `plot_vector()` method, which takes `qx`, `qy`, and `qz` vector arrays (ex. specific discharge or flow across a cell faces). The flow array values can be extracted from the cell by cell flow file using the `flopy.utils.CellBudgetFile` object as shown below. Once they are extracted, they either be can be passed to the `plot_vector()` method or they can be post processed into specific discharge using `postprocessing.get_specific_discharge`. Note that `get_specific_discharge()` also takes the head array as an argument. The head array is used by `get_specific_discharge()` to convert the volumetric flow in dimensions of L^3/T to specific discharge in dimensions of L/T and to plot the specific discharge in the center of each saturated cell. For this problem, there is no 'FLOW LOWER FACE' array since the Freyberg Model is a one layer model.

```
[15]: fname = os.path.join(str(modelpth), "freyberg.cbc")
cbb = flopy.utils.CellBudgetFile(fname)
frf = cbb.get_data(text="FLOW RIGHT FACE")[0]
fff = cbb.get_data(text="FLOW FRONT FACE")[0]
qx, qy, qz = flopy.utils.postprocessing.get_specific_discharge(
    (frf, fff, None), ml, head=head
)
```

```
[16]: fig = plt.figure(figsize=(18, 5))
ax = fig.add_subplot(1, 1, 1)

ax.set_title("plot_array() and plot_vector()")
xsect = flopy.plot.PlotCrossSection(model=ml, ax=ax, line={"Column": 5})
csa = xsect.plot_array(head, head=head, alpha=0.5)
patches = xsect.plot_ibound(head=head)
linecollection = xsect.plot_grid()
quiver = xsect.plot_vector(
    qx,
    qy,
    qz,
```

(continues on next page)

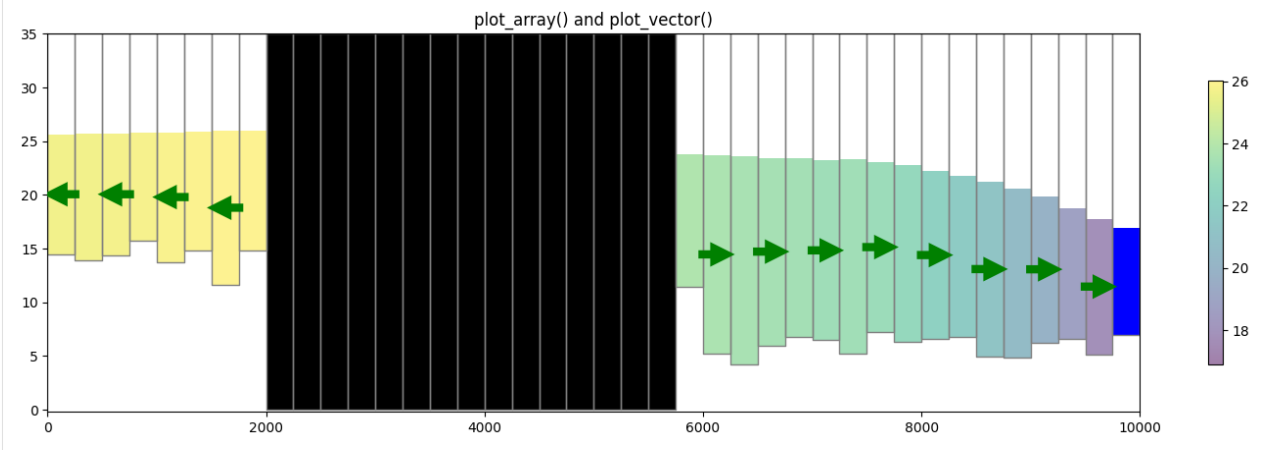
(continued from previous page)

```

    head=head,
    hstep=2,
    normalize=True,
    color="green",
    scale=30,
    headwidth=3,
    headlength=3,
    headaxislength=3,
    zorder=10,
)

cb = plt.colorbar(csa, shrink=0.75)

```



Plotting a cross section from Shapefile data

A shapefile can be used to define the vertices for a instance of the `PlotCrossSection` class. The function `flopy.plot.plotutil.shapefile_get_vertices()` will return a list of vertices for each polyline in a shapefile.

Let's plot the shapefiles and the Freyberg model using `PlotMapView` for visualization purposes and then plot the cross-section.

```

[17]: # Setup the figure and PlotMapView. Show a very faint map of ibound and
      # model grid by specifying a transparency alpha value.

      # set the modelgrid rotation and offset
      ml.modelgrid.set_coord_info(
          xoff=-2419.2189559966773, yoff=297.0427372400354, angrot=-14
      )

      fig = plt.figure(figsize=(12, 12))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      mapview = flopy.plot.PlotMapView(model=ml)

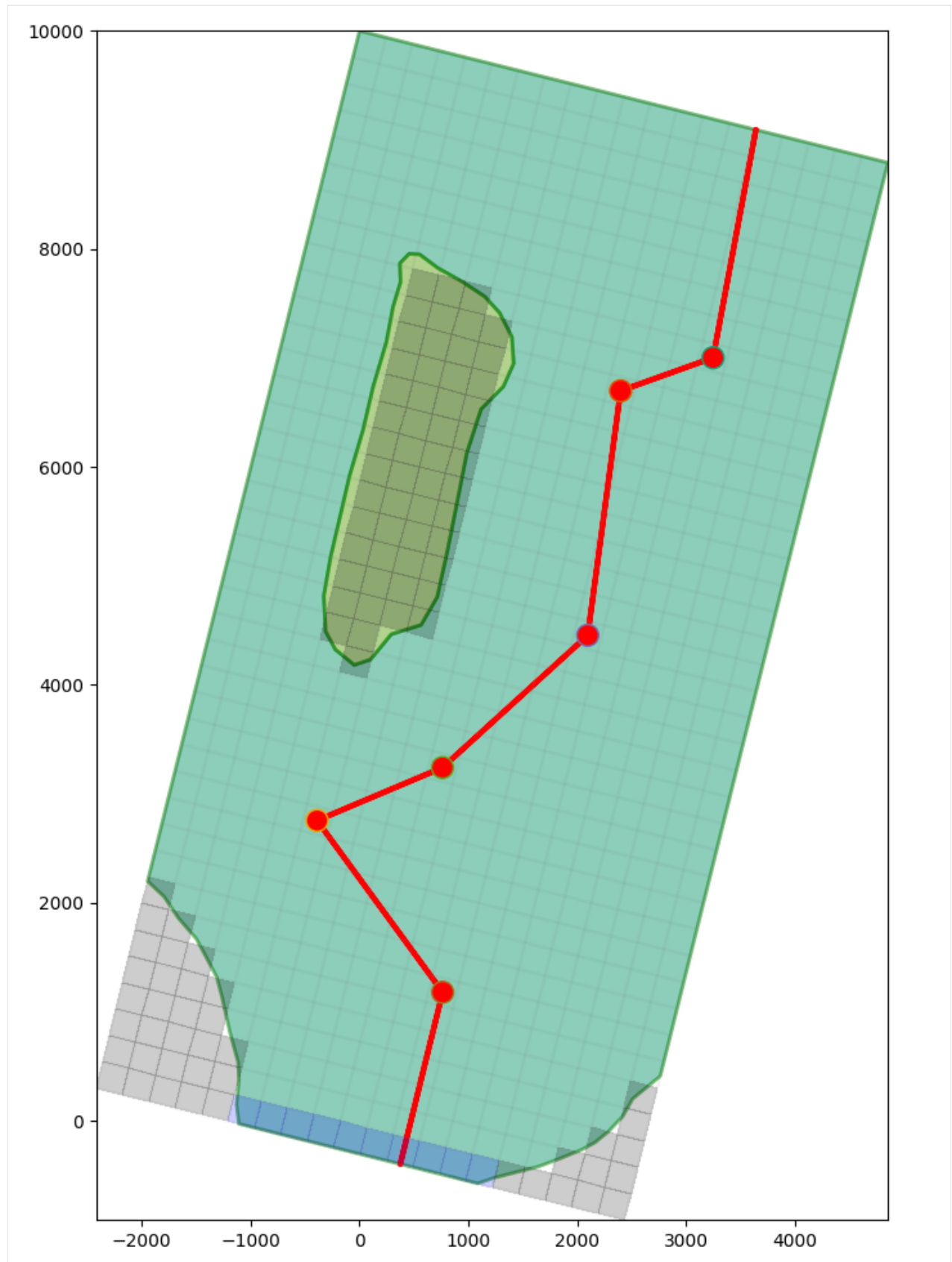
      # Plot a shapefile of
      shp = os.path.join(loadpth, "gis", "bedrock_outcrop_hole_rotate14")
      patch_collection = mapview.plot_shapefile(
          shp, edgecolor="green", linewidths=2, alpha=0.5 # facecolor='none',

```

(continues on next page)

(continued from previous page)

```
)  
# Plot a shapefile of a cross-section line  
shp = os.path.join(loadpth, "gis", "cross_section_rotate14")  
patch_collection = mapview.plot_shapefile(  
    shp, radius=0, lw=3, edgecolor="red", facecolor="None"  
)  
  
# Plot a shapefile of well locations  
shp = os.path.join(loadpth, "gis", "wells_locations_rotate14")  
patch_collection = mapview.plot_shapefile(shp, radius=100, facecolor="red")  
  
# Plot the grid and boundary conditions over the top  
quadmesh = mapview.plot_ibound(alpha=0.1)  
linecollection = mapview.plot_grid(alpha=0.1)
```



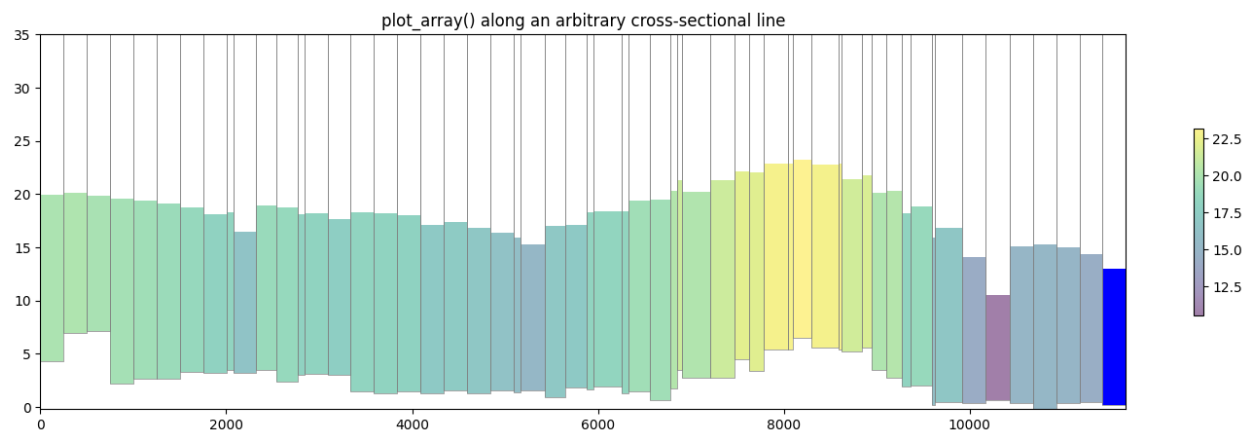
Now let's make a cross section based on this arbitrary cross-sectional line. We can load the cross sectional line vertices using `flopy.plot.plotutil.shapefile_get_vertices()`

Note: in previous examples we passed `line={'column', 5}` to plot a cross section along a column. In this example we pass vertex information into `PlotCrossSection` using `line={'line', line[0]}` where `line[0]` is a list of vertices.

```
[18]: # get the vertices for cross-section lines in a shapefile
fpth = os.path.join(loadpth, "gis", "cross_section_rotate14")
line = flopy.plot.plotutil.shapefile_get_vertices(fpth)

# Set up the figure
fig = plt.figure(figsize=(18, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_title("plot_array() along an arbitrary cross-sectional line")

# plot head values along the cross sectional line
xsect = flopy.plot.PlotCrossSection(model=m1, line={"line": line[0]})
csa = xsect.plot_array(head, head=head, alpha=0.5)
patches = xsect.plot_ibound(head=head)
linecollection = xsect.plot_grid(lw=0.5)
cb = fig.colorbar(csa, ax=ax, shrink=0.5)
```



Plotting geographic coordinates on the x-axis using the `PlotCrossSection` class

The default cross section plotting method plots cells with regard to their intersection distance along the cross sectional line defined by the user. While this method is perfectly acceptable and in many cases may be preferred for plotting arbitrary cross sections, a flag has been added to plot based on geographic coordinates.

The flag `geographic_coords` defaults to `False` which maintains FloPy's previous method of plotting cross sections.

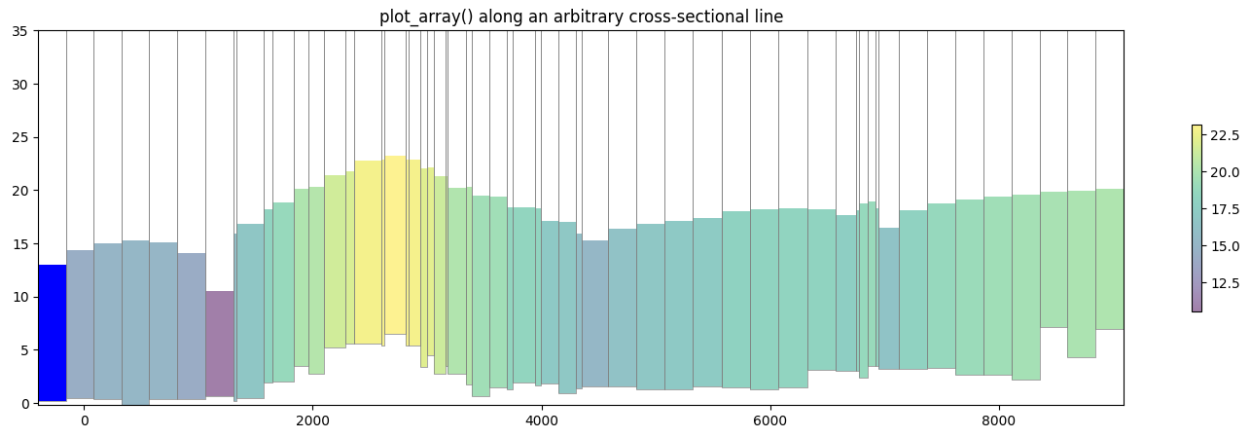
```
[19]: # get the vertices for cross-section lines in a shapefile
fpth = os.path.join(loadpth, "gis", "cross_section_rotate14")
line = flopy.plot.plotutil.shapefile_get_vertices(fpth)

# Set up the figure
fig = plt.figure(figsize=(18, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_title("plot_array() along an arbitrary cross-sectional line")
```

(continues on next page)

(continued from previous page)

```
# plot head values along the cross sectional line
xssect = flopy.plot.PlotCrossSection(
    model=ml, line={"line": line[0]}, geographic_coords=True
)
csa = xssect.plot_array(head, head=head, alpha=0.5)
patches = xssect.plot_ibound(head=head)
linecollection = xssect.plot_grid(lw=0.5)
cb = fig.colorbar(csa, ax=ax, shrink=0.5)
```



Plotting Cross Sections with MODFLOW-6 models

PlotCrossSection has support for MODFLOW-6 models and operates in the same fashion for Structured Grids, Vertex Grids, and Unstructured Grids. Here is a short example on how to plot with MODFLOW-6 structured grids using a version of the Freyberg model created for MODFLOW-6

```
[20]: # load the Freyberg model into mf6-flopy and run the simulation
sim_name = "mfsim.nam"
sim_path = str(prj_root / "examples" / "data" / "mf6-freyberg")
sim = flopy.mf6.MFSimulation.load(
    sim_name=sim_name, version=vmf6, exe_name=exe_name_mf6, sim_ws=sim_path
)

sim.set_sim_path(modelpth)
sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Something bad happened.")
files = ["freyberg.hds", "freyberg.cbc"]
for f in files:
    if os.path.isfile(os.path.join(str(modelpth), f)):
        msg = "Output file located: {}".format(f)
        print(msg)
```

(continues on next page)

(continued from previous page)

```

else:
    errmsg = "Error. Output file cannot be found: {}".format(f)
    print(errmsg)

```

loading simulation...
 loading simulation name file...
 loading tdis package...
 loading model gwf6...
 loading package dis...
 loading package ic...
 WARNING: Block "options" is not a valid block name for file type ic.
 loading package oc...
 loading package npf...
 loading package sto...
 loading package chd...
 loading package riv...
 loading package wel...
 loading package rch...
 loading solution package freyberg...
 writing simulation...
 writing simulation name file...
 writing simulation tdis package...
 writing solution package freyberg...
 writing model freyberg...
 writing model name file...
 writing package dis...
 writing package ic...
 writing package oc...
 writing package npf...
 writing package sto...
 writing package chd-1...
 writing package riv-1...
 writing package wel-1...
 writing package rch-1...

MODFLOW 6
 U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
 VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
 Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
 Build 20220726_000000

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water

(continues on next page)

(continued from previous page)

Resources Software User Rights Notice for complete use, copyright,
and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:46:05

Writing simulation list file: mfsim.lst

Using Simulation name file: mfsim.nam

Solving: Stress period: 1 Time step: 1

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:46:05

Elapsed run time: 0.069 Seconds

Normal termination of simulation.

Output file located: freyberg.hds

Output file located: freyberg.cbc

Plotting boundary conditions and arrays

This works the same as modflow-2005, however the simulation object can host a number of modflow-6 models so we need to grab a model before attempting to plot with PlotCrossSection

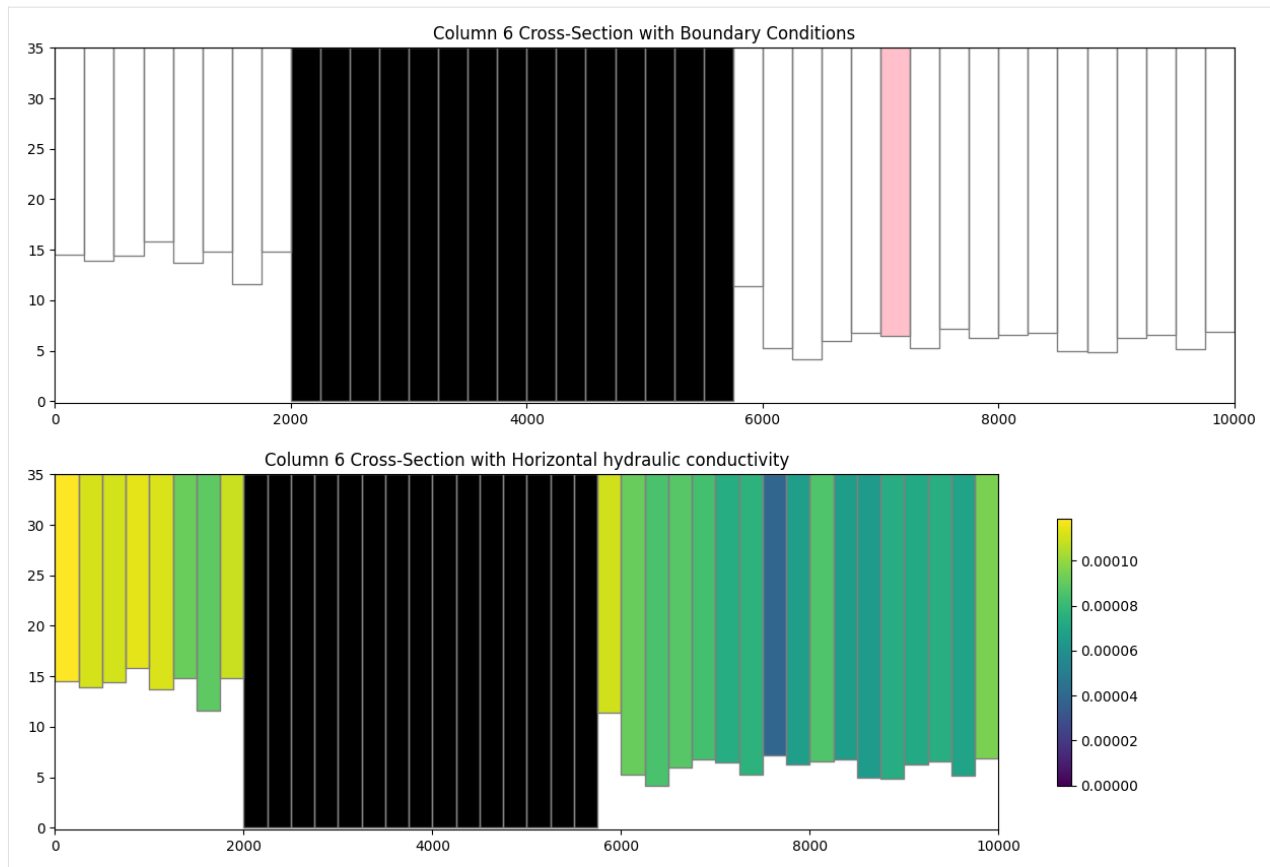
```
[21]: # get the modflow-6 model we want to plot
ml6 = sim.get_model("freyberg")

fig = plt.figure(figsize=(15, 10))
ax = fig.add_subplot(2, 1, 1)

# plot boundary conditions
xsect = flopy.plot.PlotCrossSection(model=ml6, line={"Column": 5})
patches = xsect.plot_bc("WEL", color="pink")
patches = xsect.plot_ibound()
linecollection = xsect.plot_grid()
t = ax.set_title("Column 6 Cross-Section with Boundary Conditions")

# plot xxxx
ax = fig.add_subplot(2, 1, 2)
# plot the horizontal hydraulic conductivities
a = ml6.npf.k.array

xsect = flopy.plot.PlotCrossSection(model=ml6, line={"Column": 5})
csa = xsect.plot_array(a)
patches = xsect.plot_ibound()
linecollection = xsect.plot_grid()
t = ax.set_title(
    "Column 6 Cross-Section with Horizontal hydraulic conductivity"
)
cb = plt.colorbar(csa, shrink=0.75)
```



Plotting specific discharge with a MODFLOW-6 model

MODFLOW-6 includes a the PLOT_SPECIFIC_DISCHARGE flag in the NPF package to calculate and store discharge vectors for easy plotting. The `postprocessing.get_specific_discharge()` method will preprocess the data into vectors and `PlotCrossSection` has the `plot_vector()` method to use this data. The specific discharge array is stored in the cell budget file.

```
[22]: # get the head from the head file
head_file = os.path.join(modelpth, "freyberg.hds")
hds = flopy.utils.HeadFile(head_file)
head = hds.get_alldata()[0]

# get the specific discharge from the cell budget file
cbc_file = os.path.join(modelpth, "freyberg.cbc")
cbc = flopy.utils.CellBudgetFile(cbc_file, precision="double")
spdis = cbc.get_data(text="SPDIS")[-1]
qx, qy, qz = flopy.utils.postprocessing.get_specific_discharge(
    spdis, m16, head=head
)

fig = plt.figure(figsize=(18, 5))
ax = fig.add_subplot(1, 1, 1)

ax.set_title("plot_array() and plot_vector()")
```

(continues on next page)

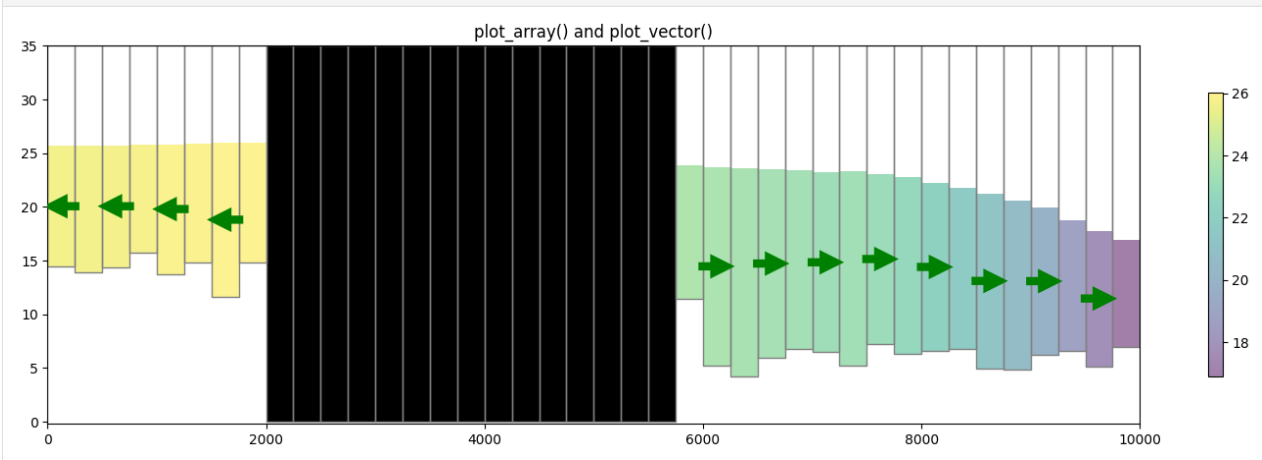
(continued from previous page)

```

xsect = flopy.plot.PlotCrossSection(model=m16, ax=ax, line={"Column": 5})
csa = xsect.plot_array(head, head=head, alpha=0.5)
patches = xsect.plot_ibound(head=head)
linecollection = xsect.plot_grid()
quiver = xsect.plot_vector(
    qx,
    qy,
    qz,
    head=head,
    hstep=2,
    normalize=True,
    color="green",
    scale=30,
    headwidth=3,
    headlength=3,
    headaxislength=3,
    zorder=10,
)

cb = plt.colorbar(csa, shrink=0.75)

```



Vertex cross section plotting with MODFLOW-6 (DISV)

FloPy fully supports vertex discretization (DISV) plotting through the `PlotCrossSection` class. The method calls are identical to the ones presented previously for Structured discretization (DIS) and the same matplotlib keyword arguments are supported. Let's run through an example using a vertex model grid.

```

[23]: # build and run vertex model grid demo problem
notebook_utils.run(modelpth)

# check if model ran properly
modelpth = os.path.join(modelpth, "mp7_ex2", "mf6")
# from pprint import pprint
# pprint([str(p) for p in (Path(tempdir.name) / "mp7_ex2").glob('*')])
files = ["mp7p2.hds", "mp7p2.cbb"]
for f in files:

```

(continues on next page)

(continued from previous page)

```

if os.path.isfile(os.path.join(modelpth, f)):
    msg = "Output file located: {}".format(f)
    print(msg)
else:
    errmsg = "Error. Output file cannot be found: {}".format(f)
    print(errmsg)

```

WARNING: Unable to resolve dimension of ('gwf6', 'disv', 'cell2d', 'cell2d', 'icvert')
↳ based on shape "ncvert".

writing simulation...

```

writing simulation name file...
writing simulation tdis package...
writing solution package ims...
writing model mp7p2...
writing model name file...
writing package disv...
writing package ic...
writing package npf...
writing package wel_0...

```

INFORMATION: maxbound in ('gwf6', 'wel', 'dimensions') changed to 1 based on size of
↳ stress_period_data

```

writing package rcha_0...
writing package riv_0...

```

INFORMATION: maxbound in ('gwf6', 'riv', 'dimensions') changed to 21 based on size of
↳ stress_period_data

```

writing package oc...

```

MODFLOW 6

U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_0000000

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:46:07

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

(continues on next page)

(continued from previous page)

Solving: Stress period: 1 Time step: 1

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:46:07
Elapsed run time: 0.150 Seconds

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'mp7p2.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'mp7p2.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.

Normal termination of simulation.

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.00000E+03 Steady-state flow

Particle Summary:

- 0 particles are pending release.
- 0 particles remain active.
- 16 particles terminated at boundary faces.
- 0 particles terminated at weak sink cells.
- 0 particles terminated at weak source cells.
- 0 particles terminated at strong source/sink cells.
- 0 particles terminated in cells with a specified zone number.
- 0 particles were stranded in inactive or dry cells.
- 0 particles were unreleased.
- 0 particles have an unknown status.

Normal termination.

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.00000E+03 Steady-state flow

Particle Summary:

- 0 particles are pending release.
- 0 particles remain active.
- 416 particles terminated at boundary faces.
- 0 particles terminated at weak sink cells.
- 0 particles terminated at weak source cells.
- 0 particles terminated at strong source/sink cells.
- 0 particles terminated in cells with a specified zone number.
- 0 particles were stranded in inactive or dry cells.

(continues on next page)

(continued from previous page)

```

    0 particles were unreleased.
    0 particles have an unknown status.

Normal termination.
Output file located: mp7p2.hds
Output file located: mp7p2.cbb

```

```

[24]: # load the simulation and get the model
vertex_sim_name = "mfsim.nam"
vertex_sim = flopy.mf6.MFSimulation.load(
    sim_name=vertex_sim_name,
    version=vmf6,
    exe_name=exe_name_mf6,
    sim_ws=modelpth,
)
vertex_ml6 = vertex_sim.get_model("mp7p2")

loading simulation...
  loading simulation name file...
  loading tdis package...
  loading model gw6...
  loading package disv...
WARNING: Unable to resolve dimension of ('gw6', 'disv', 'cell2d', 'cell2d', 'icvert')
↳ based on shape "ncvert".
  loading package ic...
  loading package npf...
  loading package wel...
  loading package rch...
  loading package riv...
  loading package oc...
  loading solution package mp7p2...

```

Plotting a line based cross section through the model grid

Because a `VertexGrid` has no row or column number, the cross-section line must be defined explicitly. This is done by passing a dictionary to the line parameter with key `line` — the value may be an array-like of 2 or more points, e.g. `{"line": [(x0, y0), (x1, y1), ...]}`, or a `flopy.utils.geometry.LineString` or `shapely.geometry.LineString`. Below we show an example of setting up a cross-section line with a MODFLOW-6 DISV model.

```

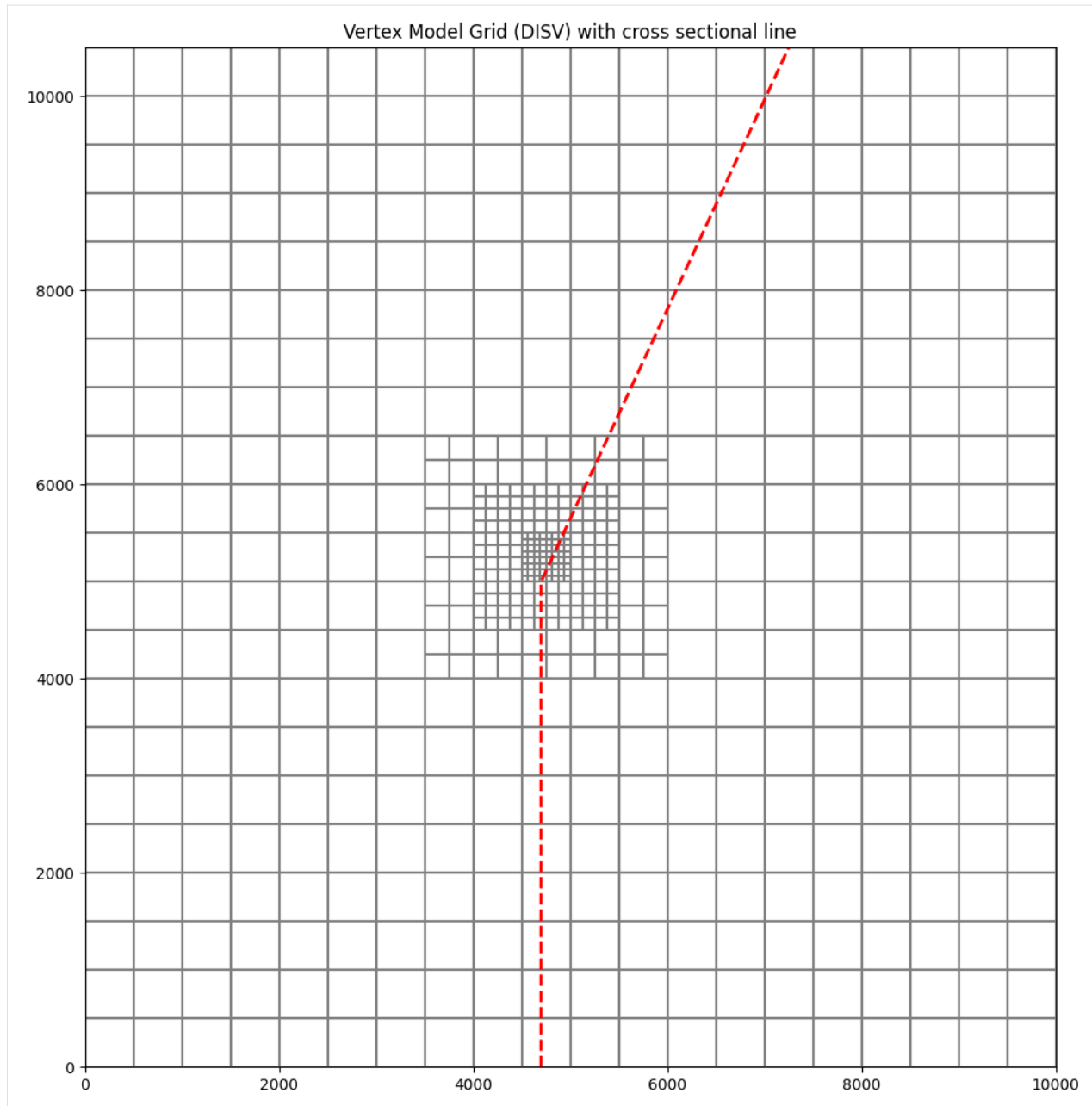
[25]: line = np.array([(4700, 0), (4700, 5000), (7250, 10500)])

# Let's plot the model grid in map view to look at it
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("Vertex Model Grid (DISV) with cross sectional line")

# use PlotMapView to plot a DISV (vertex) model
mapview = flopy.plot.PlotMapView(vertex_ml6, layer=0)
linecollection = mapview.plot_grid()

# plot the line over the model grid
lc = plt.plot(line.T[0], line.T[1], "r--", lw=2)

```

Now we can plot a cross section of the model grid defined by this line

```
[26]: fig = plt.figure(figsize=(15, 5))
      ax = fig.add_subplot(1, 1, 1)

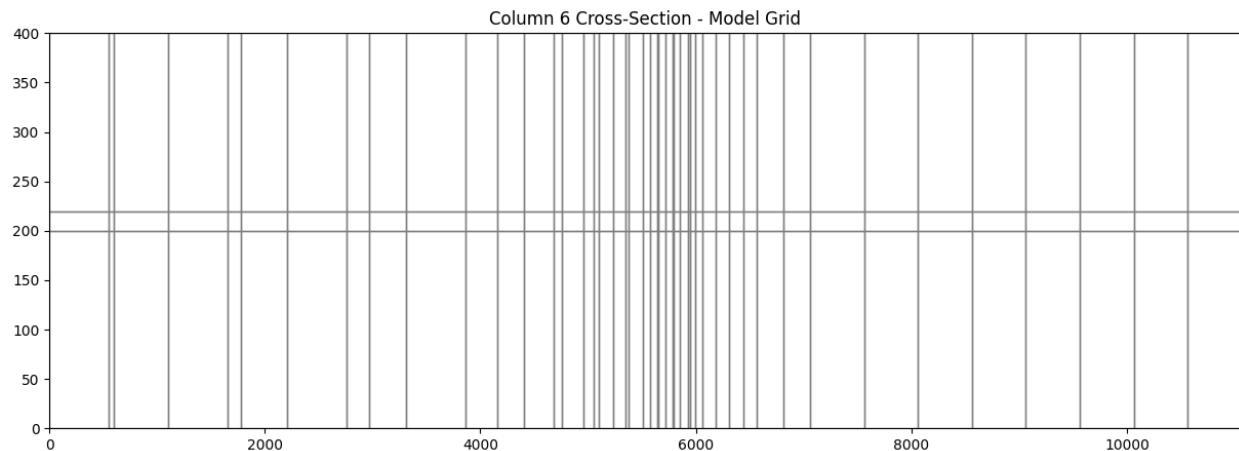
      # Next we create an instance of the PlotCrossSection class
      xssect = flopy.plot.PlotCrossSection(model=vertex_ml6, line={"line": line})

      # Then we can use the plot_grid() method to draw the grid
      # The return value for this function is a matplotlib LineCollection object,
      # which could be manipulated (or used) later if necessary.
      linecollection = xssect.plot_grid()
```

(continues on next page)

(continued from previous page)

```
t = ax.set_title("Column 6 Cross-Section - Model Grid")
```



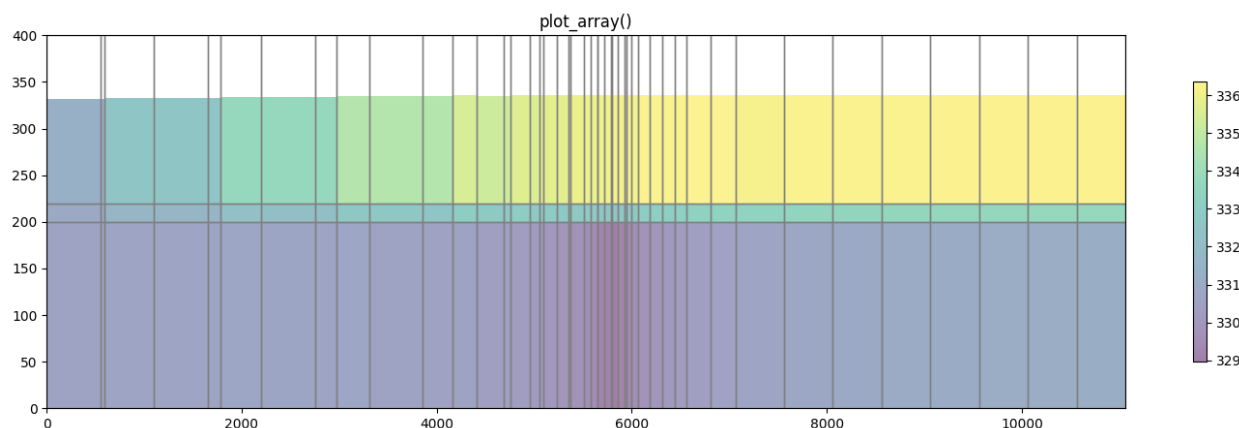
Plotting Arrays and Contouring with Vertex Model grids

`PlotCrossSection` allows the user to plot arrays and contour with DISV based discretization. The `plot_array()` method is called in the same way as using a structured grid. The only difference is that `PlotCrossSection` builds a matplotlib patch collection for Vertex based grids.

```
[27]: # get the head output for stress period 1 from the modflow6 head file
head = flopy.utils.HeadFile(os.path.join(modelpth, "mp7p2.hds"))
hdata = head.get_alldata()[0, :, :, :]

fig = plt.figure(figsize=(18, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_title("plot_array()")

xsect = flopy.plot.PlotCrossSection(model=vertex_m16, line={"line": line})
patch_collection = xsect.plot_array(hdata, head=hdata, alpha=0.5)
line_collection = xsect.plot_grid()
cb = plt.colorbar(patch_collection, shrink=0.75)
```



The `contour_array()` method operates in the same way as the structured example.

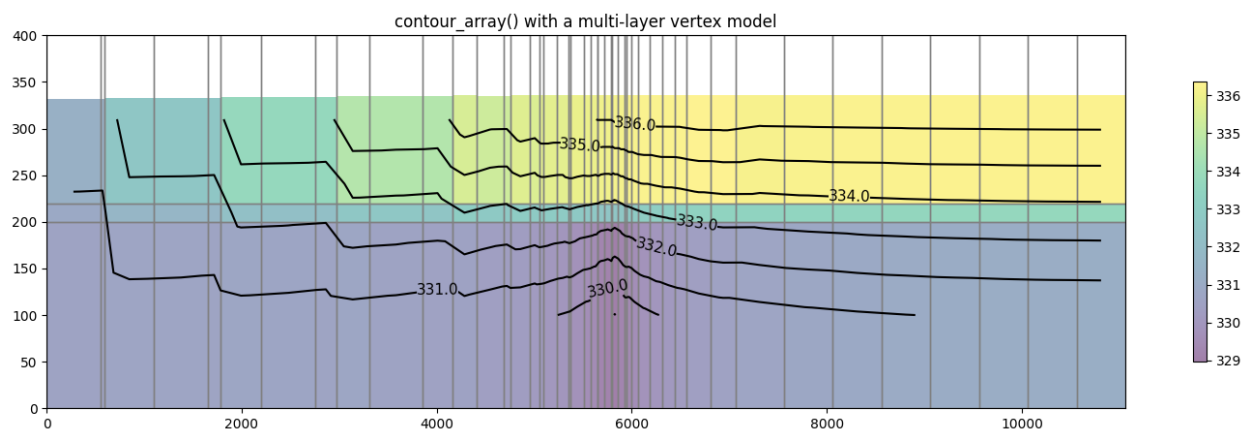
```
[28]: levels = np.arange(329, 337, 1)

fig = plt.figure(figsize=(18, 5))
ax = fig.add_subplot(1, 1, 1)
ax.set_title("contour_array() with a multi-layer vertex model")

xsect = flopy.plot.PlotCrossSection(model=vertex_ml6, line={"line": line})
patch_collection = xsect.plot_array(hdata, head=hdata, alpha=0.5)
line_collection = xsect.plot_grid()

contour_set = xsect.contour_array(hdata, levels=levels, colors="k")
plt.clabel(contour_set, fmt="%.1f", colors="k", fontsize=11)

cb = plt.colorbar(patch_collection, shrink=0.75)
```



Plotting specific discharge vectors for DISV

MODFLOW-6 includes a the PLOT_SPECIFIC_DISCHARGE flag in the NPF package to calculate and store discharge vectors for easy plotting. The `postprocessing.get_specific_discharge()` method will preprocess the data into vectors and `PlotCrossSection` has the `plot_vector()` method to use this data. The specific discharge array is stored in the cell budget file.

Note: When plotting specific discharge, an arbitrary cross section cannot be used. The cross sectional line must be orthogonal to the model grid

```
[29]: # define and plot our orthogonal line
line = np.array([(0, 4700), (10000, 4700)])

# Let's plot the model grid in map view to look at it
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("Vertex Model Grid (DISV) with cross sectional line")

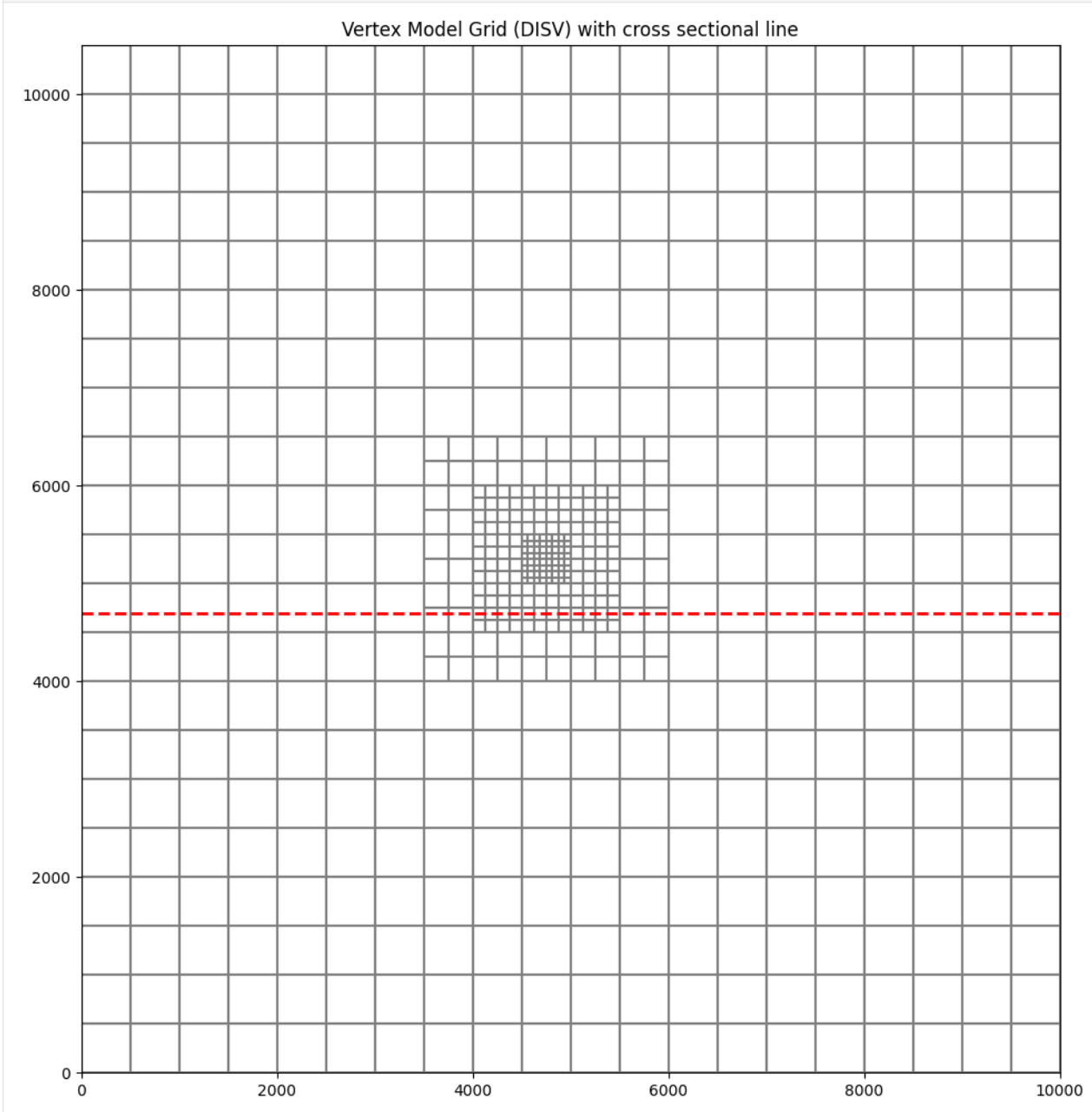
# use PlotMapView to plot a DISV (vertex) model
mapview = flopy.plot.PlotMapView(vertex_ml6, layer=0)
linecollection = mapview.plot_grid()

# plot the line over the model grid
```

(continues on next page)

(continued from previous page)

```
lc = plt.plot(line.T[0], line.T[1], "r--", lw=2)
```



```
[30]: # plot specific discharge on cross section
cbb = flopy.utils.CellBudgetFile(os.path.join(modelpth, "mp7p2.cbb"))
spdis = cbb.get_data(text="SPDIS")[-1]
qx, qy, qz = flopy.utils.postprocessing.get_specific_discharge(
    spdis, vertex_ml6, head=hdata
)

fig = plt.figure(figsize=(18, 5))
ax = fig.add_subplot(1, 1, 1)
```

(continues on next page)

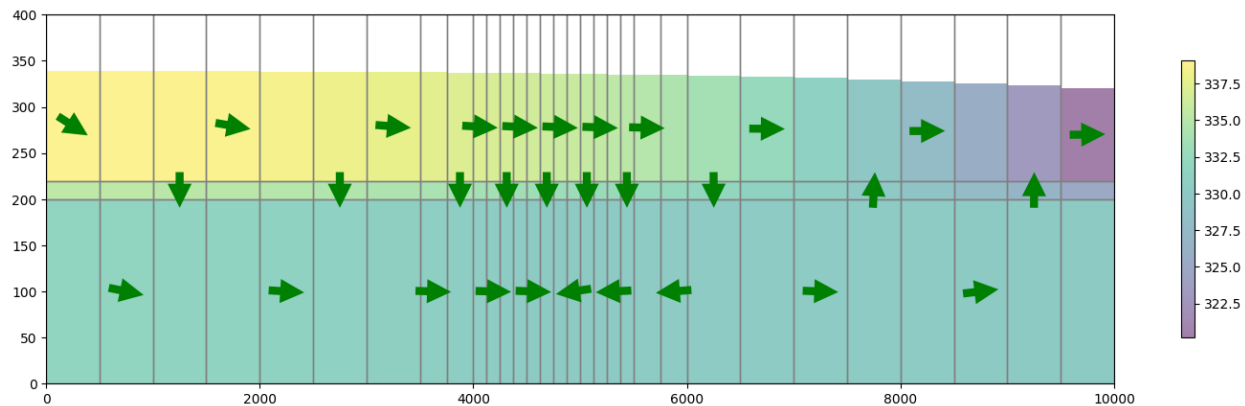
(continued from previous page)

```

xsect = flopy.plot.PlotCrossSection(model=vertex_m16, line={"line": line})
patch_collection = xsect.plot_array(hdata, head=hdata, alpha=0.5)
line_collection = xsect.plot_grid()
quiver = xsect.plot_vector(
    qx,
    qy,
    qz,
    head=hdata,
    hstep=3,
    normalize=True,
    color="green",
    scale=30,
    headwidth=3,
    headlength=3,
    headaxislength=3,
    zorder=10,
)

cb = plt.colorbar(patch_collection, shrink=0.75)

```



Plotting using built in styles

FloPy's plotting routines can be used with built in styles from the `styles` module. The `styles` module takes advantage of matplotlib's temporary styling routines by reading in pre-built style sheets. Two different types of styles have been built for flopy: `USGSMap()` and `USGSPlot()` styles which can be used to create report quality figures. The `styles` module also contains a number of methods that can be used for adding axis labels, text, annotations, headings, removing tick lines, and updating the current font.

This example will load the Keating groundwater transport model and plot results using styles

```
[31]: notebook_utils.run_keating_model(modelpth)
```

```

Running mf6gwf model...
Running mf6gwt model...

```

```
[31]: True
```

Load the flow and transport models

```
[32]: sim_path = os.path.join(modelpth, "mf6-gwt-keating", "mf6gwf")
tr_path = os.path.join(modelpth, "mf6-gwt-keating", "mf6gwt")
sim_name = "mfsim.nam"
sim = flopy.mf6.MFSimulation.load(
    sim_name=sim_name, version=vmf6, exe_name=exe_name_mf6, sim_ws=sim_path
)
gw6 = sim.get_model("flow")

sim = flopy.mf6.MFSimulation.load(
    sim_name=sim_name, version=vmf6, exe_name=exe_name_mf6, sim_ws=tr_path
)
gwt6 = sim.get_model("trans")
```

```
loading simulation...
  loading simulation name file...
  loading tdis package...
  loading model gw6...
    loading package dis...
    loading package npf...
    loading package ic...
    loading package chd...
    loading package rch...
    loading package oc...
  loading solution package flow...
loading simulation...
  loading simulation name file...
  loading tdis package...
  loading model gwt6...
    loading package dis...
    loading package ic...
    loading package mst...
    loading package adv...
    loading package dsp...
    loading package fmi...
    loading package ssm...
    loading package oc...
    loading package obs...
  loading solution package trans...
```

```
[33]: # import styles
from flopy.plot import styles

# load head file and plot
head = gw6.output.head().get_data()

with styles.USGMap():
    fig, ax = plt.subplots(1, 1, figsize=(12, 8), dpi=300, tight_layout=True)

    xsect = flopy.plot.PlotCrossSection(model=gw6, ax=ax, line={"row": 0})
    pc = xsect.plot_array(head, head=head, cmap="jet")
    xsect.plot_bc(ftype="RCH", color="red")
    xsect.plot_bc(ftype="CHD")
    plt.colorbar(pc, shrink=0.25)
```

(continues on next page)

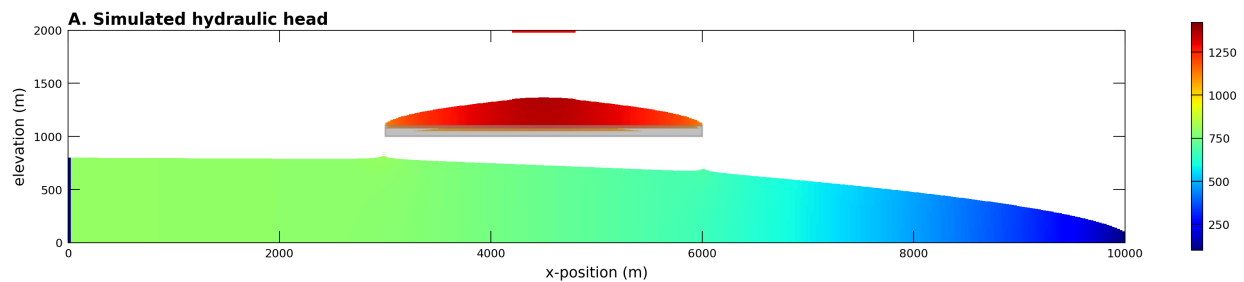
(continued from previous page)

```

# add a rectangle to show the confining layer
confining_rect = mpl.patches.Rectangle(
    (3000, 1000), 3000, 100, color="gray", alpha=0.5
)
ax.add_patch(confining_rect)

# set labels using styles
styles.xlabel(label="x-position (m)")
styles.ylabel(label="elevation (m)")
styles.heading(
    letter="A.", heading="Simulated hydraulic head", fontsize=10
)
ax.set_aspect(1.0)

```



Plotting concentration model results using the USGSMap() style

```

[34]: # load the transport output file
cobj = gwt6.output.concentration()
plot_times = [100, 1000, 3000]
obs1 = (48, 0, 118) # Layer, row, and column for observation 1
obs2 = (76, 0, 358) # Layer, row, and column for observation 2
xgrid, _, zgrid = gw6.modelgrid.xyzcellcenters

with styles.USGSPlot():
    fig, axes = plt.subplots(3, 1, figsize=(15, 9), tight_layout=True)
    for ix, totim in enumerate(plot_times):
        heading = "Time = {}".format(totim)
        conc = cobj.get_data(totim=totim)
        ax = axes[ix]
        xsect = flopy.plot.PlotCrossSection(model=gw6, ax=ax, line={"row": 0})
        pc = xsect.plot_array(conc, head=head, cmap="jet", vmin=0, vmax=1)
        xsect.plot_bc(ftype="RCH", color="red")
        xsect.plot_bc(ftype="CHD")

    # plot confining layer
    confining_rect = mpl.patches.Rectangle(
        (3000, 1000), 3000, 100, color="gray", alpha=0.5
    )
    ax.add_patch(confining_rect)

```

(continues on next page)

(continued from previous page)

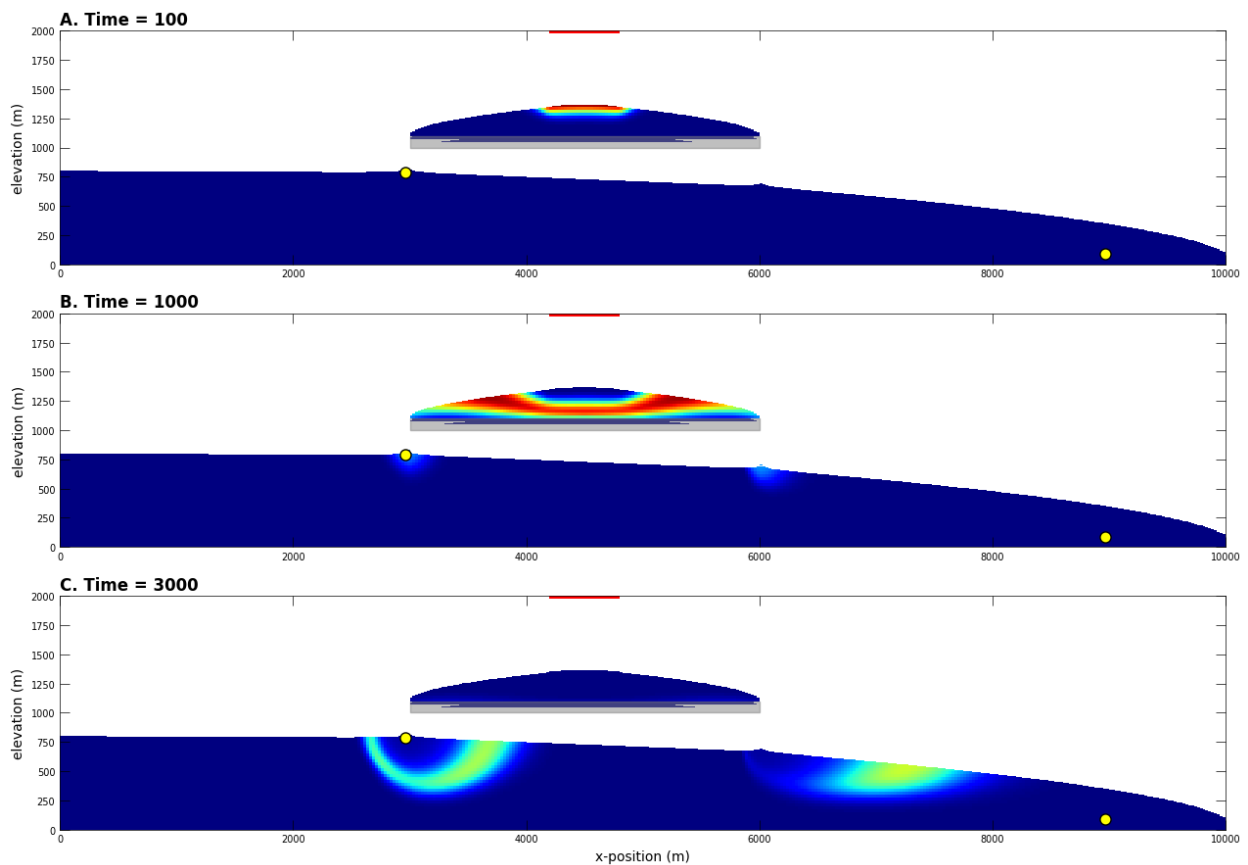
```

# set axis labels and title using styles
styles.ylabel(ax=ax, label="elevation (m)", fontsize=10)
if ix == 2:
    styles.xlabel(ax=ax, label="x-position (m)", fontsize=10)
styles.heading(ax=ax, heading=heading, idx=ix, fontsize=12)

ax.set_aspect(1.0)

# add observation locations based on grid cell centers
for k, i, j in [obs1, obs2]:
    x = xgrid[i, j]
    z = zgrid[k, i, j]
    ax.plot(x, z, mfc="yellow", mec="black", marker="o", ms="8")

```



Summary

This notebook demonstrates some of the plotting functionality available with flopy. Although not described here, the plotting functionality tries to be general by passing keyword arguments passed to the `PlotCrossSection` methods down into the `matplotlib.pyplot` routines that do the actual plotting. For those looking to customize these plots, it may be necessary to search for the available keywords by understanding the types of objects that are created by the `PlotCrossSection` methods. The `PlotCrossSection` methods return these `matplotlib.collections` objects so that they could be fine-tuned later in the script before plotting.

Hope this gets you started!

```
[35]: try:
        # ignore PermissionError on Windows
        tempdir.cleanup()
    except:
        pass
```

6.2.4 Making Maps of Your Model

This notebook demonstrates the mapping capabilities of FloPy. It demonstrates these capabilities by loading and running existing models and then showing how the `PlotMapView` object and its methods can be used to make nice plots of the model grid, boundary conditions, model results, shape files, etc.

Mapping is demonstrated for MODFLOW-2005, MODFLOW-USG, and MODFLOW-6 models in this notebook

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import shapefile

sys.path.append(os.path.join("../", "common"))
import notebook_utils

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("../", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))
```

```
3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[2]: # Set name of MODFLOW exe
# assumes executable is in users path statement
v2005 = "mf2005"
exe_name_2005 = "mf2005"
vmf6 = "mf6"
exe_name_mf6 = "mf6"
exe_mp = "mp6"

# Set the paths
prj_root = notebook_utils.get_project_root_path()
loadpth = str(prj_root / "examples" / "data" / "freyberg")
tempdir = TemporaryDirectory()
modelpth = tempdir.name
```

Load and Run an Existing MODFLOW-2005 Model

A model called the “Freyberg Model” is located in the loadpth folder. In the following code block, we load that model, then change into a new workspace (modelpth) where we recreate and run the model. For this to work properly, the MODFLOW-2005 executable (mf2005) must be in the path. We verify that it worked correctly by checking for the presence of freyberg.hds and freyberg.cbc.

```
[3]: ml = flopy.modflow.Modflow.load(
    "freyberg.nam", model_ws=loadpth, exe_name=exe_name_2005, version=v2005
)
ml.change_model_ws(new_pth=modelpth)
ml.write_input()
success, buff = ml.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

files = ["freyberg.hds", "freyberg.cbc"]
for f in files:
    if os.path.isfile(os.path.join(modelpth, f)):
        msg = "Output file located: {}".format(f)
        print(msg)
    else:
        errmsg = "Error. Output file cannot be found: {}".format(f)
        print(errmsg)
```

```
MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017
```

(continues on next page)

(continued from previous page)

```

Using NAME file: freyberg.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:48:46

Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:48:47
Elapsed run time: 0.017 Seconds

Normal termination of simulation
Output file located: freyberg.hds
Output file located: freyberg.cbc

```

Create and Run MODPATH 6 model

The MODFLOW-2005 model created in the previous code block will be used to create a endpoint capture zone and pathline analysis for the pumping wells in the model.

```

[4]: mp = flopy.modpath.Modpath6(
    "freybergmp", exe_name=exe_mp, modflowmodel=ml, model_ws=modelpth
)
mpbas = flopy.modpath.Modpath6Bas(
    mp,
    hnoflo=ml.bas6.hnoflo,
    hdry=ml.lpf.hdry,
    ibound=ml.bas6.ibound.array,
    prsity=0.2,
    prsityCB=0.2,
)
sim = mp.create_mpsim(trackdir="forward", simtype="endpoint", packages="RCH")
mp.write_input()
success, buff = mp.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

mpp = flopy.modpath.Modpath6(
    "freybergmpp", exe_name=exe_mp, modflowmodel=ml, model_ws=modelpth
)
mpbas = flopy.modpath.Modpath6Bas(
    mpp,
    hnoflo=ml.bas6.hnoflo,
    hdry=ml.lpf.hdry,
    ibound=ml.bas6.ibound.array,
    prsity=0.2,
    prsityCB=0.2,
)
sim = mpp.create_mpsim(trackdir="backward", simtype="pathline", packages="WEL")
mpp.write_input()
mpp.run_model()

```

```
Processing basic data ...
Checking head file ...
Checking budget file and building index ...

Run particle tracking simulation ...
Processing Time Step      1 Period      1. Time = 1.000000E+01
Particle tracking complete. Writing endpoint file ...
End of MODPATH simulation. Normal termination.
FloPy is using the following executable to run the model: ../../home/runner/.local/bin/
↳ modflow/mp6
Processing basic data ...
Checking head file ...
Checking budget file and building index ...

Run particle tracking simulation ...
Processing Time Step      1 Period      1. Time = 1.000000E+01
Particle tracking complete. Writing endpoint file ...
End of MODPATH simulation. Normal termination.
```

```
[4]: (True, [])
```

Creating a Map of the Model Grid

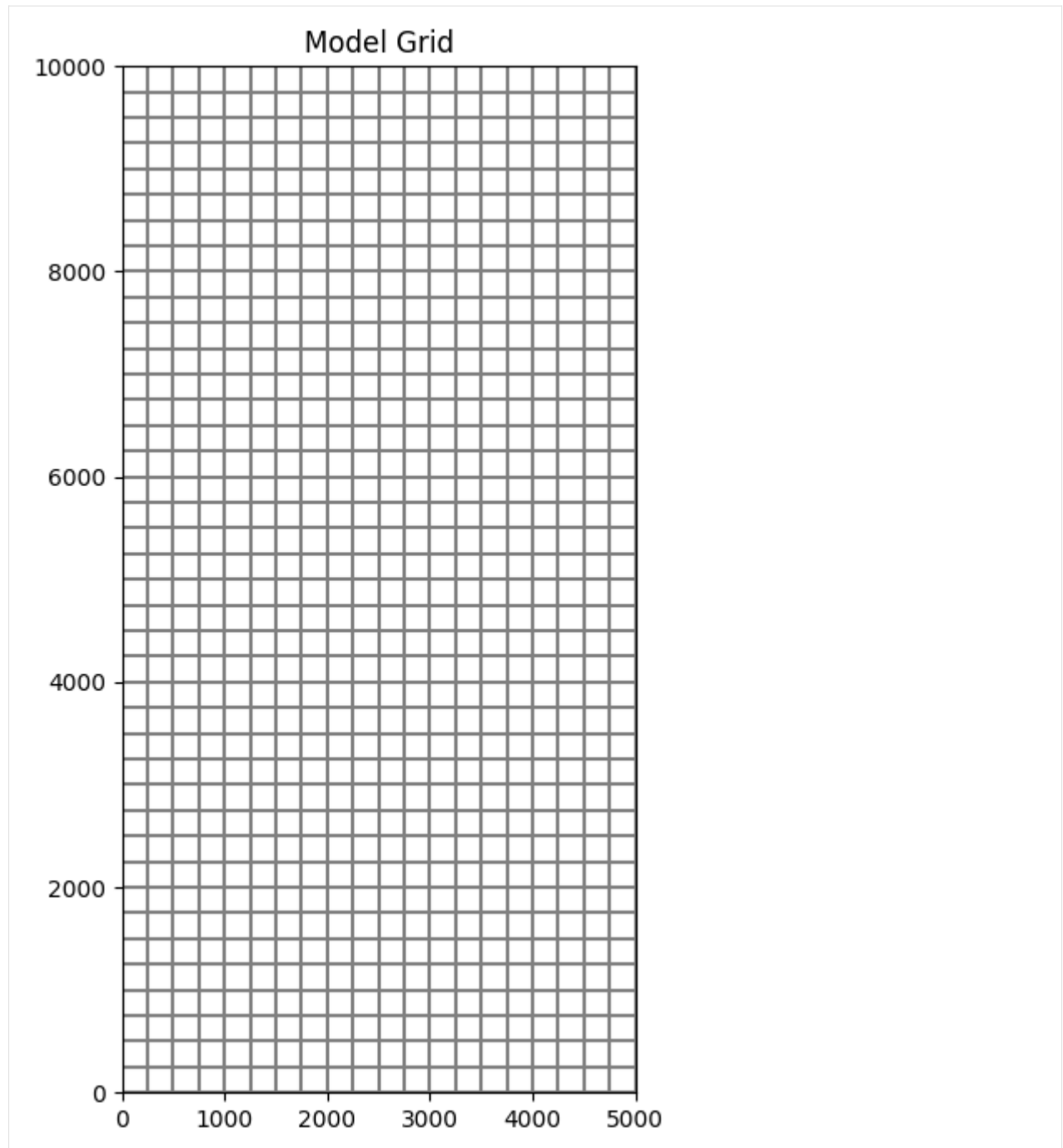
Now that we have a model, we can use the flopy plotting utilities to make maps. We will start by making a map of the model grid using the `PlotMapView` class and the `plot_grid()` method of that class.

```
[5]: # First step is to set up the plot
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

# Next we create an instance of the PlotMapView class
mapview = flopy.plot.PlotMapView(model=m1)

# Then we can use the plot_grid() method to draw the grid
# The return value for this function is a matplotlib LineCollection object,
# which could be manipulated (or used) later if necessary.
linecollection = mapview.plot_grid()

t = ax.set_title("Model Grid")
```



Grid transformations and setting coordinate information

The `PlotMapView` class can plot the position of the model grid in space. However, transformations must be done on the modelgrid using `set_coord_info()`. This allows the user to set the coordinate information once, and then they are able to generate as many instances of `PlotMapView` as they wish, without providing the coordinate info again.

Here we demonstrate the effects of these values. In the first two plots, the grid origin (lower left corner) remains fixed at (0, 0). These first two plots demonstrate how work with coordinate info in the `PlotMapView` class. The third example shows the grid origin set at (507000 E, 2927000 N)

```
[6]: fig = plt.figure(figsize=(18, 6))

ax = fig.add_subplot(1, 3, 1, aspect="equal")

# set modelgrid rotation
ml.modelgrid.set_coord_info(angrot=14)

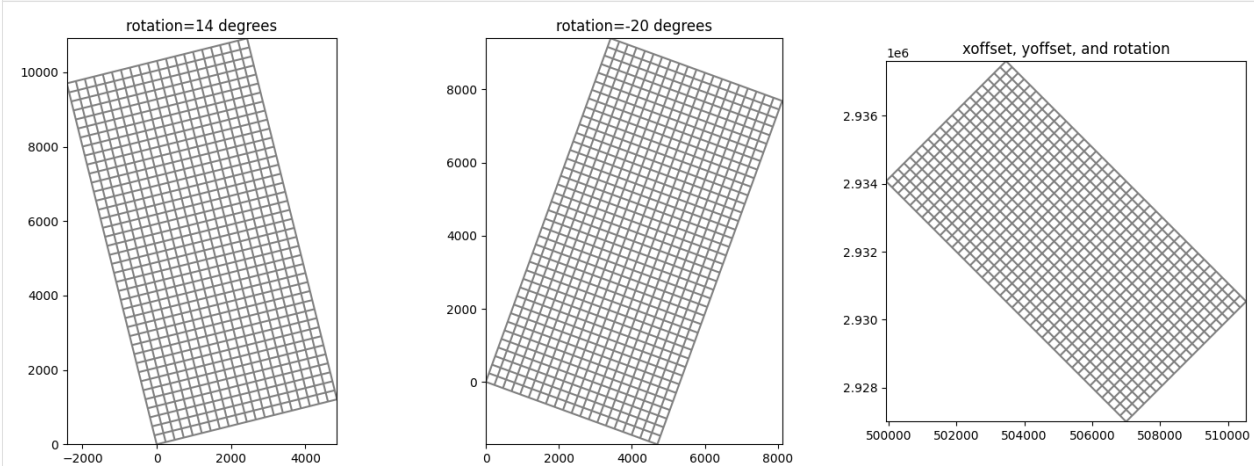
# generate a plot
mapview = flopy.plot.PlotMapView(model=ml)
linecollection = mapview.plot_grid()
t = ax.set_title("rotation=14 degrees")

# re-set the modelgrid rotation
ml.modelgrid.set_coord_info(angrot=-20)

ax = fig.add_subplot(1, 3, 2, aspect="equal")
mapview = flopy.plot.PlotMapView(model=ml)
linecollection = mapview.plot_grid()
t = ax.set_title("rotation=-20 degrees")

# re-set the modelgrid origin and rotation
ml.modelgrid.set_coord_info(xoff=507000, yoff=2927000, angrot=45)

ax = fig.add_subplot(1, 3, 3, aspect="equal")
mapview = flopy.plot.PlotMapView(model=ml)
linecollection = mapview.plot_grid()
t = ax.set_title("xoffset, yoffset, and rotation")
```

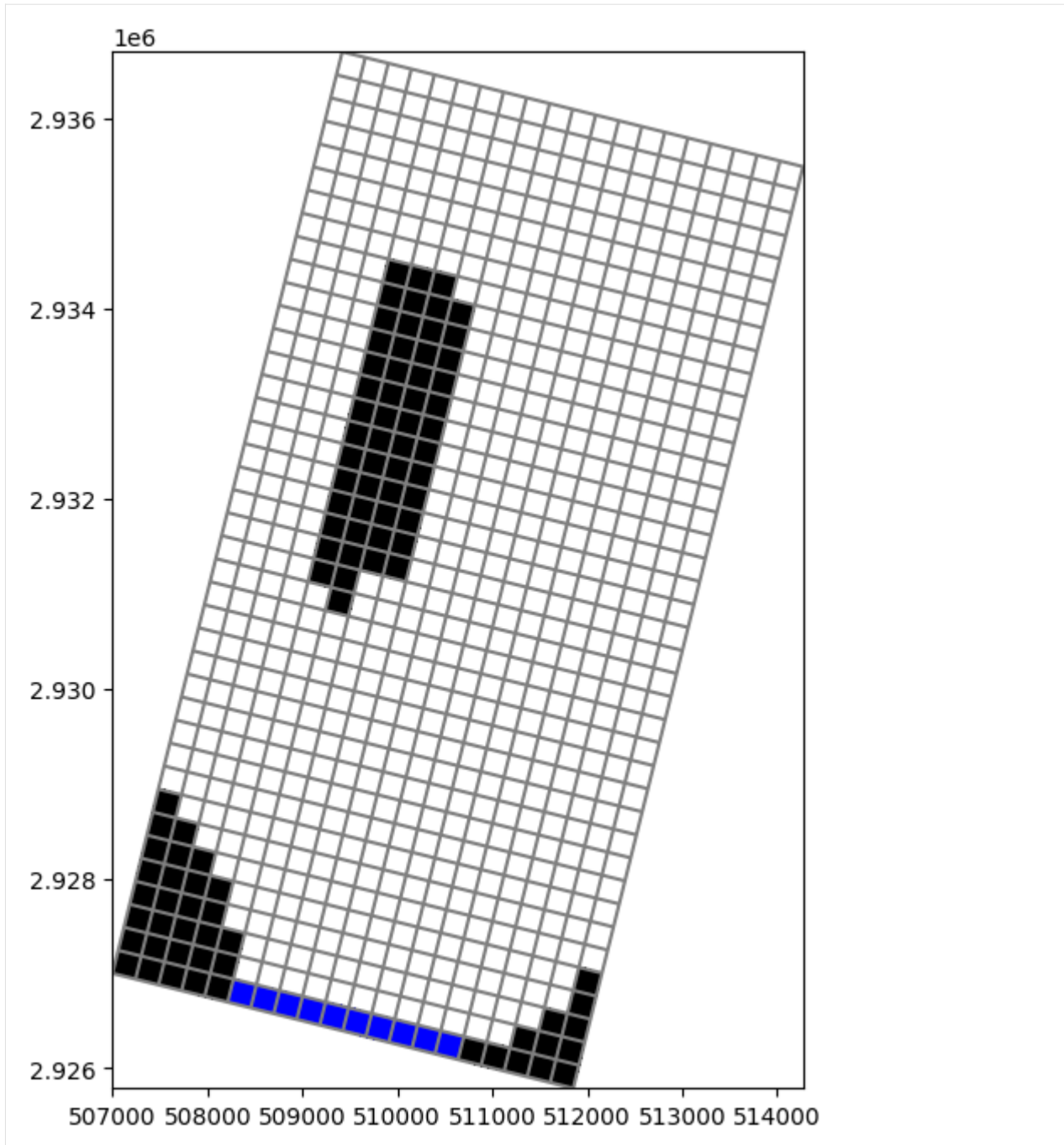


Plotting Ibound

The `plot_ibound()` method can be used to plot the boundary conditions contained in the `ibound` array, which is part of the MODFLOW Basic Package. The `plot_ibound()` method returns a matplotlib QuadMesh object (`matplotlib.collections.QuadMesh`). If you are familiar with the matplotlib collections, then this may be important to you, but if not, then don't worry about the return objects of these plotting function.

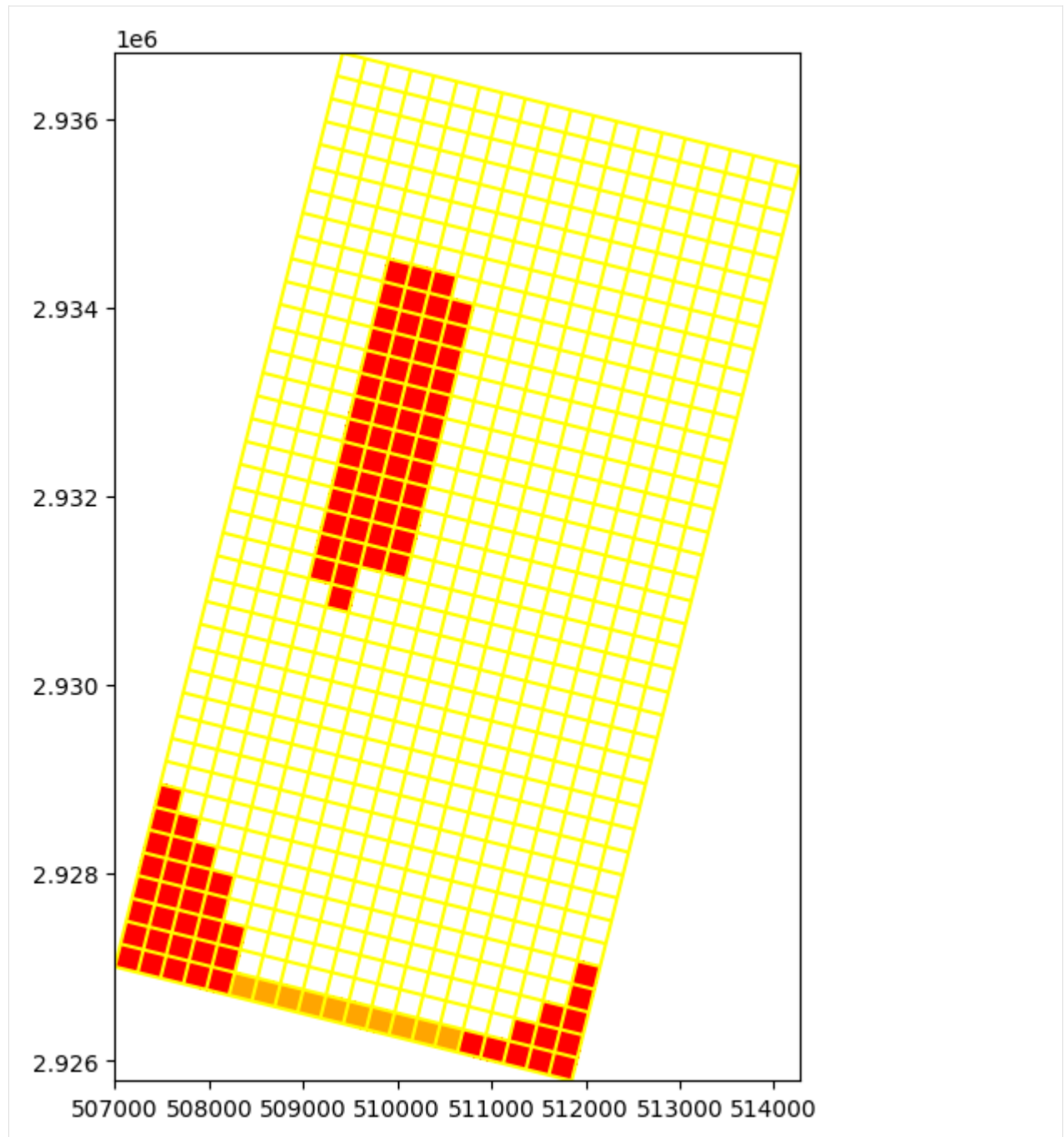
```
[7]: fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(1, 1, 1, aspect="equal")

    # set the grid rotation and then plot
    ml.modelgrid.set_coord_info(angrot=-14)
    mapview = flopy.plot.PlotMapView(model=ml)
    quadmesh = mapview.plot_ibound()
    linecollection = mapview.plot_grid()
```



We can also change the colors by calling the `color_noflow` and `color_ch` parameters in `plot_ibound()` and the `colors` parameter in `plot_grid()`

```
[8]: fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mapview = flopy.plot.PlotMapView(model=ml)
quadmesh = mapview.plot_ibound(color_noflow="red", color_ch="orange")
linecollection = mapview.plot_grid(colors="yellow")
```

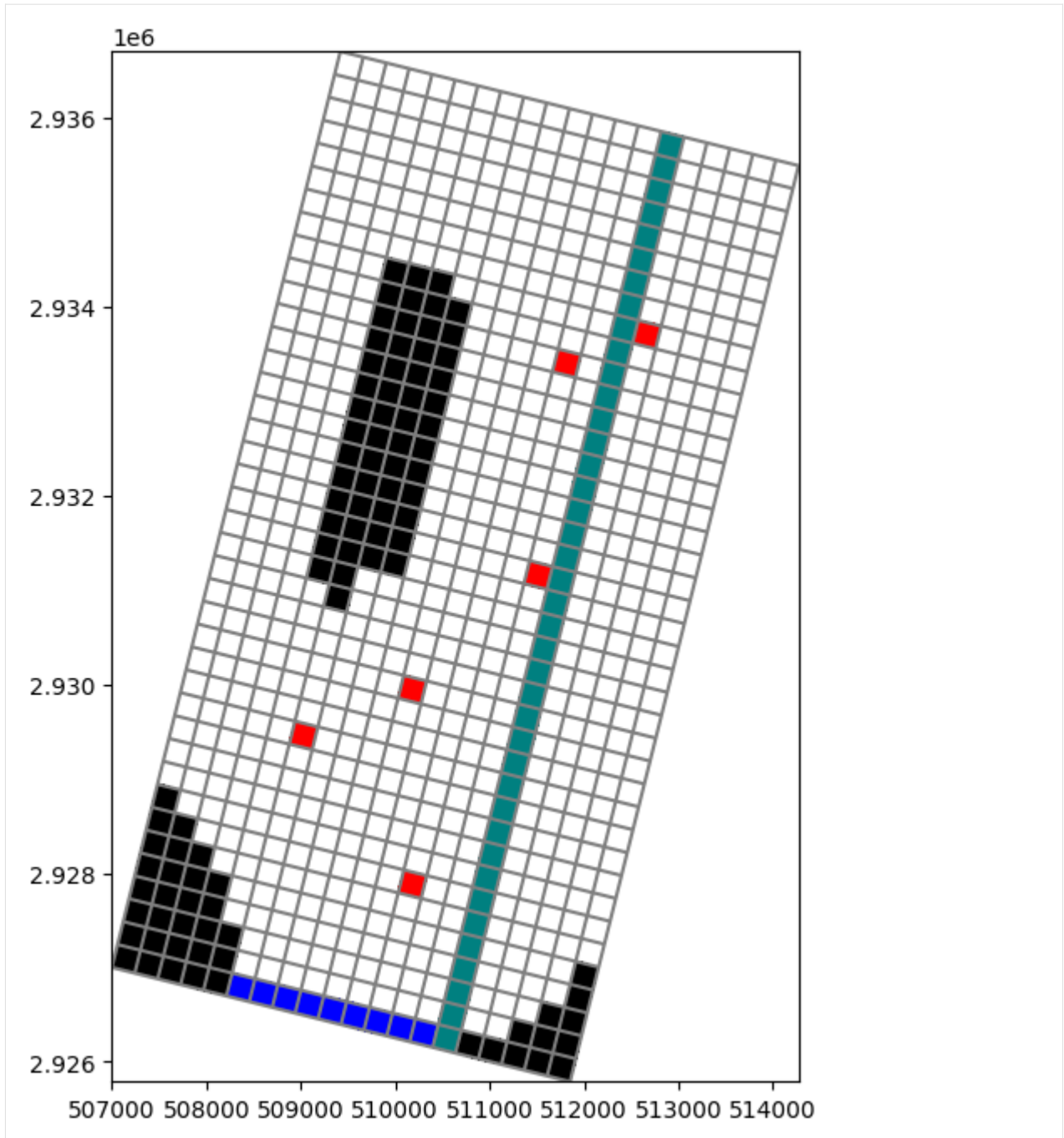
Plotting Boundary Conditions

The `plot_bc()` method can be used to plot boundary conditions. It is setup to use the following dictionary to assign colors, however, these colors can be changed in the method call.

```
bc_color_dict = {'default': 'black', 'WEL': 'red', 'DRN': 'yellow',  
                'RIV': 'green', 'GHB': 'cyan', 'CHD': 'navy'}
```

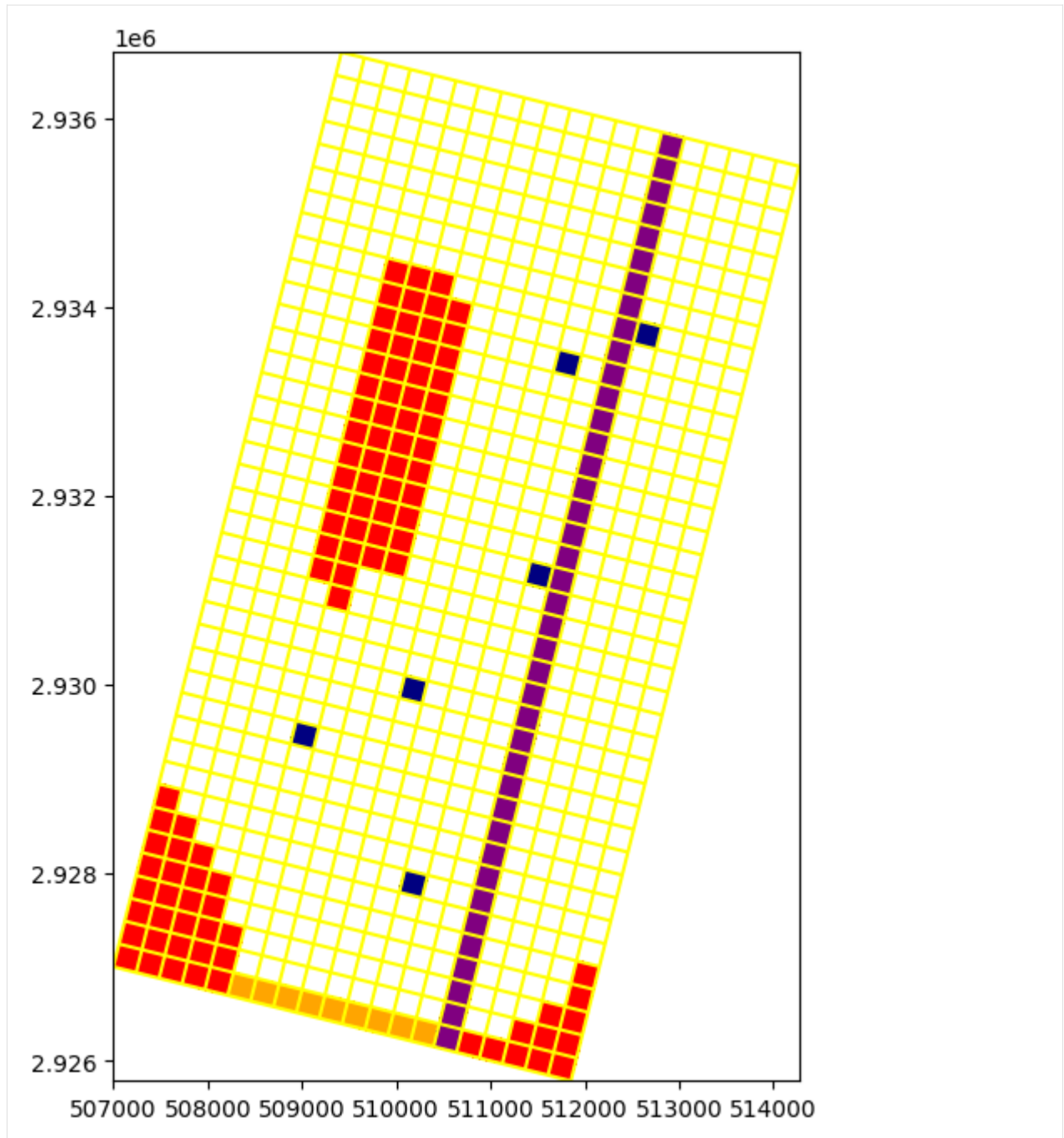
Here, we plot the location of river cells and the location of well cells.

```
[9]: fig = plt.figure(figsize=(8, 8))  
     ax = fig.add_subplot(1, 1, 1, aspect="equal")  
     mapview = flopy.plot.PlotMapView(model=ml)  
     quadmesh = mapview.plot_ibound()  
     quadmesh = mapview.plot_bc("RIV")  
     quadmesh = mapview.plot_bc("WEL")  
     linecollection = mapview.plot_grid()
```



The colors can be changed by using the `color_noflow` and `color_ch` parameters in `plot_ibound()`, the `color` parameter in `plot_bc()`, and the `colors` parameter in `plot_grid()`

```
[10]: fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mapview = flopy.plot.PlotMapView(model=m1)
quadmesh = mapview.plot_ibound(color_noflow="red", color_ch="orange")
quadmesh = mapview.plot_bc("RIV", color="purple")
quadmesh = mapview.plot_bc("WEL", color="navy")
linecollection = mapview.plot_grid(colors="yellow")
```

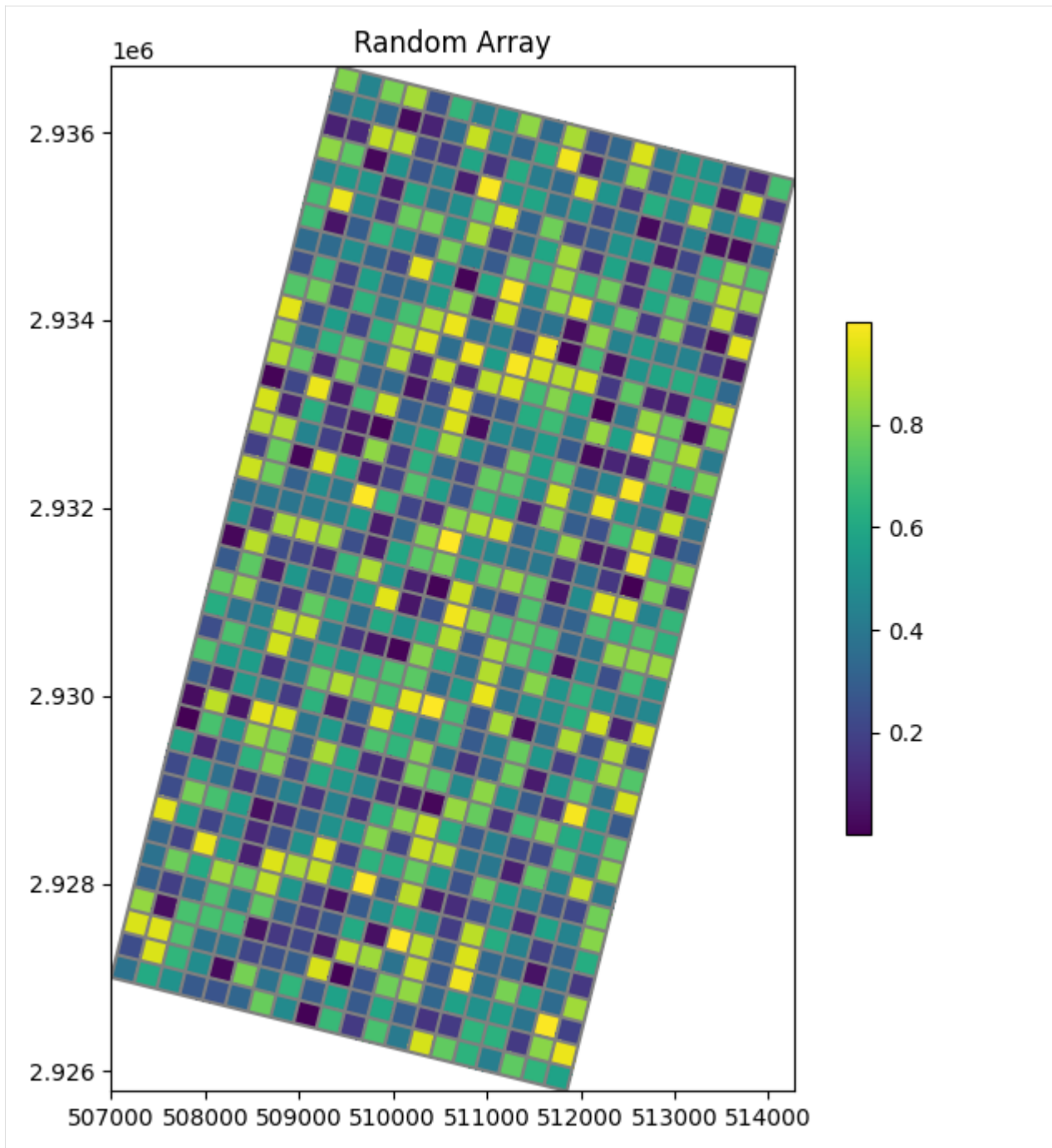


Plotting an Array

PlotMapView has a `plot_array()` method. The `plot_array()` method will accept either a 2D or 3D array. If a 3D array is passed, then the `layer` parameter for the PlotMapView object will be used (note that the PlotMapView object can be created with a `layer=` argument).

```
[11]: # Create a random array and plot it
a = np.random.random((ml.dis.nrow, ml.dis.ncol))

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("Random Array")
mapview = flopy.plot.PlotMapView(model=ml, layer=0)
quadmesh = mapview.plot_array(a)
linecollection = mapview.plot_grid()
cb = plt.colorbar(quadmesh, shrink=0.5)
```



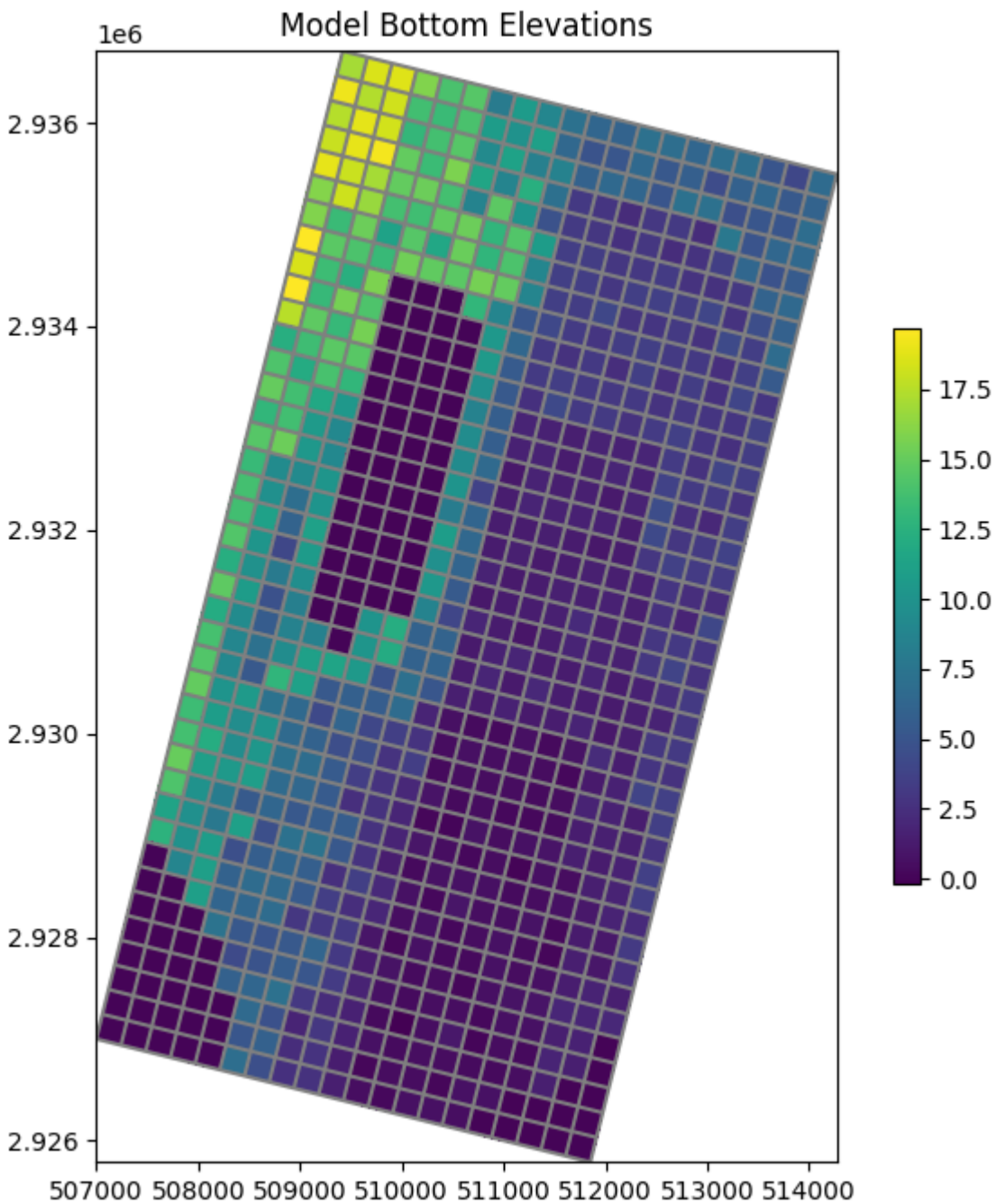
```
[12]: # Plot the model bottom array
a = ml.dis.botm.array

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("Model Bottom Elevations")
mapview = flopy.plot.PlotMapView(model=ml, layer=0)
quadmesh = mapview.plot_array(a)
```

(continues on next page)

(continued from previous page)

```
linecollection = mapview.plot_grid()  
cb = plt.colorbar(quadmesh, shrink=0.5)
```



Contouring an Array

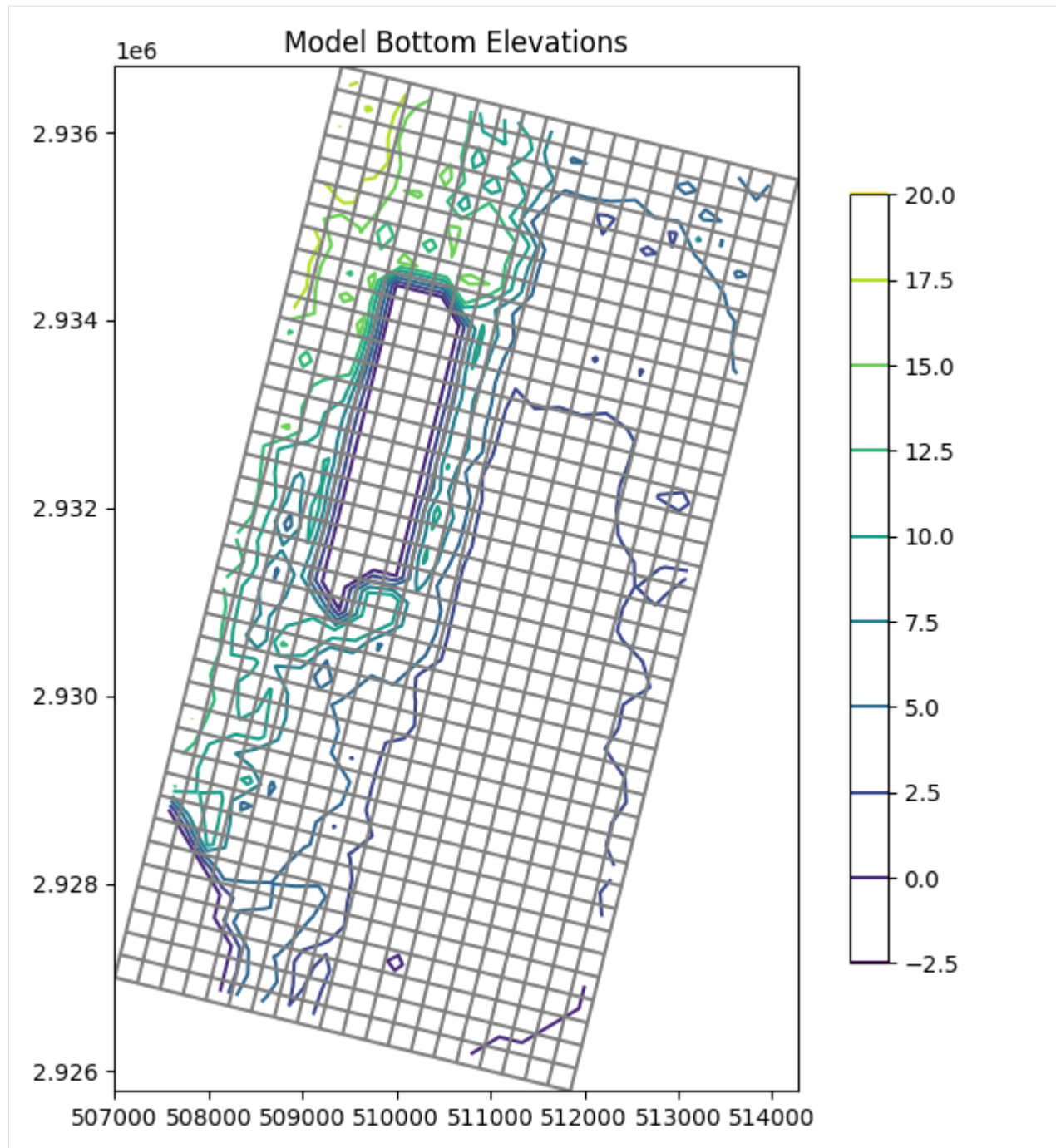
PlotMapView also has a `contour_array()` method. It also takes a 2D or 3D array and will contour the layer slice if 3D.

```
[13]: # Contour the model bottom array
a = ml.dis.botm.array

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("Model Bottom Elevations")
mapview = flopy.plot.PlotMapView(model=ml, layer=0)
contour_set = mapview.contour_array(a)
linecollection = mapview.plot_grid()

plt.colorbar(contour_set, shrink=0.75)

[13]: <matplotlib.colorbar.Colorbar at 0x7f9f97ea0430>
```

```
[14]: # The contour_array() method will take any keywords
# that can be used by the matplotlib.pyplot.contour
# function. So we can pass in levels, for example.
a = ml.dis.botm.array
levels = np.arange(0, 20, 0.5)

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
```

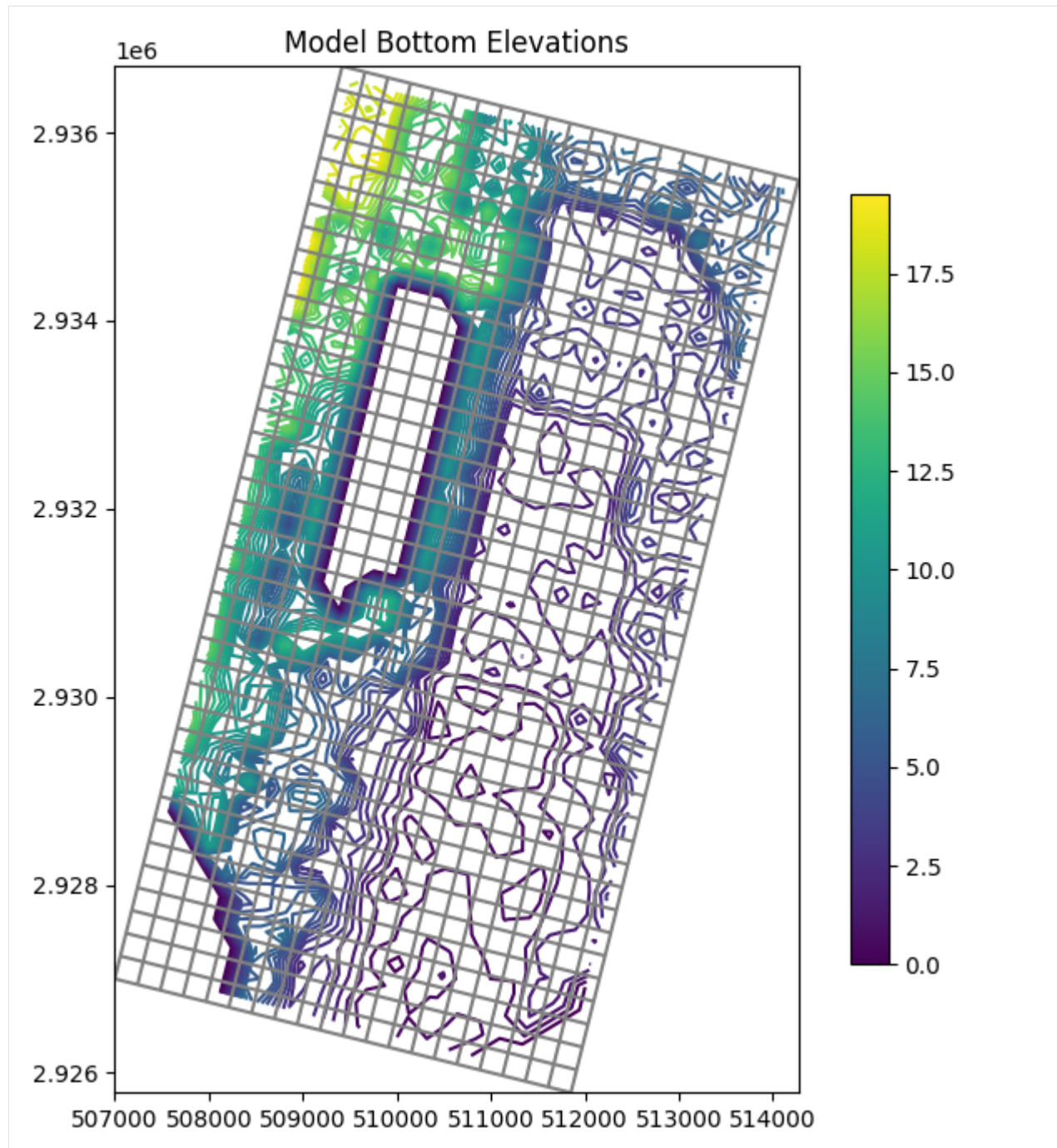
(continues on next page)

(continued from previous page)

```
ax.set_title("Model Bottom Elevations")
mapview = flopy.plot.PlotMapView(model=ml, layer=0)
contour_set = mapview.contour_array(a, levels=levels)
linecollection = mapview.plot_grid()

# set up and plot a continuous colorbar in matplotlib for a contour plot
norm = mpl.colors.Normalize(
    vmin=contour_set.cvalues.min(), vmax=contour_set.cvalues.max()
)
sm = plt.cm.ScalarMappable(norm=norm, cmap=contour_set.cmap)
sm.set_array([])
fig.colorbar(sm, shrink=0.75, ax=ax)
```

```
[14]: <matplotlib.colorbar.Colorbar at 0x7f9f949e6670>
```



Array contours can be exported directly to a shapefile.

```
[15]: from flopy.export.utils import ( # use export_contourf for filled contours
      export_contours,
      )

      shp_path = os.path.join(modelpth, "contours.shp")
      export_contours(shp_path, contour_set)
```

(continues on next page)

(continued from previous page)

```

from shapefile import Reader

with Reader(shp_path) as r:
    nshapes = len(r.shapes())
    print("Contours:", nshapes)

```

No CRS information for writing a .prj file.

Supply an valid coordinate system reference to the attached modelgrid object or .

→ export() method.

wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks

Contours: 360

Plotting Heads

So this means that we can easily plot results from the simulation by extracting heads using `flopy.utils.HeadFile`. Here we plot the simulated heads.

```

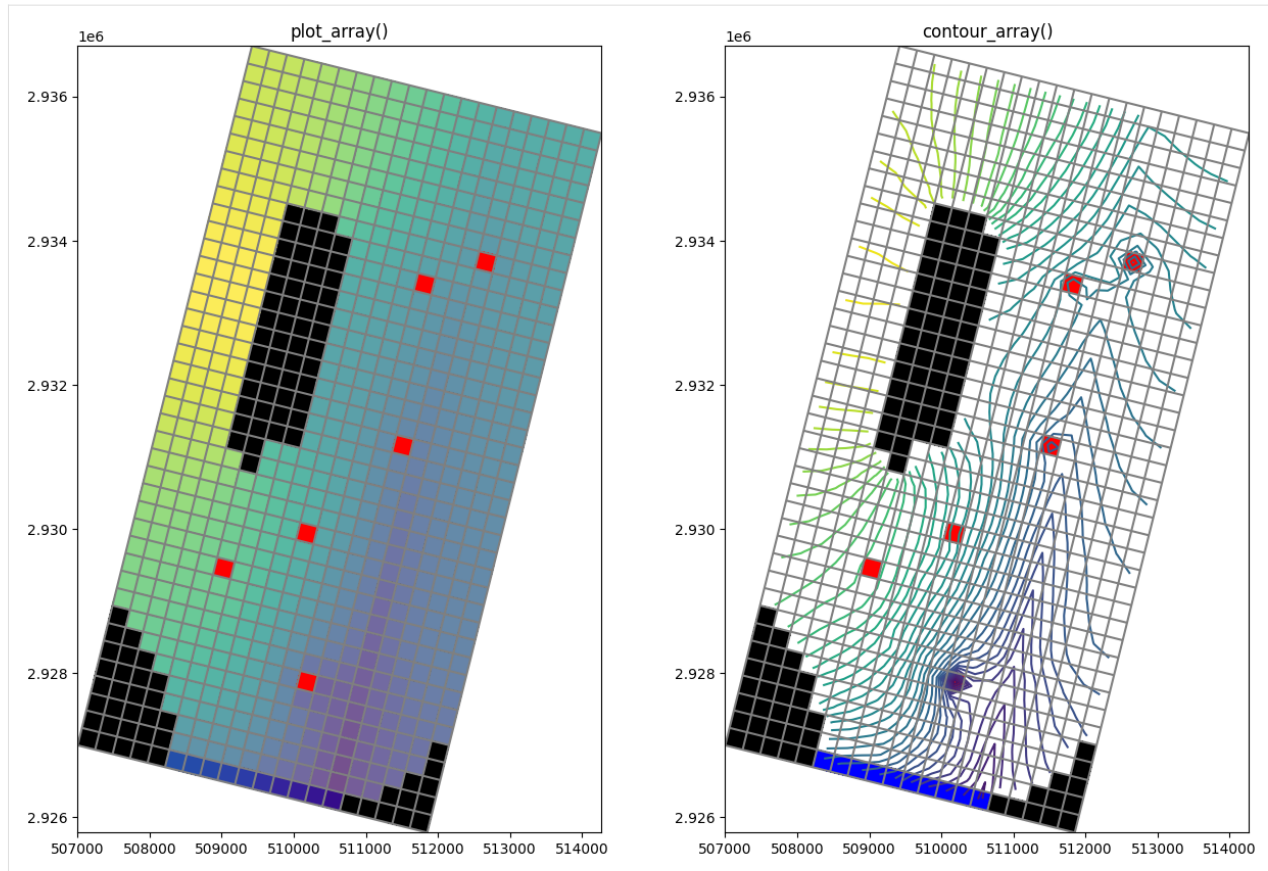
[16]: fname = os.path.join(modelpth, "freyberg.hds")
      hdoobj = flopy.utils.HeadFile(fname)
      head = hdoobj.get_data()
      levels = np.arange(10, 30, 0.5)

      fig = plt.figure(figsize=(15, 10))

      ax = fig.add_subplot(1, 2, 1, aspect="equal")
      ax.set_title("plot_array()")
      mapview = flopy.plot.PlotMapView(model=ml)
      quadmesh = mapview.plot_ibound()
      quadmesh = mapview.plot_array(head, alpha=0.5)
      mapview.plot_bc("WEL")
      linecollection = mapview.plot_grid()

      ax = fig.add_subplot(1, 2, 2, aspect="equal")
      ax.set_title("contour_array()")
      mapview = flopy.plot.PlotMapView(model=ml)
      quadmesh = mapview.plot_ibound()
      mapview.plot_bc("WEL")
      contour_set = mapview.contour_array(head, levels=levels)
      linecollection = mapview.plot_grid()

```



Plotting Discharge Vectors

PlotMapView has a `plot_vector()` method, which takes vector components in the x- and y-directions at the cell centers. The x- and y-vector components are calculated from the 'FLOW RIGHT FACE' and 'FLOW FRONT FACE' arrays, which can be written by MODFLOW to the cell by cell budget file. These array can be extracted from the cell by cell flow file using the `flopy.utils.CellBudgetFile` object as shown below. Once they are extracted, they can be passed to the `postprocessing.get_specific_discharge()` method to get the discharge vectors and plotted using the `plot_vector()` method.

Note: `postprocessing.get_specific_discharge()` also takes the head array as an optional argument. The head array is used to convert the volumetric discharge in dimensions of L^3/T to specific discharge in dimensions of L/T .

```
[17]: fname = os.path.join(modelpth, "freyberg.cbc")
      cbb = flopy.utils.CellBudgetFile(fname)
      head = hdoobj.get_data()
      frf = cbb.get_data(text="FLOW RIGHT FACE")[0]
      fff = cbb.get_data(text="FLOW FRONT FACE")[0]
      flf = None

      qx, qy, qz = flopy.utils.postprocessing.get_specific_discharge(
          (frf, fff, None), ml
      ) # no head array for volumetric discharge
      sqx, sqy, sqz = flopy.utils.postprocessing.get_specific_discharge(
          (frf, fff, None), ml, head
```

(continues on next page)

(continued from previous page)

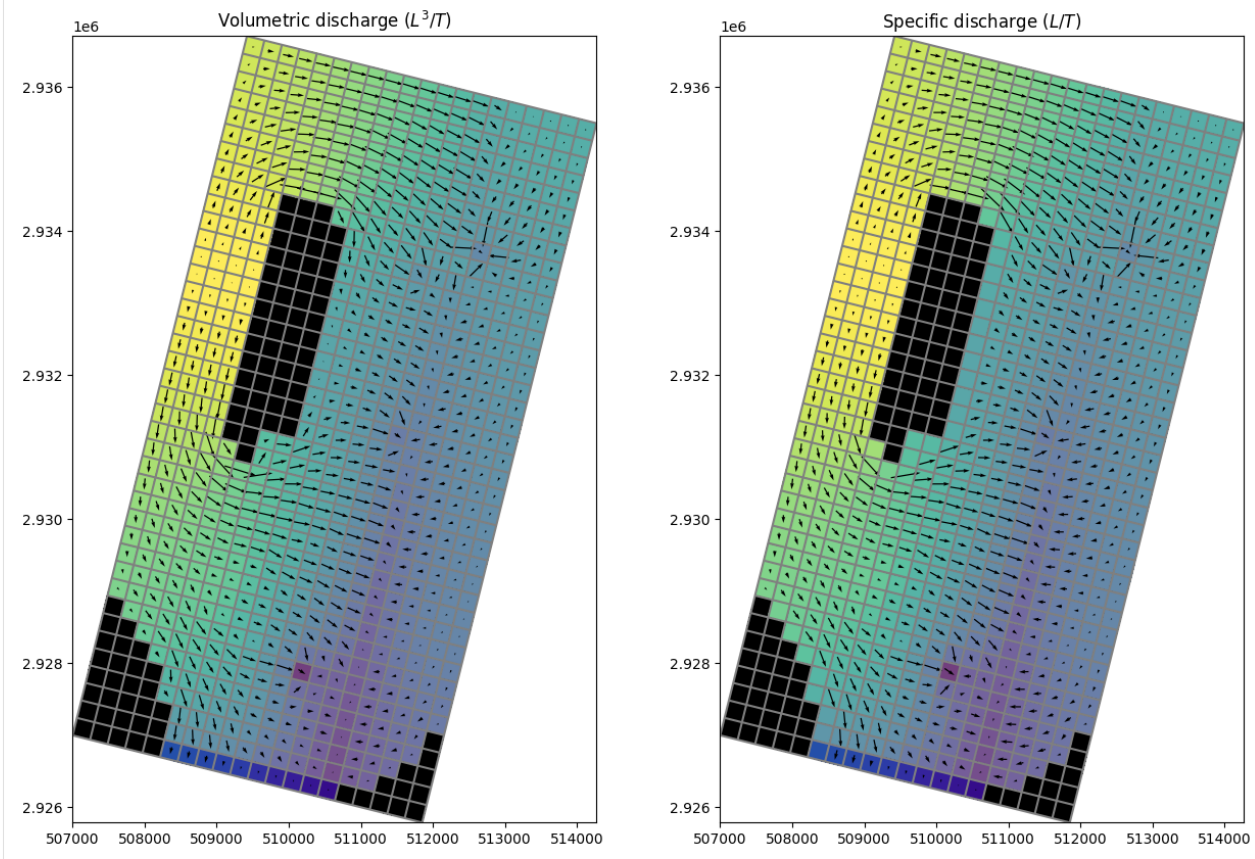
```

)

fig = plt.figure(figsize=(15, 10))

ax = fig.add_subplot(1, 2, 1, aspect="equal")
ax.set_title("Volumetric discharge (" + r"$L^3/T$" + ")")
mapview = flopy.plot.PlotMapView(model=ml)
quadmesh = mapview.plot_ibound()
quadmesh = mapview.plot_array(head, alpha=0.5)
quiver = mapview.plot_vector(qx, qy)
linecollection = mapview.plot_grid()

ax = fig.add_subplot(1, 2, 2, aspect="equal")
ax.set_title("Specific discharge (" + r"$L/T$" + ")")
mapview = flopy.plot.PlotMapView(model=ml)
quadmesh = mapview.plot_ibound()
quadmesh = mapview.plot_array(head, alpha=0.5)
quiver = mapview.plot_vector(
    sqx, sqy
) # include the head array for specific discharge
linecollection = mapview.plot_grid()
    
```



Plotting MODPATH endpoints and pathlines

PlotMapView has a `plot_endpoint()` and `plot_pathline()` method, which takes MODPATH endpoint and pathline data and plots them on the map object. Here we load the endpoint and pathline data and plot them on the head and discharge data previously plotted. Pathlines are shown for all times less than or equal to 200 years. Recharge capture zone data for all of the pumping wells are plotted as circle markers colored by travel time.

```
[18]: # load the endpoint data
endfile = os.path.join(modelpth, mp.sim.endpoint_file)
endobj = flopy.utils.EndpointFile(endfile)
ept = endobj.get_alldata()

# load the pathline data
pthfile = os.path.join(modelpth, mpp.sim.pathline_file)
pthobj = flopy.utils.PathlineFile(pthfile)
plines = pthobj.get_alldata()

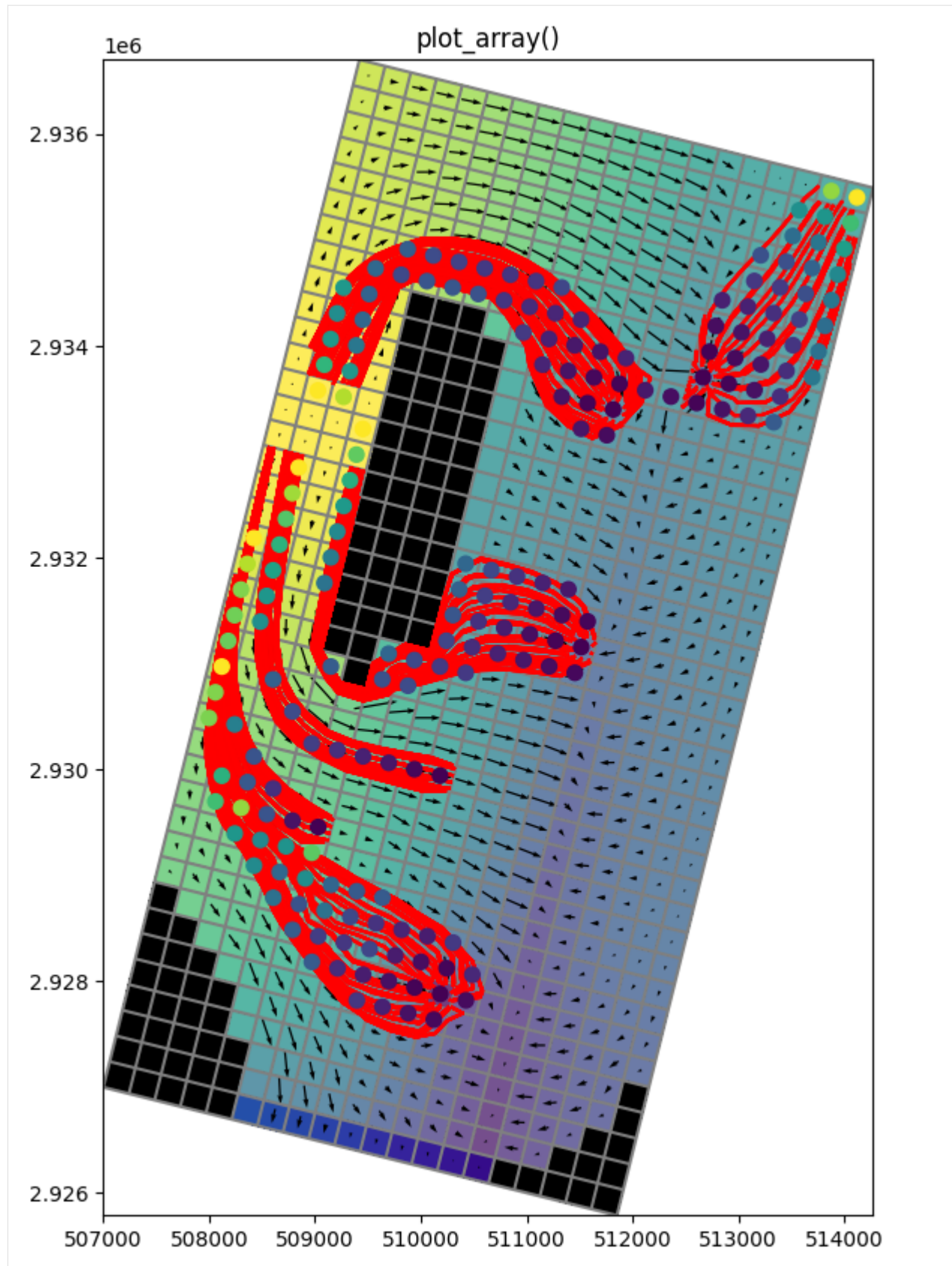
# plot the data
fig = plt.figure(figsize=(10, 10))

ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("plot_array()")
mapview = flopy.plot.PlotMapView(model=ml)
quadmesh = mapview.plot_ibound()
quadmesh = mapview.plot_array(head, alpha=0.5)
quiver = mapview.plot_vector(sqx, sqy)
linecollection = mapview.plot_grid()
for d in ml.wel.stress_period_data[0]:
    mapview.plot_endpoint(
        ept,
        direction="starting",
        selection_direction="ending",
        selection=(d[0], d[1], d[2]),
        zorder=100,
    )

# construct maximum travel time to plot (200 years - MODFLOW time unit is seconds)
travel_time_max = 200.0 * 365.25 * 24.0 * 60.0 * 60.0
ctt = "<={}".format(travel_time_max)

# plot the pathlines
mapview.plot_pathline(plines, layer="all", colors="red", travel_time=ctt)
```

```
[18]: <matplotlib.collections.LineCollection at 0x7f9f8e7dbfd0>
```



Plotting a Shapefile

PlotMapView has a `plot_shapefile()` method that can be used to quickly plot a shapefile on your map. In order to use the `plot_shapefile()` method, you must be able to “import shapefile”. The command `import shapefile` is part of the `pyshp` package.

The `plot_shapefile()` function can plot points, lines, and polygons and will return a `patch_collection` of objects from the shapefile. For a shapefile of polygons, the `plot_shapefile()` function will try to plot and fill them all using a different color. For a shapefile of points, you may need to specify a radius, in model units, in order for the circles to show up properly.

The shapefile must have intersecting geographic coordinates as the `PlotMapView` object in order for it to overlay correctly on the plot. The `plot_shapefile()` method and function do not use any of the projection information that may be stored with the shapefile. If you reset `xoff`, `yoff`, and `angrot` in the `ml.modelgrid.set_coord_info()` call below, you will see that the grid will no longer overlay correctly with the shapefile.

```
[19]: # Setup the figure and PlotMapView. Show a very faint map of ibound and
# model grid by specifying a transparency alpha value.
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

# reset the grid rotation and offsets to 0
ml.modelgrid.set_coord_info(xoff=0, yoff=0, angrot=0)

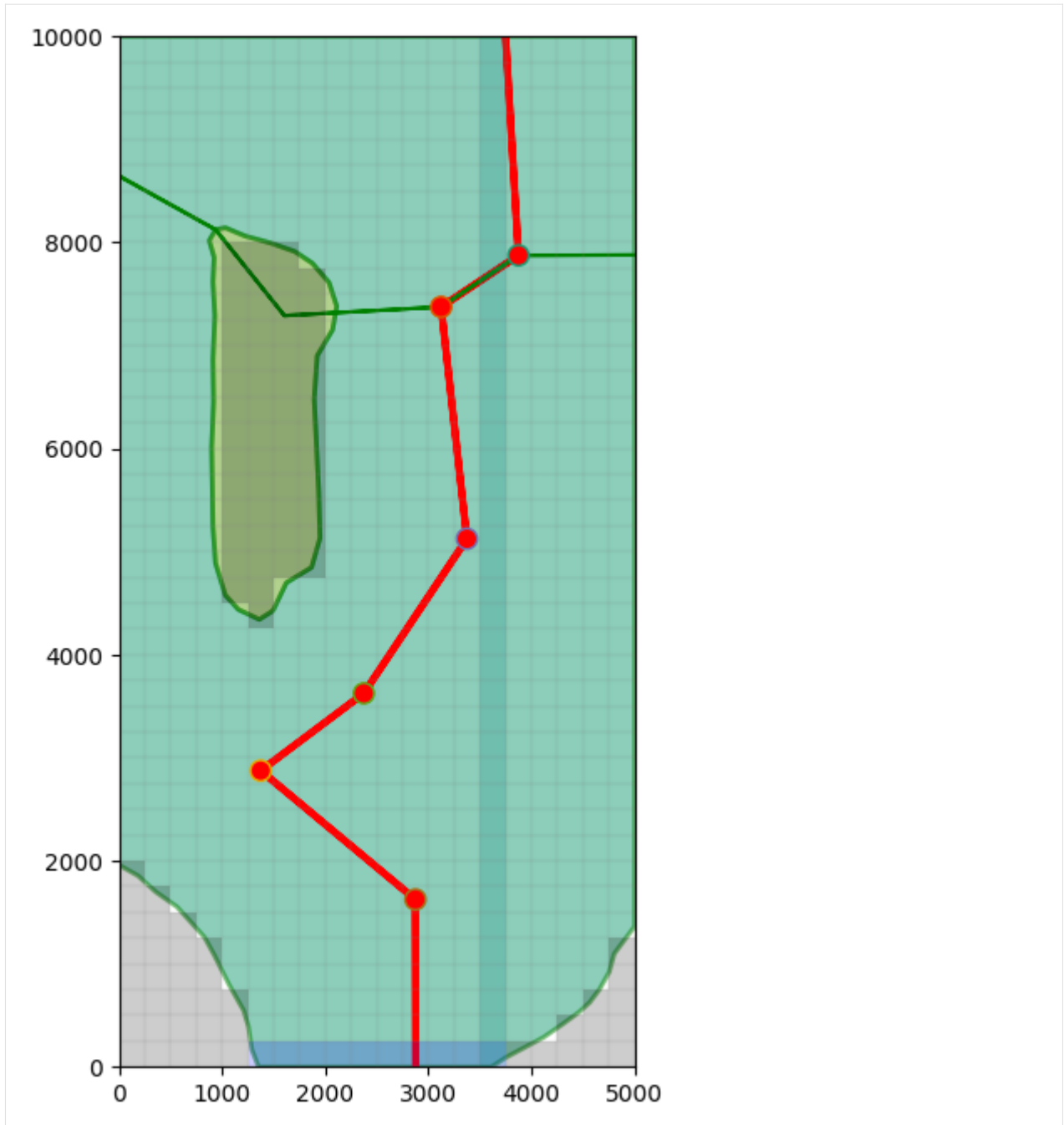
mapview = flopy.plot.PlotMapView(model=ml, ax=ax)

# Plot a shapefile of
shp = os.path.join(loadpth, "gis", "bedrock_outcrop_hole")
patch_collection = mapview.plot_shapefile(
    shp, edgecolor="green", linewidths=2, alpha=0.5 # facecolor='none',
)

# Plot a shapefile of a cross-section line
shp = os.path.join(loadpth, "gis", "cross_section")
patch_collection = mapview.plot_shapefile(
    shp, radius=0, lw=[3, 1.5], edgecolor=["red", "green"], facecolor="None"
)

# Plot a shapefile of well locations
shp = os.path.join(loadpth, "gis", "wells_locations")
patch_collection = mapview.plot_shapefile(shp, radius=100, facecolor="red")

# Plot the grid and boundary conditions over the top
quadmesh = mapview.plot_ibound(alpha=0.1)
quadmesh = mapview.plot_bc("RIV", alpha=0.1)
linecollection = mapview.plot_grid(alpha=0.1)
```



Although the `PlotMapView`'s `plot_shapefile()` method does not consider projection information when plotting maps, it can be used to plot shapefiles when a `PlotMapView` instance is rotated and offset into geographic coordinates. The same shapefiles plotted above (but in geographic coordinates rather than model coordinates) are plotted on the rotated model grid. The offset from model coordinates to geographic coordinates relative to the lower left corner are `xoff=-2419.22`, `yoff=297.04` and the rotation angle is 14° .

```
[20]: # Setup the figure and PlotMapView. Show a very faint map of ibound and
      # model grid by specifying a transparency alpha value.

      # set the modelgrid rotation and offset
```

(continues on next page)

(continued from previous page)

```

ml.modelgrid.set_coord_info(
    xoff=-2419.2189559966773, yoff=297.0427372400354, angrot=-14
)

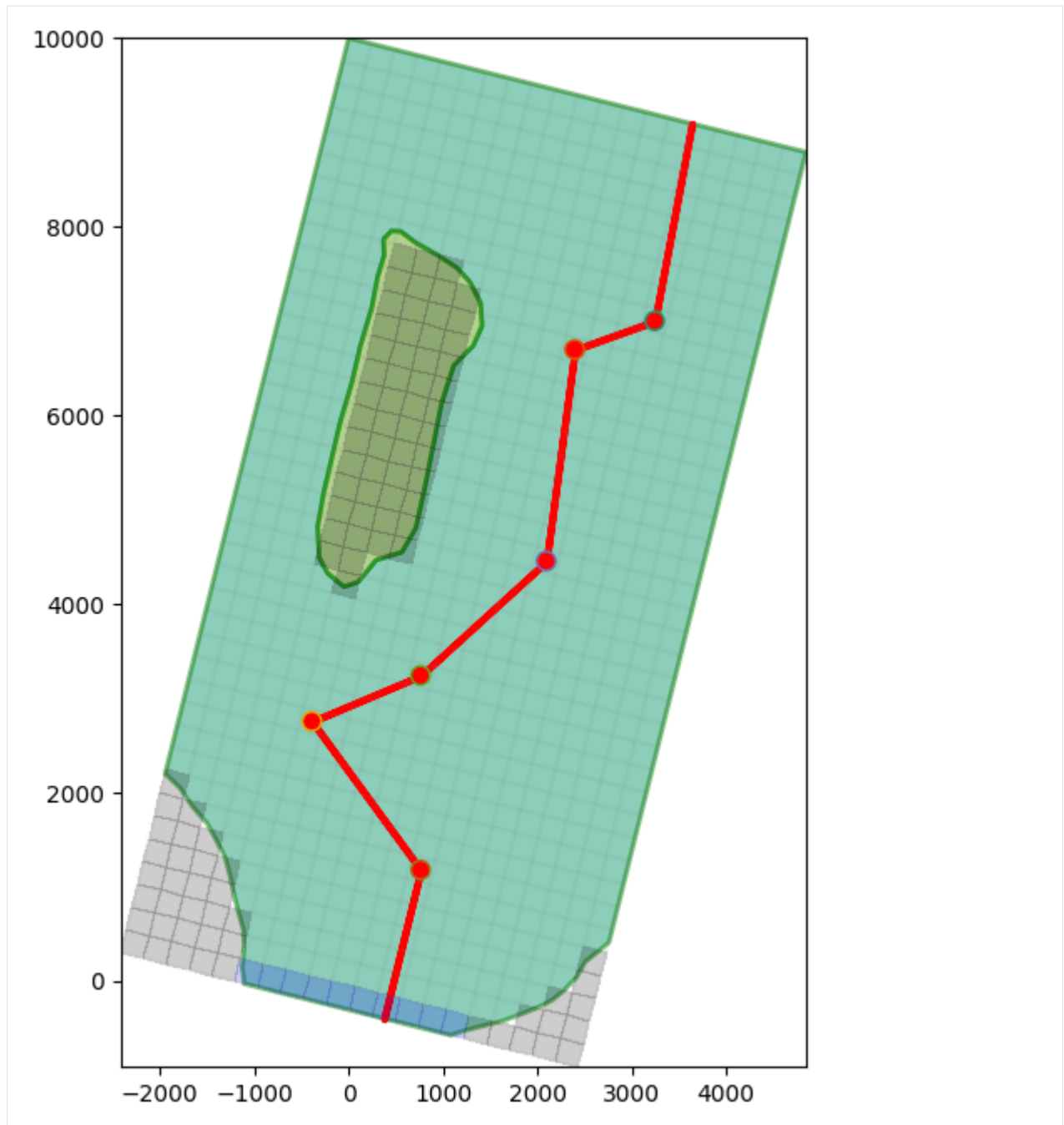
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mapview = flopy.plot.PlotMapView(model=ml)

# Plot a shapefile of
shp = os.path.join(loadpth, "gis", "bedrock_outcrop_hole_rotate14")
patch_collection = mapview.plot_shapefile(
    shp, edgecolor="green", linewidths=2, alpha=0.5 # facecolor='none',
)
# Plot a shapefile of a cross-section line
shp = os.path.join(loadpth, "gis", "cross_section_rotate14")
patch_collection = mapview.plot_shapefile(shp, lw=3, edgecolor="red")

# Plot a shapefile of well locations
shp = os.path.join(loadpth, "gis", "wells_locations_rotate14")
patch_collection = mapview.plot_shapefile(shp, radius=100, facecolor="red")

# Plot the grid and boundary conditions over the top
quadmesh = mapview.plot_ibound(alpha=0.1)
linecollection = mapview.plot_grid(alpha=0.1)

```



Plotting GIS Shapes

PlotMapView has a `plot_shapes()` method that can be used to quickly plot GIS based shapes on your map. In order to use the `plot_shapes()` method, you must be able to “import shapefile”. The command `import shapefile` is part of the `pyshp` package.

The `plot_shapes()` function can plot points, lines, polygons, and multipolygons and will return a `patch_collection`. For a list or collection of polygons, the `plot_shapes()` function will try to plot and fill them all using a different color. For a list or collection of points, you may need to specify a radius, in model units, in order for the circles to show up properly.

Note: The supplied shapes must have intersecting geographic coordinates as the `PlotMapView` object in order for it to overlay correctly on the plot.

`plot_shapes()` supports many GIS based input types and they are listed below:

```
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[ ]:
[21]: #
      # Here is a basic example of how to use the method:

[22]: # lets extract some shapes from our shapefiles
      shp = os.path.join(loadpth, "gis", "bedrock_outcrop_hole_rotate14")
      with shapefile.Reader(shp) as r:
          polygon_w_hole = [
              r.shape(0),
          ]

      shp = os.path.join(loadpth, "gis", "cross_section_rotate14")
      with shapefile.Reader(shp) as r:
          cross_section = r.shapes()

      # Plot a shapefile of well locations
      shp = os.path.join(loadpth, "gis", "wells_locations_rotate14")
      with shapefile.Reader(shp) as r:
          wells = r.shapes()
```

Now that the shapes are extracted from the shapefiles, they can be plotted using `plot_shapes()`

```
[23]: # set the modelgrid rotation and offset
ml.modelgrid.set_coord_info(
    xoff=-2419.2189559966773, yoff=297.0427372400354, angrot=-14
)

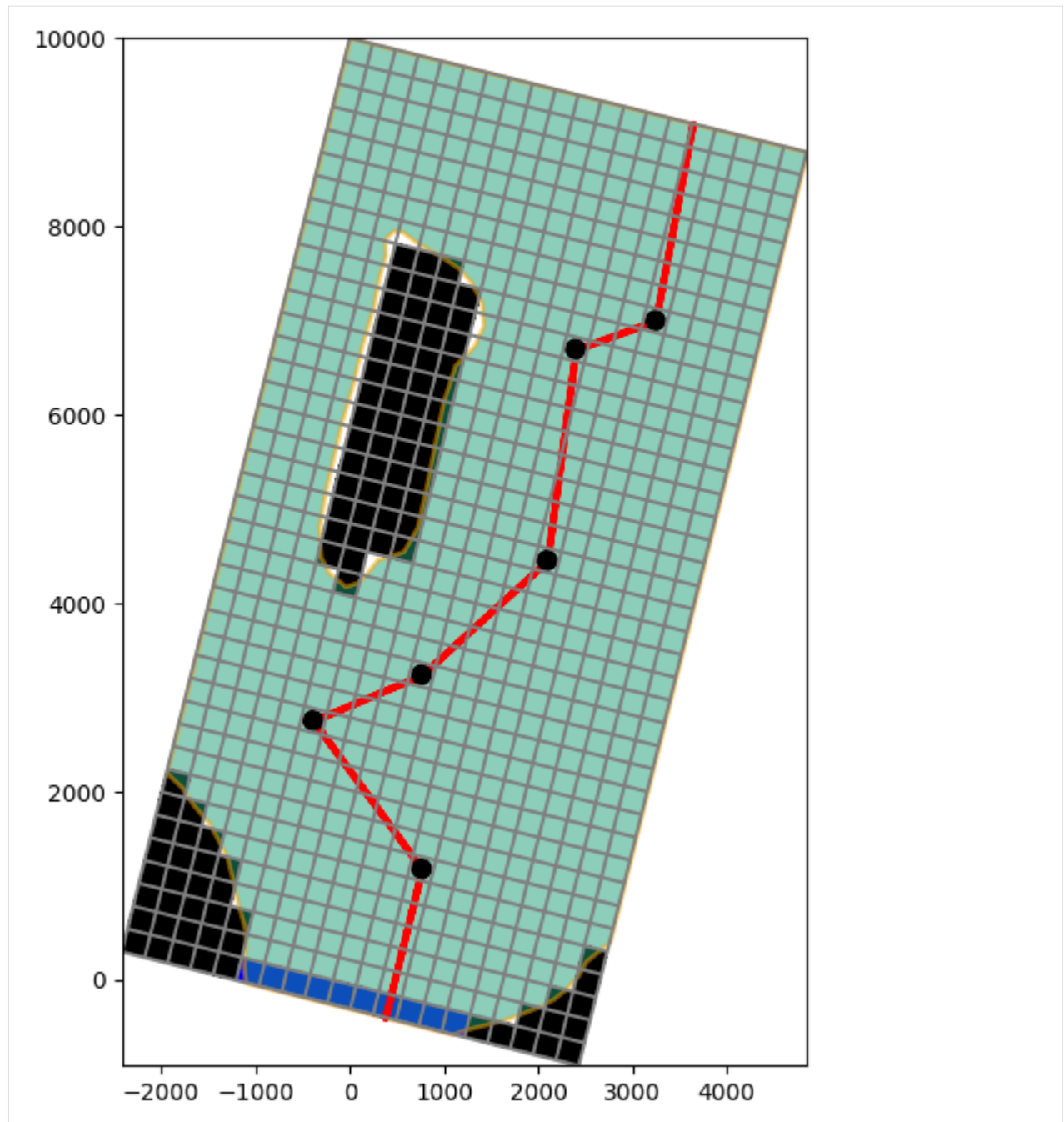
fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mapview = flopy.plot.PlotMapView(model=ml)

# Plot the grid and boundary conditions
quadmesh = mapview.plot_ibound()
linecollection = mapview.plot_grid()

# plot polygon(s)
patch_collection0 = mapview.plot_shapes(
    polygon_w_hole, edgecolor="orange", linewidths=2, alpha=0.5
)

# plot_line(s)
patch_collection1 = mapview.plot_shapes(cross_section, lw=3, edgecolor="red")

# plot_point(s)
patch_collection3 = mapview.plot_shapes(
    wells, radius=100, facecolor="k", edgecolor="k"
)
```



Working with MODFLOW-6 models

PlotMapView has support for MODFLOW-6 models and operates in the same fashion for Structured Grids, Vertex Grids, and Unstructured Grids. Here is a short example on how to plot with MODFLOW-6 structured grids using a version of the Freyberg model created for MODFLOW-6

```
[24]: # load the Freyberg model into mf6-flopy and run the simulation
sim_name = "mfsim.nam"
sim_path = str(prj_root / "examples" / "data" / "mf6-freyberg")
sim = flopy.mf6.MFSimulation.load(
    sim_name=sim_name, version=vmf6, exe_name=exe_name_mf6, sim_ws=sim_path
)

newpth = os.path.join(modelpth)
sim.set_sim_path(newpth)
sim.write_simulation()
success, buff = sim.run_simulation()
if not success:
    print("Something bad happened.")
files = ["freyberg.hds", "freyberg.cbc"]
for f in files:
    if os.path.isfile(os.path.join(modelpth, f)):
        msg = "Output file located: {}".format(f)
        print(msg)
    else:
        errmsg = "Error. Output file cannot be found: {}".format(f)
        print(errmsg)
```

```
loading simulation...
  loading simulation name file...
  loading tdis package...
  loading model gwf6...
    loading package dis...
    loading package ic...
WARNING: Block "options" is not a valid block name for file type ic.
    loading package oc...
    loading package npf...
    loading package sto...
    loading package chd...
    loading package riv...
    loading package wel...
    loading package rch...
  loading solution package freyberg...
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package freyberg...
  writing model freyberg...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package oc...
    writing package npf...
    writing package sto...
```

(continues on next page)

(continued from previous page)

```
writing package chd-1...
writing package riv-1...
writing package wel-1...
writing package rch-1...
FloPy is using the following executable to run the model: ../../home/runner/.local/bin/
↪modflow/mf6
```

```
MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023
```

```
MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000
```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

```
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:48:57
```

```
Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam
```

```
Solving: Stress period:      1      Time step:      1
```

```
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:48:57
Elapsed run time: 0.103 Seconds
```

```
Normal termination of simulation.
Output file located: freyberg.hds
Output file located: freyberg.cbc
```

Plotting boundary conditions and arrays

This works the same as modflow-2005, however the simulation object can host a number of modflow-6 models so we need to grab a model before attempting to plot with PlotMapView

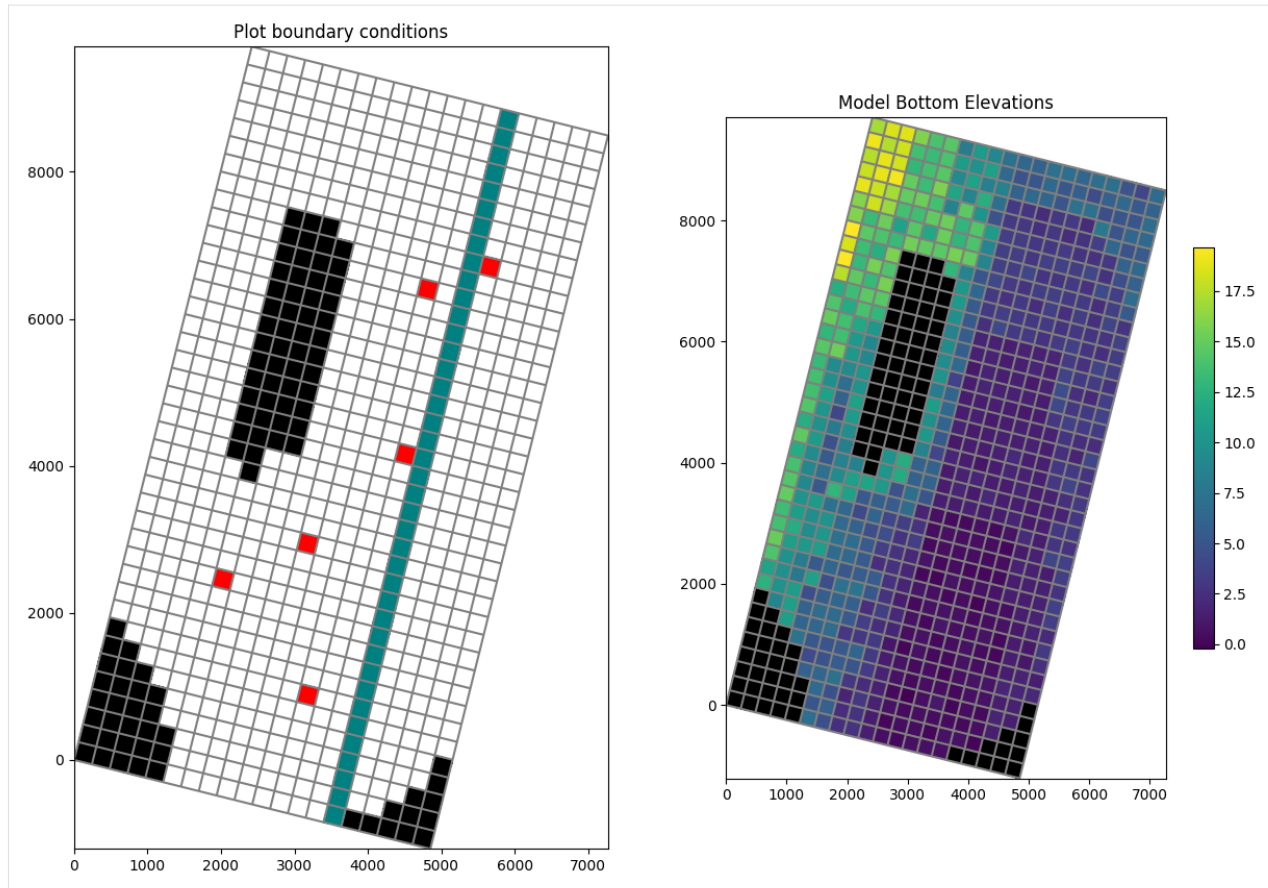
```
[25]: # get the modflow-6 model we want to plot
ml6 = sim.get_model("freyberg")
ml6.modelgrid.set_coord_info(angrot=-14)

fig = plt.figure(figsize=(15, 10))

# plot boundary conditions
ax = fig.add_subplot(1, 2, 1, aspect="equal")
mapview = flopy.plot.PlotMapView(model=ml6)
quadmesh = mapview.plot_ibound()
quadmesh = mapview.plot_bc("RIV")
quadmesh = mapview.plot_bc("WEL")
linecollection = mapview.plot_grid()
ax.set_title("Plot boundary conditions")

# plot model bottom elevations
a = ml6.dis.botm.array

ax = fig.add_subplot(1, 2, 2, aspect="equal")
ax.set_title("Model Bottom Elevations")
mapview = flopy.plot.PlotMapView(model=ml6, layer=0)
quadmesh = mapview.plot_array(a)
inactive = mapview.plot_inactive()
linecollection = mapview.plot_grid()
cb = plt.colorbar(quadmesh, shrink=0.5, ax=ax)
```



Contouring Arrays

Contouring arrays follows the same code signature for MODFLOW-6 as the MODFLOW-2005 example. Just use the `contour_array()` method

```
[26]: # The contour_array() method will take any keywords
# that can be used by the matplotlib.pyplot.contour
# function. So we can pass in levels, for example.
a = ml6.dis.botm.array
levels = np.arange(0, 20, 0.5)

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("Model Bottom Elevations")
mapview = flopy.plot.PlotMapView(model=ml6, layer=0)
contour_set = mapview.contour_array(a, levels=levels)
linecollection = mapview.plot_grid()

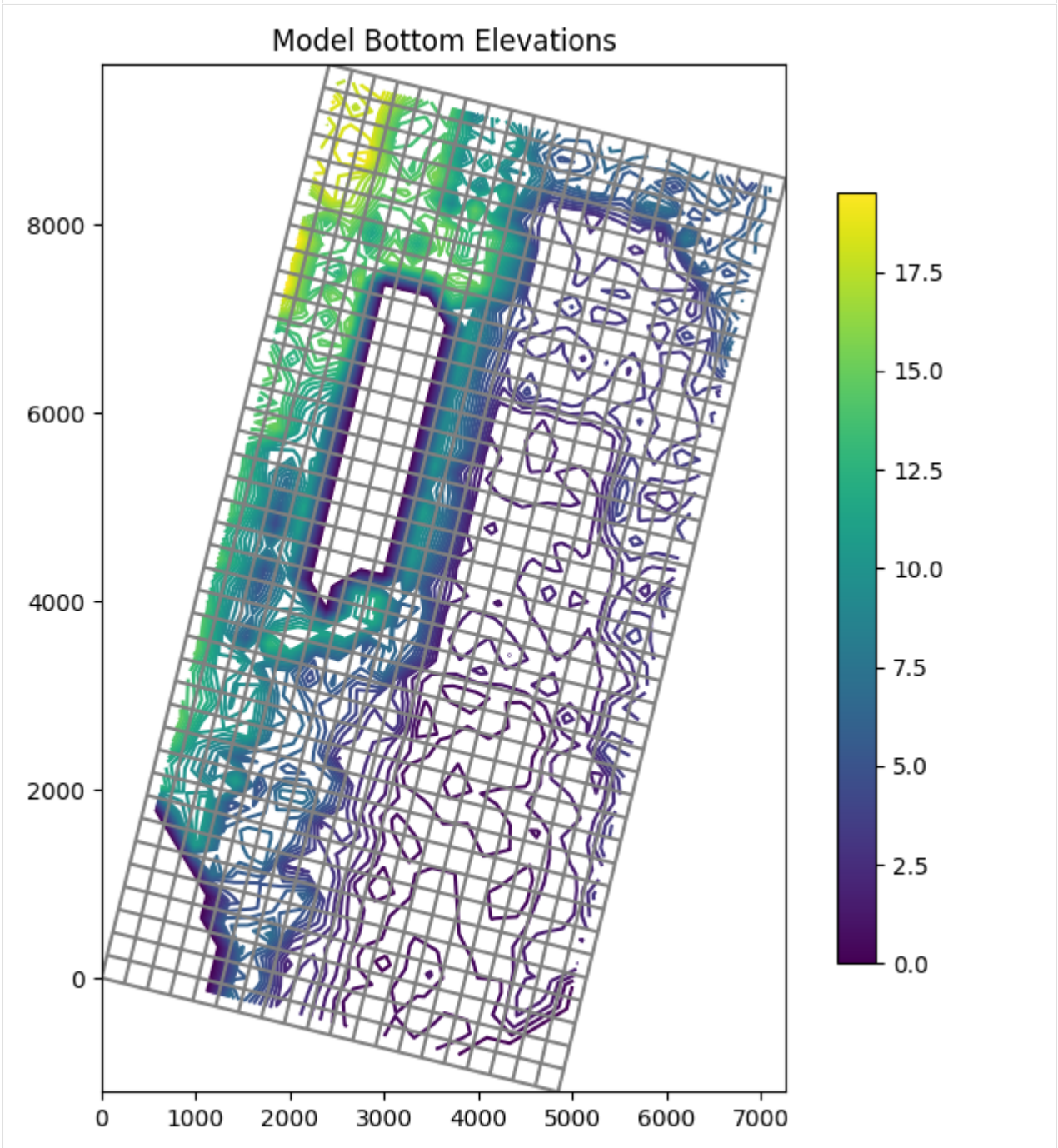
# set up and plot a continuous colorbar in matplotlib for a contour plot
norm = mpl.colors.Normalize(
    vmin=contour_set.cvalues.min(), vmax=contour_set.cvalues.max()
)
sm = plt.cm.ScalarMappable(norm=norm, cmap=contour_set.cmap)
```

(continues on next page)

(continued from previous page)

```
sm.set_array([])
fig.colorbar(sm, shrink=0.75, ax=ax)
```

[26]: <matplotlib.colorbar.Colorbar at 0x7f9f8e026ca0>



Plotting specific discharge with a MODFLOW-6 model

MODFLOW-6 includes a the PLOT_SPECIFIC_DISCHARGE flag in the NPF package to calculate and store discharge vectors for easy plotting. The postprocessing module will translate the specific discharge into vector array and PlotMapView has the plot_vector() method to use this data. The specific discharge array is stored in the cell budget file.

```
[27]: # get the specific discharge from the cell budget file
cbc_file = os.path.join(newpth, "freyberg.cbc")
cbc = flopy.utils.CellBudgetFile(cbc_file)
spdis = cbc.get_data(text="SPDIS")[0]

qx, qy, qz = flopy.utils.postprocessing.get_specific_discharge(spdis, ml6)

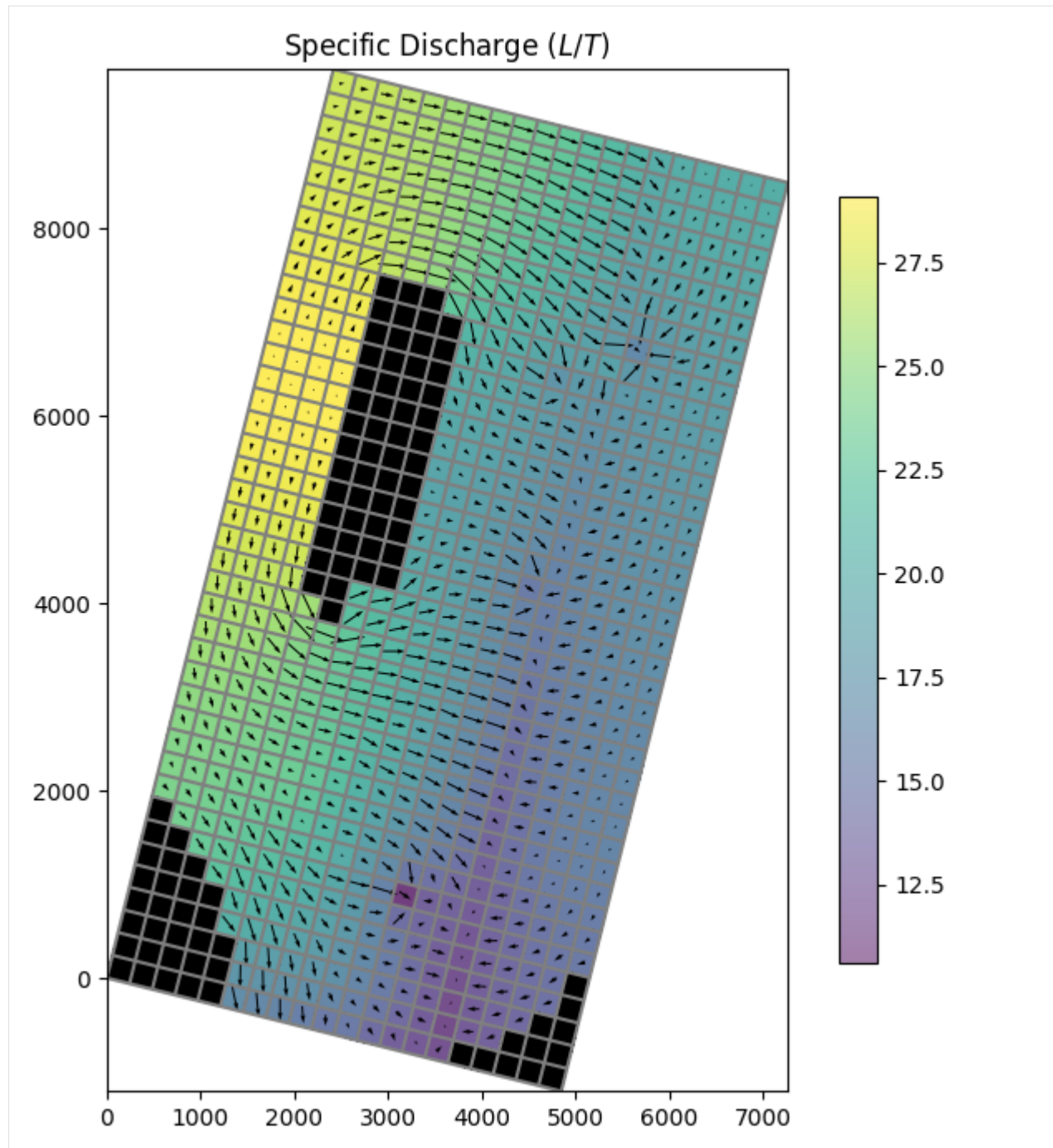
# get the head from the head file
head_file = os.path.join(newpth, "freyberg.hds")
head = flopy.utils.HeadFile(head_file)
hdata = head.get_alldata()[0]

# plot specific discharge using PlotMapView
fig = plt.figure(figsize=(8, 8))

mapview = flopy.plot.PlotMapView(model=ml6, layer=0)
linecollection = mapview.plot_grid()
quadmesh = mapview.plot_array(a=hdata, alpha=0.5)
quiver = mapview.plot_vector(qx, qy)
inactive = mapview.plot_inactive()

plt.title("Specific Discharge (" + r"$L/T$" + ")")
plt.colorbar(quadmesh, shrink=0.75)

[27]: <matplotlib.colorbar.Colorbar at 0x7f9f8df2aeb0>
```



Vertex model plotting with MODFLOW-6

FloPy fully supports vertex discretization (DISV) plotting through the `PlotMapView` class. The method calls are identical to the ones presented previously for Structured discretization (DIS) and the same matplotlib keyword arguments are supported. Let's run through an example using a vertex model grid.

[28]: *# build and run vertex model grid demo problem*

```
notebook_utils.run(modelpth)
```

```
# check if model ran properly
```

```
modelpth = os.path.join(modelpth, "mp7_ex2", "mf6")
```

```
files = ["mp7p2.hds", "mp7p2.cbb"]
```

```
for f in files:
```

```
    if os.path.isfile(os.path.join(modelpth, f)):
```

```
        msg = "Output file located: {}".format(f)
```

```
        print(msg)
```

```
    else:
```

```
        errmsg = "Error. Output file cannot be found: {}".format(f)
```

```
        print(errmsg)
```

```
WARNING: Unable to resolve dimension of ('gwf6', 'disv', 'cell2d', 'cell2d', 'icvert')_
↳based on shape "ncvert".
```

```
writing simulation...
```

```
    writing simulation name file...
```

```
    writing simulation tdis package...
```

```
    writing solution package ims...
```

```
    writing model mp7p2...
```

```
        writing model name file...
```

```
        writing package disv...
```

```
        writing package ic...
```

```
        writing package npf...
```

```
        writing package wel_0...
```

```
INFORMATION: maxbound in ('gwf6', 'wel', 'dimensions') changed to 1 based on size of_
```

```
↳stress_period_data
```

```
    writing package rcha_0...
```

```
    writing package riv_0...
```

```
INFORMATION: maxbound in ('gwf6', 'riv', 'dimensions') changed to 21 based on size of_
```

```
↳stress_period_data
```

```
    writing package oc...
```

```
MODFLOW 6
```

```
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
```

```
VERSION 6.4.2 06/28/2023
```

```
MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
```

```
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
```

```
Build 20220726_000000
```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the

(continues on next page)

(continued from previous page)

software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:49:00

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

Solving: Stress period: 1 Time step: 1

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:49:00
Elapsed run time: 0.147 Seconds

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'mp7p2.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'mp7p2.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.

Normal termination of simulation.

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.000000E+03 Steady-state flow

Particle Summary:

- 0 particles are pending release.
- 0 particles remain active.
- 16 particles terminated at boundary faces.
- 0 particles terminated at weak sink cells.
- 0 particles terminated at weak source cells.
- 0 particles terminated at strong source/sink cells.
- 0 particles terminated in cells with a specified zone number.
- 0 particles were stranded in inactive or dry cells.
- 0 particles were unreleased.
- 0 particles have an unknown status.

Normal termination.

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

(continues on next page)

(continued from previous page)

Processing Time Step 1 Period 1. Time = 1.000000E+03 Steady-state flow

Particle Summary:

```

    0 particles are pending release.
    0 particles remain active.
  416 particles terminated at boundary faces.
    0 particles terminated at weak sink cells.
    0 particles terminated at weak source cells.
    0 particles terminated at strong source/sink cells.
    0 particles terminated in cells with a specified zone number.
    0 particles were stranded in inactive or dry cells.
    0 particles were unreleased.
    0 particles have an unknown status.

```

Normal termination.

Output file located: mp7p2.hds

Output file located: mp7p2.cbb

```

[29]: # load the simulation and get the model
vertex_sim_name = "mfsim.nam"
vertex_sim = flopy.mf6.MFSimulation.load(
    sim_name=vertex_sim_name,
    version=vmf6,
    exe_name=exe_name_mf6,
    sim_ws=modelpth,
)
vertex_ml6 = vertex_sim.get_model("mp7p2")

```

```

loading simulation...
  loading simulation name file...
  loading tdis package...
  loading model gw6...
  loading package disv...
WARNING: Unable to resolve dimension of ('gw6', 'disv', 'cell2d', 'cell2d', 'icvert')_
↳ based on shape "ncvert".
  loading package ic...
  loading package npf...
  loading package wel...
  loading package rch...
  loading package riv...
  loading package oc...
loading solution package mp7p2...

```

Setting MODFLOW-6 Vertex Model Grid offsets, rotation and plotting

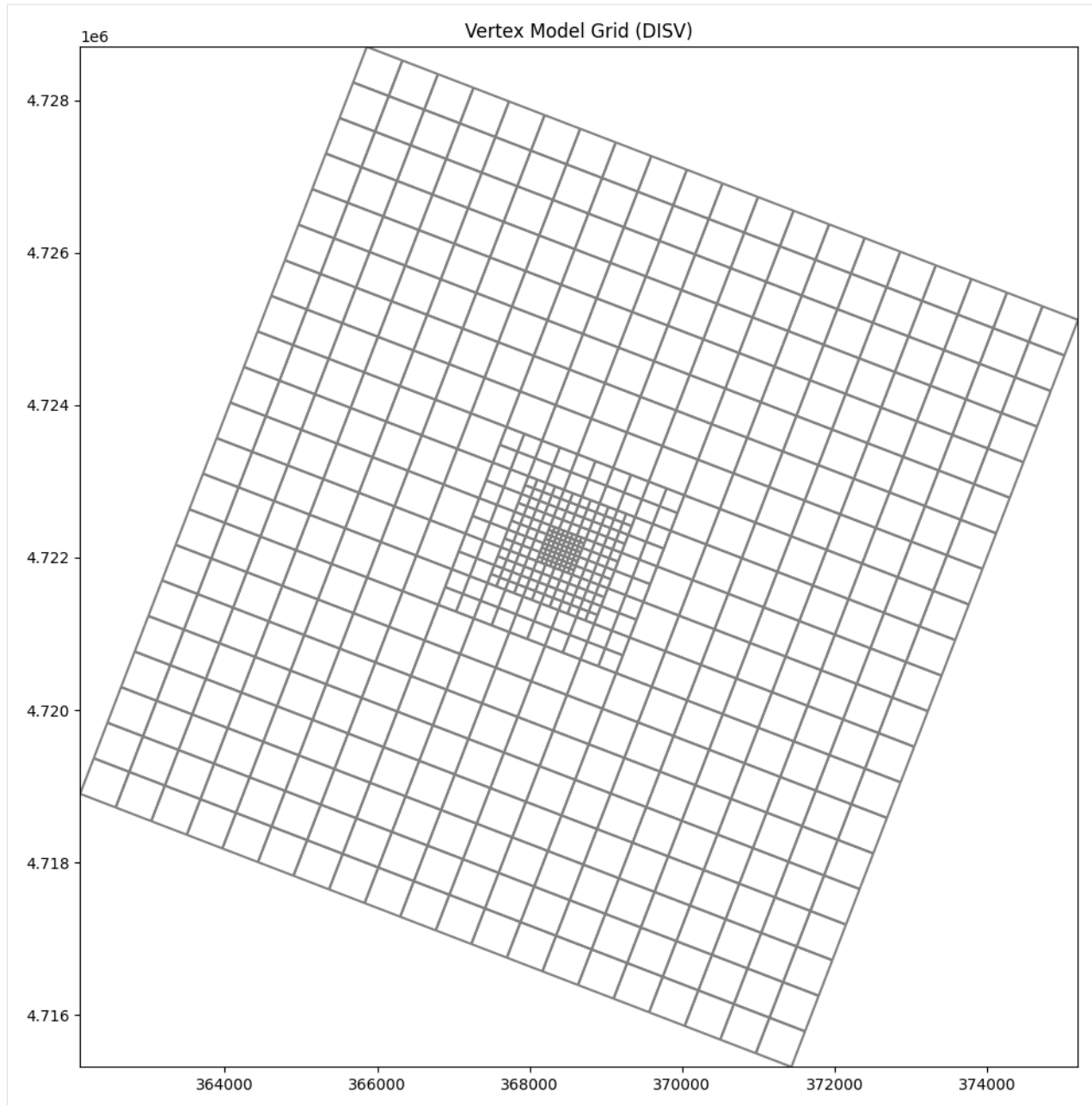
Setting the Grid offsets and rotation is consistent in FloPy, no matter which type of discretization the user is using. The `set_coord_info()` method on the `modelgrid` is used.

Plotting works consistently too, the user just calls the `PlotMapView` class and it accounts for the discretization type

```
[30]: # set coordinate information on the modelgrid
vertex_ml6.modelgrid.set_coord_info(xoff=362100, yoff=4718900, angrot=-21)

fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("Vertex Model Grid (DISV)")

# use PlotMapView to plot a DISV (vertex) model
mapview = flopy.plot.PlotMapView(vertex_ml6, layer=0)
linecollection = mapview.plot_grid()
```



Plotting boundary conditions with Vertex Model grids

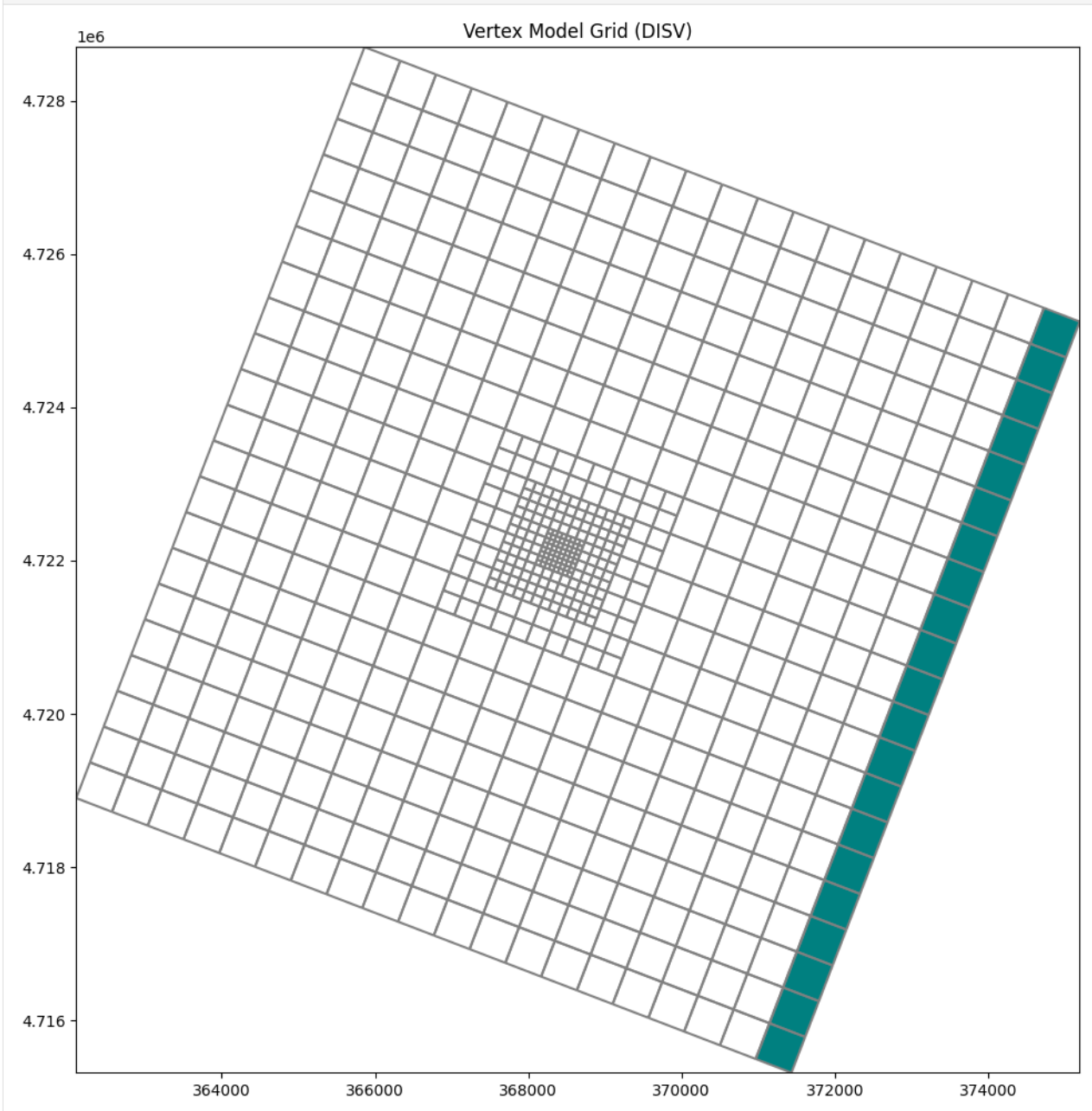
The `plot_bc()` method can be used to plot boundary conditions. It is setup to use the following dictionary to assign colors, however, these colors can be changed in the method call.

```
bc_color_dict = {'default': 'black', 'WEL': 'red', 'DRN': 'yellow',
                 'RIV': 'green', 'GHB': 'cyan', 'CHD': 'navy'}
```

Here we plot river (RIV) cell locations

```
[31]: fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("Vertex Model Grid (DISV)")

# use PlotMapView to plot a DISV (vertex) model
mapview = flopy.plot.PlotMapView(vertex_ml6, layer=0)
riv = mapview.plot_bc("RIV")
linecollection = mapview.plot_grid()
```



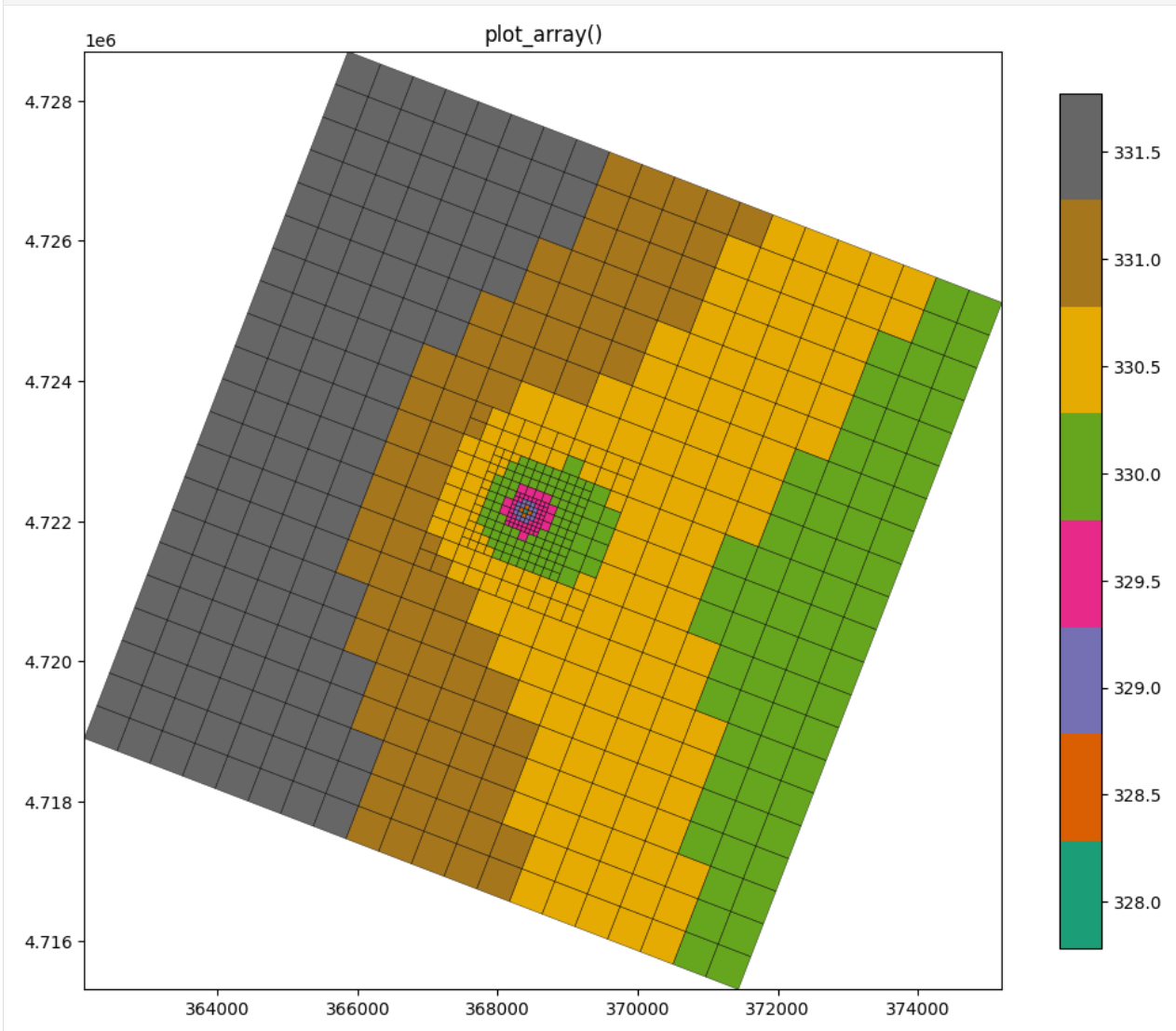
Plotting Arrays and Contouring with Vertex Model grids

PlotMapView allows the user to plot arrays and contour with DISV based discretization. The `plot_array()` method is called in the same way as using a structured grid. The only difference is that PlotMapView builds a matplotlib patch collection for Vertex based grids.

```
[32]: # get the head output for stress period 1 from the modflow6 head file
head = flopy.utils.HeadFile(os.path.join(modelpth, "mp7p2.hds"))
hdata = head.get_alldata()[0, :, :, :]

fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("plot_array()")

mapview = flopy.plot.PlotMapView(model=vertex_ml6, layer=2)
patch_collection = mapview.plot_array(hdata, cmap="Dark2")
linecollection = mapview.plot_grid(lw=0.25, color="k")
cb = plt.colorbar(patch_collection, shrink=0.75)
```



The `contour_array()` method operates in the same way as the structured example.

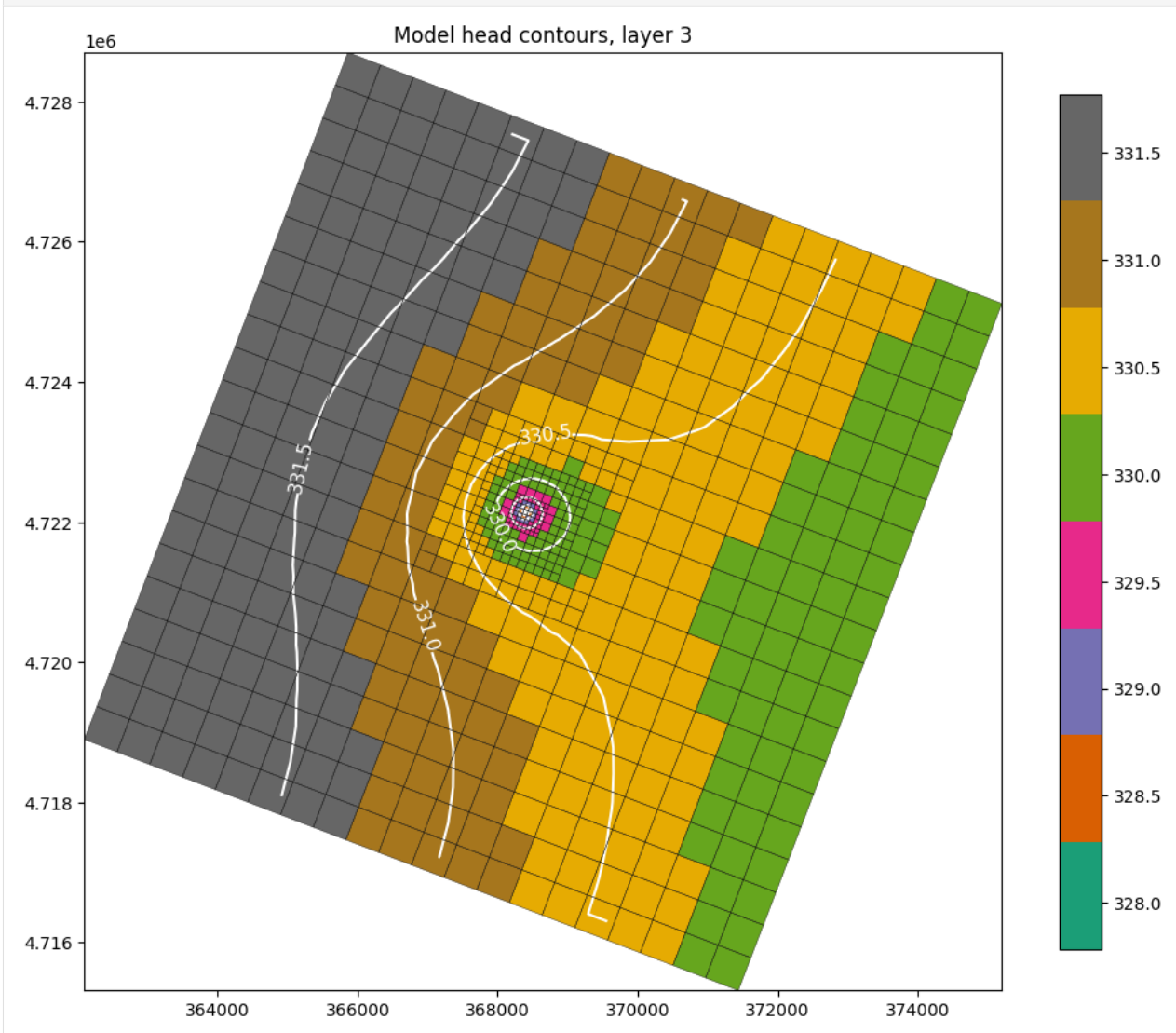
```
[33]: # plotting head array and then contouring the array!
levels = np.arange(327, 332, 0.5)

fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("Model head contours, layer 3")

mapview = flopy.plot.PlotMapView(model=vertex_ml6, layer=2)
pc = mapview.plot_array(hdata, cmap="Dark2")

# contouring the head array
contour_set = mapview.contour_array(hdata, levels=levels, colors="white")
plt.clabel(contour_set, fmt="%.1f", colors="white", fontsize=11)
linecollection = mapview.plot_grid(lw=0.25, color="k")

cb = plt.colorbar(pc, shrink=0.75, ax=ax)
```



Plotting MODPATH 7 results on a vertex model

MODPATH-7 results can be plotted using the same built in methods as used previously to plot MODPATH-6 results. The `plot_pathline()` and `plot_timeseries()` methods are layered on the previous example to show modpath simulation results

```
[34]: # load the MODPATH-7 results
mp_namea = "mp7p2a_mp"
fpth = os.path.join(modelpth, mp_namea + ".mppth")
p = flopy.utils.PathlineFile(fpth)
p0 = p.get_alldata()

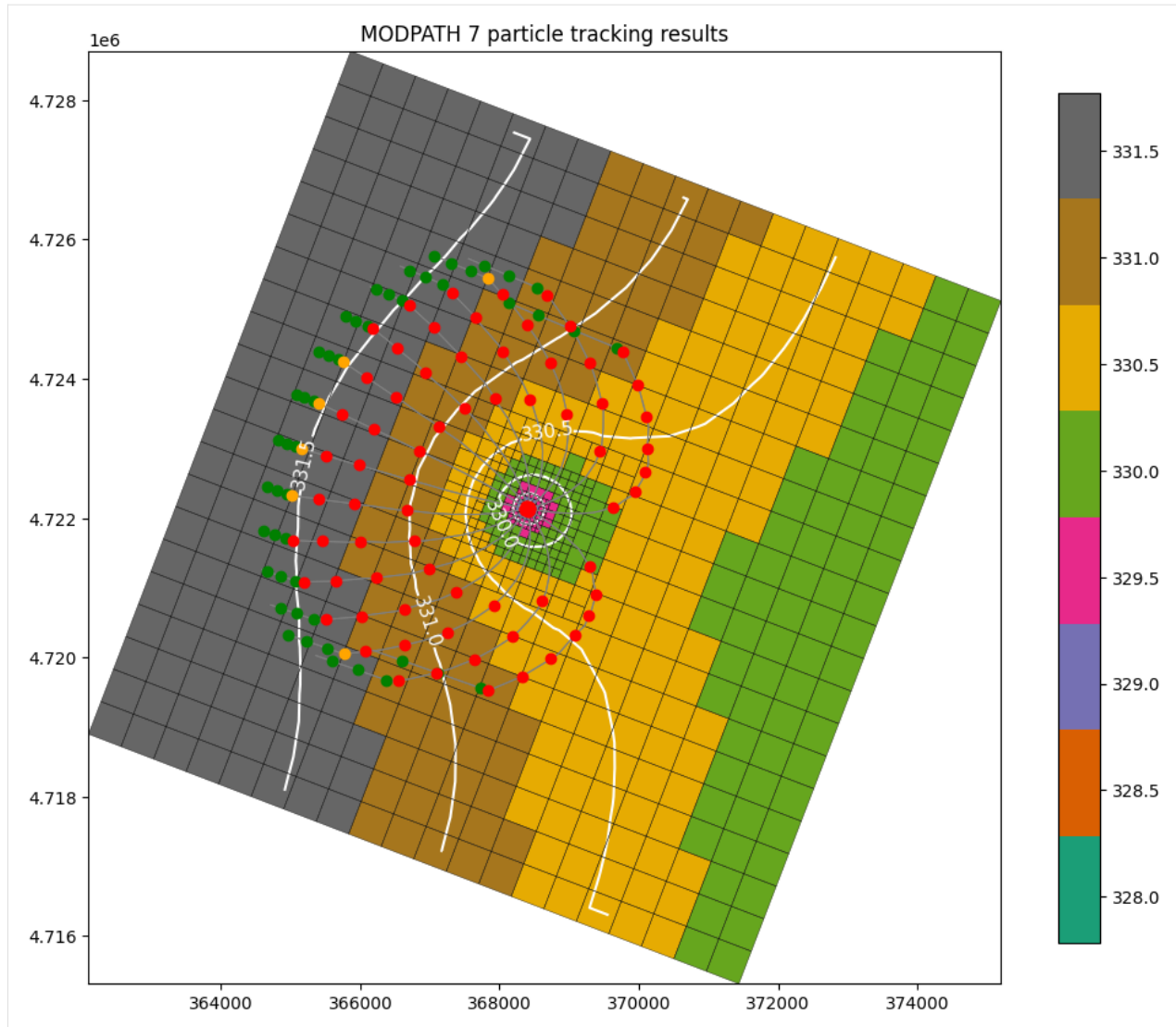
fpth = os.path.join(modelpth, mp_namea + ".timeseries")
ts = flopy.utils.TimeseriesFile(fpth)
ts0 = ts.get_alldata()

[35]: # setup the plot
fig = plt.figure(figsize=(12, 12))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("MODPATH 7 particle tracking results")

mapview = flopy.plot.PlotMapView(vertex_ml6, layer=2)

# plot and contour head arrays
pc = mapview.plot_array(hdata, cmap="Dark2")
contour_set = mapview.contour_array(hdata, levels=levels, colors="white")
plt.clabel(contour_set, fmt="%.1f", colors="white", fontsize=11)
linecollection = mapview.plot_grid(lw=0.25, color="k")
cb = plt.colorbar(pc, shrink=0.75, ax=ax)

# plot the modpath results
pline = mapview.plot_pathline(p0, layer="all", color="blue", lw=0.75)
colors = ["green", "orange", "red"]
for k in range(3):
    tseries = mapview.plot_timeseries(
        ts0, layer=k, marker="o", lw=0, color=colors[k]
    )
```

Plotting specific discharge vectors for DISV

MODFLOW-6 includes a the PLOT_SPECIFIC_DISCHARGE flag in the NPF package to calculate and store discharge vectors for easy plotting. The postprocessing module will translate the specific discharge into vector array and PlotMapView has the `plot_vector()` method to use this data. The specific discharge array is stored in the cell budget file.

```
[36]: cbb = flopy.utils.CellBudgetFile(
        os.path.join(modelpth, "mp7p2.cbb"), precision="double"
    )
    spdis = cbb.get_data(text="SPDIS")[0]
    qx, qy, qz = flopy.utils.postprocessing.get_specific_discharge(
        spdis, vertex_m16
    )

    fig = plt.figure(figsize=(12, 12))
```

(continues on next page)

(continued from previous page)

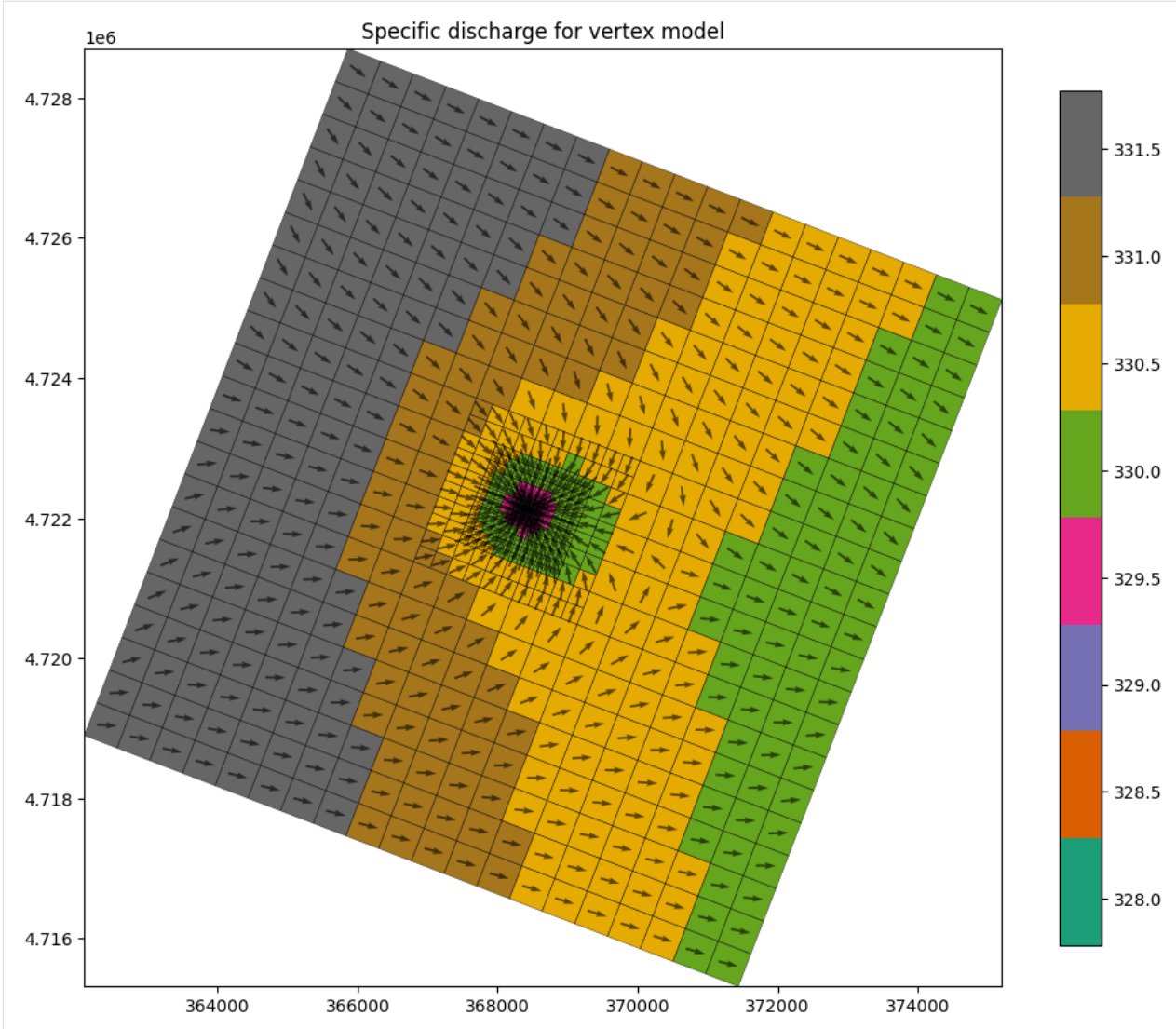
```

ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.set_title("Specific discharge for vertex model")

mapview = flopy.plot.PlotMapView(vertex_ml6, layer=2)
pc = mapview.plot_array(hdata, cmap="Dark2")
linecollection = mapview.plot_grid(lw=0.25, color="k")
cb = plt.colorbar(pc, shrink=0.75, ax=ax)

# plot specific discharge
quiver = mapview.plot_vector(qx, qy, normalize=True, alpha=0.60)

```



Unstructured grid (DISU) plotting with MODFLOW-USG and MODFLOW-6

Unstructured grid (DISU) plotting has support through the `PlotMapView` class and the `UnstructuredGrid` discretization object. The method calls are identical to those used for vertex (DISV) and structured (DIS) model grids. Let's run through a few unstructured grid examples

```
[37]: # set up the notebook for unstructured grid plotting
from flopy.discretization import UnstructuredGrid

# this is a folder containing some unstructured grids
datapath = str(prj_root / "examples" / "data" / "unstructured")

# simple functions to load vertices and incidence lists
def load_verts(fname):
    verts = np.genfromtxt(
        fname, dtype=[int, float, float], names=["iv", "x", "y"]
    )
    verts["iv"] -= 1 # zero based
    return verts

def load_iverts(fname):
    f = open(fname, "r")
    iverts = []
    xc = []
    yc = []
    for line in f:
        ll = line.strip().split()
        iverts.append([int(i) - 1 for i in ll[4:]])
        xc.append(float(ll[1]))
        yc.append(float(ll[2]))
    return iverts, np.array(xc), np.array(yc)
```

```
[38]: # load vertices
fname = os.path.join(datapath, "ugrid_verts.dat")
verts = load_verts(fname)

# load the incidence list into iverts
fname = os.path.join(datapath, "ugrid_iverts.dat")
iverts, xc, yc = load_iverts(fname)
```

In this case, `verts` is just a 2-dimensional list of x,y vertex pairs. `iverts` is also a 2-dimensional list, where the outer list is of size `ncells`, and the inner list is a list of the vertex numbers that comprise the cell.

```
[39]: # Print the first 5 entries in verts and iverts
for ivert, v in enumerate(verts[:5]):
    print("Vertex coordinate pair for vertex {}: {}".format(ivert, v))
print("...\n")

for icell, vertlist in enumerate(iverts[:5]):
    print("List of vertices for cell {}: {}".format(icell, vertlist))
```

```

Vertex coordinate pair for vertex 0: (0, 0., 700.)
Vertex coordinate pair for vertex 1: (1, 100., 700.)
Vertex coordinate pair for vertex 2: (2, 100., 600.)
Vertex coordinate pair for vertex 3: (3, 0., 600.)
Vertex coordinate pair for vertex 4: (4, 200., 700.)
...

List of vertices for cell 0: [0, 1, 2, 3, 0]
List of vertices for cell 1: [1, 4, 5, 2, 1]
List of vertices for cell 2: [4, 6, 7, 5, 4]
List of vertices for cell 3: [6, 8, 9, 7, 6]
List of vertices for cell 4: [8, 10, 11, 9, 8]

```

A flopy `UnstructuredGrid` object can now be created using the vertices and incidence list. The `UnstructuredGrid` object is a key part of the plotting capabilities in flopy. In addition to the vertex information, the `UnstructuredGrid` object also needs to know how many cells are in each layer. This is specified in the `ncpl` variable, which is a list of cells per layer.

```

[40]: ncpl = np.array(5 * [len(iverts)])
      umg = UnstructuredGrid(verts, iverts, xc, yc, ncpl=ncpl, angrot=10)
      print(ncpl)
      print(umg)

[218 218 218 218 218]
xll:0.0; yll:0.0; rotation:10; units:undefined; lenuni:0

```

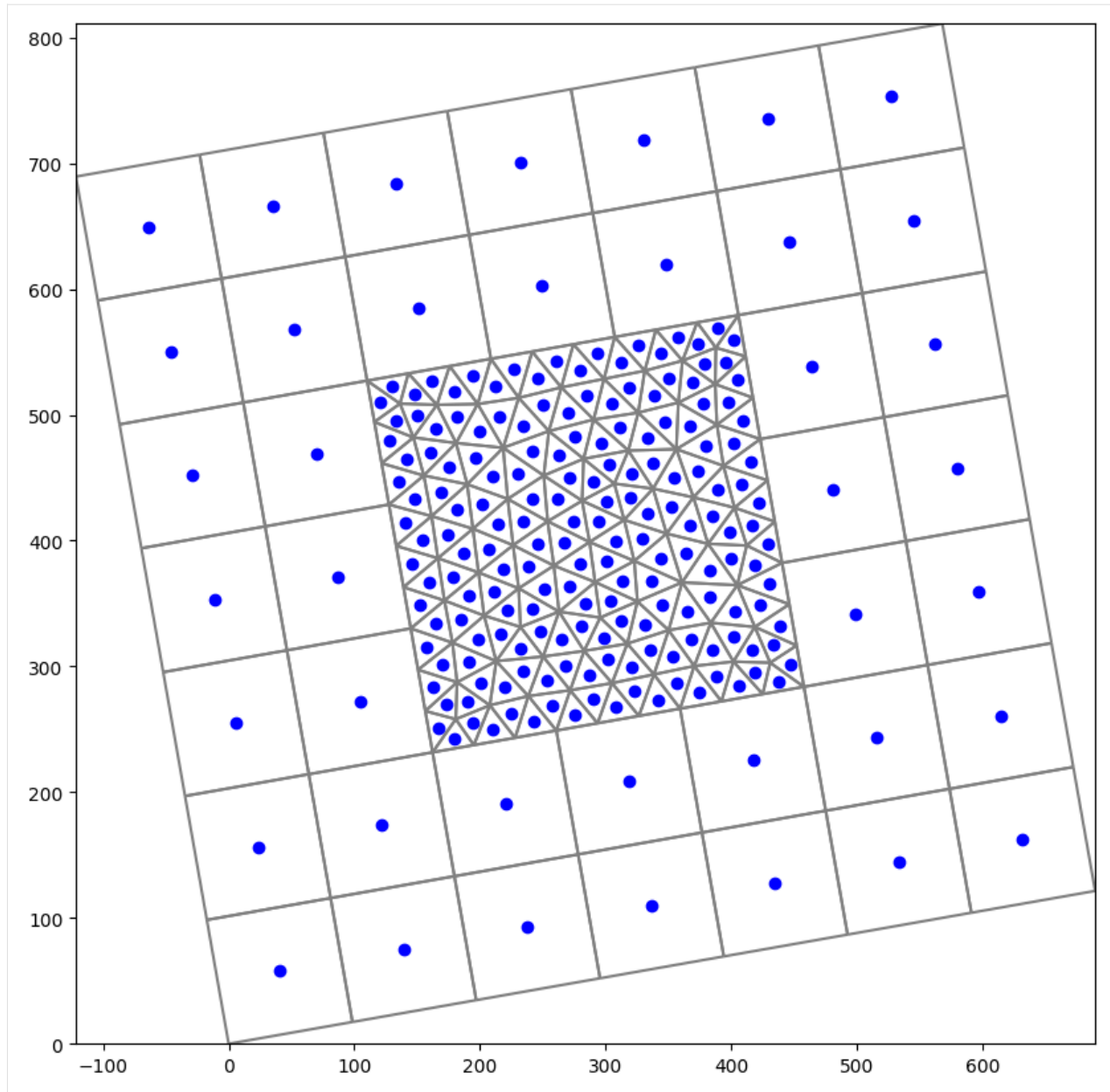
Now that we have an `UnstructuredGrid`, we can use the flopy `PlotMapView` object to create different types of plots, just like we do for structured grids.

```

[41]: f = plt.figure(figsize=(10, 10))
      mapview = flopy.plot.PlotMapView(modelgrid=umg)
      mapview.plot_grid()
      plt.plot(umg.xcellcenters, umg.ycellcenters, "bo")

[41]: [<matplotlib.lines.Line2D at 0x7f9f8cb47700>]

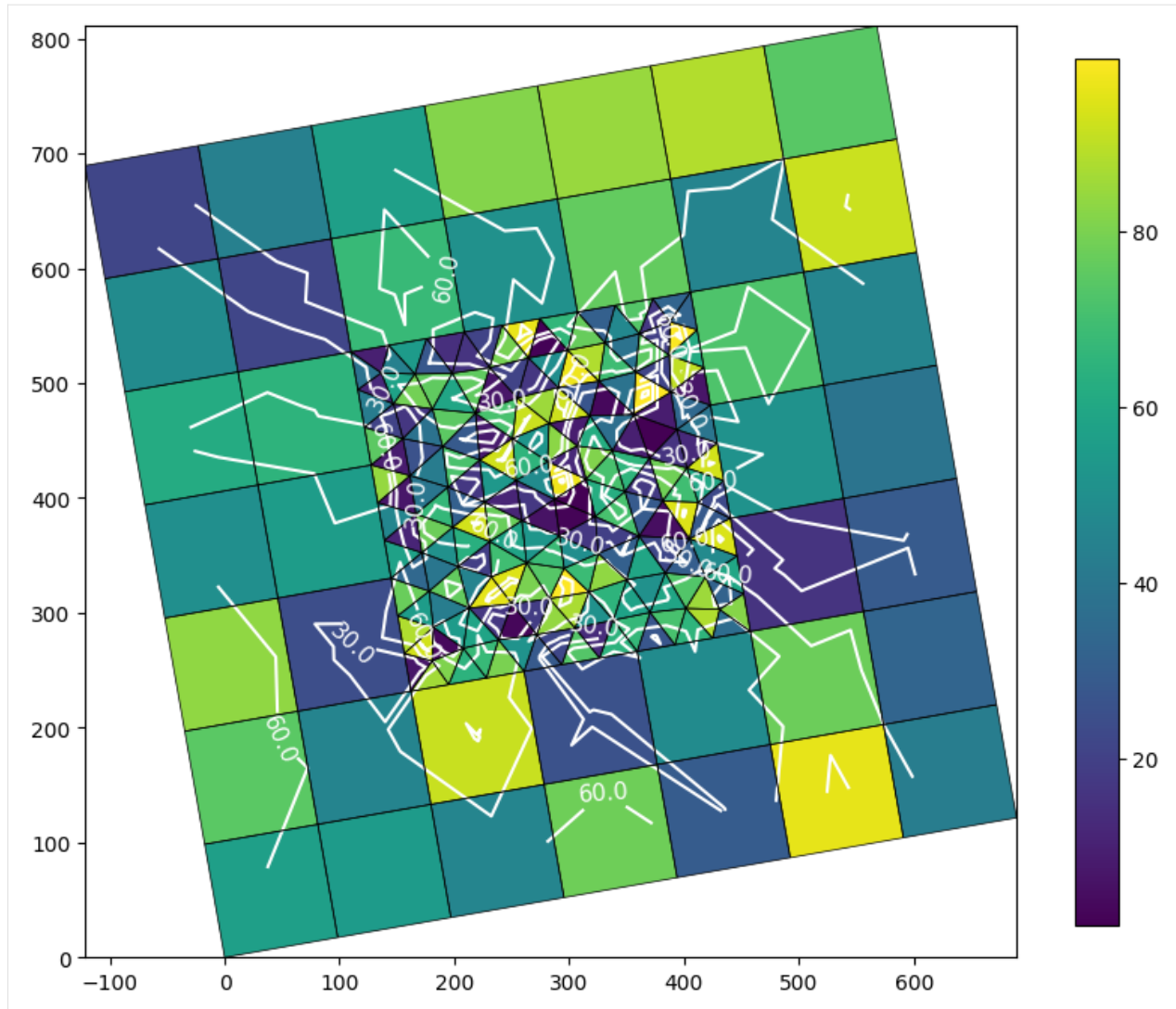
```



```
[42]: # Create a random array for layer 0, and then plot it with a color flood and contours
f = plt.figure(figsize=(10, 10))

a = np.random.random((ncpl[0])) * 100
levels = np.arange(0, 100, 30)

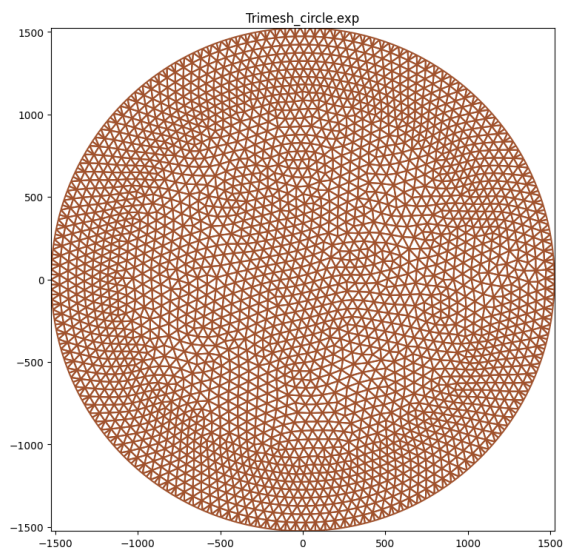
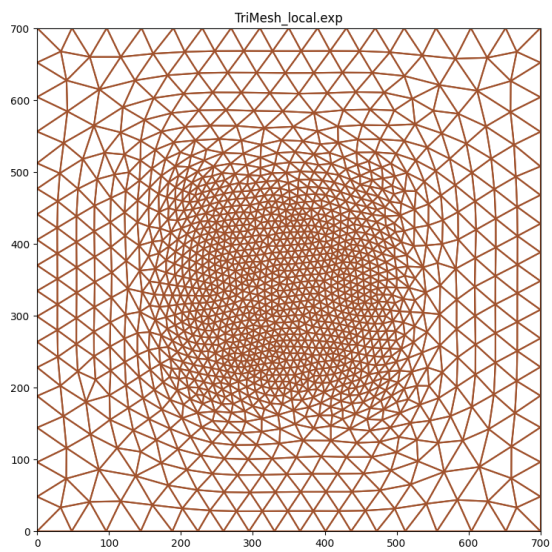
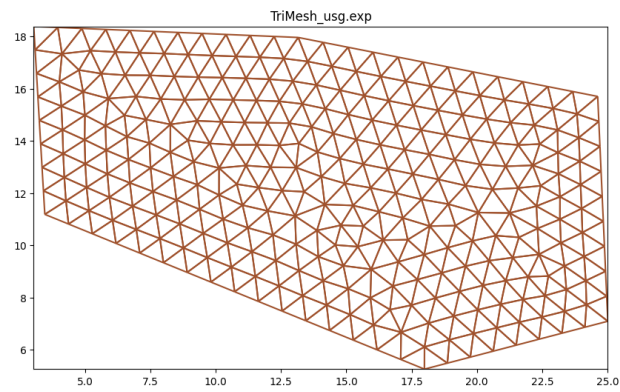
mapview = flopy.plot.PlotMapView(modelgrid=umg)
pc = mapview.plot_array(a, cmap="viridis")
contour_set = mapview.contour_array(a, levels=levels, colors="white")
plt.clabel(contour_set, fmt="%.1f", colors="white", fontsize=11)
linecollection = mapview.plot_grid(color="k", lw=0.5)
colorbar = plt.colorbar(pc, shrink=0.75)
```



Here are some examples of some other types of grids. The data files for these grids are located in the datapth folder.

```
[43]: from pathlib import Path

fig = plt.figure(figsize=(10, 30))
fnames = [fname for fname in os.listdir(datapth) if fname.endswith(".exp")]
nplot = len(fnames)
for i, f in enumerate(fnames):
    ax = fig.add_subplot(nplot, 1, i + 1, aspect="equal")
    fname = os.path.join(datapth, f)
    umga = UnstructuredGrid.from_argus_export(fname, nlay=1)
    mapview = flopy.plot.PlotMapView(modelgrid=umga, ax=ax)
    linecollection = mapview.plot_grid(colors="sienna")
    ax.set_title(Path(fname).name)
```



Plotting using built in styles

FloPy's plotting routines can be used with built in styles from the `styles` module. The `styles` module takes advantage of matplotlib's temporary styling routines by reading in pre-built style sheets. Two different types of styles have been built for flopy: `USGSMap()` and `USGSPlot()` styles which can be used to create report quality figures. The `styles` module also contains a number of methods that can be used for adding axis labels, text, annotations, headings, removing tick lines, and updating the current font.

```
[44]: # import flopy's styles
      from flopy.plot import styles

[45]: # get the specific discharge from the cell budget file
      cbc_file = os.path.join(newpth, "freyberg.cbc")
      cbc = flopy.utils.CellBudgetFile(cbc_file)
      spdis = cbc.get_data(text="SPDIS")[0]

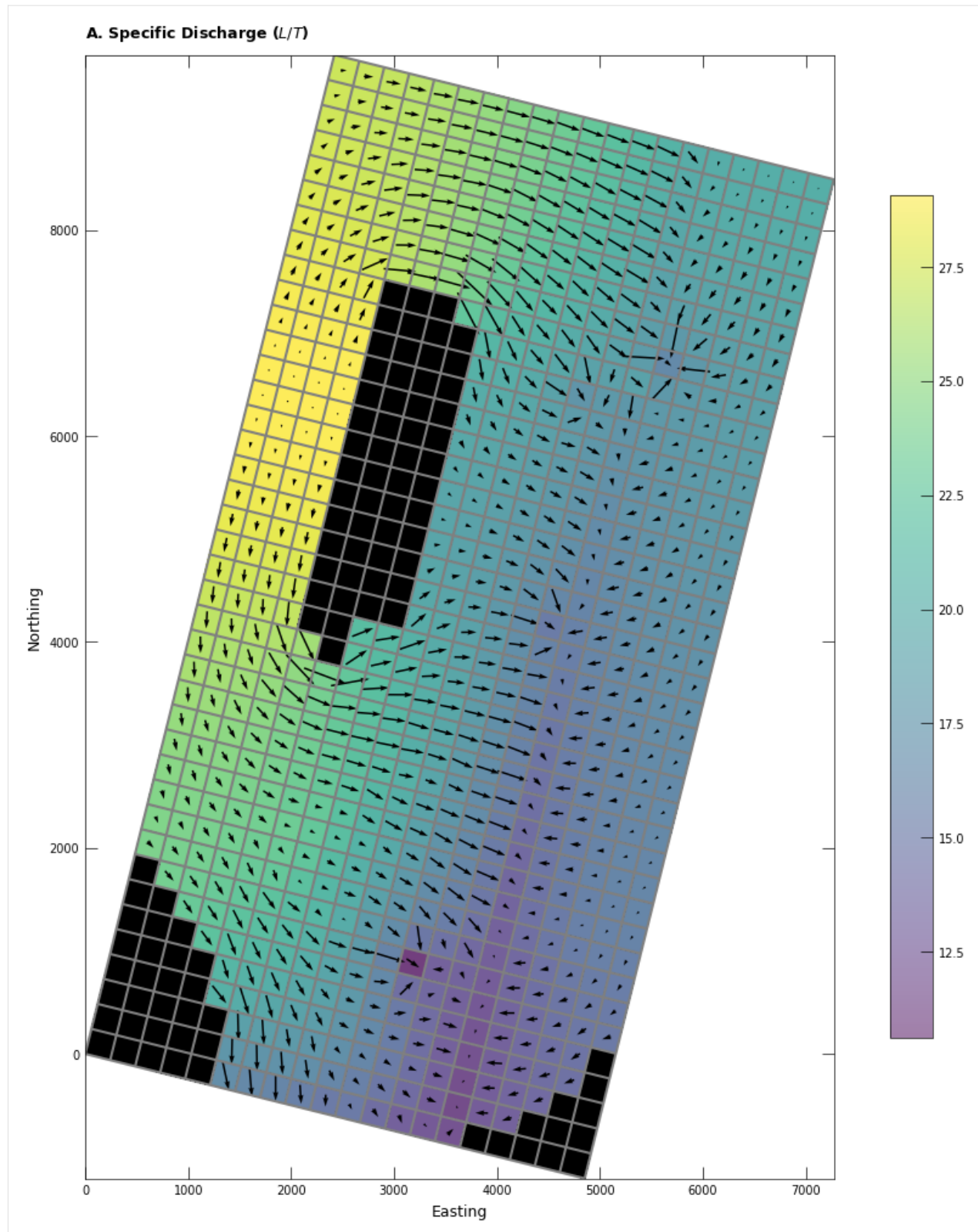
      qx, qy, qz = flopy.utils.postprocessing.get_specific_discharge(spdis, ml6)

      # get the head from the head file
      head_file = os.path.join(newpth, "freyberg.hds")
      head = flopy.utils.HeadFile(head_file)
      hdata = head.get_alldata()[0]

      # use USGSMap style to create a discharge figure:
      with styles.USGSMap():
          fig = plt.figure(figsize=(12, 12))

          mapview = flopy.plot.PlotMapView(model=ml6, layer=0)
          linecollection = mapview.plot_grid()
          quadmesh = mapview.plot_array(a=hdata, alpha=0.5)
          quiver = mapview.plot_vector(qx, qy)
          inactive = mapview.plot_inactive()
          plt.colorbar(quadmesh, shrink=0.75)

      # use styles to add a heading, xlabel, ylabel
      styles.heading(
          letter="A.", heading="Specific Discharge (" + r"$L/T$" + ")")
      )
      styles.xlabel(label="Easting")
      styles.ylabel(label="Northing")
```



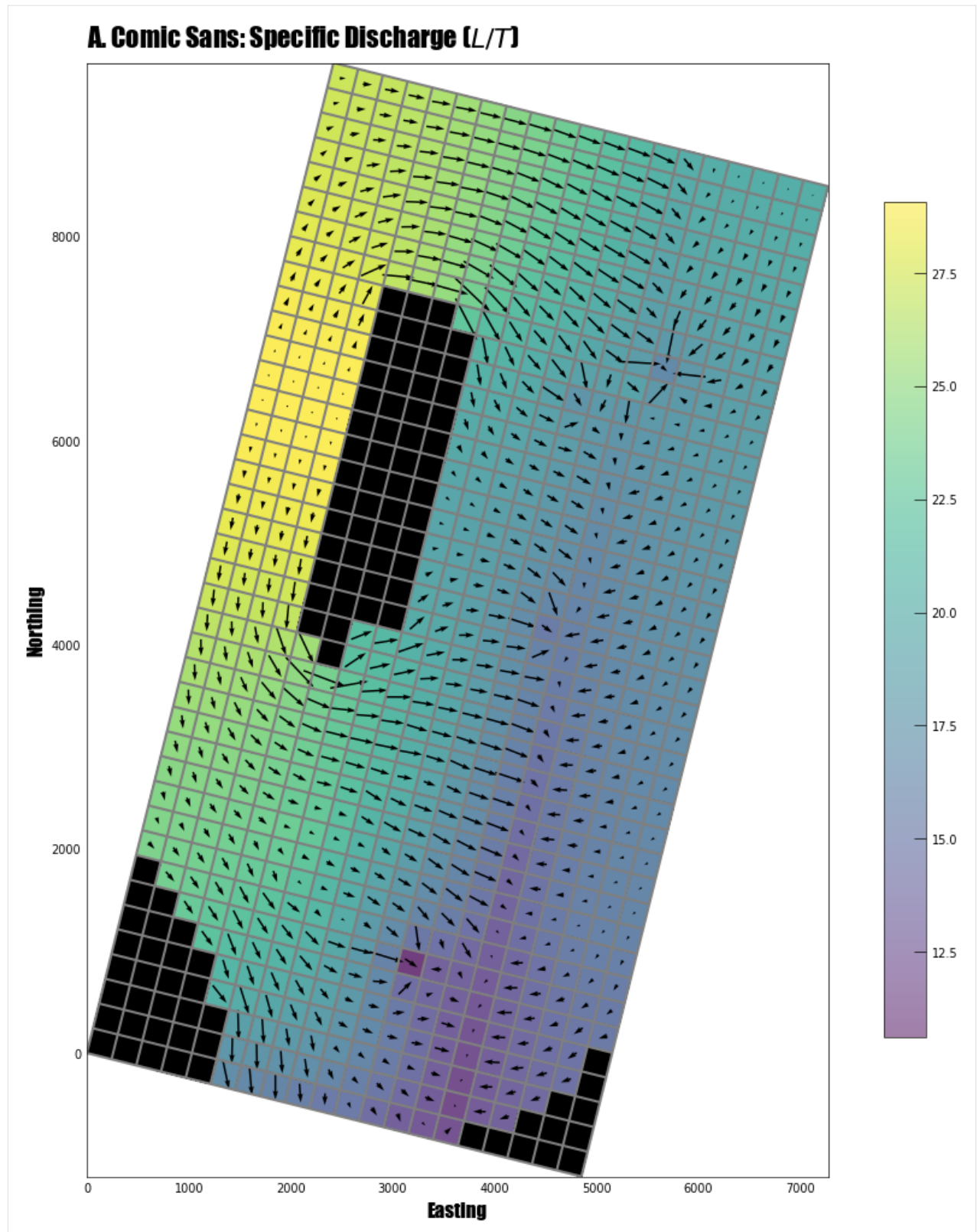
Here is a second example showing how to change the font type using `styles`


```
[46]: # use USGSMap style, change font type, and plot without tick lines:
with styles.USGSMap():
    fig = plt.figure(figsize=(12, 12))

    mapview = flopy.plot.PlotMapView(model=m16, layer=0)
    linecollection = mapview.plot_grid()
    quadmesh = mapview.plot_array(a=hdata, alpha=0.5)
    quiver = mapview.plot_vector(qx, qy)
    inactive = mapview.plot_inactive()
    plt.colorbar(quadmesh, shrink=0.75)

    # change the font type to comic sans
    styles.set_font_type(family="fantasy", fontname="Comic Sans MS"),

    # use styles to add a heading, xlabel, ylabel, and remove tick marks
    styles.heading(
        letter="A.",
        heading="Comic Sans: Specific Discharge (" + r"$L/T$" + ")",
        fontsize=16,
    )
    styles.xlabel(label="Easting", fontsize=12)
    styles.ylabel(label="Northing", fontsize=12)
    styles.remove_edge_ticks()
```



Summary

This notebook demonstrates some of the plotting functionality available with FloPy. Although not described here, the plotting functionality tries to be general by passing keyword arguments passed to `PlotMapView` methods down into the `matplotlib.pyplot` routines that do the actual plotting. For those looking to customize these plots, it may be necessary to search for the available keywords by understanding the types of objects that are created by the `PlotMapView` methods. The `PlotMapView` methods return these `matplotlib.collections` objects so that they could be fine-tuned later in the script before plotting.

Hope this gets you started!

```
[47]: try:
        # ignore PermissionError on Windows
        tempdir.cleanup()
    except:
        pass
```

6.3 Exporting data

6.3.1 Shapefile export demo

The goal of this notebook is to demonstrate ways to export model information to shapefiles. This example will cover:

- * basic exporting of information for a model, individual package, or dataset
- * custom exporting of combined data from different packages
- * general exporting and importing of geographic data from other sources

```
[1]: import os
```

```
[2]: import sys
    from tempfile import TemporaryDirectory

    import matplotlib as mpl
    import matplotlib.pyplot as plt
    import numpy as np
    import pandas as pd

    import flopy

    print(sys.version)
    print("numpy version: {}".format(np.__version__))
    print("matplotlib version: {}".format(mpl.__version__))
    print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[3]: # temporary directory
    temp_dir = TemporaryDirectory()
    outdir = os.path.join(temp_dir.name, "shapefile_export")
```

(continues on next page)

(continued from previous page)

```
# load an existing model
model_ws = "../examples/data/freyberg"
m = flopy.modflow.Modflow.load(
    "freyberg.nam",
    model_ws=model_ws,
    verbose=False,
    check=False,
    exe_name="mfnwt",
)
```

```
[4]: m.get_package_list()
```

```
[4]: ['DIS', 'BAS6', 'LPF', 'WEL', 'RIV', 'RCH', 'OC', 'PCG']
```

set the model coordinate information

the coordinate information where the grid is located in a projected coordinate system (e.g. UTM)

```
[5]: grid = m.modelgrid
grid.set_coord_info(xoff=273170, yoff=5088657, epsg=26916)
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/utils/crs.py:
↳119: PendingDeprecationWarning: the epsg argument will be deprecated and will be
↳removed in version 3.4. Use crs instead.
warnings.warn(
```

```
[6]: grid.extent
```

```
[6]: (273170.0, 278170.0, 5088657.0, 5098657.0)
```

Declarative export using attached .export() methods

Export the whole model to a single shapefile

```
[7]: fname = "{}model.shp".format(outdir)
m.export(fname)
```

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for delc array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for delr array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for laycbd array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for nstp array, DIS package
warn(
```

(continues on next page)

(continued from previous page)

```

/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for perlen array, DIS package
  warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for steady array, DIS package
  warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for tsmult array, DIS package
  warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for chani array, LPF package
  warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for layavg array, LPF package
  warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for laytyp array, LPF package
  warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for layvka array, LPF package
  warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳shapefile_utils.py:307: UserWarning: Failed to get data for laywet array, LPF package
  warn(

```

```

[8]: ax = plt.subplot(1, 1, 1, aspect="equal")
      extents = grid.extent
      pc = flopy.plot.plot_shapefile(fname, ax=ax, edgecolor="k", facecolor="none")
      ax.set_xlim(extents[0], extents[1])
      ax.set_ylim(extents[2], extents[3])
      ax.set_title(fname)

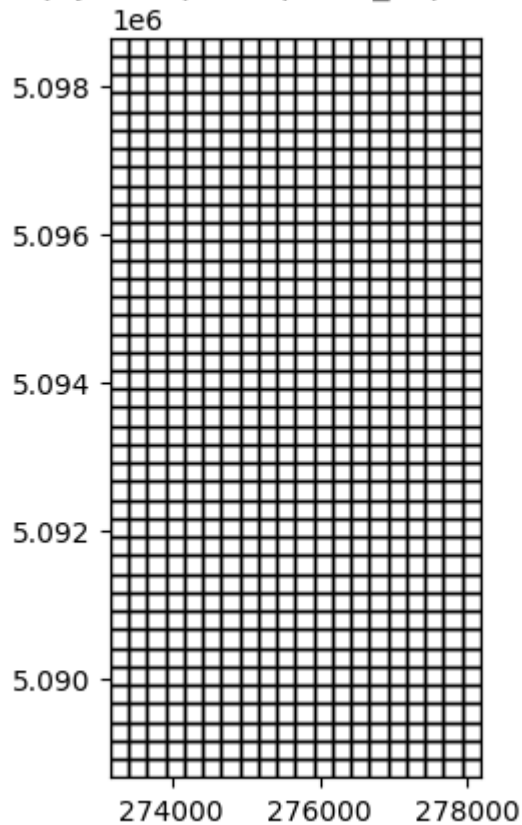
```

```

[8]: Text(0.5, 1.0, '/tmp/tmptjfs5qe/shapefile_export/model.shp')

```

/tmp/tmptjfs5qe/shapefile_export/model.shp



```
[9]: fname = "{}wel.shp".format(outdir)
     m.wel.export(fname)
```

Export a package to a shapefile

Export a FloPy list or array object

```
[10]: m.lpf.hk
```

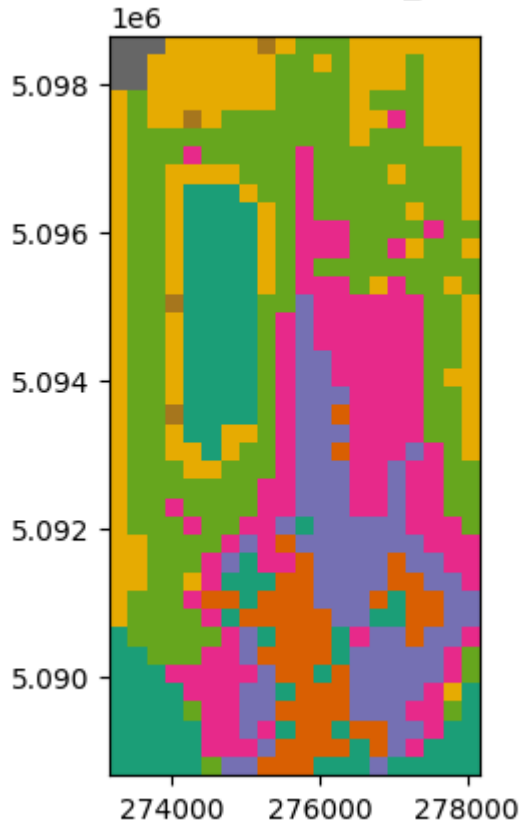
```
[10]: <flopy.utils.util_array.Util3d at 0x7ff0c656a1c0>
```

```
[11]: fname = "{}hk.shp".format(outdir)
     m.lpf.hk.export("{}hk.shp".format(outdir))
```

```
[12]: ax = plt.subplot(1, 1, 1, aspect="equal")
     extents = grid.extent
     a = m.lpf.hk.array.ravel()
     pc = flopy.plot.plot_shapefile(fname, ax=ax, a=a)
     ax.set_xlim(extents[0], extents[1])
     ax.set_ylim(extents[2], extents[3])
     ax.set_title(fname)
```

```
[12]: Text(0.5, 1.0, '/tmp/tmptjfs5qe/shapefile_export/hk.shp')
```

/tmp/tmptjfs5qe/shapefile_export/hk.shp



```
[13]: m.riv.stress_period_data
```

```
[13]: <flopy.utils.util_list.MfList at 0x7ff0c656a6a0>
```

```
[14]: m.riv.stress_period_data.export("{}riv_spd.shp".format(outdir))
```

MfList.export() exports the whole grid by default, regardless of the locations of the boundary cells

`sparse=True` only exports the boundary cells in the MfList

```
[15]: m.riv.stress_period_data.export("{}riv_spd.shp".format(outdir), sparse=True)
```

```
wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks
```

```
[16]: m.wel.stress_period_data.export("{}wel_spd.shp".format(outdir), sparse=True)
```

```
wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks
```

Ad-hoc exporting using recarray2shp

- The main idea is to create a recarray with all of the attribute information, and a list of geometry features (one feature per row in the recarray)
- each geometry feature is an instance of the `Point`, `LineString` or `Polygon` classes in `flopy.utils.geometry`. The shapefile format requires all the features to be of the same type.
- We will use pandas dataframes for these examples because they are easy to work with, and then convert them to recarrays prior to exporting.

```
[17]: from flopy.export.shapefile_utils import recarray2shp
```

combining data from different packages

write a shapefile of RIV and WEL package cells

```
[18]: wellspd = pd.DataFrame(m.wel.stress_period_data[0])
rivspd = pd.DataFrame(m.riv.stress_period_data[0])
spd = pd.concat([wellspd, rivspd])
spd.head()
```

```
[18]:
```

	k	i	j	flux	iface	stage	cond	rbot
0	0	8	15	-0.00820	0.0	NaN	NaN	NaN
1	0	10	12	-0.00410	0.0	NaN	NaN	NaN
2	0	19	13	-0.00390	0.0	NaN	NaN	NaN
3	0	25	9	-0.00083	0.0	NaN	NaN	NaN
4	0	28	5	-0.00072	0.0	NaN	NaN	NaN

```
[19]: from flopy.utils.geometry import Polygon
```

```
vertices = []
for row, col in zip(spd.i, spd.j):
    vertices.append(grid.get_cell_vertices(row, col))
polygons = [Polygon(vrt for vrt in vertices)]
polygons
```

```
[19]: [<flopy.utils.geometry.Polygon at 0x7ff0c54bde80>,
<flopy.utils.geometry.Polygon at 0x7ff0c54ad730>,
<flopy.utils.geometry.Polygon at 0x7ff0c54ade50>,
<flopy.utils.geometry.Polygon at 0x7ff0c54ad160>,
<flopy.utils.geometry.Polygon at 0x7ff0c54ad940>,
<flopy.utils.geometry.Polygon at 0x7ff0c54ad3d0>,
<flopy.utils.geometry.Polygon at 0x7ff0c54adbb0>,
<flopy.utils.geometry.Polygon at 0x7ff0c54adee0>,
<flopy.utils.geometry.Polygon at 0x7ff0c54adc40>,
<flopy.utils.geometry.Polygon at 0x7ff0c54ad6a0>,
<flopy.utils.geometry.Polygon at 0x7ff0c54ada00>,
<flopy.utils.geometry.Polygon at 0x7ff0c54ad070>,
<flopy.utils.geometry.Polygon at 0x7ff0c54ada90>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299070>,
<flopy.utils.geometry.Polygon at 0x7ff0c32990d0>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299220>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299340>,
```

(continues on next page)

(continued from previous page)

```

<flopy.utils.geometry.Polygon at 0x7ff0c3299310>,
<flopy.utils.geometry.Polygon at 0x7ff0c32994f0>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299580>,
<flopy.utils.geometry.Polygon at 0x7ff0c32996a0>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299730>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299700>,
<flopy.utils.geometry.Polygon at 0x7ff0c32998e0>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299820>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299a00>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299a60>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299b20>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299b80>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299c40>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299ca0>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299d60>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299df0>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299dc0>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299f10>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299fa0>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299160>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299040>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299670>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299790>,
<flopy.utils.geometry.Polygon at 0x7ff0c3299940>,
<flopy.utils.geometry.Polygon at 0x7ff0c32b72b0>,
<flopy.utils.geometry.Polygon at 0x7ff0c32b7340>,
<flopy.utils.geometry.Polygon at 0x7ff0c32b7430>,
<flopy.utils.geometry.Polygon at 0x7ff0c32b74c0>,
<flopy.utils.geometry.Polygon at 0x7ff0c32b7550>]

```

```

[20]: fname = "{}bcs.shp".format(outdir)
recarray2shp(spd.to_records(), geoms=polygons, shpname=fname, epsg=grid.epsg)

wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks

/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/utils/crs.py:
↳ 119: PendingDeprecationWarning: the epsg argument will be deprecated and will be
↳ removed in version 3.4. Use crs instead.
warnings.warn(

```

```

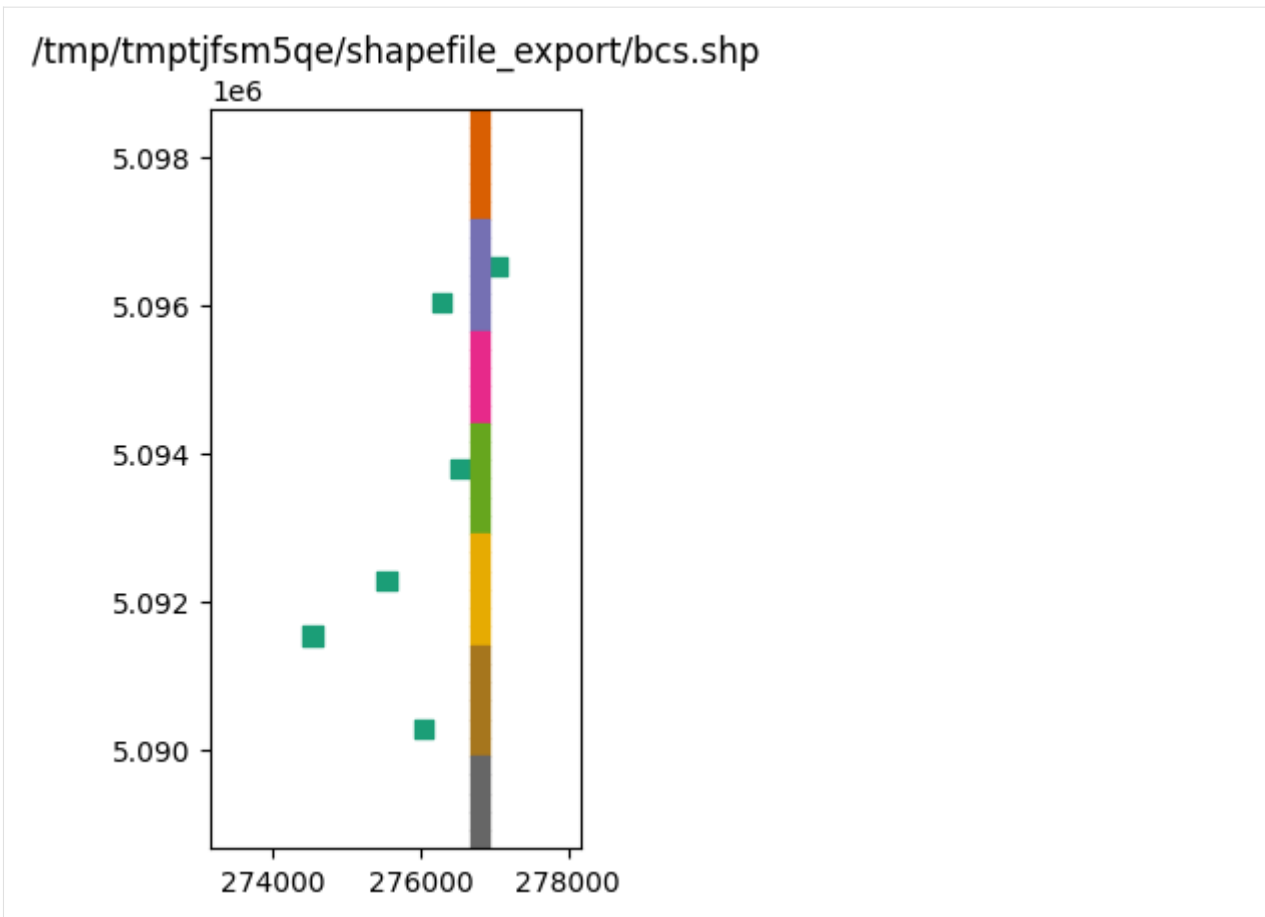
[21]: ax = plt.subplot(1, 1, 1, aspect="equal")
extents = grid.extent
pc = flopy.plot.plot_shapefile(fname, ax=ax)
ax.set_xlim(extents[0], extents[1])
ax.set_ylim(extents[2], extents[3])
ax.set_title(fname)

```

```

[21]: Text(0.5, 1.0, '/tmp/tmptjfs5qe/shapefile_export/bcs.shp')

```



exporting other data

Suppose we have some well data with actual locations that we want to export to a shapefile

```
[22]: welldata = pd.DataFrame(
    {
        "wellID": np.arange(0, 10),
        "q": np.random.randn(10) * 100 - 1000,
        "x_utm": np.random.rand(10) * 5000 + grid.xoffset,
        "y_utm": grid.yoffset + np.random.rand(10) * 10000,
    }
)
welldata.head()
```

```
[22]:
```

	wellID	q	x_utm	y_utm
0	0	-1092.359884	276660.473089	5.095095e+06
1	1	-974.519847	276306.978258	5.093062e+06
2	2	-889.823255	277969.021802	5.096678e+06
3	3	-1062.366654	277663.656335	5.090236e+06
4	4	-968.146430	277554.297277	5.090216e+06

```
[23]: from flopy.utils.geometry import Point
```

(continues on next page)

(continued from previous page)

```
geoms = [Point(x, y) for x, y in zip(welldata.x_utm, welldata.y_utm)]

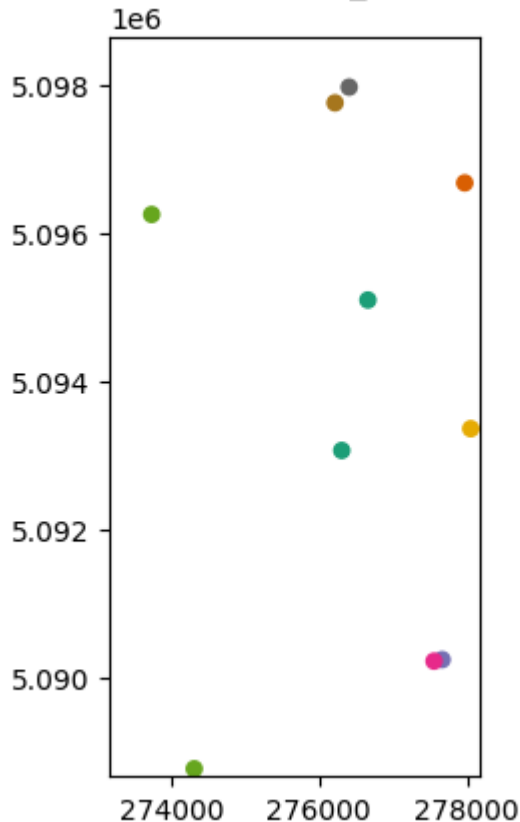
fname = "{}wel_data.shp".format(outdir)
recarray2shp(welldata.to_records(), geoms=geoms, shpname=fname, epsg=grid.epsg)

wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/utis/crs.py:
↳119: PendingDeprecationWarning: the epsg argument will be deprecated and will be
↳removed in version 3.4. Use crs instead.
warnings.warn(
```

```
[24]: ax = plt.subplot(1, 1, 1, aspect="equal")
      extents = grid.extent
      pc = flopy.plot.plot_shapefile(fname, ax=ax, radius=100)
      ax.set_xlim(extents[0], extents[1])
      ax.set_ylim(extents[2], extents[3])
      ax.set_title(fname)
```

```
[24]: Text(0.5, 1.0, '/tmp/tmptjfs5qe/shapefile_export/wel_data.shp')
```

/tmp/tmptjfs5qe/shapefile_export/wel_data.shp



Adding attribute data to an existing shapefile

Suppose we have a GIS coverage representing the river in the riv package

```
[25]: from flopy.utils.geometry import LineString

### make up a linestring shapefile of the river reaches
i, j = m.riv.stress_period_data[0].i, m.riv.stress_period_data[0].j
x0 = grid.xyzcellcenters[0][i[0], j[0]]
x1 = grid.xyzcellcenters[0][i[-1], j[-1]]
y0 = grid.xyzcellcenters[1][i[0], j[0]]
y1 = grid.xyzcellcenters[1][i[-1], j[-1]]
x = np.linspace(x0, x1, m.nrow + 1)
y = np.linspace(y0, y1, m.nrow + 1)
l0 = zip(list(zip(x[:-1], y[:-1])), list(zip(x[1:], y[1:])))
lines = [LineString(l) for l in l0]

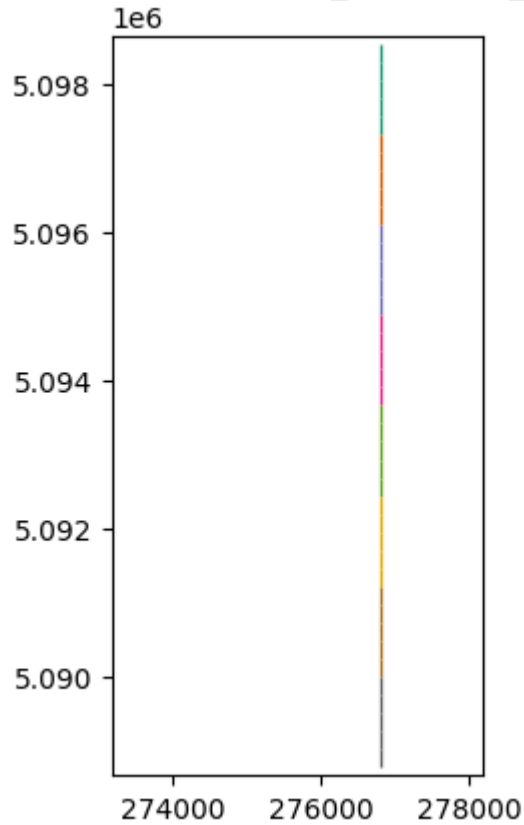
rivdata = pd.DataFrame(m.riv.stress_period_data[0])
rivdata["reach"] = np.arange(len(lines))
lines_shapefile = "{}riv_reaches.shp".format(outdir)
recarray2shp(
    rivdata.to_records(index=False),
    geoms=lines,
    shpname=lines_shapefile,
    epsg=grid.epsg,
)

wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/utils/crs.py:
↳119: PendingDeprecationWarning: the epsg argument will be deprecated and will be
↳removed in version 3.4. Use crs instead.
warnings.warn(

[26]: ax = plt.subplot(1, 1, 1, aspect="equal")
      extents = grid.extent
      pc = flopy.plot.plot_shapefile(lines_shapefile, ax=ax, radius=25)
      ax.set_xlim(extents[0], extents[1])
      ax.set_ylim(extents[2], extents[3])
      ax.set_title(lines_shapefile)

[26]: Text(0.5, 1.0, '/tmp/tmptjfs5qe/shapefile_export/riv_reaches.shp')
```

/tmp/tmptjfs5qe/shapefile_export/riv_reaches.shp



shp2recarray reads a shapefile into a numpy record array, which can easily be converted to a DataFrame

```
[27]: from flopy.export.shapefile_utils import shp2recarray
```

```
[28]: linesdata = shp2recarray(lines_shapefile)
linesdata = pd.DataFrame(linesdata)
linesdata.head()
```

```
[28]:
```

	k	i	j	stage	cond	rbot	iface	reach	\
0	0	0	14	20.100000	0.05	20.00	0.0	0	
1	0	1	14	19.870001	0.05	19.75	0.0	1	
2	0	2	14	19.650000	0.05	19.50	0.0	2	
3	0	3	14	19.420000	0.05	19.25	0.0	3	
4	0	4	14	19.190001	0.05	19.00	0.0	4	


```

                                geometry
0  <flopy.utils.geometry.LineString object at 0x7...
1  <flopy.utils.geometry.LineString object at 0x7...
2  <flopy.utils.geometry.LineString object at 0x7...
3  <flopy.utils.geometry.LineString object at 0x7...
4  <flopy.utils.geometry.LineString object at 0x7...

```

```
[29]: # make up some fluxes between the river and aquifer at each reach
q = np.random.randn(len(linesdata)) + 1
```

(continues on next page)

(continued from previous page)

q

```
[29]: array([-0.18403005,  2.35045997, -0.21018577,  2.56692767,  2.3528112 ,
          1.05254341,  2.98830938,  1.34309542,  0.99476571,  1.11428521,
          0.94711756,  1.24274964,  0.98880093, -1.35575662, -0.08662407,
          1.58616024, -0.03223463,  1.41355697, -0.52118678,  1.81407137,
          2.24146616,  1.20358632,  0.60193288,  1.76019454, -0.8627548 ,
          0.18114315, -0.49923485,  0.15026523, -0.19594543,  0.87682404,
          -0.33079539, -0.31925346,  1.1363268 , -0.23418252,  1.03831157,
          2.35007046,  0.90669645,  1.42527292, -0.14586778,  1.52432627])
```

```
[30]: linesdata["qreach"] = q
      linesdata["qstream"] = np.cumsum(q)
```

```
[31]: recarray2shp(
      linesdata.drop("geometry", axis=1).to_records(),
      geoms=linesdata.geometry.values,
      shpname=lines_shapefile,
      epsg=grid.epsg,
      )
```

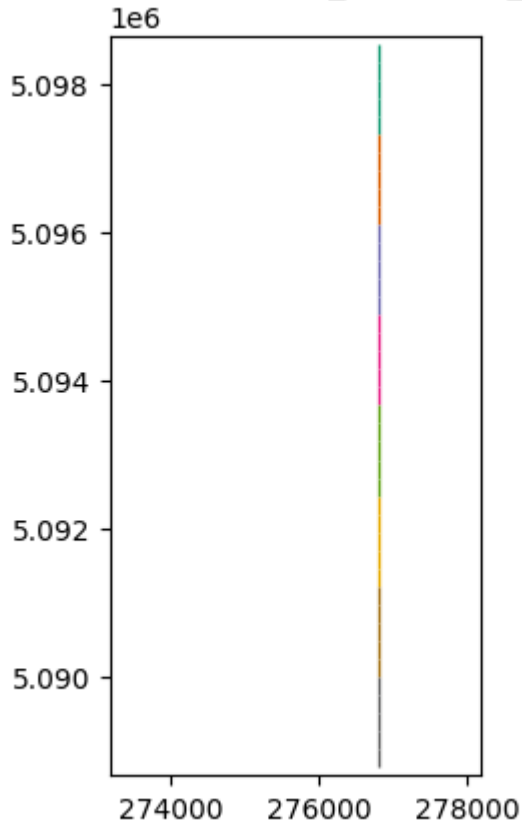
wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/utils/crs.py:
↳119: PendingDeprecationWarning: the epsg argument will be deprecated and will be
↳removed in version 3.4. Use crs instead.
      warnings.warn(
```

```
[32]: ax = plt.subplot(1, 1, 1, aspect="equal")
      extents = grid.extent
      pc = flopy.plot.plot_shapefile(lines_shapefile, ax=ax, radius=25)
      ax.set_xlim(extents[0], extents[1])
      ax.set_ylim(extents[2], extents[3])
      ax.set_title(lines_shapefile)
```

```
[32]: Text(0.5, 1.0, '/tmp/tmptjfs5qe/shapefile_export/riv_reaches.shp')
```

/tmp/tmptjfs5qe/shapefile_export/riv_reaches.shp



Overriding the model's modelgrid with a user supplied modelgrid

In some cases it may be necessary to override the model's modelgrid instance with a separate modelgrid. An example of this is if the model discretization is in feet and the user would like it projected in meters. Exporting can be accomplished by supplying a modelgrid as a kwarg in any of the `export()` methods within flopy. Below is an example:

```
[33]: mg0 = m.modelgrid

# build a new modelgrid instance with discretization in meters
modelgrid = flopy.discretization.StructuredGrid(
    delc=mg0.delc * 0.3048,
    delr=mg0.delr * 0.3048,
    top=mg0.top,
    botm=mg0.botm,
    idomain=mg0.idomain,
    xoff=mg0.xoffset * 0.3048,
    yoff=mg0.yoffset * 0.3048,
)

# exporting an entire model
m.export("{}freiberg.shp".format(outdir), modelgrid=modelgrid)
```

No CRS information for writing a .prj file.
 Supply an valid coordinate system reference to the attached modelgrid object or .
 ↪ export() method.

No CRS information for writing a .prj file.
 Supply an valid coordinate system reference to the attached modelgrid object or .
 ↪ export() method.

```
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for delc array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for delr array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for laycbd array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for nstp array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for perlen array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for steady array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for tsmult array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for chani array, LPF package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for layavg array, LPF package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for laytyp array, LPF package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for layvka array, LPF package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪ shapefile_utils.py:307: UserWarning: Failed to get data for laywet array, LPF package
warn(
```

And for a specific parameter the method is the same

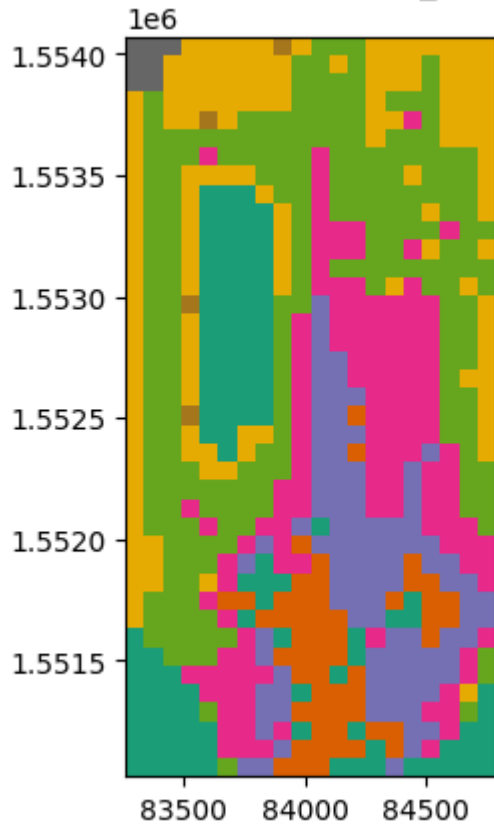
```
[34]: fname = "{}hk.shp".format(outdir)
m.lpf.hk.export(fname, modelgrid=modelgrid)
```

No CRS information for writing a .prj file.
 Supply an valid coordinate system reference to the attached modelgrid object or .
 ↪ export() method.


```
[35]: ax = plt.subplot(1, 1, 1, aspect="equal")
      extents = modelgrid.extent
      a = m.lpf.hk.array.ravel()
      pc = flopy.plot.plot_shapefile(fname, ax=ax, a=a)
      ax.set_xlim(extents[0], extents[1])
      ax.set_ylim(extents[2], extents[3])
      ax.set_title(fname)
```

```
[35]: Text(0.5, 1.0, '/tmp/tmptjfs5qe/shapefile_export/hk.shp')
```

/tmp/tmptjfs5qe/shapefile_export/hk.shp



```
[36]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.4 Other FloPy features

6.4.1 FloPy working stack demo

A short demonstration of core flopy functionality

```
[1]: import sys
    from pathlib import Path
    from tempfile import TemporaryDirectory

[2]: import matplotlib as mpl
    import matplotlib.pyplot as plt
    import numpy as np
    import pandas as pd

[3]: from IPython.display import clear_output, display

proj_root = Path.cwd().parent.parent

# run installed version of flopy or add local path
try:
    import flopy
except:
    sys.path.append(proj_root)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("pandas version: {}".format(pd.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
pandas version: 2.0.3
flopy version: 3.4.3
```

Model Inputs

```
[4]: # first lets load an existing model
model_ws = proj_root / "examples" / "data" / "freyberg_multilayer_transient"
ml = flopy.modflow.Modflow.load(
    "freyberg.nam",
    model_ws=model_ws,
    verbose=False,
    check=False,
    exe_name="mfnwt",
)
```

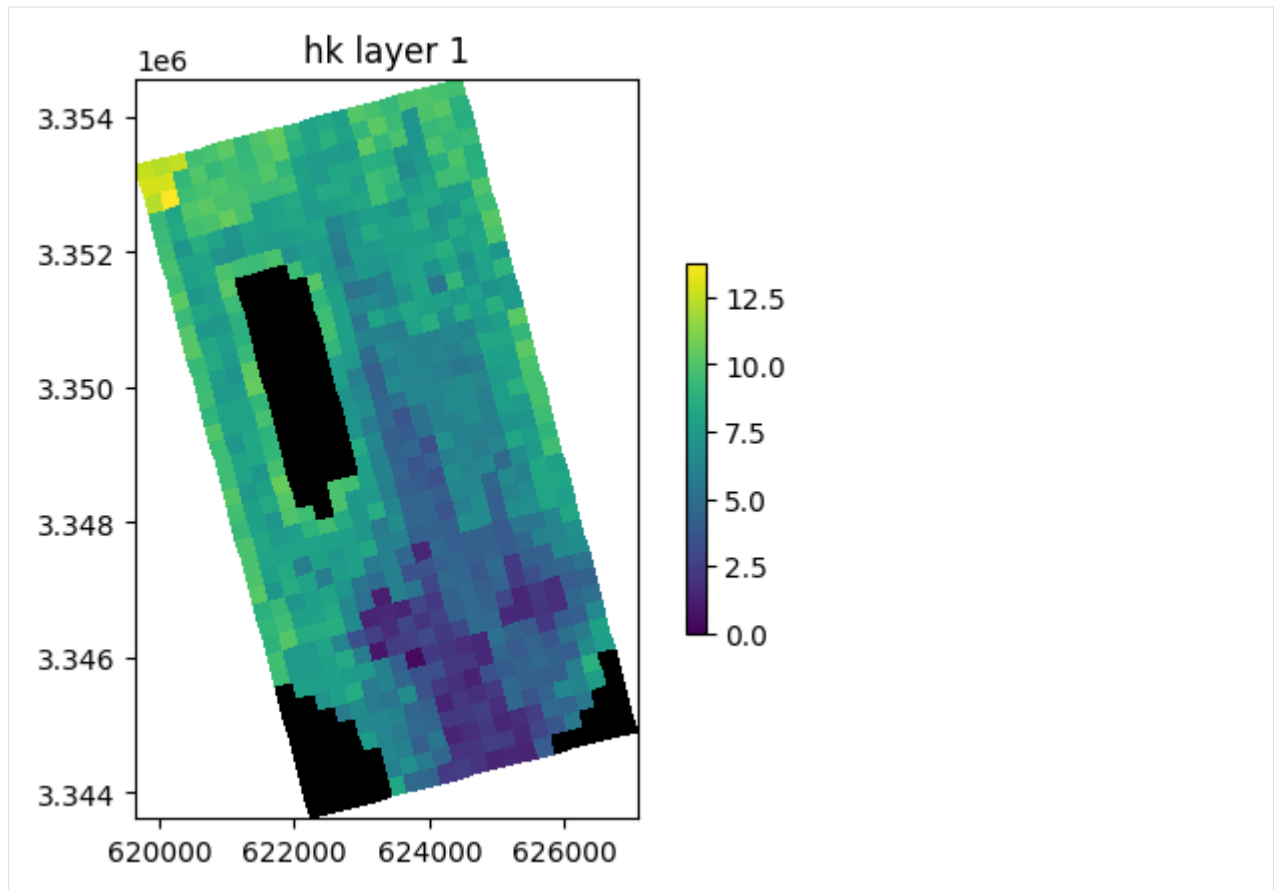
```
[5]: ml.modelgrid
```

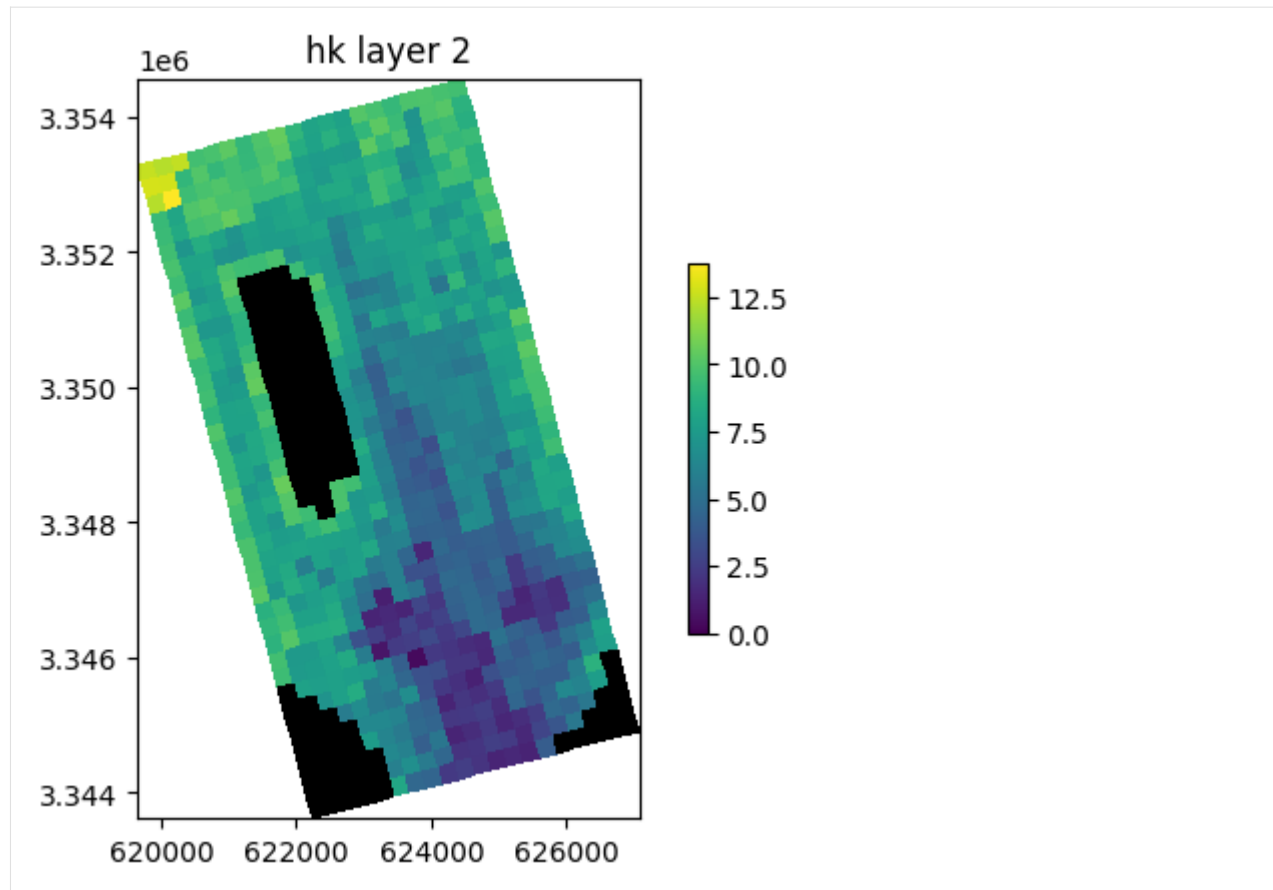
```
[5]: xll:622241.1904510253; yll:3343617.741737109; rotation:15.0; crs:EPSG:32614; units:
      ↪meters; lenuni:2
```

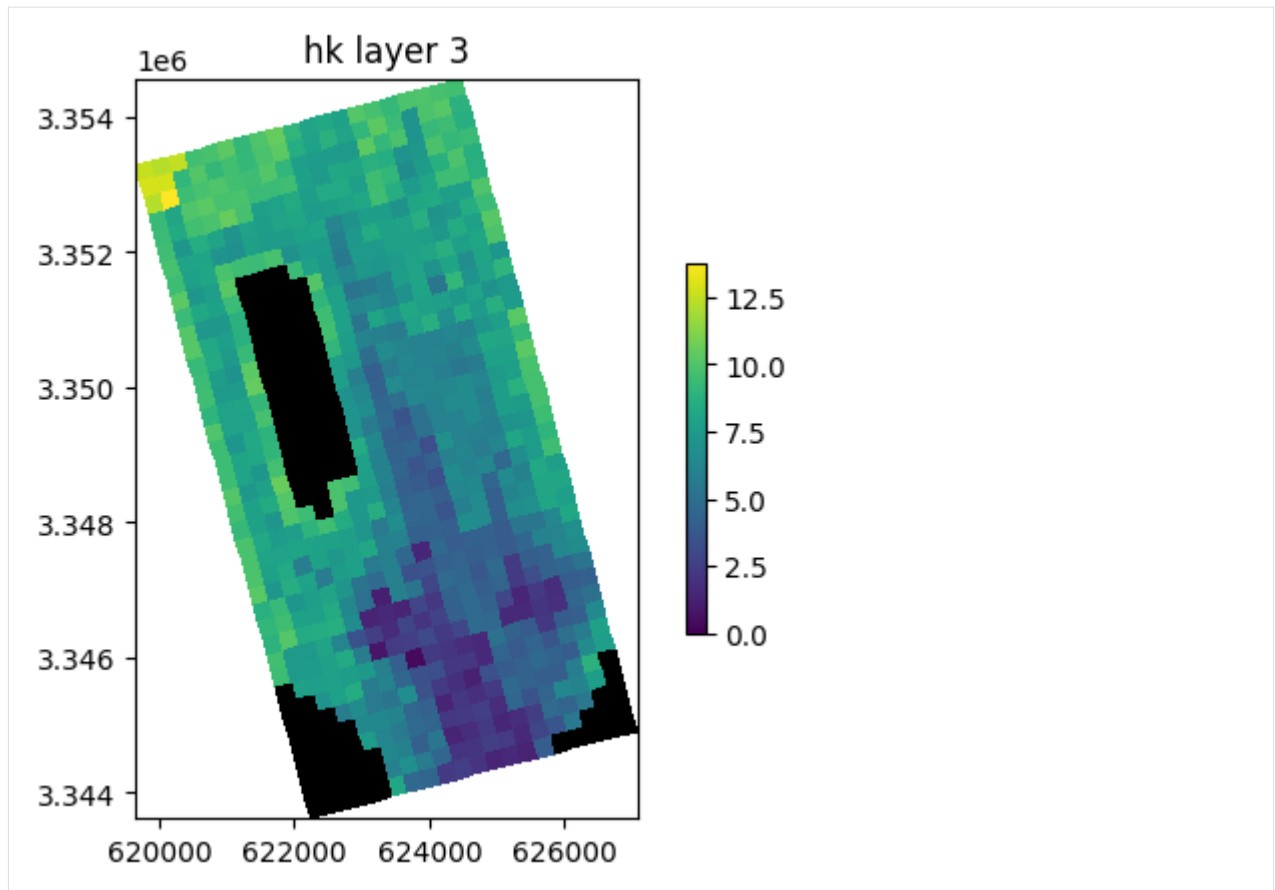
Let's look at some plots

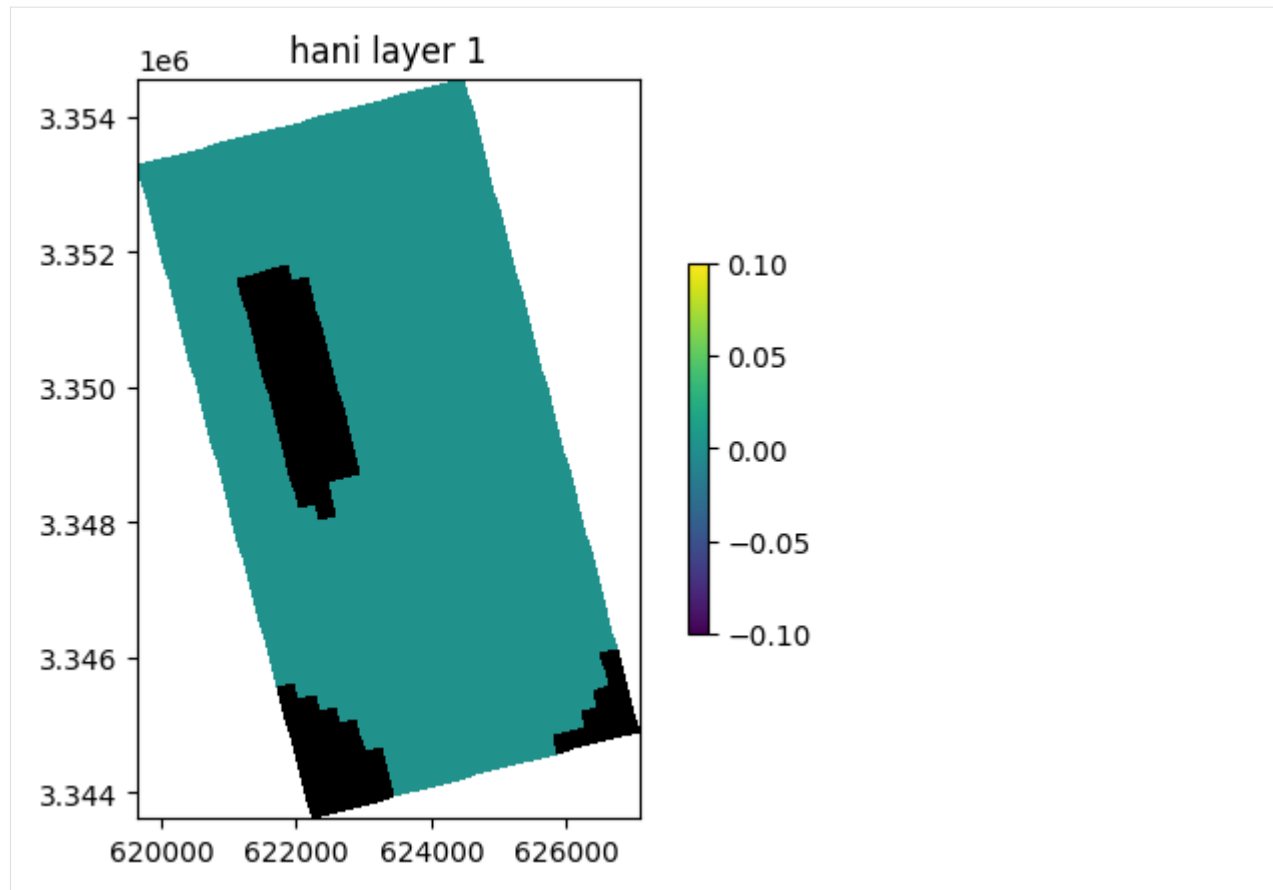
```
[6]: ml.upw.plot()
```

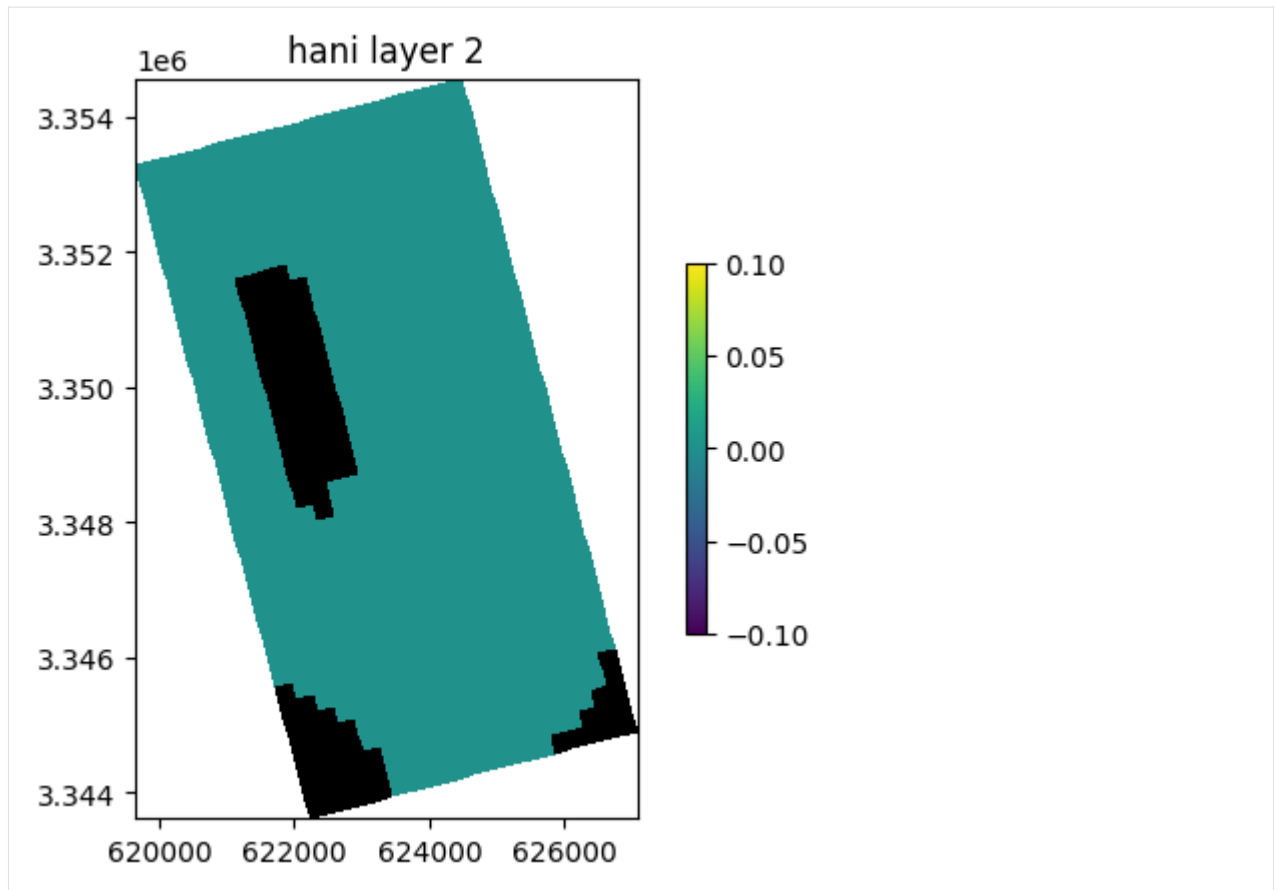
```
[6]: [<Axes: title={'center': 'hk layer 1'}>,
      <Axes: title={'center': 'hk layer 2'}>,
      <Axes: title={'center': 'hk layer 3'}>,
      <Axes: title={'center': 'hani layer 1'}>,
      <Axes: title={'center': 'hani layer 2'}>,
      <Axes: title={'center': 'hani layer 3'}>,
      <Axes: title={'center': 'vka layer 1'}>,
      <Axes: title={'center': 'vka layer 2'}>,
      <Axes: title={'center': 'vka layer 3'}>,
      <Axes: title={'center': 'ss layer 1'}>,
      <Axes: title={'center': 'ss layer 2'}>,
      <Axes: title={'center': 'ss layer 3'}>,
      <Axes: title={'center': 'sy layer 1'}>,
      <Axes: title={'center': 'sy layer 2'}>,
      <Axes: title={'center': 'sy layer 3'}>,
      <Axes: title={'center': 'vkcb layer 1'}>,
      <Axes: title={'center': 'vkcb layer 2'}>,
      <Axes: title={'center': 'vkcb layer 3'}>]
```

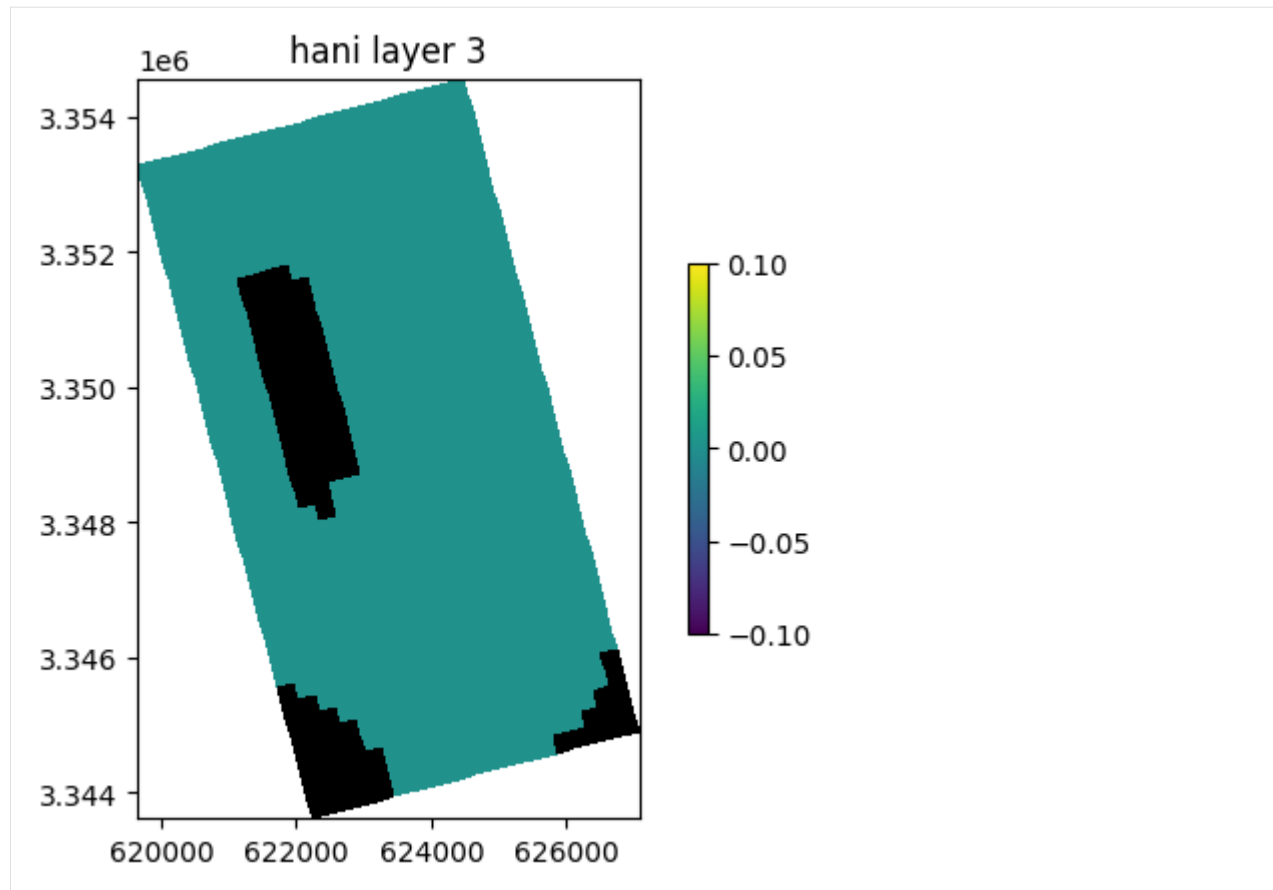


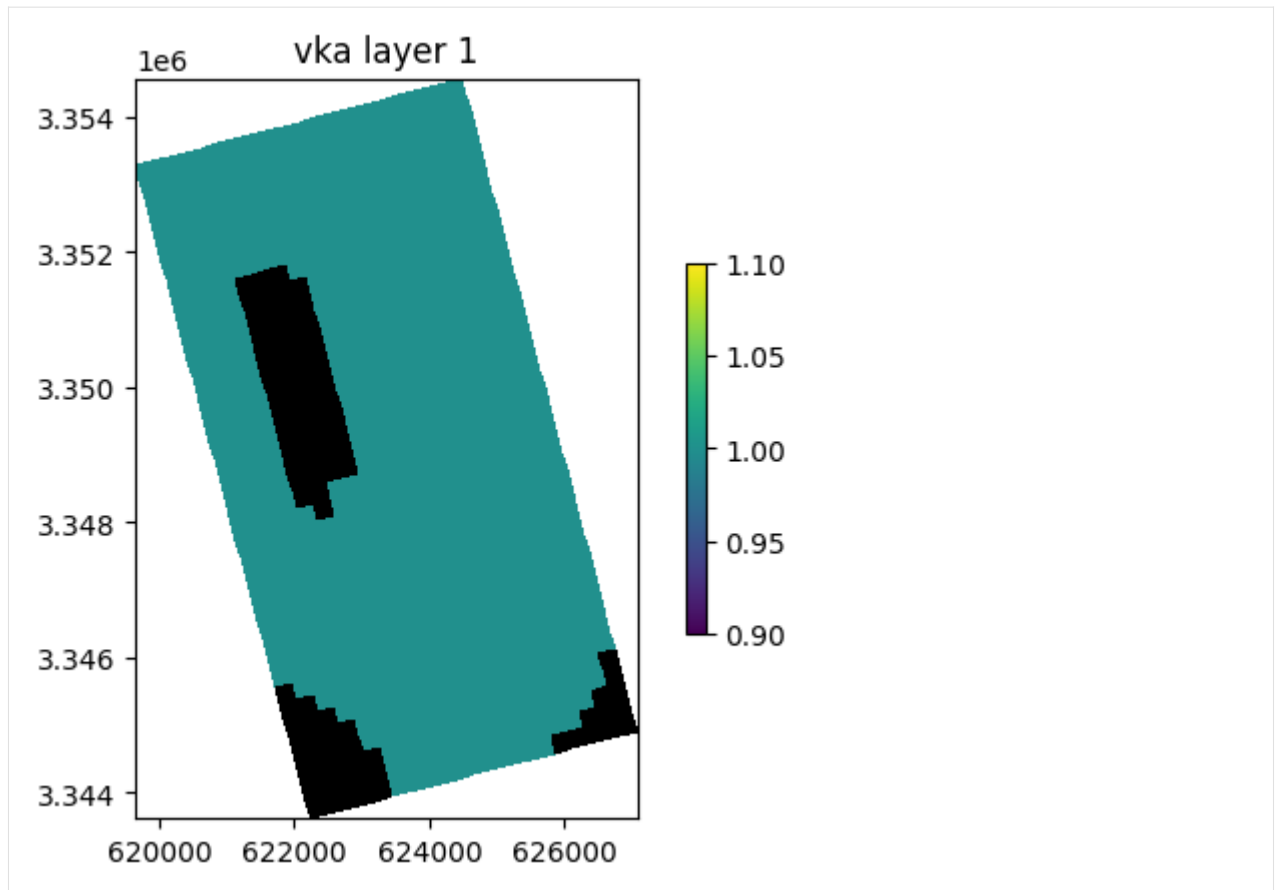


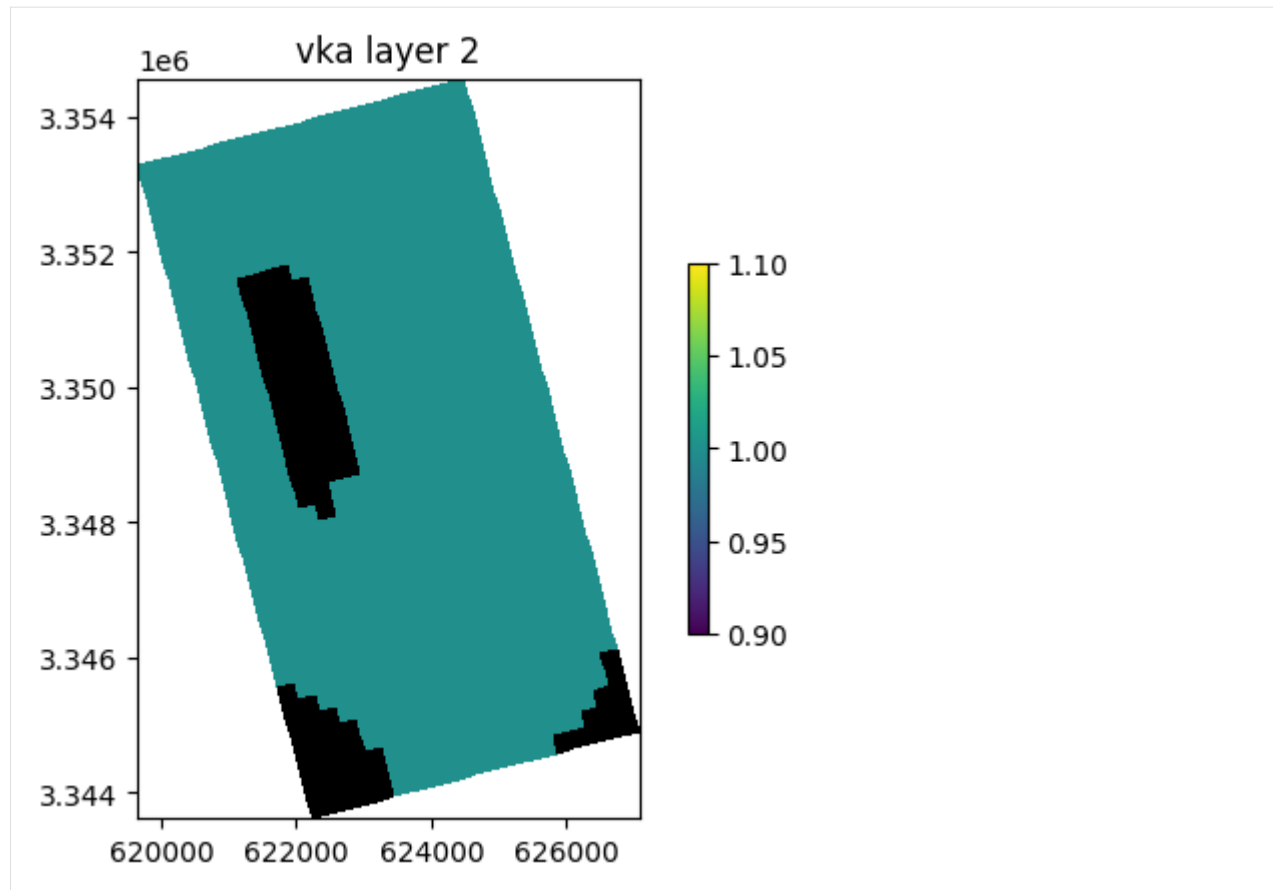


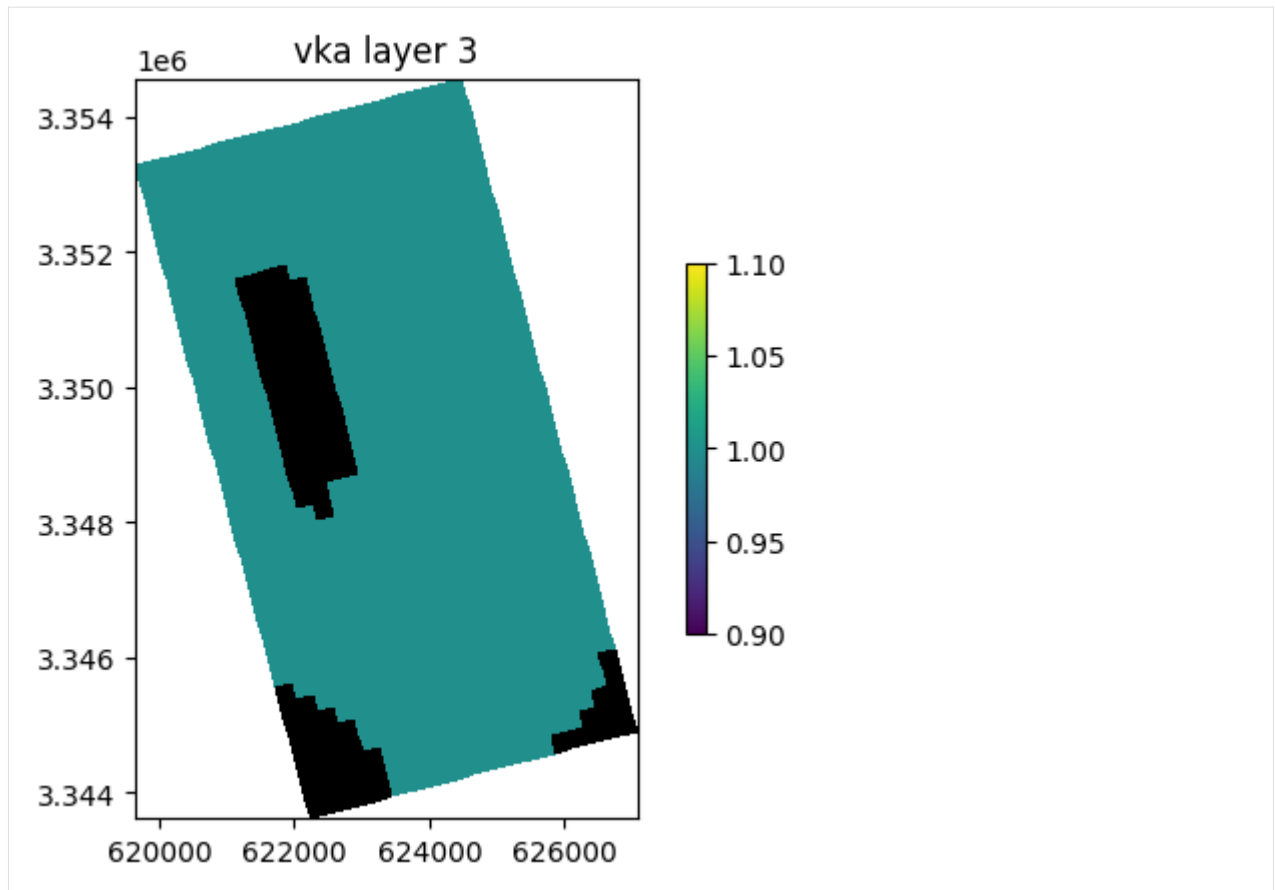


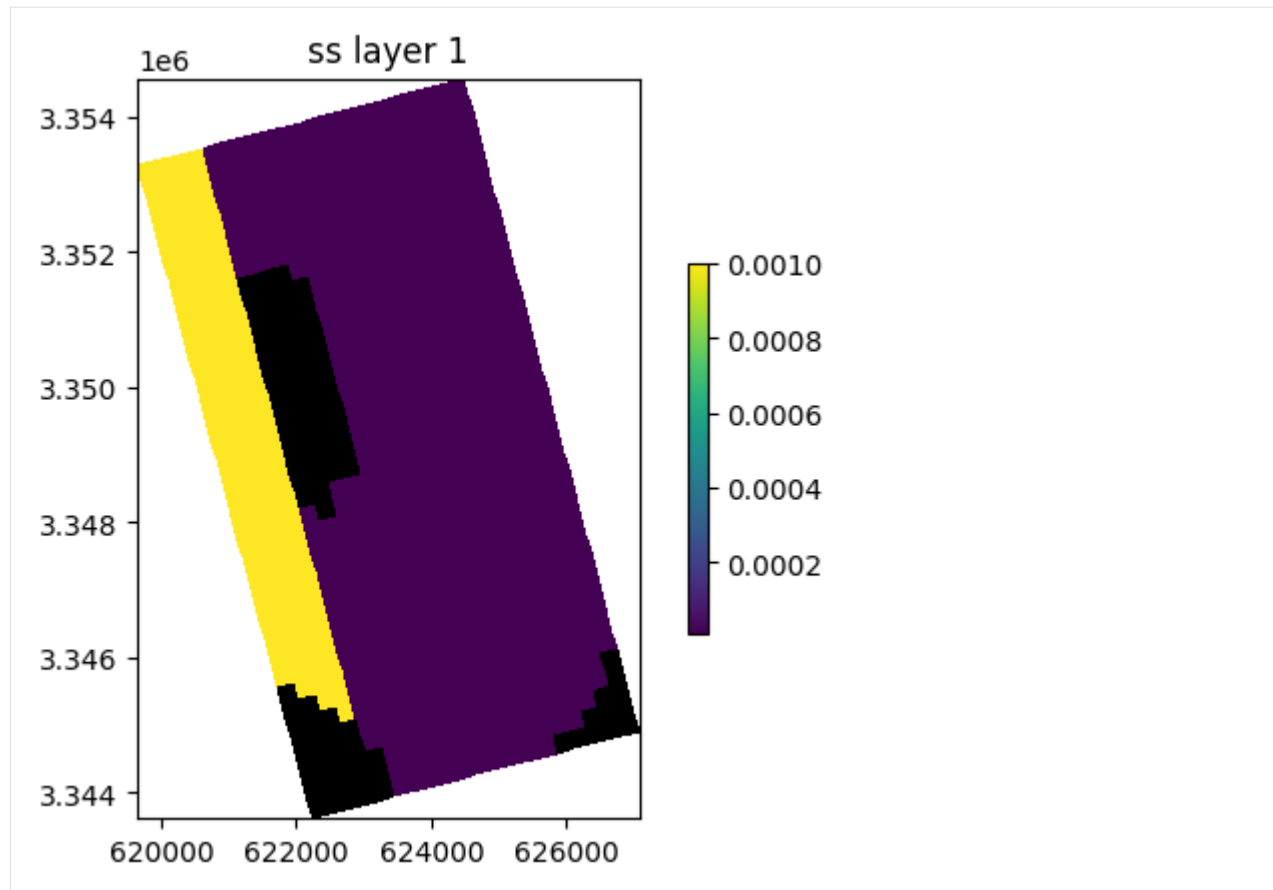


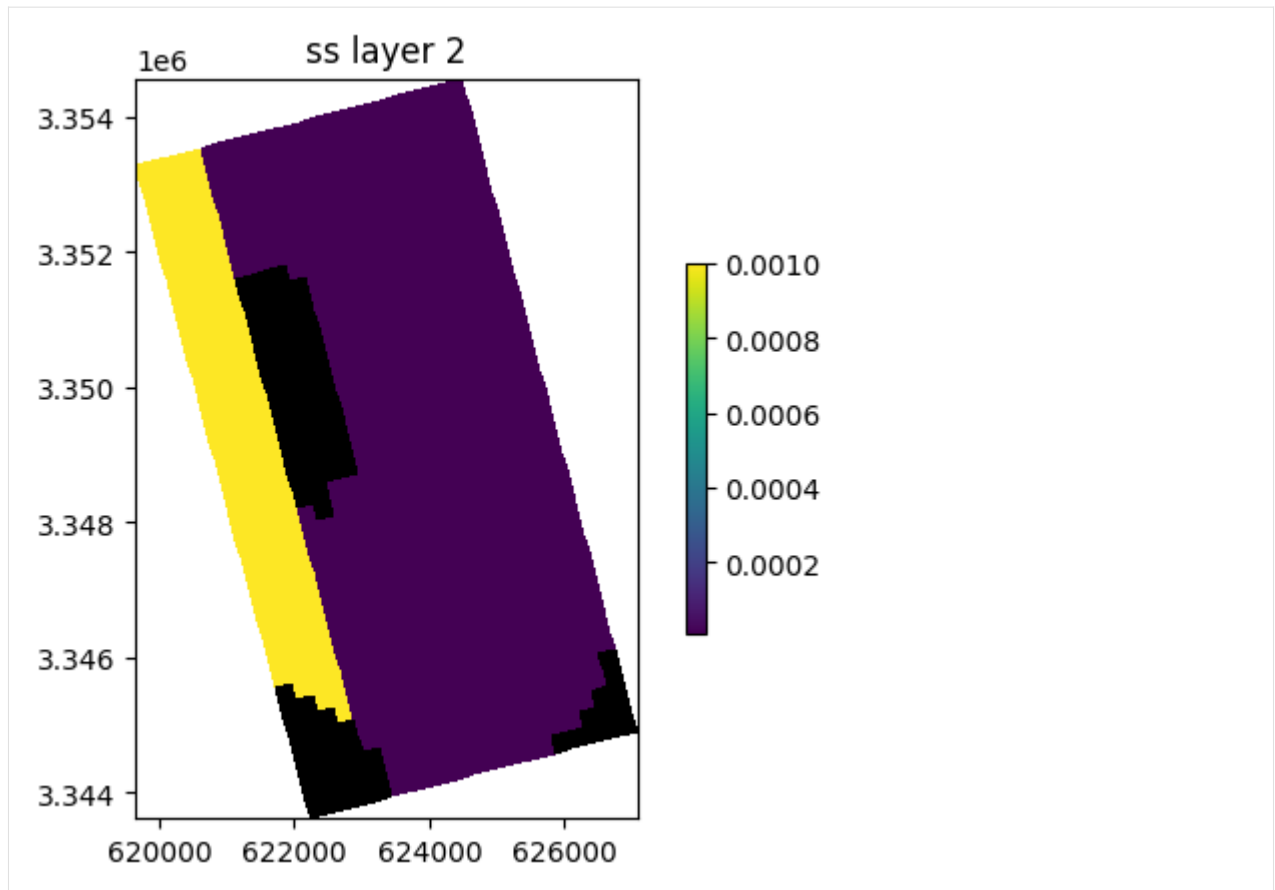


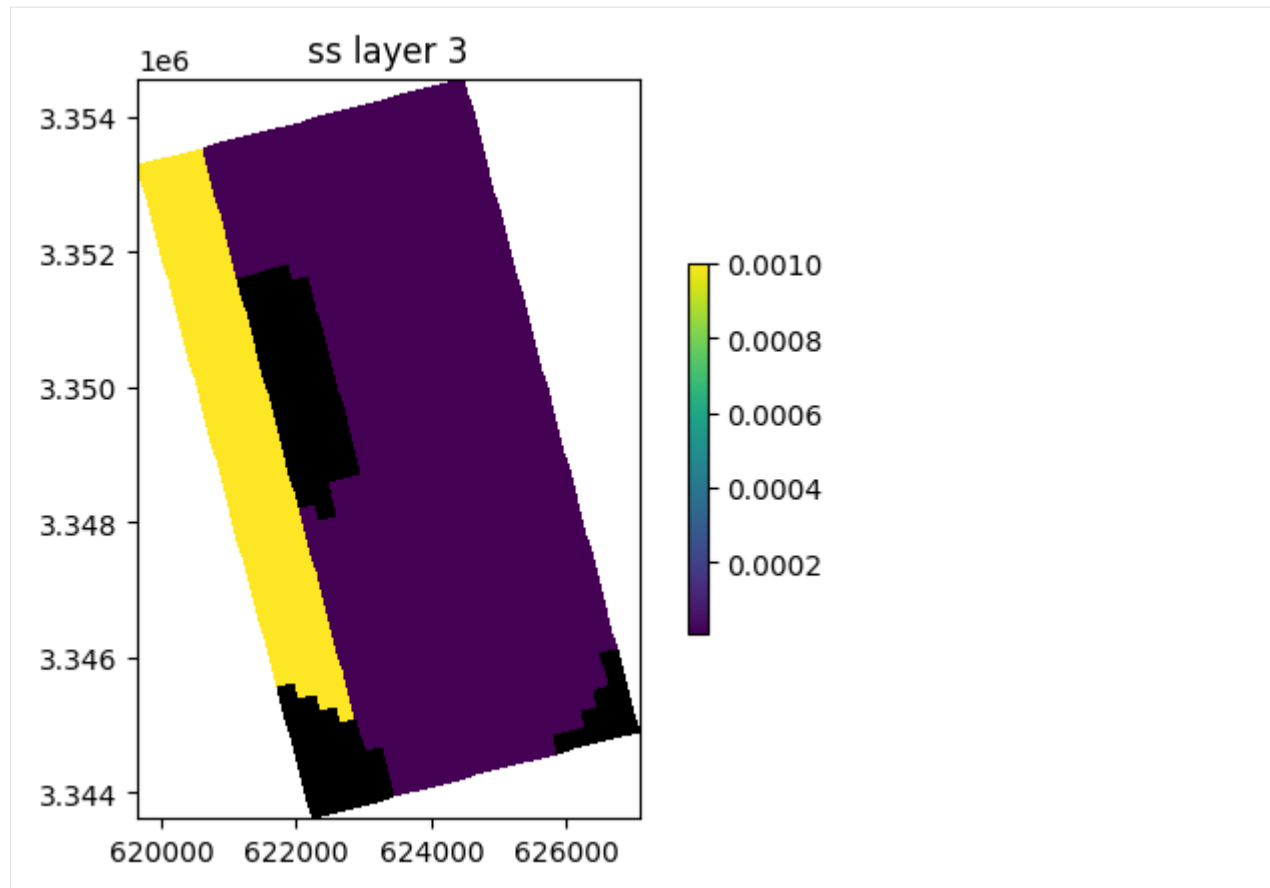


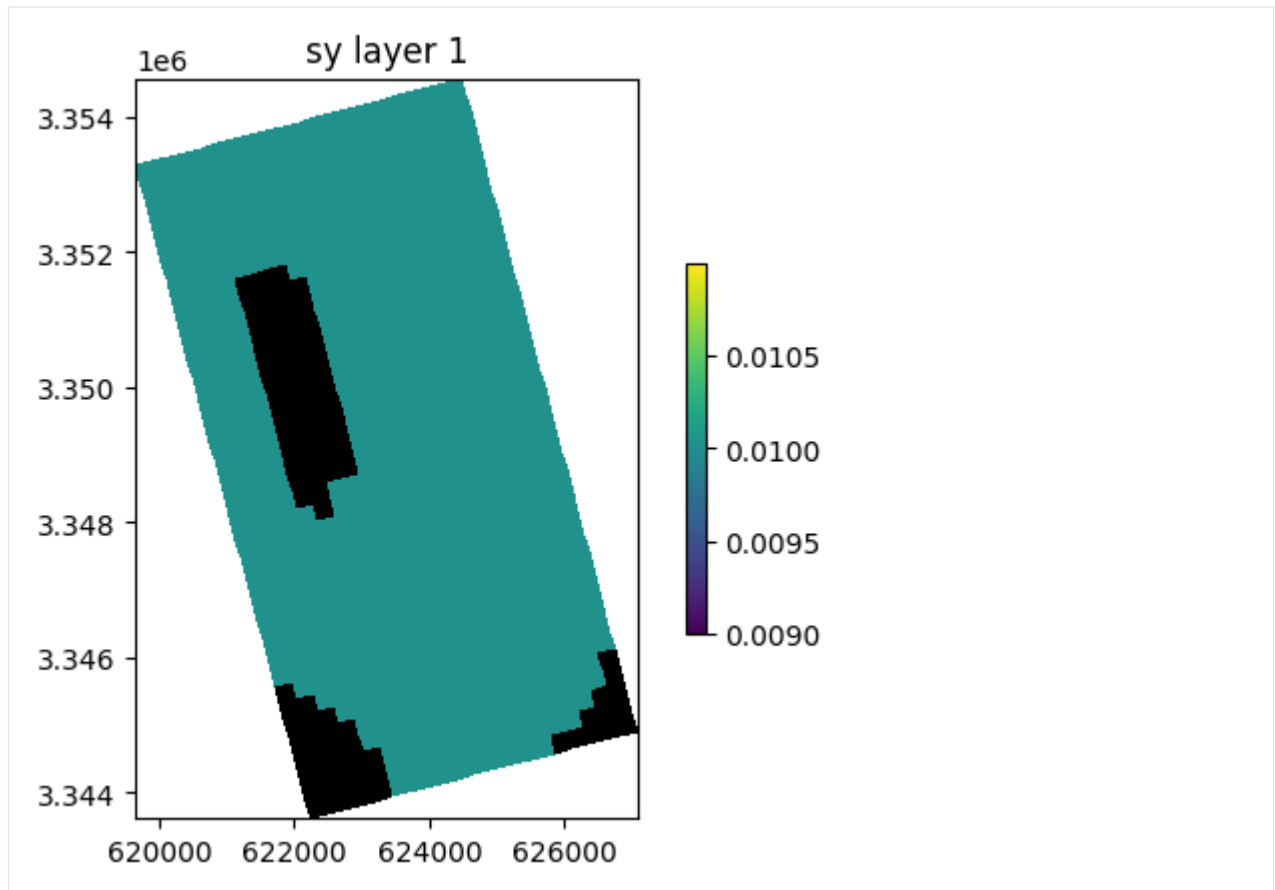


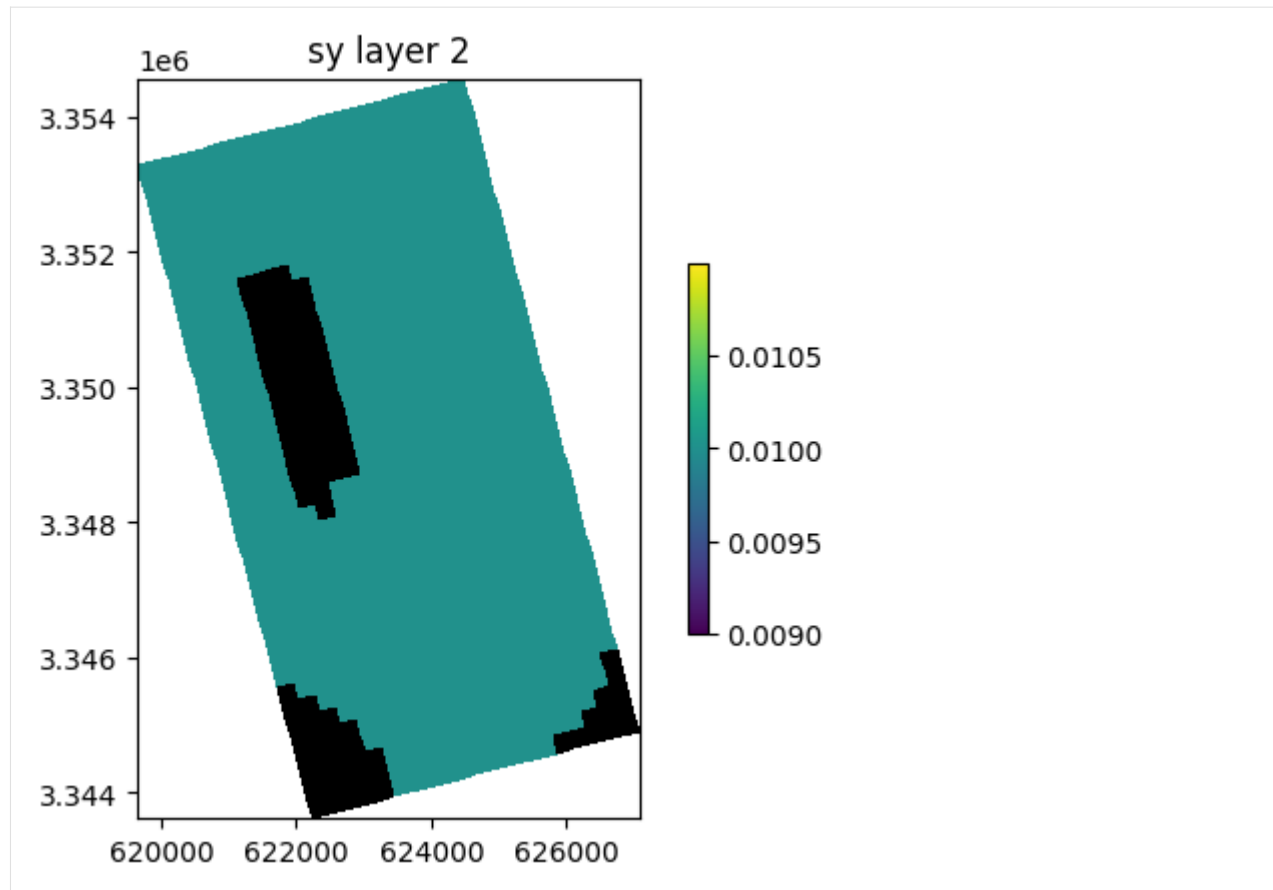


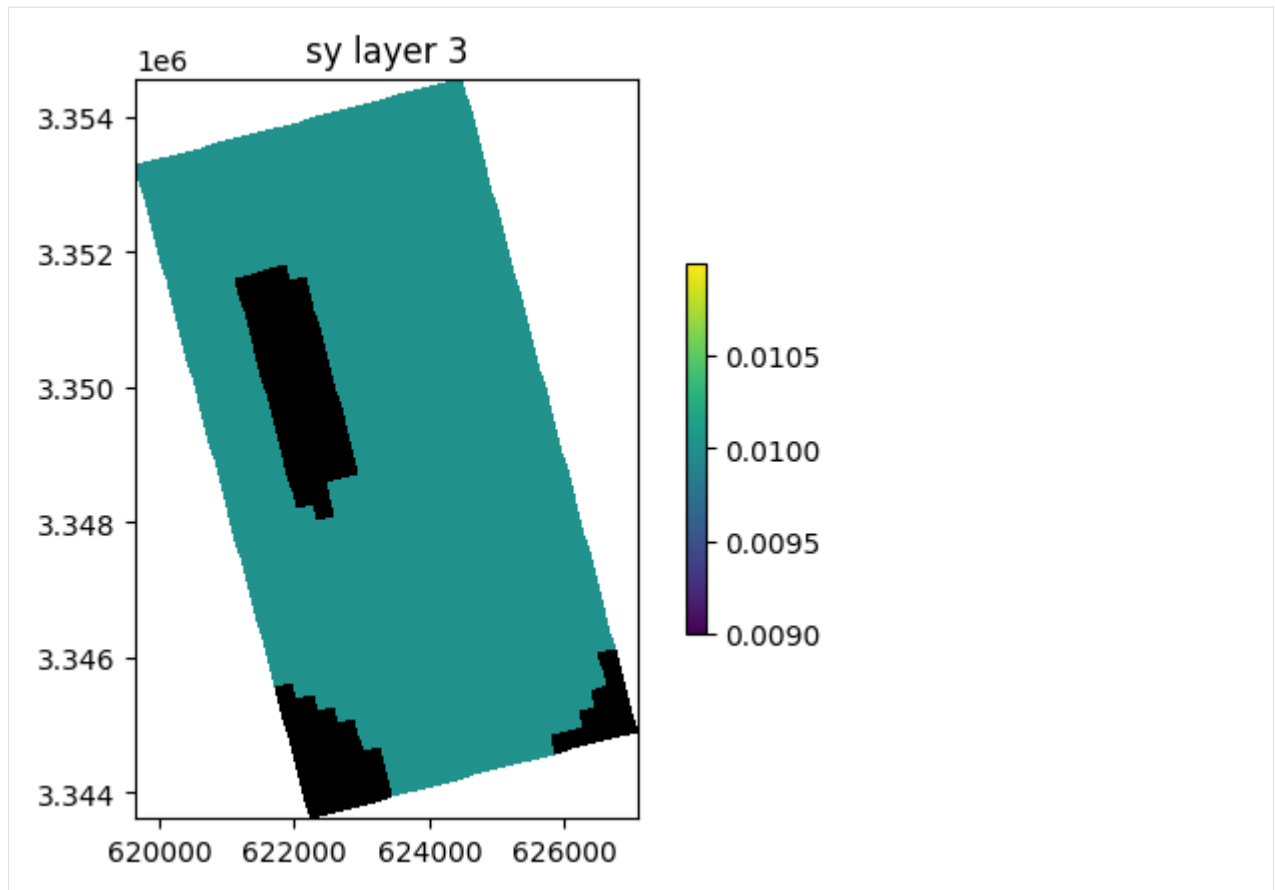


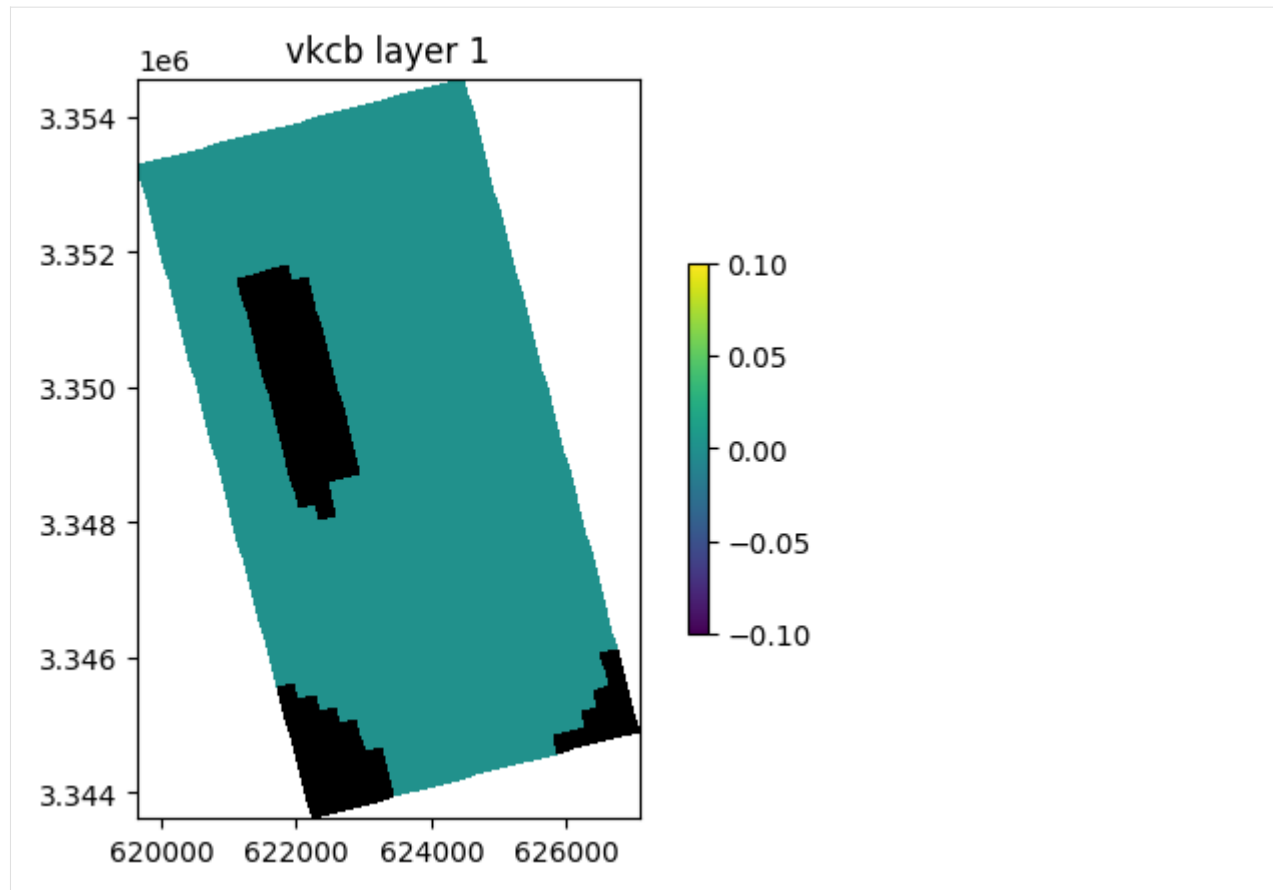


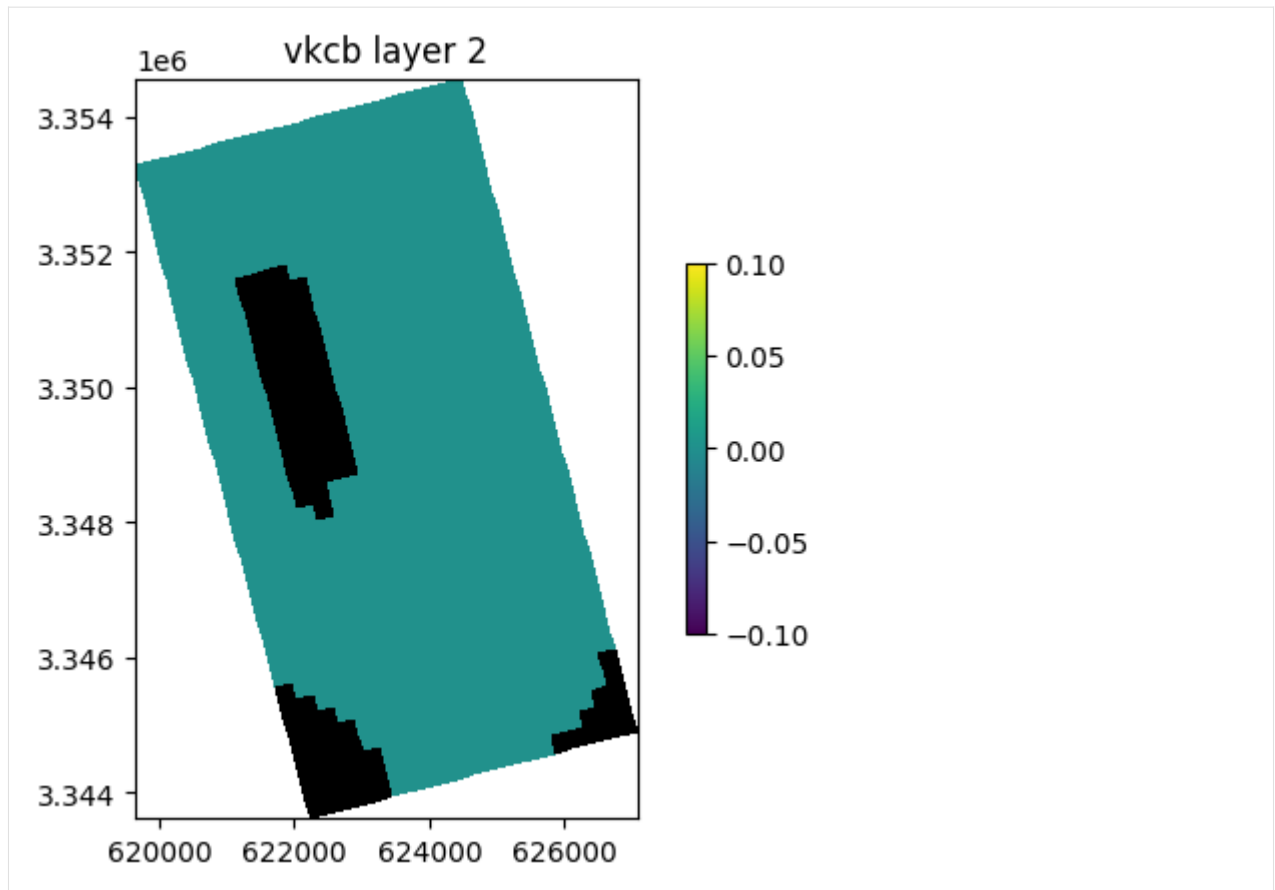


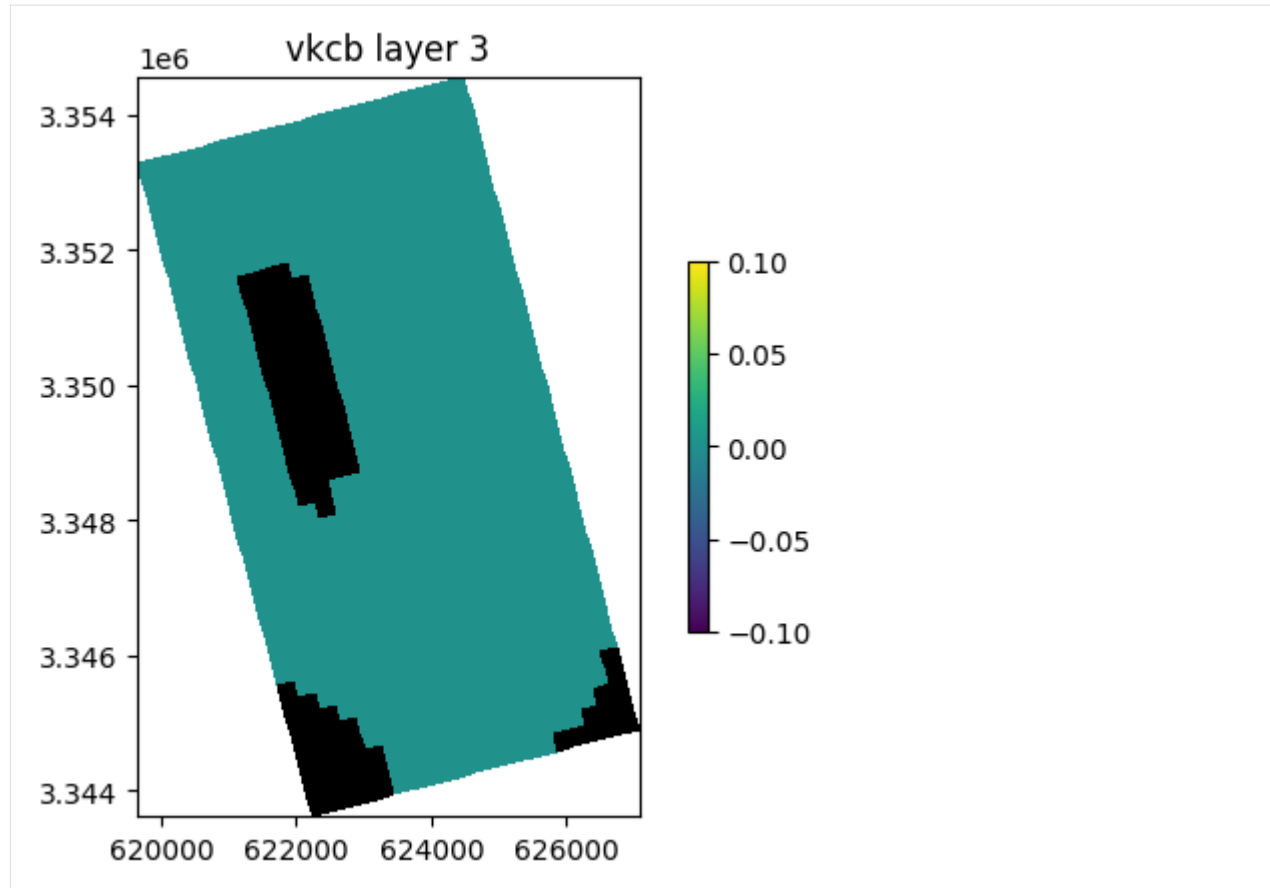






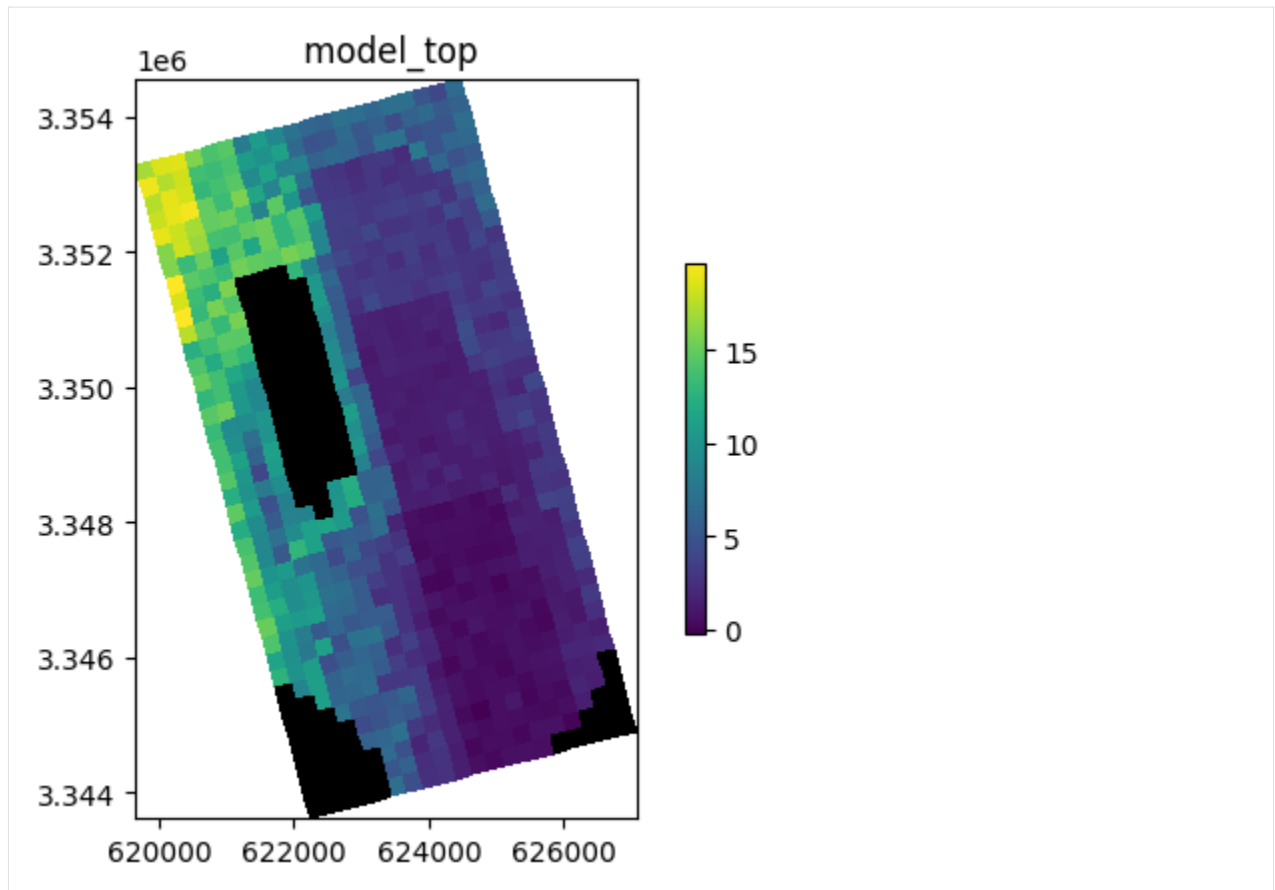


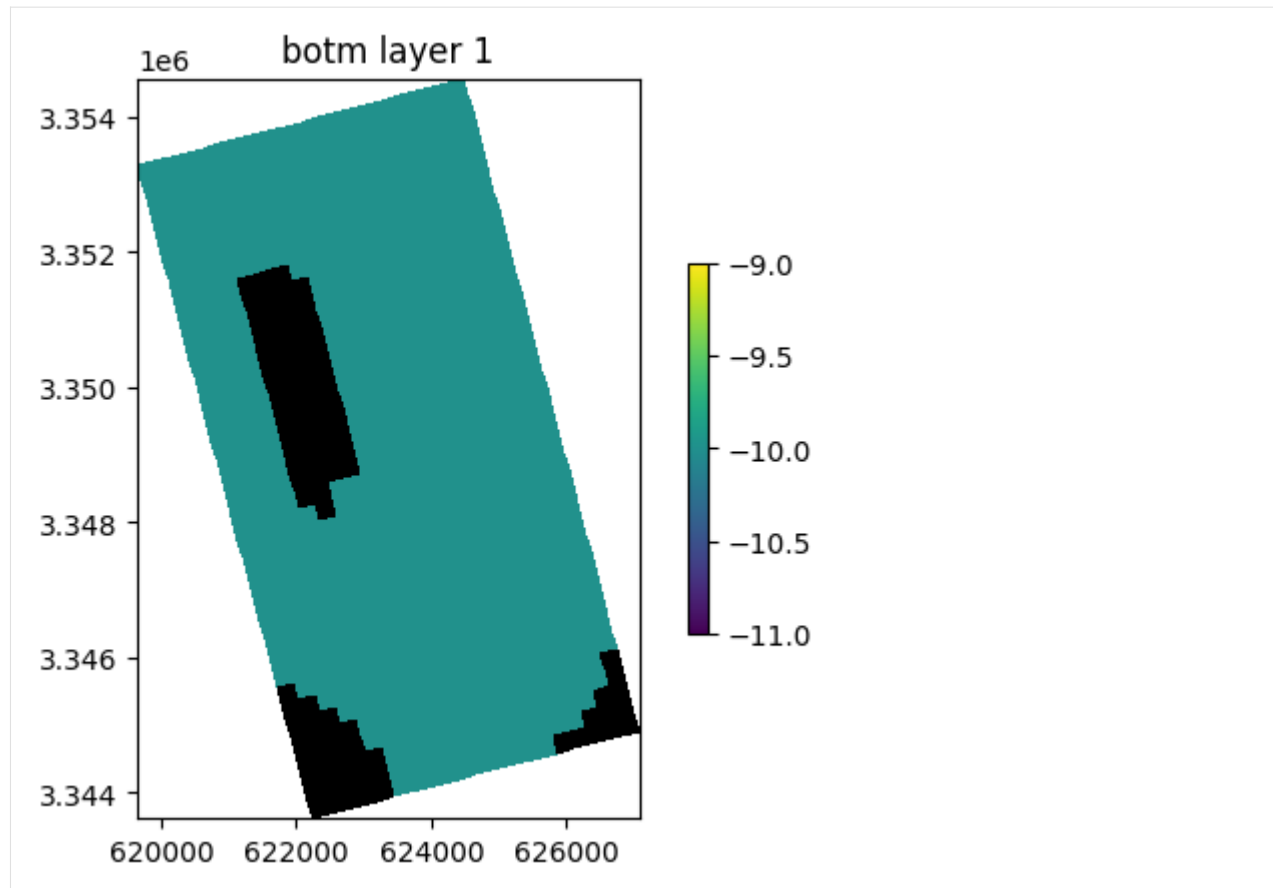


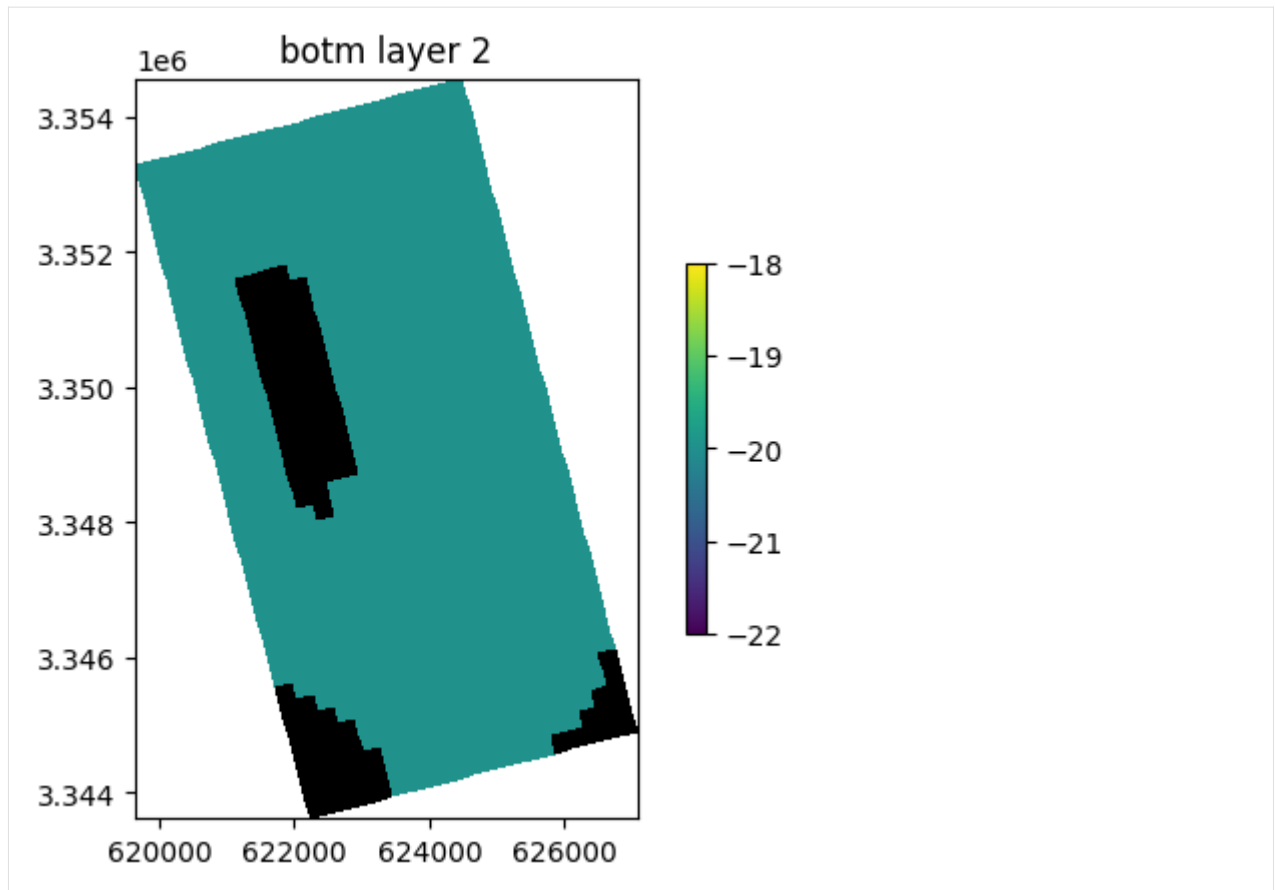


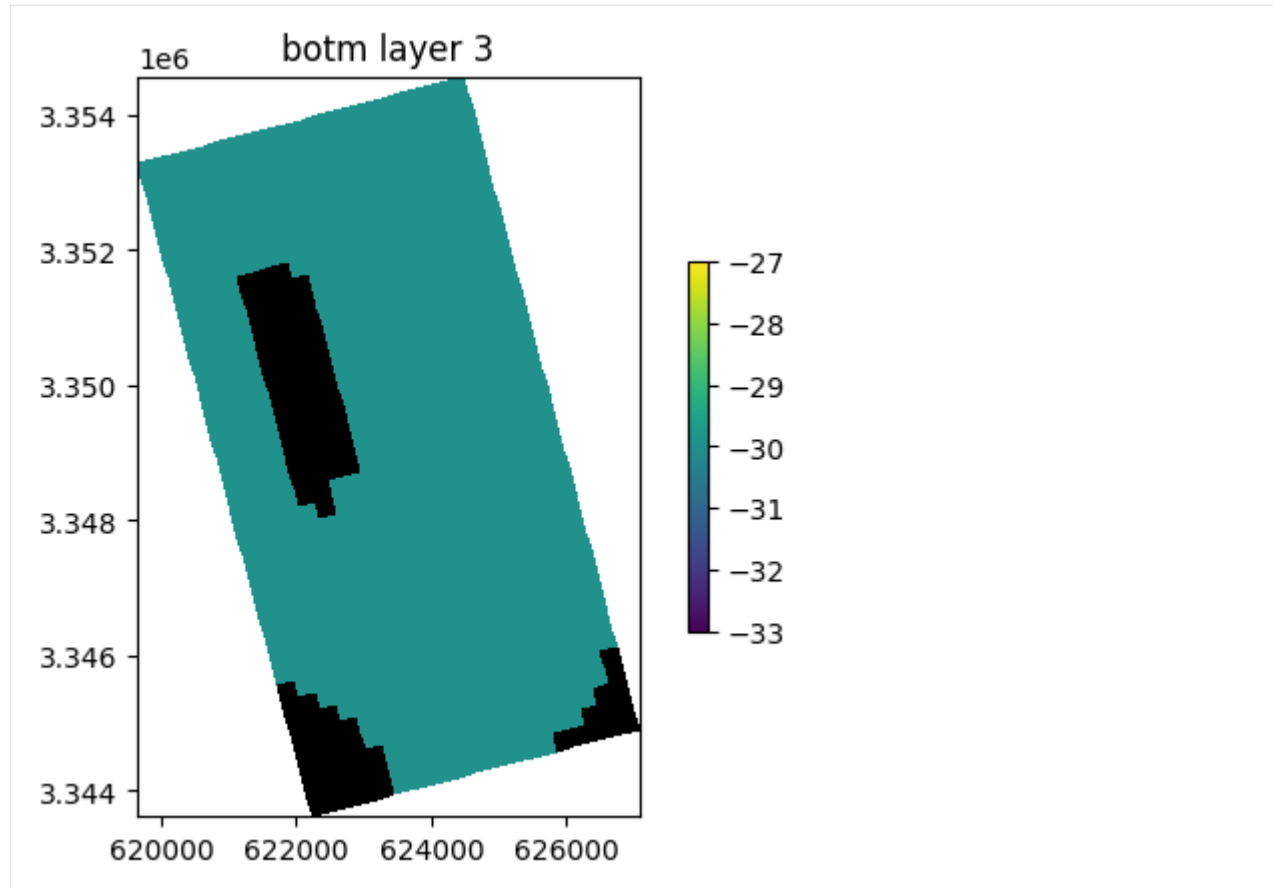
```
[7]: ml.dis.plot()
```

```
[7]: [<Axes: title={'center': ' model_top'}>,
      <Axes: title={'center': 'botm layer 1'}>,
      <Axes: title={'center': 'botm layer 2'}>,
      <Axes: title={'center': 'botm layer 3'}>]
```



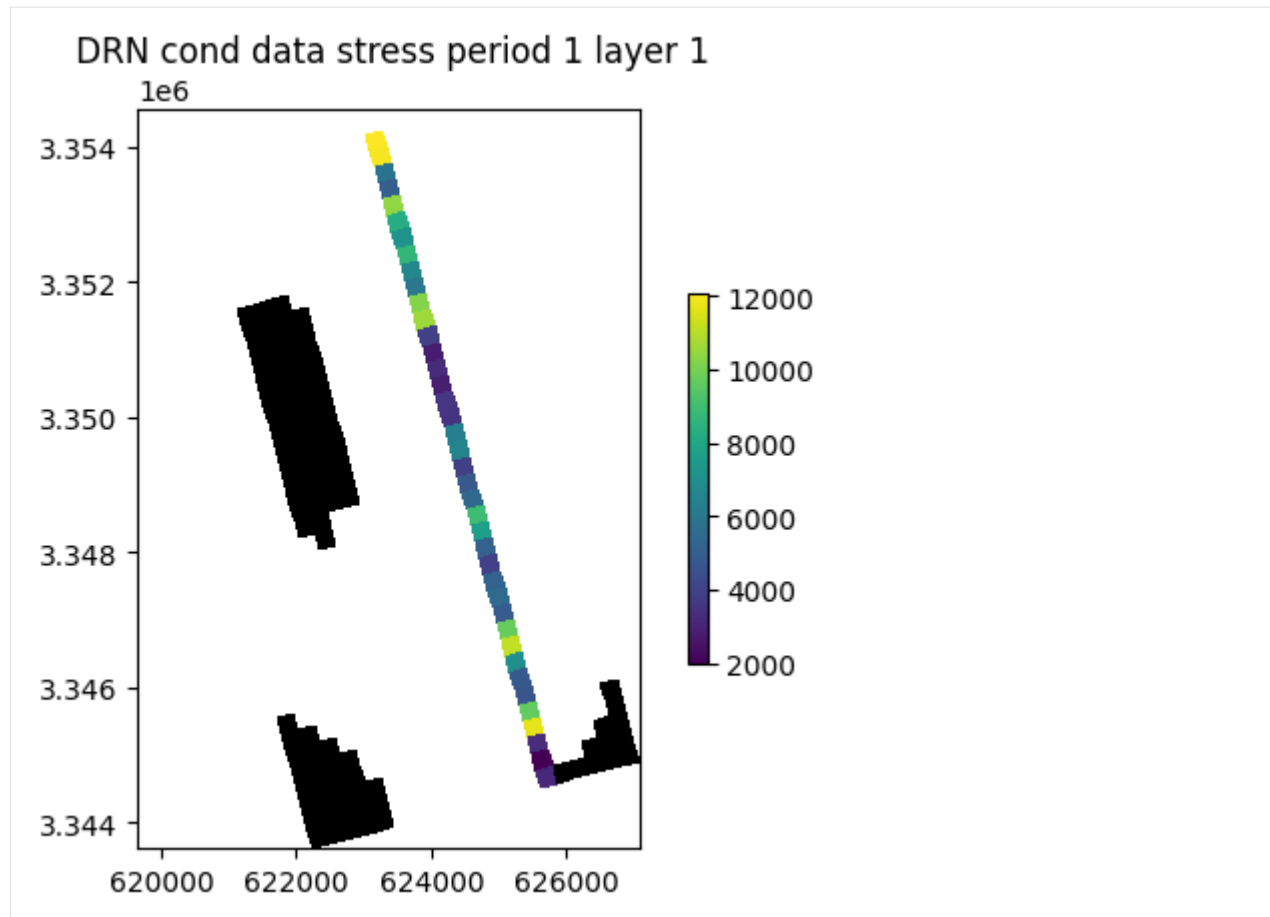


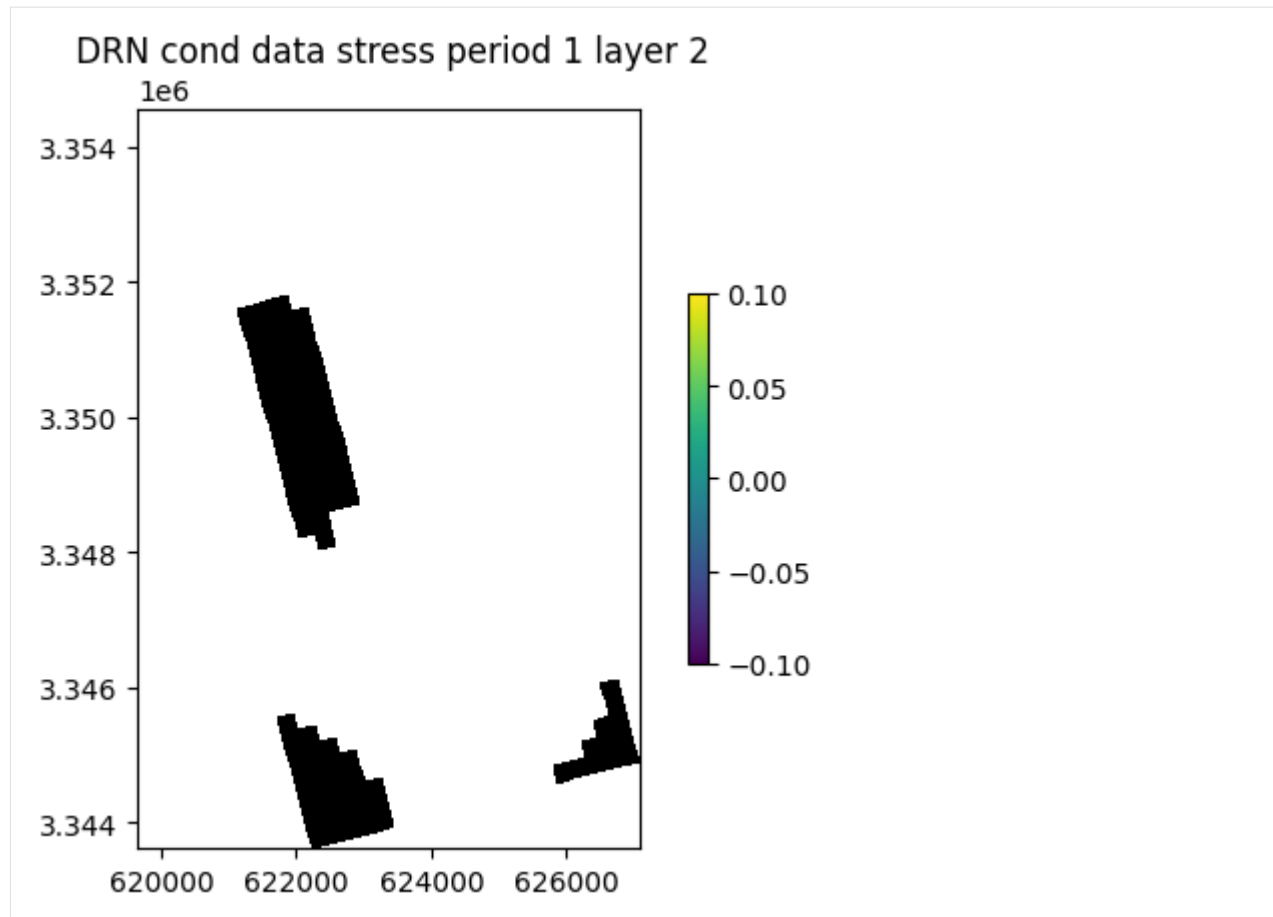


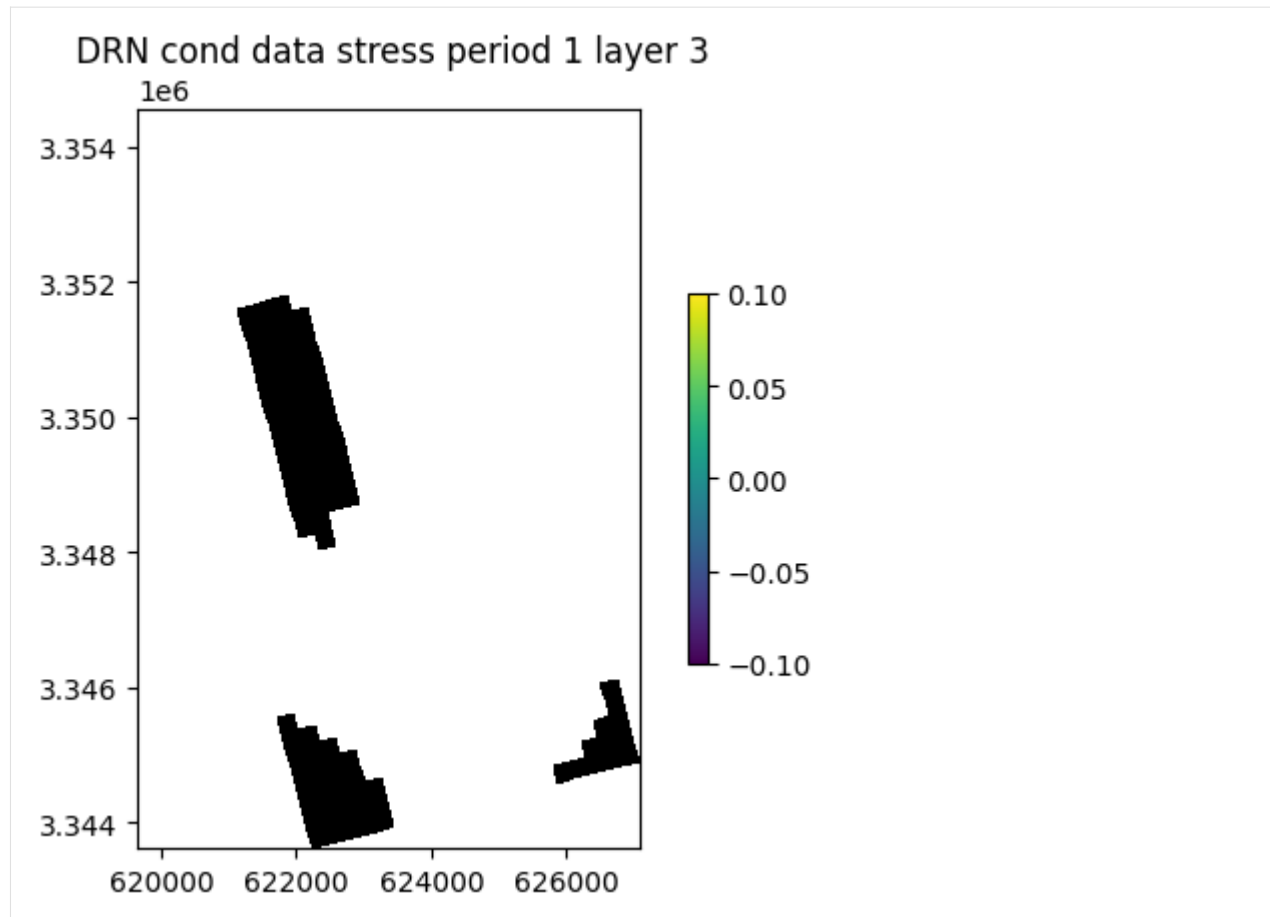


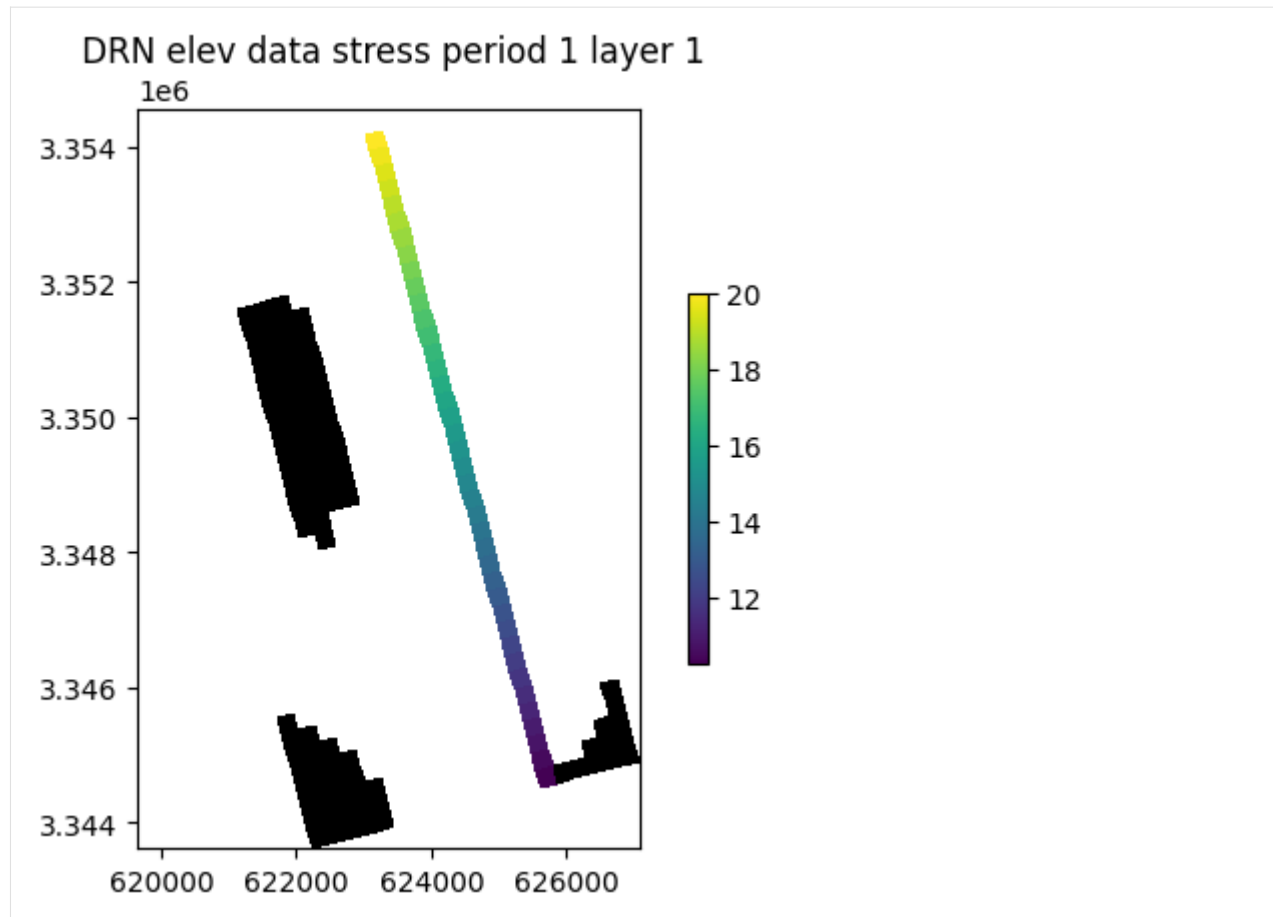
```
[8]: ml.drn.plot(key="cond")
ml.drn.plot(key="elev")
```

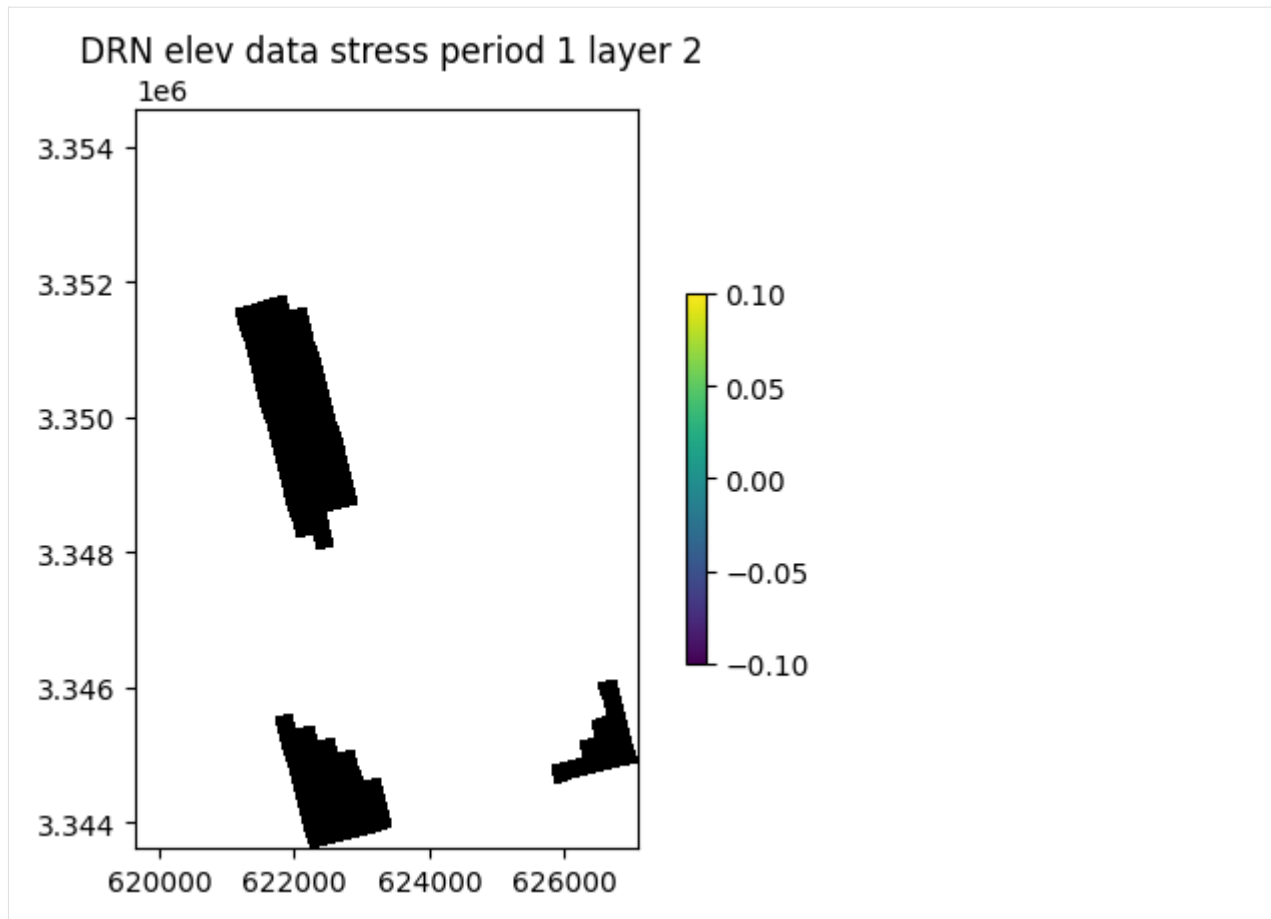
```
[8]: [<Axes: title={'center': ' DRN elev data stress period 1 layer 1'}>,
<Axes: title={'center': ' DRN elev data stress period 1 layer 2'}>,
<Axes: title={'center': ' DRN elev data stress period 1 layer 3'}>]
```

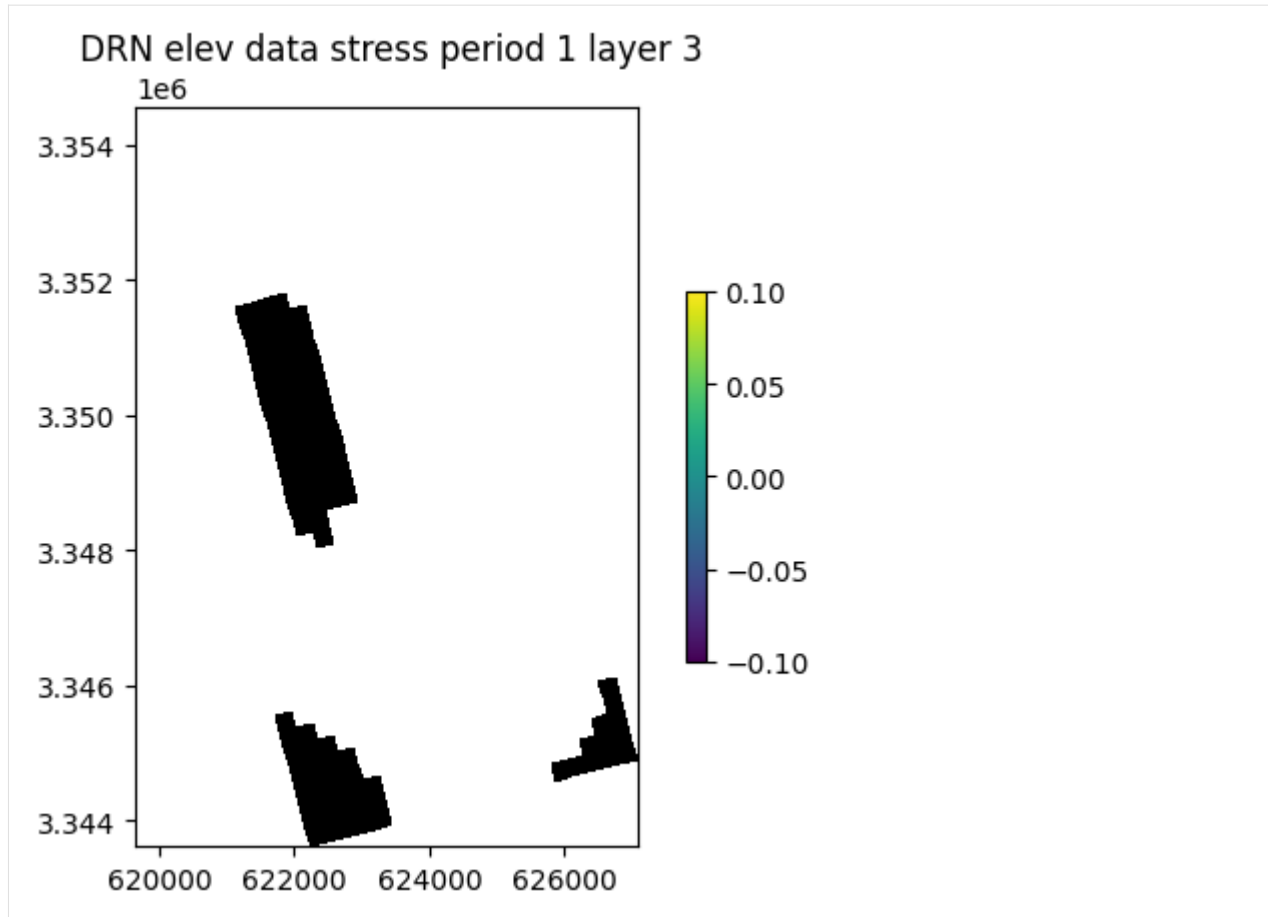












First create a temporary workspace.

```
[9]: # create a temporary workspace
temp_dir = TemporaryDirectory()
workspace = Path(temp_dir.name)
```

Write a shapefile of the DIS package.

```
[10]: # write the shapefile
ml.dis.export(workspace / "freyberg_dis.shp")

/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳ shapefile_utils.py:307: UserWarning: Failed to get data for delc array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳ shapefile_utils.py:307: UserWarning: Failed to get data for delr array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳ shapefile_utils.py:307: UserWarning: Failed to get data for laycbd array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↳ shapefile_utils.py:307: UserWarning: Failed to get data for nstp array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
```

(continues on next page)

(continued from previous page)

```
↪shapefile_utils.py:307: UserWarning: Failed to get data for perlen array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪shapefile_utils.py:307: UserWarning: Failed to get data for steady array, DIS package
warn(
/opt/hostedtoolcache/Python/3.8.18/x64/lib/python3.8/site-packages/flopy/export/
↪shapefile_utils.py:307: UserWarning: Failed to get data for tsmult array, DIS package
warn(
```

Write a netCDF file with all model inputs.

```
[11]: ml.export(workspace / "freyberg.nc")

initialize_geometry::
model crs: EPSG:32614
initialize_geometry::nc_crs = EPSG:4326
transforming coordinates using = proj=pipeline step inv proj=utm zone=14 ellps=WGS84_
↪step proj=unitconvert xy_in=rad xy_out=deg

[11]: <flopy.export.netcdf.NetCdf at 0x7ff9e4da2b20>
```

Change model directory and external path, modify inputs and write new input files.

```
[12]: ml.external_path = workspace / "ref"
ml.model_ws = workspace
ml.write_input()

Util2d:delr: resetting 'how' to external
Util2d:delc: resetting 'how' to external
Util2d:model_top: resetting 'how' to external
Util2d:botm_layer_0: resetting 'how' to external
Util2d:botm_layer_1: resetting 'how' to external
Util2d:botm_layer_2: resetting 'how' to external
Util2d:ibound_layer_0: resetting 'how' to external
Util2d:ibound_layer_1: resetting 'how' to external
Util2d:ibound_layer_2: resetting 'how' to external
Util2d:strt_layer_0: resetting 'how' to external
Util2d:strt_layer_1: resetting 'how' to external
Util2d:strt_layer_2: resetting 'how' to external
Util2d:rech_1: resetting 'how' to external
Util2d:rech_2: resetting 'how' to external
Util2d:rech_3: resetting 'how' to external
Util2d:rech_4: resetting 'how' to external
Util2d:rech_5: resetting 'how' to external
Util2d:rech_6: resetting 'how' to external
Util2d:rech_7: resetting 'how' to external
Util2d:rech_8: resetting 'how' to external
Util2d:rech_9: resetting 'how' to external
Util2d:rech_10: resetting 'how' to external
Util2d:rech_11: resetting 'how' to external
Util2d:rech_12: resetting 'how' to external
Util2d:rech_13: resetting 'how' to external
Util2d:rech_14: resetting 'how' to external
Util2d:rech_15: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_16: resetting 'how' to external
Util2d:rech_17: resetting 'how' to external
Util2d:rech_18: resetting 'how' to external
Util2d:rech_19: resetting 'how' to external
Util2d:rech_20: resetting 'how' to external
Util2d:rech_21: resetting 'how' to external
Util2d:rech_22: resetting 'how' to external
Util2d:rech_23: resetting 'how' to external
Util2d:rech_24: resetting 'how' to external
Util2d:rech_25: resetting 'how' to external
Util2d:rech_26: resetting 'how' to external
Util2d:rech_27: resetting 'how' to external
Util2d:rech_28: resetting 'how' to external
Util2d:rech_29: resetting 'how' to external
Util2d:rech_30: resetting 'how' to external
Util2d:rech_31: resetting 'how' to external
Util2d:rech_32: resetting 'how' to external
Util2d:rech_33: resetting 'how' to external
Util2d:rech_34: resetting 'how' to external
Util2d:rech_35: resetting 'how' to external
Util2d:rech_36: resetting 'how' to external
Util2d:rech_37: resetting 'how' to external
Util2d:rech_38: resetting 'how' to external
Util2d:rech_39: resetting 'how' to external
Util2d:rech_40: resetting 'how' to external
Util2d:rech_41: resetting 'how' to external
Util2d:rech_42: resetting 'how' to external
Util2d:rech_43: resetting 'how' to external
Util2d:rech_44: resetting 'how' to external
Util2d:rech_45: resetting 'how' to external
Util2d:rech_46: resetting 'how' to external
Util2d:rech_47: resetting 'how' to external
Util2d:rech_48: resetting 'how' to external
Util2d:rech_49: resetting 'how' to external
Util2d:rech_50: resetting 'how' to external
Util2d:rech_51: resetting 'how' to external
Util2d:rech_52: resetting 'how' to external
Util2d:rech_53: resetting 'how' to external
Util2d:rech_54: resetting 'how' to external
Util2d:rech_55: resetting 'how' to external
Util2d:rech_56: resetting 'how' to external
Util2d:rech_57: resetting 'how' to external
Util2d:rech_58: resetting 'how' to external
Util2d:rech_59: resetting 'how' to external
Util2d:rech_60: resetting 'how' to external
Util2d:rech_61: resetting 'how' to external
Util2d:rech_62: resetting 'how' to external
Util2d:rech_63: resetting 'how' to external
Util2d:rech_64: resetting 'how' to external
Util2d:rech_65: resetting 'how' to external
Util2d:rech_66: resetting 'how' to external
Util2d:rech_67: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_68: resetting 'how' to external
Util2d:rech_69: resetting 'how' to external
Util2d:rech_70: resetting 'how' to external
Util2d:rech_71: resetting 'how' to external
Util2d:rech_72: resetting 'how' to external
Util2d:rech_73: resetting 'how' to external
Util2d:rech_74: resetting 'how' to external
Util2d:rech_75: resetting 'how' to external
Util2d:rech_76: resetting 'how' to external
Util2d:rech_77: resetting 'how' to external
Util2d:rech_78: resetting 'how' to external
Util2d:rech_79: resetting 'how' to external
Util2d:rech_80: resetting 'how' to external
Util2d:rech_81: resetting 'how' to external
Util2d:rech_82: resetting 'how' to external
Util2d:rech_83: resetting 'how' to external
Util2d:rech_84: resetting 'how' to external
Util2d:rech_85: resetting 'how' to external
Util2d:rech_86: resetting 'how' to external
Util2d:rech_87: resetting 'how' to external
Util2d:rech_88: resetting 'how' to external
Util2d:rech_89: resetting 'how' to external
Util2d:rech_90: resetting 'how' to external
Util2d:rech_91: resetting 'how' to external
Util2d:rech_92: resetting 'how' to external
Util2d:rech_93: resetting 'how' to external
Util2d:rech_94: resetting 'how' to external
Util2d:rech_95: resetting 'how' to external
Util2d:rech_96: resetting 'how' to external
Util2d:rech_97: resetting 'how' to external
Util2d:rech_98: resetting 'how' to external
Util2d:rech_99: resetting 'how' to external
Util2d:rech_100: resetting 'how' to external
Util2d:rech_101: resetting 'how' to external
Util2d:rech_102: resetting 'how' to external
Util2d:rech_103: resetting 'how' to external
Util2d:rech_104: resetting 'how' to external
Util2d:rech_105: resetting 'how' to external
Util2d:rech_106: resetting 'how' to external
Util2d:rech_107: resetting 'how' to external
Util2d:rech_108: resetting 'how' to external
Util2d:rech_109: resetting 'how' to external
Util2d:rech_110: resetting 'how' to external
Util2d:rech_111: resetting 'how' to external
Util2d:rech_112: resetting 'how' to external
Util2d:rech_113: resetting 'how' to external
Util2d:rech_114: resetting 'how' to external
Util2d:rech_115: resetting 'how' to external
Util2d:rech_116: resetting 'how' to external
Util2d:rech_117: resetting 'how' to external
Util2d:rech_118: resetting 'how' to external
Util2d:rech_119: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```

Util2d:rech_120: resetting 'how' to external
Util2d:rech_121: resetting 'how' to external
Util2d:rech_122: resetting 'how' to external
Util2d:rech_123: resetting 'how' to external
Util2d:rech_124: resetting 'how' to external
Util2d:rech_125: resetting 'how' to external
Util2d:rech_126: resetting 'how' to external
Util2d:rech_127: resetting 'how' to external
Util2d:rech_128: resetting 'how' to external
Util2d:rech_129: resetting 'how' to external
Util2d:rech_130: resetting 'how' to external
Util2d:rech_131: resetting 'how' to external
Util2d:rech_132: resetting 'how' to external
Util2d:rech_133: resetting 'how' to external
Util2d:rech_134: resetting 'how' to external
Util2d:rech_135: resetting 'how' to external
Util2d:rech_136: resetting 'how' to external
Util2d:rech_137: resetting 'how' to external
Util2d:rech_138: resetting 'how' to external
Util2d:rech_139: resetting 'how' to external
Util2d:rech_140: resetting 'how' to external
Util2d:rech_141: resetting 'how' to external
Util2d:rech_142: resetting 'how' to external
Util2d:rech_143: resetting 'how' to external
Util2d:rech_144: resetting 'how' to external
Util2d:rech_145: resetting 'how' to external
Util2d:rech_146: resetting 'how' to external
Util2d:rech_147: resetting 'how' to external
Util2d:rech_148: resetting 'how' to external
Util2d:rech_149: resetting 'how' to external
Util2d:rech_150: resetting 'how' to external
Util2d:rech_151: resetting 'how' to external
Util2d:rech_152: resetting 'how' to external
Util2d:rech_153: resetting 'how' to external
Util2d:rech_154: resetting 'how' to external
Util2d:rech_155: resetting 'how' to external
Util2d:rech_156: resetting 'how' to external
Util2d:rech_157: resetting 'how' to external
Util2d:rech_158: resetting 'how' to external
Util2d:rech_159: resetting 'how' to external
Util2d:rech_160: resetting 'how' to external
Util2d:rech_161: resetting 'how' to external
Util2d:rech_162: resetting 'how' to external
Util2d:rech_163: resetting 'how' to external
Util2d:rech_164: resetting 'how' to external
Util2d:rech_165: resetting 'how' to external
Util2d:rech_166: resetting 'how' to external
Util2d:rech_167: resetting 'how' to external
Util2d:rech_168: resetting 'how' to external
Util2d:rech_169: resetting 'how' to external
Util2d:rech_170: resetting 'how' to external
Util2d:rech_171: resetting 'how' to external

```

(continues on next page)

(continued from previous page)

```
Util2d:rech_172: resetting 'how' to external
Util2d:rech_173: resetting 'how' to external
Util2d:rech_174: resetting 'how' to external
Util2d:rech_175: resetting 'how' to external
Util2d:rech_176: resetting 'how' to external
Util2d:rech_177: resetting 'how' to external
Util2d:rech_178: resetting 'how' to external
Util2d:rech_179: resetting 'how' to external
Util2d:rech_180: resetting 'how' to external
Util2d:rech_181: resetting 'how' to external
Util2d:rech_182: resetting 'how' to external
Util2d:rech_183: resetting 'how' to external
Util2d:rech_184: resetting 'how' to external
Util2d:rech_185: resetting 'how' to external
Util2d:rech_186: resetting 'how' to external
Util2d:rech_187: resetting 'how' to external
Util2d:rech_188: resetting 'how' to external
Util2d:rech_189: resetting 'how' to external
Util2d:rech_190: resetting 'how' to external
Util2d:rech_191: resetting 'how' to external
Util2d:rech_192: resetting 'how' to external
Util2d:rech_193: resetting 'how' to external
Util2d:rech_194: resetting 'how' to external
Util2d:rech_195: resetting 'how' to external
Util2d:rech_196: resetting 'how' to external
Util2d:rech_197: resetting 'how' to external
Util2d:rech_198: resetting 'how' to external
Util2d:rech_199: resetting 'how' to external
Util2d:rech_200: resetting 'how' to external
Util2d:rech_201: resetting 'how' to external
Util2d:rech_202: resetting 'how' to external
Util2d:rech_203: resetting 'how' to external
Util2d:rech_204: resetting 'how' to external
Util2d:rech_205: resetting 'how' to external
Util2d:rech_206: resetting 'how' to external
Util2d:rech_207: resetting 'how' to external
Util2d:rech_208: resetting 'how' to external
Util2d:rech_209: resetting 'how' to external
Util2d:rech_210: resetting 'how' to external
Util2d:rech_211: resetting 'how' to external
Util2d:rech_212: resetting 'how' to external
Util2d:rech_213: resetting 'how' to external
Util2d:rech_214: resetting 'how' to external
Util2d:rech_215: resetting 'how' to external
Util2d:rech_216: resetting 'how' to external
Util2d:rech_217: resetting 'how' to external
Util2d:rech_218: resetting 'how' to external
Util2d:rech_219: resetting 'how' to external
Util2d:rech_220: resetting 'how' to external
Util2d:rech_221: resetting 'how' to external
Util2d:rech_222: resetting 'how' to external
Util2d:rech_223: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```

Util2d:rech_224: resetting 'how' to external
Util2d:rech_225: resetting 'how' to external
Util2d:rech_226: resetting 'how' to external
Util2d:rech_227: resetting 'how' to external
Util2d:rech_228: resetting 'how' to external
Util2d:rech_229: resetting 'how' to external
Util2d:rech_230: resetting 'how' to external
Util2d:rech_231: resetting 'how' to external
Util2d:rech_232: resetting 'how' to external
Util2d:rech_233: resetting 'how' to external
Util2d:rech_234: resetting 'how' to external
Util2d:rech_235: resetting 'how' to external
Util2d:rech_236: resetting 'how' to external
Util2d:rech_237: resetting 'how' to external
Util2d:rech_238: resetting 'how' to external
Util2d:rech_239: resetting 'how' to external
Util2d:rech_240: resetting 'how' to external
Util2d:rech_241: resetting 'how' to external
Util2d:rech_242: resetting 'how' to external
Util2d:rech_243: resetting 'how' to external
Util2d:rech_244: resetting 'how' to external
Util2d:rech_245: resetting 'how' to external
Util2d:rech_246: resetting 'how' to external
Util2d:rech_247: resetting 'how' to external
Util2d:rech_248: resetting 'how' to external
Util2d:rech_249: resetting 'how' to external
Util2d:rech_250: resetting 'how' to external
Util2d:rech_251: resetting 'how' to external
Util2d:rech_252: resetting 'how' to external
Util2d:rech_253: resetting 'how' to external
Util2d:rech_254: resetting 'how' to external
Util2d:rech_255: resetting 'how' to external
Util2d:rech_256: resetting 'how' to external
Util2d:rech_257: resetting 'how' to external
Util2d:rech_258: resetting 'how' to external
Util2d:rech_259: resetting 'how' to external
Util2d:rech_260: resetting 'how' to external
Util2d:rech_261: resetting 'how' to external
Util2d:rech_262: resetting 'how' to external
Util2d:rech_263: resetting 'how' to external
Util2d:rech_264: resetting 'how' to external
Util2d:rech_265: resetting 'how' to external
Util2d:rech_266: resetting 'how' to external
Util2d:rech_267: resetting 'how' to external
Util2d:rech_268: resetting 'how' to external
Util2d:rech_269: resetting 'how' to external
Util2d:rech_270: resetting 'how' to external
Util2d:rech_271: resetting 'how' to external
Util2d:rech_272: resetting 'how' to external
Util2d:rech_273: resetting 'how' to external
Util2d:rech_274: resetting 'how' to external
Util2d:rech_275: resetting 'how' to external

```

(continues on next page)

(continued from previous page)

```
Util2d:rech_276: resetting 'how' to external
Util2d:rech_277: resetting 'how' to external
Util2d:rech_278: resetting 'how' to external
Util2d:rech_279: resetting 'how' to external
Util2d:rech_280: resetting 'how' to external
Util2d:rech_281: resetting 'how' to external
Util2d:rech_282: resetting 'how' to external
Util2d:rech_283: resetting 'how' to external
Util2d:rech_284: resetting 'how' to external
Util2d:rech_285: resetting 'how' to external
Util2d:rech_286: resetting 'how' to external
Util2d:rech_287: resetting 'how' to external
Util2d:rech_288: resetting 'how' to external
Util2d:rech_289: resetting 'how' to external
Util2d:rech_290: resetting 'how' to external
Util2d:rech_291: resetting 'how' to external
Util2d:rech_292: resetting 'how' to external
Util2d:rech_293: resetting 'how' to external
Util2d:rech_294: resetting 'how' to external
Util2d:rech_295: resetting 'how' to external
Util2d:rech_296: resetting 'how' to external
Util2d:rech_297: resetting 'how' to external
Util2d:rech_298: resetting 'how' to external
Util2d:rech_299: resetting 'how' to external
Util2d:rech_300: resetting 'how' to external
Util2d:rech_301: resetting 'how' to external
Util2d:rech_302: resetting 'how' to external
Util2d:rech_303: resetting 'how' to external
Util2d:rech_304: resetting 'how' to external
Util2d:rech_305: resetting 'how' to external
Util2d:rech_306: resetting 'how' to external
Util2d:rech_307: resetting 'how' to external
Util2d:rech_308: resetting 'how' to external
Util2d:rech_309: resetting 'how' to external
Util2d:rech_310: resetting 'how' to external
Util2d:rech_311: resetting 'how' to external
Util2d:rech_312: resetting 'how' to external
Util2d:rech_313: resetting 'how' to external
Util2d:rech_314: resetting 'how' to external
Util2d:rech_315: resetting 'how' to external
Util2d:rech_316: resetting 'how' to external
Util2d:rech_317: resetting 'how' to external
Util2d:rech_318: resetting 'how' to external
Util2d:rech_319: resetting 'how' to external
Util2d:rech_320: resetting 'how' to external
Util2d:rech_321: resetting 'how' to external
Util2d:rech_322: resetting 'how' to external
Util2d:rech_323: resetting 'how' to external
Util2d:rech_324: resetting 'how' to external
Util2d:rech_325: resetting 'how' to external
Util2d:rech_326: resetting 'how' to external
Util2d:rech_327: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```

Util2d:rech_328: resetting 'how' to external
Util2d:rech_329: resetting 'how' to external
Util2d:rech_330: resetting 'how' to external
Util2d:rech_331: resetting 'how' to external
Util2d:rech_332: resetting 'how' to external
Util2d:rech_333: resetting 'how' to external
Util2d:rech_334: resetting 'how' to external
Util2d:rech_335: resetting 'how' to external
Util2d:rech_336: resetting 'how' to external
Util2d:rech_337: resetting 'how' to external
Util2d:rech_338: resetting 'how' to external
Util2d:rech_339: resetting 'how' to external
Util2d:rech_340: resetting 'how' to external
Util2d:rech_341: resetting 'how' to external
Util2d:rech_342: resetting 'how' to external
Util2d:rech_343: resetting 'how' to external
Util2d:rech_344: resetting 'how' to external
Util2d:rech_345: resetting 'how' to external
Util2d:rech_346: resetting 'how' to external
Util2d:rech_347: resetting 'how' to external
Util2d:rech_348: resetting 'how' to external
Util2d:rech_349: resetting 'how' to external
Util2d:rech_350: resetting 'how' to external
Util2d:rech_351: resetting 'how' to external
Util2d:rech_352: resetting 'how' to external
Util2d:rech_353: resetting 'how' to external
Util2d:rech_354: resetting 'how' to external
Util2d:rech_355: resetting 'how' to external
Util2d:rech_356: resetting 'how' to external
Util2d:rech_357: resetting 'how' to external
Util2d:rech_358: resetting 'how' to external
Util2d:rech_359: resetting 'how' to external
Util2d:rech_360: resetting 'how' to external
Util2d:rech_361: resetting 'how' to external
Util2d:rech_362: resetting 'how' to external
Util2d:rech_363: resetting 'how' to external
Util2d:rech_364: resetting 'how' to external
Util2d:rech_365: resetting 'how' to external
Util2d:rech_366: resetting 'how' to external
Util2d:rech_367: resetting 'how' to external
Util2d:rech_368: resetting 'how' to external
Util2d:rech_369: resetting 'how' to external
Util2d:rech_370: resetting 'how' to external
Util2d:rech_371: resetting 'how' to external
Util2d:rech_372: resetting 'how' to external
Util2d:rech_373: resetting 'how' to external
Util2d:rech_374: resetting 'how' to external
Util2d:rech_375: resetting 'how' to external
Util2d:rech_376: resetting 'how' to external
Util2d:rech_377: resetting 'how' to external
Util2d:rech_378: resetting 'how' to external
Util2d:rech_379: resetting 'how' to external

```

(continues on next page)

(continued from previous page)

```
Util2d:rech_380: resetting 'how' to external
Util2d:rech_381: resetting 'how' to external
Util2d:rech_382: resetting 'how' to external
Util2d:rech_383: resetting 'how' to external
Util2d:rech_384: resetting 'how' to external
Util2d:rech_385: resetting 'how' to external
Util2d:rech_386: resetting 'how' to external
Util2d:rech_387: resetting 'how' to external
Util2d:rech_388: resetting 'how' to external
Util2d:rech_389: resetting 'how' to external
Util2d:rech_390: resetting 'how' to external
Util2d:rech_391: resetting 'how' to external
Util2d:rech_392: resetting 'how' to external
Util2d:rech_393: resetting 'how' to external
Util2d:rech_394: resetting 'how' to external
Util2d:rech_395: resetting 'how' to external
Util2d:rech_396: resetting 'how' to external
Util2d:rech_397: resetting 'how' to external
Util2d:rech_398: resetting 'how' to external
Util2d:rech_399: resetting 'how' to external
Util2d:rech_400: resetting 'how' to external
Util2d:rech_401: resetting 'how' to external
Util2d:rech_402: resetting 'how' to external
Util2d:rech_403: resetting 'how' to external
Util2d:rech_404: resetting 'how' to external
Util2d:rech_405: resetting 'how' to external
Util2d:rech_406: resetting 'how' to external
Util2d:rech_407: resetting 'how' to external
Util2d:rech_408: resetting 'how' to external
Util2d:rech_409: resetting 'how' to external
Util2d:rech_410: resetting 'how' to external
Util2d:rech_411: resetting 'how' to external
Util2d:rech_412: resetting 'how' to external
Util2d:rech_413: resetting 'how' to external
Util2d:rech_414: resetting 'how' to external
Util2d:rech_415: resetting 'how' to external
Util2d:rech_416: resetting 'how' to external
Util2d:rech_417: resetting 'how' to external
Util2d:rech_418: resetting 'how' to external
Util2d:rech_419: resetting 'how' to external
Util2d:rech_420: resetting 'how' to external
Util2d:rech_421: resetting 'how' to external
Util2d:rech_422: resetting 'how' to external
Util2d:rech_423: resetting 'how' to external
Util2d:rech_424: resetting 'how' to external
Util2d:rech_425: resetting 'how' to external
Util2d:rech_426: resetting 'how' to external
Util2d:rech_427: resetting 'how' to external
Util2d:rech_428: resetting 'how' to external
Util2d:rech_429: resetting 'how' to external
Util2d:rech_430: resetting 'how' to external
Util2d:rech_431: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```

Util2d:rech_432: resetting 'how' to external
Util2d:rech_433: resetting 'how' to external
Util2d:rech_434: resetting 'how' to external
Util2d:rech_435: resetting 'how' to external
Util2d:rech_436: resetting 'how' to external
Util2d:rech_437: resetting 'how' to external
Util2d:rech_438: resetting 'how' to external
Util2d:rech_439: resetting 'how' to external
Util2d:rech_440: resetting 'how' to external
Util2d:rech_441: resetting 'how' to external
Util2d:rech_442: resetting 'how' to external
Util2d:rech_443: resetting 'how' to external
Util2d:rech_444: resetting 'how' to external
Util2d:rech_445: resetting 'how' to external
Util2d:rech_446: resetting 'how' to external
Util2d:rech_447: resetting 'how' to external
Util2d:rech_448: resetting 'how' to external
Util2d:rech_449: resetting 'how' to external
Util2d:rech_450: resetting 'how' to external
Util2d:rech_451: resetting 'how' to external
Util2d:rech_452: resetting 'how' to external
Util2d:rech_453: resetting 'how' to external
Util2d:rech_454: resetting 'how' to external
Util2d:rech_455: resetting 'how' to external
Util2d:rech_456: resetting 'how' to external
Util2d:rech_457: resetting 'how' to external
Util2d:rech_458: resetting 'how' to external
Util2d:rech_459: resetting 'how' to external
Util2d:rech_460: resetting 'how' to external
Util2d:rech_461: resetting 'how' to external
Util2d:rech_462: resetting 'how' to external
Util2d:rech_463: resetting 'how' to external
Util2d:rech_464: resetting 'how' to external
Util2d:rech_465: resetting 'how' to external
Util2d:rech_466: resetting 'how' to external
Util2d:rech_467: resetting 'how' to external
Util2d:rech_468: resetting 'how' to external
Util2d:rech_469: resetting 'how' to external
Util2d:rech_470: resetting 'how' to external
Util2d:rech_471: resetting 'how' to external
Util2d:rech_472: resetting 'how' to external
Util2d:rech_473: resetting 'how' to external
Util2d:rech_474: resetting 'how' to external
Util2d:rech_475: resetting 'how' to external
Util2d:rech_476: resetting 'how' to external
Util2d:rech_477: resetting 'how' to external
Util2d:rech_478: resetting 'how' to external
Util2d:rech_479: resetting 'how' to external
Util2d:rech_480: resetting 'how' to external
Util2d:rech_481: resetting 'how' to external
Util2d:rech_482: resetting 'how' to external
Util2d:rech_483: resetting 'how' to external

```

(continues on next page)

(continued from previous page)

```
Util2d:rech_484: resetting 'how' to external
Util2d:rech_485: resetting 'how' to external
Util2d:rech_486: resetting 'how' to external
Util2d:rech_487: resetting 'how' to external
Util2d:rech_488: resetting 'how' to external
Util2d:rech_489: resetting 'how' to external
Util2d:rech_490: resetting 'how' to external
Util2d:rech_491: resetting 'how' to external
Util2d:rech_492: resetting 'how' to external
Util2d:rech_493: resetting 'how' to external
Util2d:rech_494: resetting 'how' to external
Util2d:rech_495: resetting 'how' to external
Util2d:rech_496: resetting 'how' to external
Util2d:rech_497: resetting 'how' to external
Util2d:rech_498: resetting 'how' to external
Util2d:rech_499: resetting 'how' to external
Util2d:rech_500: resetting 'how' to external
Util2d:rech_501: resetting 'how' to external
Util2d:rech_502: resetting 'how' to external
Util2d:rech_503: resetting 'how' to external
Util2d:rech_504: resetting 'how' to external
Util2d:rech_505: resetting 'how' to external
Util2d:rech_506: resetting 'how' to external
Util2d:rech_507: resetting 'how' to external
Util2d:rech_508: resetting 'how' to external
Util2d:rech_509: resetting 'how' to external
Util2d:rech_510: resetting 'how' to external
Util2d:rech_511: resetting 'how' to external
Util2d:rech_512: resetting 'how' to external
Util2d:rech_513: resetting 'how' to external
Util2d:rech_514: resetting 'how' to external
Util2d:rech_515: resetting 'how' to external
Util2d:rech_516: resetting 'how' to external
Util2d:rech_517: resetting 'how' to external
Util2d:rech_518: resetting 'how' to external
Util2d:rech_519: resetting 'how' to external
Util2d:rech_520: resetting 'how' to external
Util2d:rech_521: resetting 'how' to external
Util2d:rech_522: resetting 'how' to external
Util2d:rech_523: resetting 'how' to external
Util2d:rech_524: resetting 'how' to external
Util2d:rech_525: resetting 'how' to external
Util2d:rech_526: resetting 'how' to external
Util2d:rech_527: resetting 'how' to external
Util2d:rech_528: resetting 'how' to external
Util2d:rech_529: resetting 'how' to external
Util2d:rech_530: resetting 'how' to external
Util2d:rech_531: resetting 'how' to external
Util2d:rech_532: resetting 'how' to external
Util2d:rech_533: resetting 'how' to external
Util2d:rech_534: resetting 'how' to external
Util2d:rech_535: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_536: resetting 'how' to external
Util2d:rech_537: resetting 'how' to external
Util2d:rech_538: resetting 'how' to external
Util2d:rech_539: resetting 'how' to external
Util2d:rech_540: resetting 'how' to external
Util2d:rech_541: resetting 'how' to external
Util2d:rech_542: resetting 'how' to external
Util2d:rech_543: resetting 'how' to external
Util2d:rech_544: resetting 'how' to external
Util2d:rech_545: resetting 'how' to external
Util2d:rech_546: resetting 'how' to external
Util2d:rech_547: resetting 'how' to external
Util2d:rech_548: resetting 'how' to external
Util2d:rech_549: resetting 'how' to external
Util2d:rech_550: resetting 'how' to external
Util2d:rech_551: resetting 'how' to external
Util2d:rech_552: resetting 'how' to external
Util2d:rech_553: resetting 'how' to external
Util2d:rech_554: resetting 'how' to external
Util2d:rech_555: resetting 'how' to external
Util2d:rech_556: resetting 'how' to external
Util2d:rech_557: resetting 'how' to external
Util2d:rech_558: resetting 'how' to external
Util2d:rech_559: resetting 'how' to external
Util2d:rech_560: resetting 'how' to external
Util2d:rech_561: resetting 'how' to external
Util2d:rech_562: resetting 'how' to external
Util2d:rech_563: resetting 'how' to external
Util2d:rech_564: resetting 'how' to external
Util2d:rech_565: resetting 'how' to external
Util2d:rech_566: resetting 'how' to external
Util2d:rech_567: resetting 'how' to external
Util2d:rech_568: resetting 'how' to external
Util2d:rech_569: resetting 'how' to external
Util2d:rech_570: resetting 'how' to external
Util2d:rech_571: resetting 'how' to external
Util2d:rech_572: resetting 'how' to external
Util2d:rech_573: resetting 'how' to external
Util2d:rech_574: resetting 'how' to external
Util2d:rech_575: resetting 'how' to external
Util2d:rech_576: resetting 'how' to external
Util2d:rech_577: resetting 'how' to external
Util2d:rech_578: resetting 'how' to external
Util2d:rech_579: resetting 'how' to external
Util2d:rech_580: resetting 'how' to external
Util2d:rech_581: resetting 'how' to external
Util2d:rech_582: resetting 'how' to external
Util2d:rech_583: resetting 'how' to external
Util2d:rech_584: resetting 'how' to external
Util2d:rech_585: resetting 'how' to external
Util2d:rech_586: resetting 'how' to external
Util2d:rech_587: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_588: resetting 'how' to external
Util2d:rech_589: resetting 'how' to external
Util2d:rech_590: resetting 'how' to external
Util2d:rech_591: resetting 'how' to external
Util2d:rech_592: resetting 'how' to external
Util2d:rech_593: resetting 'how' to external
Util2d:rech_594: resetting 'how' to external
Util2d:rech_595: resetting 'how' to external
Util2d:rech_596: resetting 'how' to external
Util2d:rech_597: resetting 'how' to external
Util2d:rech_598: resetting 'how' to external
Util2d:rech_599: resetting 'how' to external
Util2d:rech_600: resetting 'how' to external
Util2d:rech_601: resetting 'how' to external
Util2d:rech_602: resetting 'how' to external
Util2d:rech_603: resetting 'how' to external
Util2d:rech_604: resetting 'how' to external
Util2d:rech_605: resetting 'how' to external
Util2d:rech_606: resetting 'how' to external
Util2d:rech_607: resetting 'how' to external
Util2d:rech_608: resetting 'how' to external
Util2d:rech_609: resetting 'how' to external
Util2d:rech_610: resetting 'how' to external
Util2d:rech_611: resetting 'how' to external
Util2d:rech_612: resetting 'how' to external
Util2d:rech_613: resetting 'how' to external
Util2d:rech_614: resetting 'how' to external
Util2d:rech_615: resetting 'how' to external
Util2d:rech_616: resetting 'how' to external
Util2d:rech_617: resetting 'how' to external
Util2d:rech_618: resetting 'how' to external
Util2d:rech_619: resetting 'how' to external
Util2d:rech_620: resetting 'how' to external
Util2d:rech_621: resetting 'how' to external
Util2d:rech_622: resetting 'how' to external
Util2d:rech_623: resetting 'how' to external
Util2d:rech_624: resetting 'how' to external
Util2d:rech_625: resetting 'how' to external
Util2d:rech_626: resetting 'how' to external
Util2d:rech_627: resetting 'how' to external
Util2d:rech_628: resetting 'how' to external
Util2d:rech_629: resetting 'how' to external
Util2d:rech_630: resetting 'how' to external
Util2d:rech_631: resetting 'how' to external
Util2d:rech_632: resetting 'how' to external
Util2d:rech_633: resetting 'how' to external
Util2d:rech_634: resetting 'how' to external
Util2d:rech_635: resetting 'how' to external
Util2d:rech_636: resetting 'how' to external
Util2d:rech_637: resetting 'how' to external
Util2d:rech_638: resetting 'how' to external
Util2d:rech_639: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_640: resetting 'how' to external
Util2d:rech_641: resetting 'how' to external
Util2d:rech_642: resetting 'how' to external
Util2d:rech_643: resetting 'how' to external
Util2d:rech_644: resetting 'how' to external
Util2d:rech_645: resetting 'how' to external
Util2d:rech_646: resetting 'how' to external
Util2d:rech_647: resetting 'how' to external
Util2d:rech_648: resetting 'how' to external
Util2d:rech_649: resetting 'how' to external
Util2d:rech_650: resetting 'how' to external
Util2d:rech_651: resetting 'how' to external
Util2d:rech_652: resetting 'how' to external
Util2d:rech_653: resetting 'how' to external
Util2d:rech_654: resetting 'how' to external
Util2d:rech_655: resetting 'how' to external
Util2d:rech_656: resetting 'how' to external
Util2d:rech_657: resetting 'how' to external
Util2d:rech_658: resetting 'how' to external
Util2d:rech_659: resetting 'how' to external
Util2d:rech_660: resetting 'how' to external
Util2d:rech_661: resetting 'how' to external
Util2d:rech_662: resetting 'how' to external
Util2d:rech_663: resetting 'how' to external
Util2d:rech_664: resetting 'how' to external
Util2d:rech_665: resetting 'how' to external
Util2d:rech_666: resetting 'how' to external
Util2d:rech_667: resetting 'how' to external
Util2d:rech_668: resetting 'how' to external
Util2d:rech_669: resetting 'how' to external
Util2d:rech_670: resetting 'how' to external
Util2d:rech_671: resetting 'how' to external
Util2d:rech_672: resetting 'how' to external
Util2d:rech_673: resetting 'how' to external
Util2d:rech_674: resetting 'how' to external
Util2d:rech_675: resetting 'how' to external
Util2d:rech_676: resetting 'how' to external
Util2d:rech_677: resetting 'how' to external
Util2d:rech_678: resetting 'how' to external
Util2d:rech_679: resetting 'how' to external
Util2d:rech_680: resetting 'how' to external
Util2d:rech_681: resetting 'how' to external
Util2d:rech_682: resetting 'how' to external
Util2d:rech_683: resetting 'how' to external
Util2d:rech_684: resetting 'how' to external
Util2d:rech_685: resetting 'how' to external
Util2d:rech_686: resetting 'how' to external
Util2d:rech_687: resetting 'how' to external
Util2d:rech_688: resetting 'how' to external
Util2d:rech_689: resetting 'how' to external
Util2d:rech_690: resetting 'how' to external
Util2d:rech_691: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_692: resetting 'how' to external
Util2d:rech_693: resetting 'how' to external
Util2d:rech_694: resetting 'how' to external
Util2d:rech_695: resetting 'how' to external
Util2d:rech_696: resetting 'how' to external
Util2d:rech_697: resetting 'how' to external
Util2d:rech_698: resetting 'how' to external
Util2d:rech_699: resetting 'how' to external
Util2d:rech_700: resetting 'how' to external
Util2d:rech_701: resetting 'how' to external
Util2d:rech_702: resetting 'how' to external
Util2d:rech_703: resetting 'how' to external
Util2d:rech_704: resetting 'how' to external
Util2d:rech_705: resetting 'how' to external
Util2d:rech_706: resetting 'how' to external
Util2d:rech_707: resetting 'how' to external
Util2d:rech_708: resetting 'how' to external
Util2d:rech_709: resetting 'how' to external
Util2d:rech_710: resetting 'how' to external
Util2d:rech_711: resetting 'how' to external
Util2d:rech_712: resetting 'how' to external
Util2d:rech_713: resetting 'how' to external
Util2d:rech_714: resetting 'how' to external
Util2d:rech_715: resetting 'how' to external
Util2d:rech_716: resetting 'how' to external
Util2d:rech_717: resetting 'how' to external
Util2d:rech_718: resetting 'how' to external
Util2d:rech_719: resetting 'how' to external
Util2d:rech_720: resetting 'how' to external
Util2d:rech_721: resetting 'how' to external
Util2d:rech_722: resetting 'how' to external
Util2d:rech_723: resetting 'how' to external
Util2d:rech_724: resetting 'how' to external
Util2d:rech_725: resetting 'how' to external
Util2d:rech_726: resetting 'how' to external
Util2d:rech_727: resetting 'how' to external
Util2d:rech_728: resetting 'how' to external
Util2d:rech_729: resetting 'how' to external
Util2d:rech_730: resetting 'how' to external
Util2d:rech_731: resetting 'how' to external
Util2d:rech_732: resetting 'how' to external
Util2d:rech_733: resetting 'how' to external
Util2d:rech_734: resetting 'how' to external
Util2d:rech_735: resetting 'how' to external
Util2d:rech_736: resetting 'how' to external
Util2d:rech_737: resetting 'how' to external
Util2d:rech_738: resetting 'how' to external
Util2d:rech_739: resetting 'how' to external
Util2d:rech_740: resetting 'how' to external
Util2d:rech_741: resetting 'how' to external
Util2d:rech_742: resetting 'how' to external
Util2d:rech_743: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_744: resetting 'how' to external
Util2d:rech_745: resetting 'how' to external
Util2d:rech_746: resetting 'how' to external
Util2d:rech_747: resetting 'how' to external
Util2d:rech_748: resetting 'how' to external
Util2d:rech_749: resetting 'how' to external
Util2d:rech_750: resetting 'how' to external
Util2d:rech_751: resetting 'how' to external
Util2d:rech_752: resetting 'how' to external
Util2d:rech_753: resetting 'how' to external
Util2d:rech_754: resetting 'how' to external
Util2d:rech_755: resetting 'how' to external
Util2d:rech_756: resetting 'how' to external
Util2d:rech_757: resetting 'how' to external
Util2d:rech_758: resetting 'how' to external
Util2d:rech_759: resetting 'how' to external
Util2d:rech_760: resetting 'how' to external
Util2d:rech_761: resetting 'how' to external
Util2d:rech_762: resetting 'how' to external
Util2d:rech_763: resetting 'how' to external
Util2d:rech_764: resetting 'how' to external
Util2d:rech_765: resetting 'how' to external
Util2d:rech_766: resetting 'how' to external
Util2d:rech_767: resetting 'how' to external
Util2d:rech_768: resetting 'how' to external
Util2d:rech_769: resetting 'how' to external
Util2d:rech_770: resetting 'how' to external
Util2d:rech_771: resetting 'how' to external
Util2d:rech_772: resetting 'how' to external
Util2d:rech_773: resetting 'how' to external
Util2d:rech_774: resetting 'how' to external
Util2d:rech_775: resetting 'how' to external
Util2d:rech_776: resetting 'how' to external
Util2d:rech_777: resetting 'how' to external
Util2d:rech_778: resetting 'how' to external
Util2d:rech_779: resetting 'how' to external
Util2d:rech_780: resetting 'how' to external
Util2d:rech_781: resetting 'how' to external
Util2d:rech_782: resetting 'how' to external
Util2d:rech_783: resetting 'how' to external
Util2d:rech_784: resetting 'how' to external
Util2d:rech_785: resetting 'how' to external
Util2d:rech_786: resetting 'how' to external
Util2d:rech_787: resetting 'how' to external
Util2d:rech_788: resetting 'how' to external
Util2d:rech_789: resetting 'how' to external
Util2d:rech_790: resetting 'how' to external
Util2d:rech_791: resetting 'how' to external
Util2d:rech_792: resetting 'how' to external
Util2d:rech_793: resetting 'how' to external
Util2d:rech_794: resetting 'how' to external
Util2d:rech_795: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_796: resetting 'how' to external
Util2d:rech_797: resetting 'how' to external
Util2d:rech_798: resetting 'how' to external
Util2d:rech_799: resetting 'how' to external
Util2d:rech_800: resetting 'how' to external
Util2d:rech_801: resetting 'how' to external
Util2d:rech_802: resetting 'how' to external
Util2d:rech_803: resetting 'how' to external
Util2d:rech_804: resetting 'how' to external
Util2d:rech_805: resetting 'how' to external
Util2d:rech_806: resetting 'how' to external
Util2d:rech_807: resetting 'how' to external
Util2d:rech_808: resetting 'how' to external
Util2d:rech_809: resetting 'how' to external
Util2d:rech_810: resetting 'how' to external
Util2d:rech_811: resetting 'how' to external
Util2d:rech_812: resetting 'how' to external
Util2d:rech_813: resetting 'how' to external
Util2d:rech_814: resetting 'how' to external
Util2d:rech_815: resetting 'how' to external
Util2d:rech_816: resetting 'how' to external
Util2d:rech_817: resetting 'how' to external
Util2d:rech_818: resetting 'how' to external
Util2d:rech_819: resetting 'how' to external
Util2d:rech_820: resetting 'how' to external
Util2d:rech_821: resetting 'how' to external
Util2d:rech_822: resetting 'how' to external
Util2d:rech_823: resetting 'how' to external
Util2d:rech_824: resetting 'how' to external
Util2d:rech_825: resetting 'how' to external
Util2d:rech_826: resetting 'how' to external
Util2d:rech_827: resetting 'how' to external
Util2d:rech_828: resetting 'how' to external
Util2d:rech_829: resetting 'how' to external
Util2d:rech_830: resetting 'how' to external
Util2d:rech_831: resetting 'how' to external
Util2d:rech_832: resetting 'how' to external
Util2d:rech_833: resetting 'how' to external
Util2d:rech_834: resetting 'how' to external
Util2d:rech_835: resetting 'how' to external
Util2d:rech_836: resetting 'how' to external
Util2d:rech_837: resetting 'how' to external
Util2d:rech_838: resetting 'how' to external
Util2d:rech_839: resetting 'how' to external
Util2d:rech_840: resetting 'how' to external
Util2d:rech_841: resetting 'how' to external
Util2d:rech_842: resetting 'how' to external
Util2d:rech_843: resetting 'how' to external
Util2d:rech_844: resetting 'how' to external
Util2d:rech_845: resetting 'how' to external
Util2d:rech_846: resetting 'how' to external
Util2d:rech_847: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_848: resetting 'how' to external
Util2d:rech_849: resetting 'how' to external
Util2d:rech_850: resetting 'how' to external
Util2d:rech_851: resetting 'how' to external
Util2d:rech_852: resetting 'how' to external
Util2d:rech_853: resetting 'how' to external
Util2d:rech_854: resetting 'how' to external
Util2d:rech_855: resetting 'how' to external
Util2d:rech_856: resetting 'how' to external
Util2d:rech_857: resetting 'how' to external
Util2d:rech_858: resetting 'how' to external
Util2d:rech_859: resetting 'how' to external
Util2d:rech_860: resetting 'how' to external
Util2d:rech_861: resetting 'how' to external
Util2d:rech_862: resetting 'how' to external
Util2d:rech_863: resetting 'how' to external
Util2d:rech_864: resetting 'how' to external
Util2d:rech_865: resetting 'how' to external
Util2d:rech_866: resetting 'how' to external
Util2d:rech_867: resetting 'how' to external
Util2d:rech_868: resetting 'how' to external
Util2d:rech_869: resetting 'how' to external
Util2d:rech_870: resetting 'how' to external
Util2d:rech_871: resetting 'how' to external
Util2d:rech_872: resetting 'how' to external
Util2d:rech_873: resetting 'how' to external
Util2d:rech_874: resetting 'how' to external
Util2d:rech_875: resetting 'how' to external
Util2d:rech_876: resetting 'how' to external
Util2d:rech_877: resetting 'how' to external
Util2d:rech_878: resetting 'how' to external
Util2d:rech_879: resetting 'how' to external
Util2d:rech_880: resetting 'how' to external
Util2d:rech_881: resetting 'how' to external
Util2d:rech_882: resetting 'how' to external
Util2d:rech_883: resetting 'how' to external
Util2d:rech_884: resetting 'how' to external
Util2d:rech_885: resetting 'how' to external
Util2d:rech_886: resetting 'how' to external
Util2d:rech_887: resetting 'how' to external
Util2d:rech_888: resetting 'how' to external
Util2d:rech_889: resetting 'how' to external
Util2d:rech_890: resetting 'how' to external
Util2d:rech_891: resetting 'how' to external
Util2d:rech_892: resetting 'how' to external
Util2d:rech_893: resetting 'how' to external
Util2d:rech_894: resetting 'how' to external
Util2d:rech_895: resetting 'how' to external
Util2d:rech_896: resetting 'how' to external
Util2d:rech_897: resetting 'how' to external
Util2d:rech_898: resetting 'how' to external
Util2d:rech_899: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_900: resetting 'how' to external
Util2d:rech_901: resetting 'how' to external
Util2d:rech_902: resetting 'how' to external
Util2d:rech_903: resetting 'how' to external
Util2d:rech_904: resetting 'how' to external
Util2d:rech_905: resetting 'how' to external
Util2d:rech_906: resetting 'how' to external
Util2d:rech_907: resetting 'how' to external
Util2d:rech_908: resetting 'how' to external
Util2d:rech_909: resetting 'how' to external
Util2d:rech_910: resetting 'how' to external
Util2d:rech_911: resetting 'how' to external
Util2d:rech_912: resetting 'how' to external
Util2d:rech_913: resetting 'how' to external
Util2d:rech_914: resetting 'how' to external
Util2d:rech_915: resetting 'how' to external
Util2d:rech_916: resetting 'how' to external
Util2d:rech_917: resetting 'how' to external
Util2d:rech_918: resetting 'how' to external
Util2d:rech_919: resetting 'how' to external
Util2d:rech_920: resetting 'how' to external
Util2d:rech_921: resetting 'how' to external
Util2d:rech_922: resetting 'how' to external
Util2d:rech_923: resetting 'how' to external
Util2d:rech_924: resetting 'how' to external
Util2d:rech_925: resetting 'how' to external
Util2d:rech_926: resetting 'how' to external
Util2d:rech_927: resetting 'how' to external
Util2d:rech_928: resetting 'how' to external
Util2d:rech_929: resetting 'how' to external
Util2d:rech_930: resetting 'how' to external
Util2d:rech_931: resetting 'how' to external
Util2d:rech_932: resetting 'how' to external
Util2d:rech_933: resetting 'how' to external
Util2d:rech_934: resetting 'how' to external
Util2d:rech_935: resetting 'how' to external
Util2d:rech_936: resetting 'how' to external
Util2d:rech_937: resetting 'how' to external
Util2d:rech_938: resetting 'how' to external
Util2d:rech_939: resetting 'how' to external
Util2d:rech_940: resetting 'how' to external
Util2d:rech_941: resetting 'how' to external
Util2d:rech_942: resetting 'how' to external
Util2d:rech_943: resetting 'how' to external
Util2d:rech_944: resetting 'how' to external
Util2d:rech_945: resetting 'how' to external
Util2d:rech_946: resetting 'how' to external
Util2d:rech_947: resetting 'how' to external
Util2d:rech_948: resetting 'how' to external
Util2d:rech_949: resetting 'how' to external
Util2d:rech_950: resetting 'how' to external
Util2d:rech_951: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_952: resetting 'how' to external
Util2d:rech_953: resetting 'how' to external
Util2d:rech_954: resetting 'how' to external
Util2d:rech_955: resetting 'how' to external
Util2d:rech_956: resetting 'how' to external
Util2d:rech_957: resetting 'how' to external
Util2d:rech_958: resetting 'how' to external
Util2d:rech_959: resetting 'how' to external
Util2d:rech_960: resetting 'how' to external
Util2d:rech_961: resetting 'how' to external
Util2d:rech_962: resetting 'how' to external
Util2d:rech_963: resetting 'how' to external
Util2d:rech_964: resetting 'how' to external
Util2d:rech_965: resetting 'how' to external
Util2d:rech_966: resetting 'how' to external
Util2d:rech_967: resetting 'how' to external
Util2d:rech_968: resetting 'how' to external
Util2d:rech_969: resetting 'how' to external
Util2d:rech_970: resetting 'how' to external
Util2d:rech_971: resetting 'how' to external
Util2d:rech_972: resetting 'how' to external
Util2d:rech_973: resetting 'how' to external
Util2d:rech_974: resetting 'how' to external
Util2d:rech_975: resetting 'how' to external
Util2d:rech_976: resetting 'how' to external
Util2d:rech_977: resetting 'how' to external
Util2d:rech_978: resetting 'how' to external
Util2d:rech_979: resetting 'how' to external
Util2d:rech_980: resetting 'how' to external
Util2d:rech_981: resetting 'how' to external
Util2d:rech_982: resetting 'how' to external
Util2d:rech_983: resetting 'how' to external
Util2d:rech_984: resetting 'how' to external
Util2d:rech_985: resetting 'how' to external
Util2d:rech_986: resetting 'how' to external
Util2d:rech_987: resetting 'how' to external
Util2d:rech_988: resetting 'how' to external
Util2d:rech_989: resetting 'how' to external
Util2d:rech_990: resetting 'how' to external
Util2d:rech_991: resetting 'how' to external
Util2d:rech_992: resetting 'how' to external
Util2d:rech_993: resetting 'how' to external
Util2d:rech_994: resetting 'how' to external
Util2d:rech_995: resetting 'how' to external
Util2d:rech_996: resetting 'how' to external
Util2d:rech_997: resetting 'how' to external
Util2d:rech_998: resetting 'how' to external
Util2d:rech_999: resetting 'how' to external
Util2d:rech_1000: resetting 'how' to external
Util2d:rech_1001: resetting 'how' to external
Util2d:rech_1002: resetting 'how' to external
Util2d:rech_1003: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_1004: resetting 'how' to external
Util2d:rech_1005: resetting 'how' to external
Util2d:rech_1006: resetting 'how' to external
Util2d:rech_1007: resetting 'how' to external
Util2d:rech_1008: resetting 'how' to external
Util2d:rech_1009: resetting 'how' to external
Util2d:rech_1010: resetting 'how' to external
Util2d:rech_1011: resetting 'how' to external
Util2d:rech_1012: resetting 'how' to external
Util2d:rech_1013: resetting 'how' to external
Util2d:rech_1014: resetting 'how' to external
Util2d:rech_1015: resetting 'how' to external
Util2d:rech_1016: resetting 'how' to external
Util2d:rech_1017: resetting 'how' to external
Util2d:rech_1018: resetting 'how' to external
Util2d:rech_1019: resetting 'how' to external
Util2d:rech_1020: resetting 'how' to external
Util2d:rech_1021: resetting 'how' to external
Util2d:rech_1022: resetting 'how' to external
Util2d:rech_1023: resetting 'how' to external
Util2d:rech_1024: resetting 'how' to external
Util2d:rech_1025: resetting 'how' to external
Util2d:rech_1026: resetting 'how' to external
Util2d:rech_1027: resetting 'how' to external
Util2d:rech_1028: resetting 'how' to external
Util2d:rech_1029: resetting 'how' to external
Util2d:rech_1030: resetting 'how' to external
Util2d:rech_1031: resetting 'how' to external
Util2d:rech_1032: resetting 'how' to external
Util2d:rech_1033: resetting 'how' to external
Util2d:rech_1034: resetting 'how' to external
Util2d:rech_1035: resetting 'how' to external
Util2d:rech_1036: resetting 'how' to external
Util2d:rech_1037: resetting 'how' to external
Util2d:rech_1038: resetting 'how' to external
Util2d:rech_1039: resetting 'how' to external
Util2d:rech_1040: resetting 'how' to external
Util2d:rech_1041: resetting 'how' to external
Util2d:rech_1042: resetting 'how' to external
Util2d:rech_1043: resetting 'how' to external
Util2d:rech_1044: resetting 'how' to external
Util2d:rech_1045: resetting 'how' to external
Util2d:rech_1046: resetting 'how' to external
Util2d:rech_1047: resetting 'how' to external
Util2d:rech_1048: resetting 'how' to external
Util2d:rech_1049: resetting 'how' to external
Util2d:rech_1050: resetting 'how' to external
Util2d:rech_1051: resetting 'how' to external
Util2d:rech_1052: resetting 'how' to external
Util2d:rech_1053: resetting 'how' to external
Util2d:rech_1054: resetting 'how' to external
Util2d:rech_1055: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:rech_1056: resetting 'how' to external
Util2d:rech_1057: resetting 'how' to external
Util2d:rech_1058: resetting 'how' to external
Util2d:rech_1059: resetting 'how' to external
Util2d:rech_1060: resetting 'how' to external
Util2d:rech_1061: resetting 'how' to external
Util2d:rech_1062: resetting 'how' to external
Util2d:rech_1063: resetting 'how' to external
Util2d:rech_1064: resetting 'how' to external
Util2d:rech_1065: resetting 'how' to external
Util2d:rech_1066: resetting 'how' to external
Util2d:rech_1067: resetting 'how' to external
Util2d:rech_1068: resetting 'how' to external
Util2d:rech_1069: resetting 'how' to external
Util2d:rech_1070: resetting 'how' to external
Util2d:rech_1071: resetting 'how' to external
Util2d:rech_1072: resetting 'how' to external
Util2d:rech_1073: resetting 'how' to external
Util2d:rech_1074: resetting 'how' to external
Util2d:rech_1075: resetting 'how' to external
Util2d:rech_1076: resetting 'how' to external
Util2d:rech_1077: resetting 'how' to external
Util2d:rech_1078: resetting 'how' to external
Util2d:rech_1079: resetting 'how' to external
Util2d:rech_1080: resetting 'how' to external
Util2d:rech_1081: resetting 'how' to external
Util2d:rech_1082: resetting 'how' to external
Util2d:rech_1083: resetting 'how' to external
Util2d:rech_1084: resetting 'how' to external
Util2d:rech_1085: resetting 'how' to external
Util2d:rech_1086: resetting 'how' to external
Util2d:rech_1087: resetting 'how' to external
Util2d:rech_1088: resetting 'how' to external
Util2d:rech_1089: resetting 'how' to external
Util2d:rech_1090: resetting 'how' to external
Util2d:rech_1091: resetting 'how' to external
Util2d:rech_1092: resetting 'how' to external
Util2d:rech_1093: resetting 'how' to external
Util2d:rech_1094: resetting 'how' to external
Util2d:rech_1095: resetting 'how' to external
Util2d:rech_1096: resetting 'how' to external
Util2d:rech_1097: resetting 'how' to external
Util2d:hk: resetting 'how' to external
Util2d:vk: resetting 'how' to external
Util2d:ss: resetting 'how' to external
Util2d:sy: resetting 'how' to external
Util2d:hk: resetting 'how' to external
Util2d:vk: resetting 'how' to external
Util2d:ss: resetting 'how' to external
Util2d:sy: resetting 'how' to external
Util2d:hk: resetting 'how' to external
Util2d:vk: resetting 'how' to external
```

(continues on next page)

(continued from previous page)

```
Util2d:ss: resetting 'how' to external
Util2d:sy: resetting 'how' to external
```

Now run the model.

```
[13]: ml.run_model(silent=True)
```

```
[13]: (True, [])
```

Inspecting outputs

First, let's look at the list file. The list file summarizes the model's results.

```
[14]: mfl = flopy.utils.MfListBudget(model_ws / "freyberg.list")
df_flux, df_vol = mfl.get_dataframes(start_datetime="10-21-2015")
df_flux
```

```
[14]:
```

	STORAGE_IN	CONSTANT_HEAD_IN	WELLS_IN	DRAINS_IN	RECHARGE_IN	\
2015-10-22	0.000000	0.000000	0.0	0.0	6276.861816	
2015-10-23	635.447998	0.000000	0.0	0.0	6428.198730	
2015-10-24	1361.814941	0.000000	0.0	0.0	5397.295898	
2015-10-25	677.577209	0.000000	0.0	0.0	5931.377441	
2015-10-26	697.818298	0.000000	0.0	0.0	8378.572266	
...	
2018-10-18	1298.433350	23.293699	0.0	0.0	4240.286133	
2018-10-19	920.468689	25.997499	0.0	0.0	4082.749512	
2018-10-20	496.671387	20.047001	0.0	0.0	5053.779297	
2018-10-21	230.320999	9.045700	0.0	0.0	6168.920410	
2018-10-22	0.000000	317.783386	0.0	0.0	5190.390625	

	TOTAL_IN	STORAGE_OUT	CONSTANT_HEAD_OUT	WELLS_OUT	\
2015-10-22	6276.861816	0.000000	2446.318848	0.000000	
2015-10-23	7063.646484	31.594000	2430.337891	1302.403198	
2015-10-24	6759.110840	9.152200	2369.628174	1618.676392	
2015-10-25	6608.954590	180.233307	2353.585449	1498.694702	
2015-10-26	9076.390625	1051.897461	2417.248291	3119.501953	
...	
2018-10-18	5562.013184	1606.280396	1909.538574	1930.483154	
2018-10-19	5029.215820	1659.194702	1892.605835	1279.166382	
2018-10-20	5570.497559	2428.291016	1933.045898	794.582886	
2018-10-21	6408.287109	2335.759521	2006.124268	1373.782593	
2018-10-22	5508.173828	0.000000	745.363403	4762.799805	

	DRAINS_OUT	RECHARGE_OUT	TOTAL_OUT	IN-OUT	\
2015-10-22	3830.650146	0.0	6276.968750	-0.106900	
2015-10-23	3299.415039	0.0	7063.750000	-0.103500	
2015-10-24	2761.639648	0.0	6759.096191	0.014648	
2015-10-25	2576.461670	0.0	6608.975586	-0.020996	
2015-10-26	2487.737305	0.0	9076.384766	0.005859	
...	
2018-10-18	115.724899	0.0	5562.026855	-0.013672	
2018-10-19	198.258896	0.0	5029.225586	-0.009766	

(continues on next page)

(continued from previous page)

```

2018-10-20    414.594513          0.0  5570.514648 -0.017090
2018-10-21    692.638123          0.0  6408.304688 -0.017578
2018-10-22         0.000000          0.0  5508.163086  0.010742

```

PERCENT_DISCREPANCY

```

2015-10-22          -0.0
2015-10-23          -0.0
2015-10-24           0.0
2015-10-25          -0.0
2015-10-26           0.0
...
2018-10-18          -0.0
2018-10-19          -0.0
2018-10-20          -0.0
2018-10-21          -0.0
2018-10-22           0.0

```

[1097 rows x 14 columns]

```

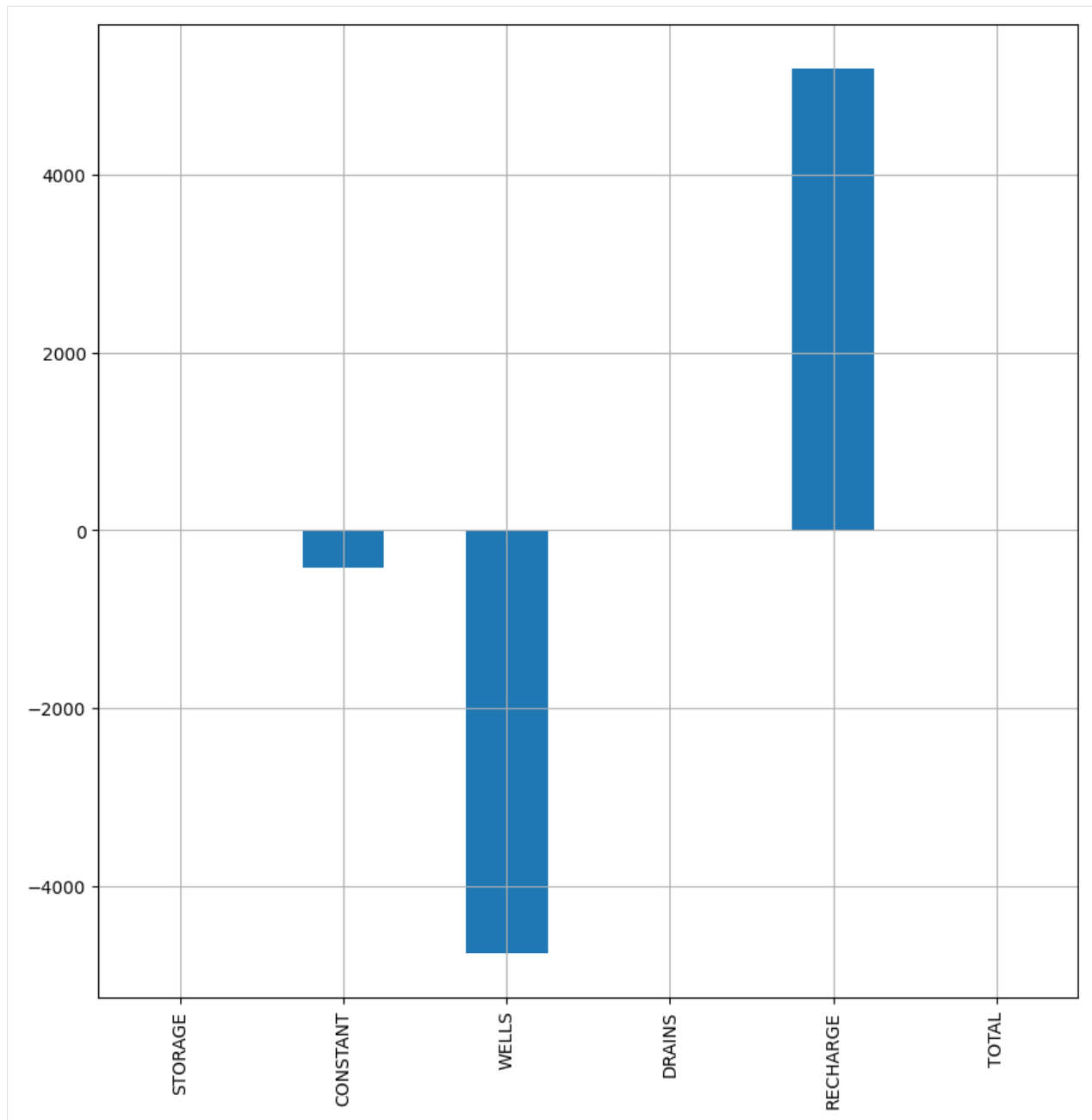
[15]: groups = df_flux.groupby(lambda x: x.split("_")[-1], axis=1).groups
df_flux_in = df_flux.loc[:, groups["IN"]]
df_flux_in.columns = df_flux_in.columns.map(lambda x: x.split("_")[0])

df_flux_out = df_flux.loc[:, groups["OUT"]]
df_flux_out.columns = df_flux_out.columns.map(lambda x: x.split("_")[0])

df_flux_delta = df_flux_in - df_flux_out
df_flux_delta.iloc[-1, :].plot(kind="bar", figsize=(10, 10), grid=True)

```

[15]: <Axes: >



Now let's look at the simulated head.

```
[16]: # if you pass the model instance, then the plots will be offset and rotated
h = flopy.utils.HeadFile(model_ws / "freyberg.hds", model=ml)
h.times
```

```
[16]: [1.0,
      2.0,
      3.0,
      4.0,
      5.0,
      6.0,
```

(continues on next page)

(continued from previous page)

7.0,
8.0,
9.0,
10.0,
11.0,
12.0,
13.0,
14.0,
15.0,
16.0,
17.0,
18.0,
19.0,
20.0,
21.0,
22.0,
23.0,
24.0,
25.0,
26.0,
27.0,
28.0,
29.0,
30.0,
31.0,
32.0,
33.0,
34.0,
35.0,
36.0,
37.0,
38.0,
39.0,
40.0,
41.0,
42.0,
43.0,
44.0,
45.0,
46.0,
47.0,
48.0,
49.0,
50.0,
51.0,
52.0,
53.0,
54.0,
55.0,
56.0,
57.0,
58.0,

(continues on next page)

(continued from previous page)

```
59.0,  
60.0,  
61.0,  
62.0,  
63.0,  
64.0,  
65.0,  
66.0,  
67.0,  
68.0,  
69.0,  
70.0,  
71.0,  
72.0,  
73.0,  
74.0,  
75.0,  
76.0,  
77.0,  
78.0,  
79.0,  
80.0,  
81.0,  
82.0,  
83.0,  
84.0,  
85.0,  
86.0,  
87.0,  
88.0,  
89.0,  
90.0,  
91.0,  
92.0,  
93.0,  
94.0,  
95.0,  
96.0,  
97.0,  
98.0,  
99.0,  
100.0,  
101.0,  
102.0,  
103.0,  
104.0,  
105.0,  
106.0,  
107.0,  
108.0,  
109.0,  
110.0,
```

(continues on next page)

(continued from previous page)

```

111.0,
112.0,
113.0,
114.0,
115.0,
116.0,
117.0,
118.0,
119.0,
120.0,
121.0,
122.0,
123.0,
124.0,
125.0,
126.0,
127.0,
128.0,
129.0,
130.0,
131.0,
132.0,
133.0,
134.0,
135.0,
136.0,
137.0,
138.0,
139.0,
140.0,
141.0,
142.0,
143.0,
144.0,
145.0,
146.0,
147.0,
148.0,
149.0,
150.0,
151.0,
152.0,
153.0,
154.0,
155.0,
156.0,
157.0,
158.0,
159.0,
160.0,
161.0,
162.0,

```

(continues on next page)

(continued from previous page)

```
163.0,  
164.0,  
165.0,  
166.0,  
167.0,  
168.0,  
169.0,  
170.0,  
171.0,  
172.0,  
173.0,  
174.0,  
175.0,  
176.0,  
177.0,  
178.0,  
179.0,  
180.0,  
181.0,  
182.0,  
183.0,  
184.0,  
185.0,  
186.0,  
187.0,  
188.0,  
189.0,  
190.0,  
191.0,  
192.0,  
193.0,  
194.0,  
195.0,  
196.0,  
197.0,  
198.0,  
199.0,  
200.0,  
201.0,  
202.0,  
203.0,  
204.0,  
205.0,  
206.0,  
207.0,  
208.0,  
209.0,  
210.0,  
211.0,  
212.0,  
213.0,  
214.0,
```

(continues on next page)

(continued from previous page)

215.0,
216.0,
217.0,
218.0,
219.0,
220.0,
221.0,
222.0,
223.0,
224.0,
225.0,
226.0,
227.0,
228.0,
229.0,
230.0,
231.0,
232.0,
233.0,
234.0,
235.0,
236.0,
237.0,
238.0,
239.0,
240.0,
241.0,
242.0,
243.0,
244.0,
245.0,
246.0,
247.0,
248.0,
249.0,
250.0,
251.0,
252.0,
253.0,
254.0,
255.0,
256.0,
257.0,
258.0,
259.0,
260.0,
261.0,
262.0,
263.0,
264.0,
265.0,
266.0,

(continues on next page)

(continued from previous page)

```
267.0,  
268.0,  
269.0,  
270.0,  
271.0,  
272.0,  
273.0,  
274.0,  
275.0,  
276.0,  
277.0,  
278.0,  
279.0,  
280.0,  
281.0,  
282.0,  
283.0,  
284.0,  
285.0,  
286.0,  
287.0,  
288.0,  
289.0,  
290.0,  
291.0,  
292.0,  
293.0,  
294.0,  
295.0,  
296.0,  
297.0,  
298.0,  
299.0,  
300.0,  
301.0,  
302.0,  
303.0,  
304.0,  
305.0,  
306.0,  
307.0,  
308.0,  
309.0,  
310.0,  
311.0,  
312.0,  
313.0,  
314.0,  
315.0,  
316.0,  
317.0,  
318.0,
```

(continues on next page)

(continued from previous page)

```

319.0,
320.0,
321.0,
322.0,
323.0,
324.0,
325.0,
326.0,
327.0,
328.0,
329.0,
330.0,
331.0,
332.0,
333.0,
334.0,
335.0,
336.0,
337.0,
338.0,
339.0,
340.0,
341.0,
342.0,
343.0,
344.0,
345.0,
346.0,
347.0,
348.0,
349.0,
350.0,
351.0,
352.0,
353.0,
354.0,
355.0,
356.0,
357.0,
358.0,
359.0,
360.0,
361.0,
362.0,
363.0,
364.0,
365.0,
366.0,
367.0,
368.0,
369.0,
370.0,

```

(continues on next page)

(continued from previous page)

```
371.0,  
372.0,  
373.0,  
374.0,  
375.0,  
376.0,  
377.0,  
378.0,  
379.0,  
380.0,  
381.0,  
382.0,  
383.0,  
384.0,  
385.0,  
386.0,  
387.0,  
388.0,  
389.0,  
390.0,  
391.0,  
392.0,  
393.0,  
394.0,  
395.0,  
396.0,  
397.0,  
398.0,  
399.0,  
400.0,  
401.0,  
402.0,  
403.0,  
404.0,  
405.0,  
406.0,  
407.0,  
408.0,  
409.0,  
410.0,  
411.0,  
412.0,  
413.0,  
414.0,  
415.0,  
416.0,  
417.0,  
418.0,  
419.0,  
420.0,  
421.0,  
422.0,
```

(continues on next page)

(continued from previous page)

423.0,
424.0,
425.0,
426.0,
427.0,
428.0,
429.0,
430.0,
431.0,
432.0,
433.0,
434.0,
435.0,
436.0,
437.0,
438.0,
439.0,
440.0,
441.0,
442.0,
443.0,
444.0,
445.0,
446.0,
447.0,
448.0,
449.0,
450.0,
451.0,
452.0,
453.0,
454.0,
455.0,
456.0,
457.0,
458.0,
459.0,
460.0,
461.0,
462.0,
463.0,
464.0,
465.0,
466.0,
467.0,
468.0,
469.0,
470.0,
471.0,
472.0,
473.0,
474.0,

(continues on next page)

(continued from previous page)

```

475.0,
476.0,
477.0,
478.0,
479.0,
480.0,
481.0,
482.0,
483.0,
484.0,
485.0,
486.0,
487.0,
488.0,
489.0,
490.0,
491.0,
492.0,
493.0,
494.0,
495.0,
496.0,
497.0,
498.0,
499.0,
500.0,
501.0,
502.0,
503.0,
504.0,
505.0,
506.0,
507.0,
508.0,
509.0,
510.0,
511.0,
512.0,
513.0,
514.0,
515.0,
516.0,
517.0,
518.0,
519.0,
520.0,
521.0,
522.0,
523.0,
524.0,
525.0,
526.0,

```

(continues on next page)

(continued from previous page)

527.0,
 528.0,
 529.0,
 530.0,
 531.0,
 532.0,
 533.0,
 534.0,
 535.0,
 536.0,
 537.0,
 538.0,
 539.0,
 540.0,
 541.0,
 542.0,
 543.0,
 544.0,
 545.0,
 546.0,
 547.0,
 548.0,
 549.0,
 550.0,
 551.0,
 552.0,
 553.0,
 554.0,
 555.0,
 556.0,
 557.0,
 558.0,
 559.0,
 560.0,
 561.0,
 562.0,
 563.0,
 564.0,
 565.0,
 566.0,
 567.0,
 568.0,
 569.0,
 570.0,
 571.0,
 572.0,
 573.0,
 574.0,
 575.0,
 576.0,
 577.0,
 578.0,

(continues on next page)

(continued from previous page)

```
579.0,  
580.0,  
581.0,  
582.0,  
583.0,  
584.0,  
585.0,  
586.0,  
587.0,  
588.0,  
589.0,  
590.0,  
591.0,  
592.0,  
593.0,  
594.0,  
595.0,  
596.0,  
597.0,  
598.0,  
599.0,  
600.0,  
601.0,  
602.0,  
603.0,  
604.0,  
605.0,  
606.0,  
607.0,  
608.0,  
609.0,  
610.0,  
611.0,  
612.0,  
613.0,  
614.0,  
615.0,  
616.0,  
617.0,  
618.0,  
619.0,  
620.0,  
621.0,  
622.0,  
623.0,  
624.0,  
625.0,  
626.0,  
627.0,  
628.0,  
629.0,  
630.0,
```

(continues on next page)

(continued from previous page)

631.0,
 632.0,
 633.0,
 634.0,
 635.0,
 636.0,
 637.0,
 638.0,
 639.0,
 640.0,
 641.0,
 642.0,
 643.0,
 644.0,
 645.0,
 646.0,
 647.0,
 648.0,
 649.0,
 650.0,
 651.0,
 652.0,
 653.0,
 654.0,
 655.0,
 656.0,
 657.0,
 658.0,
 659.0,
 660.0,
 661.0,
 662.0,
 663.0,
 664.0,
 665.0,
 666.0,
 667.0,
 668.0,
 669.0,
 670.0,
 671.0,
 672.0,
 673.0,
 674.0,
 675.0,
 676.0,
 677.0,
 678.0,
 679.0,
 680.0,
 681.0,
 682.0,

(continues on next page)

(continued from previous page)

```
683.0,  
684.0,  
685.0,  
686.0,  
687.0,  
688.0,  
689.0,  
690.0,  
691.0,  
692.0,  
693.0,  
694.0,  
695.0,  
696.0,  
697.0,  
698.0,  
699.0,  
700.0,  
701.0,  
702.0,  
703.0,  
704.0,  
705.0,  
706.0,  
707.0,  
708.0,  
709.0,  
710.0,  
711.0,  
712.0,  
713.0,  
714.0,  
715.0,  
716.0,  
717.0,  
718.0,  
719.0,  
720.0,  
721.0,  
722.0,  
723.0,  
724.0,  
725.0,  
726.0,  
727.0,  
728.0,  
729.0,  
730.0,  
731.0,  
732.0,  
733.0,  
734.0,
```

(continues on next page)

(continued from previous page)

735.0,
 736.0,
 737.0,
 738.0,
 739.0,
 740.0,
 741.0,
 742.0,
 743.0,
 744.0,
 745.0,
 746.0,
 747.0,
 748.0,
 749.0,
 750.0,
 751.0,
 752.0,
 753.0,
 754.0,
 755.0,
 756.0,
 757.0,
 758.0,
 759.0,
 760.0,
 761.0,
 762.0,
 763.0,
 764.0,
 765.0,
 766.0,
 767.0,
 768.0,
 769.0,
 770.0,
 771.0,
 772.0,
 773.0,
 774.0,
 775.0,
 776.0,
 777.0,
 778.0,
 779.0,
 780.0,
 781.0,
 782.0,
 783.0,
 784.0,
 785.0,
 786.0,

(continues on next page)

(continued from previous page)

```
787.0,  
788.0,  
789.0,  
790.0,  
791.0,  
792.0,  
793.0,  
794.0,  
795.0,  
796.0,  
797.0,  
798.0,  
799.0,  
800.0,  
801.0,  
802.0,  
803.0,  
804.0,  
805.0,  
806.0,  
807.0,  
808.0,  
809.0,  
810.0,  
811.0,  
812.0,  
813.0,  
814.0,  
815.0,  
816.0,  
817.0,  
818.0,  
819.0,  
820.0,  
821.0,  
822.0,  
823.0,  
824.0,  
825.0,  
826.0,  
827.0,  
828.0,  
829.0,  
830.0,  
831.0,  
832.0,  
833.0,  
834.0,  
835.0,  
836.0,  
837.0,  
838.0,
```

(continues on next page)

(continued from previous page)

839.0,
 840.0,
 841.0,
 842.0,
 843.0,
 844.0,
 845.0,
 846.0,
 847.0,
 848.0,
 849.0,
 850.0,
 851.0,
 852.0,
 853.0,
 854.0,
 855.0,
 856.0,
 857.0,
 858.0,
 859.0,
 860.0,
 861.0,
 862.0,
 863.0,
 864.0,
 865.0,
 866.0,
 867.0,
 868.0,
 869.0,
 870.0,
 871.0,
 872.0,
 873.0,
 874.0,
 875.0,
 876.0,
 877.0,
 878.0,
 879.0,
 880.0,
 881.0,
 882.0,
 883.0,
 884.0,
 885.0,
 886.0,
 887.0,
 888.0,
 889.0,
 890.0,

(continues on next page)

(continued from previous page)

```

891.0,
892.0,
893.0,
894.0,
895.0,
896.0,
897.0,
898.0,
899.0,
900.0,
901.0,
902.0,
903.0,
904.0,
905.0,
906.0,
907.0,
908.0,
909.0,
910.0,
911.0,
912.0,
913.0,
914.0,
915.0,
916.0,
917.0,
918.0,
919.0,
920.0,
921.0,
922.0,
923.0,
924.0,
925.0,
926.0,
927.0,
928.0,
929.0,
930.0,
931.0,
932.0,
933.0,
934.0,
935.0,
936.0,
937.0,
938.0,
939.0,
940.0,
941.0,
942.0,

```

(continues on next page)

(continued from previous page)

943.0,
 944.0,
 945.0,
 946.0,
 947.0,
 948.0,
 949.0,
 950.0,
 951.0,
 952.0,
 953.0,
 954.0,
 955.0,
 956.0,
 957.0,
 958.0,
 959.0,
 960.0,
 961.0,
 962.0,
 963.0,
 964.0,
 965.0,
 966.0,
 967.0,
 968.0,
 969.0,
 970.0,
 971.0,
 972.0,
 973.0,
 974.0,
 975.0,
 976.0,
 977.0,
 978.0,
 979.0,
 980.0,
 981.0,
 982.0,
 983.0,
 984.0,
 985.0,
 986.0,
 987.0,
 988.0,
 989.0,
 990.0,
 991.0,
 992.0,
 993.0,
 994.0,

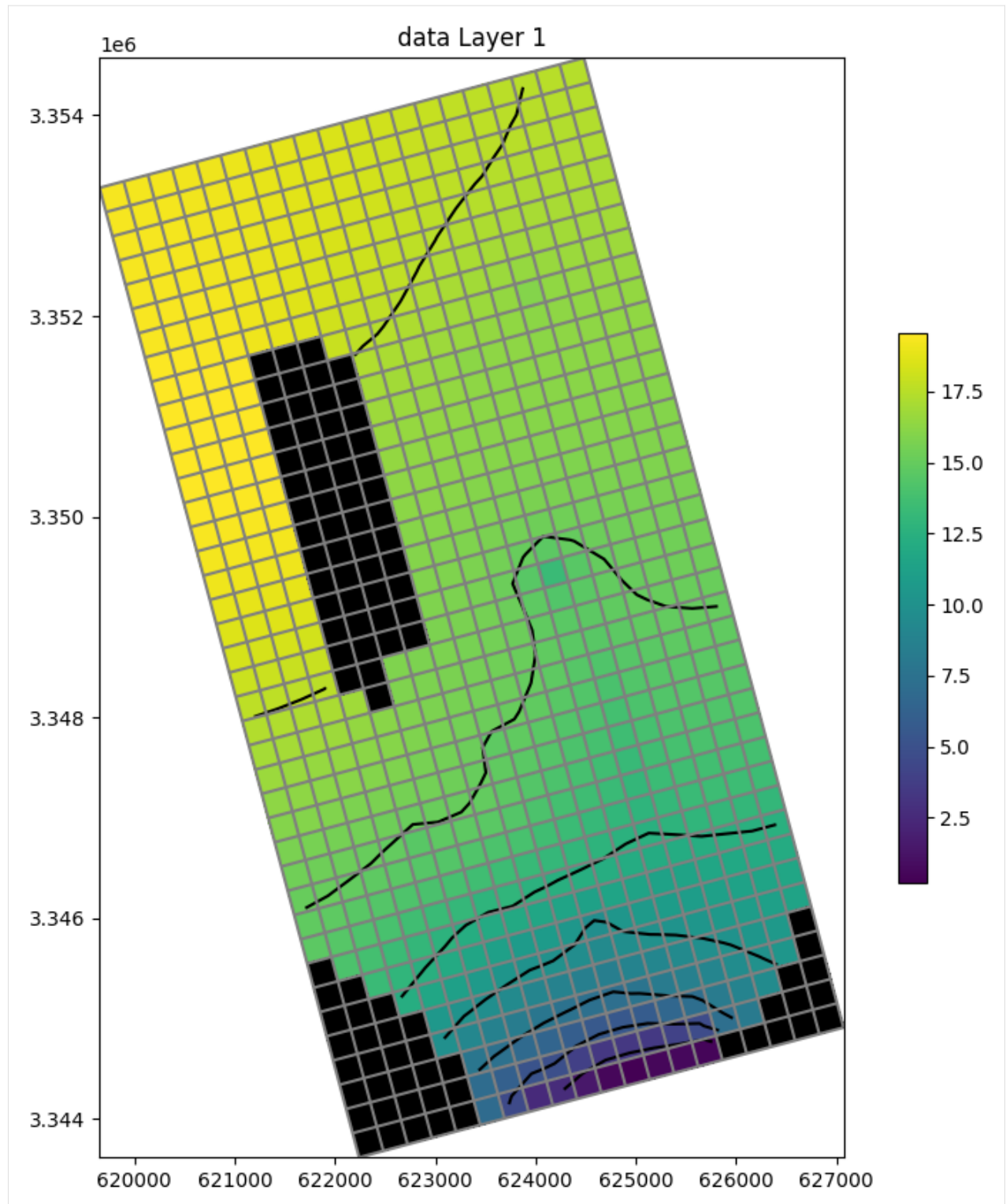
(continues on next page)

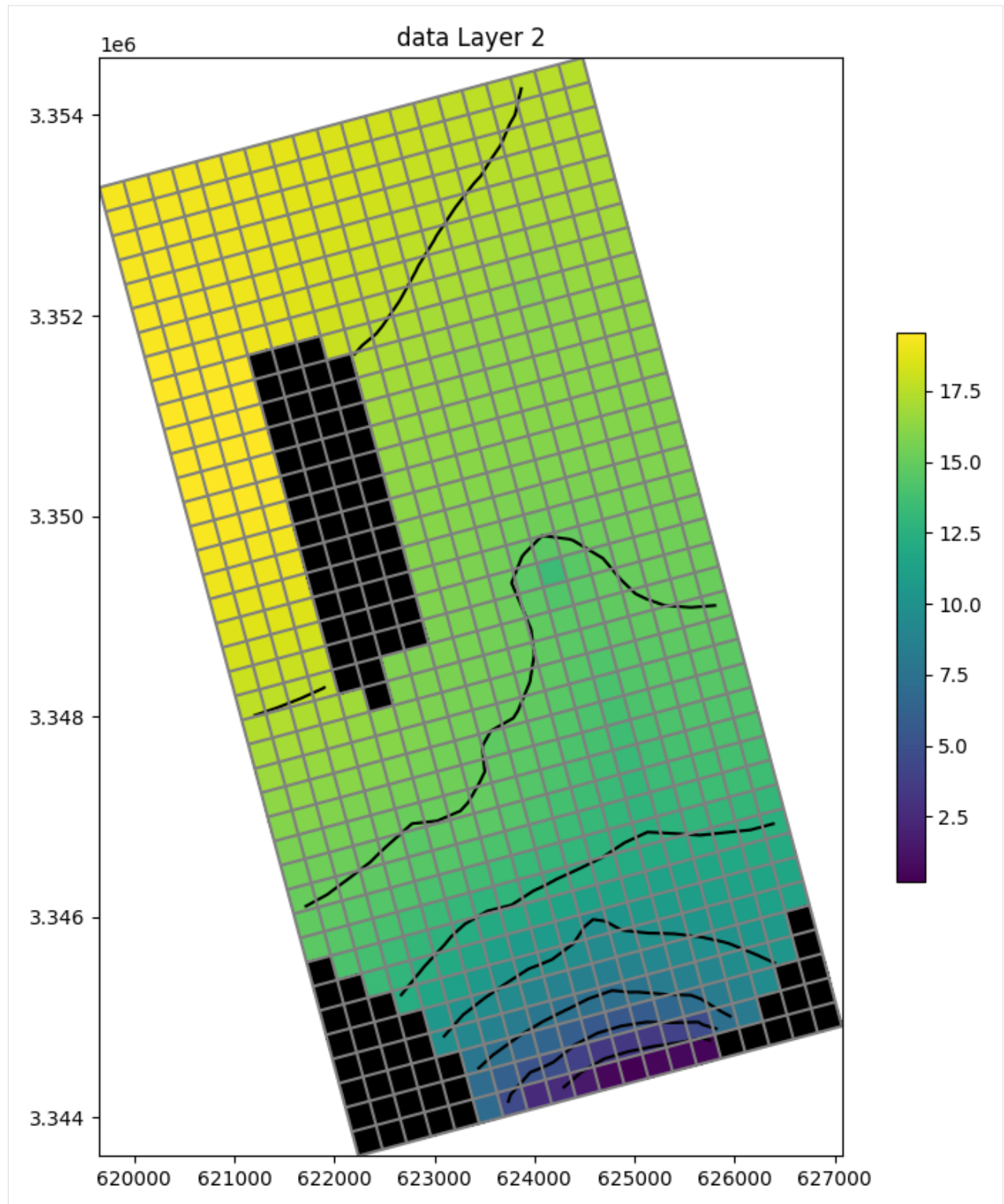
(continued from previous page)

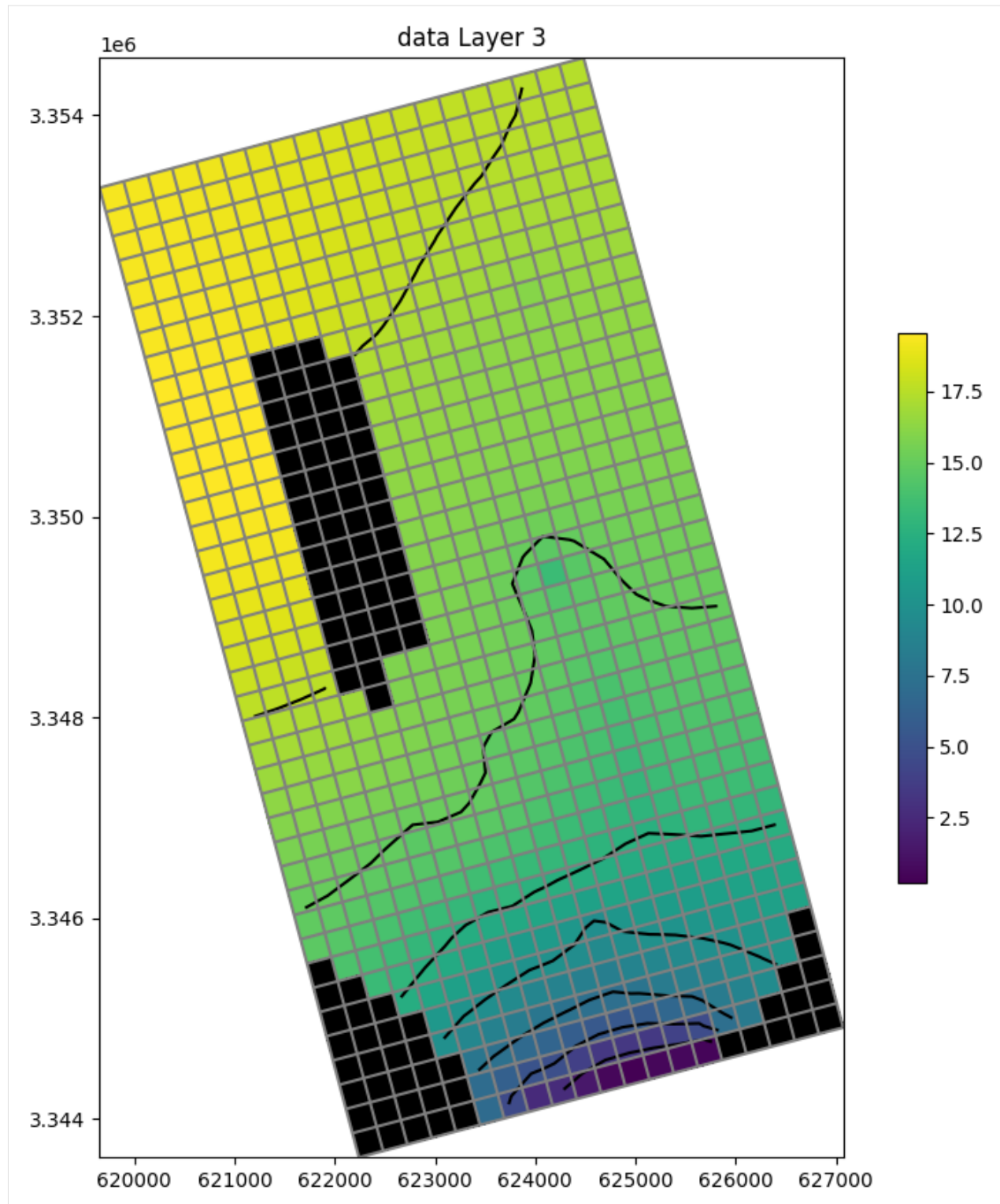
```
995.0,  
996.0,  
997.0,  
998.0,  
999.0,  
1000.0,  
...]
```

```
[17]: h.plot(totim=900, contour=True, grid=True, colorbar=True, figsize=(10, 10))
```

```
[17]: [<Axes: title={'center': 'data Layer 1'}>,  
<Axes: title={'center': 'data Layer 2'}>,  
<Axes: title={'center': 'data Layer 3'}>]
```







We can write the heads to a shapefile.

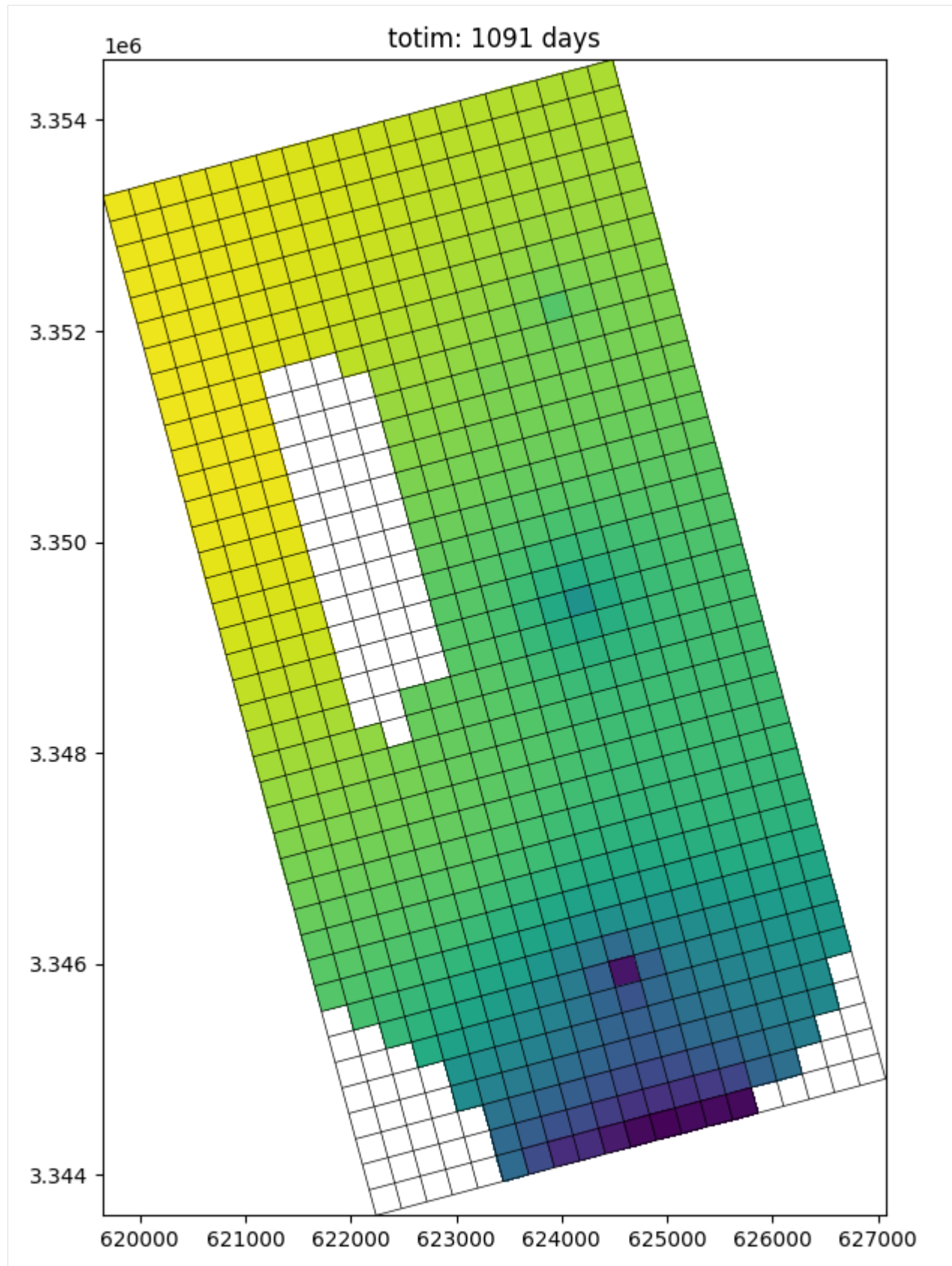
```
[18]: h.to_shapefile(ml.model_ws / "freyburg_head.shp", verbose=False)
```

Finally, let's make an animation of the simulated head over the time domain.

```
[19]: f = plt.figure(figsize=(10, 10))
      ax = plt.subplot(1, 1, 1, aspect="equal")
      for t in h.times[0:-1:10]:
          ax.cla()

          ax.set_title(f"totim: {t:4.0f} days")
          mm = flopy.plot.PlotMapView(model=ml, ax=ax)
          mm.plot_array(h.get_data(totim=t), vmin=0, vmax=20)
          mm.plot_grid(lw=0.5, color="black")

      display(f)
      clear_output(wait=True)
      plt.pause(0.1)
```

```
[20]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.4.2 Computing transmissivity-weighted averages

Demonstration of `flopy.utils.get_transmissivities` method for computing open interval transmissivities (for weighted averages of heads or fluxes) In practice this method might be used to:

- compute vertically-averaged head target values representative of observation wells of varying open intervals (including variability in saturated thickness due to the position of the water table)
- apportion boundary fluxes (e.g. from an analytic element model) among model layers based on transmissivity.
- any other analysis where a distribution of transmissivity by layer is needed for a specified vertical interval of the model.

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

Make up some open interval tops and bottoms and some heads

- (these could be lists of observation well screen tops and bottoms)
- the heads array contains the simulated head in each model layer, at the location of each observation well (for example, what you would get back from HYDMOD if you had an entry for each layer at the location of each head target).
- make up a model grid with uniform horizontal k of 2.

```
[2]: sctop = [-0.25, 0.5, 1.7, 1.5, 3.0, 2.5] # screen tops
scbot = [-1.0, -0.5, 1.2, 0.5, 1.5, -0.2] # screen bottoms
# head in each layer, for 6 head target locations
```

(continues on next page)

(continued from previous page)

```
heads = np.array(
    [
        [1.0, 2.0, 2.05, 3.0, 4.0, 2.5],
        [1.1, 2.1, 2.2, 2.0, 3.5, 3.0],
        [1.2, 2.3, 2.4, 0.6, 3.4, 3.2],
    ]
)
nl, nr = heads.shape
nc = nr
botm = np.ones((nl, nr, nc), dtype=float)
top = np.ones((nr, nc), dtype=float) * 2.1
hk = np.ones((nl, nr, nc), dtype=float) * 2.0
for i in range(nl):
    botm[nl - i - 1, :, :] = i
botm
```

```
[2]: array([[2., 2., 2., 2., 2., 2.],
           [2., 2., 2., 2., 2., 2.],
           [2., 2., 2., 2., 2., 2.],
           [2., 2., 2., 2., 2., 2.],
           [2., 2., 2., 2., 2., 2.],
           [2., 2., 2., 2., 2., 2.]],

          [[1., 1., 1., 1., 1., 1.],
           [1., 1., 1., 1., 1., 1.],
           [1., 1., 1., 1., 1., 1.],
           [1., 1., 1., 1., 1., 1.],
           [1., 1., 1., 1., 1., 1.],
           [1., 1., 1., 1., 1., 1.]],

          [[0., 0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0., 0.],
           [0., 0., 0., 0., 0., 0.]])
```

Make a flopy modflow model

```
[3]: # temporary directory
temp_dir = TemporaryDirectory()
model_ws = temp_dir.name

m = flopy.modflow.Modflow("junk", version="mfnwt", model_ws=model_ws)
dis = flopy.modflow.ModflowDis(
    m, nlay=nl, nrow=nr, ncol=nc, botm=botm, top=top
)
upw = flopy.modflow.ModflowUpw(m, hk=hk)
```

Get transmissivities along the diagonal cells

- alternatively, if a model's coordinate information has been set up, the real-world x and y coordinates could be supplied with the x and y arguments
- if sctop and scbot arguments are given, the transmissivities are computed for the open intervals only (cells that are partially within the open interval have reduced thickness, cells outside of the open interval have transmissivities of 0). If no sctop or scbot arguments are supplied, transmissivities reflect the full saturated thickness in each column of cells (see plot below, which shows different open intervals relative to the model layering)

```
[4]: r, c = np.arange(nr), np.arange(nc)
      T = flopy.utils.get_transmissivities(
          heads, m, r=r, c=c, sctop=sctop, scbot=scbot
      )
      np.round(T, 2)
```

```
[4]: array([[0. , 0. , 0. , 0. , 0.2, 0.2],
          [0. , 0. , 1. , 1. , 1. , 2. ],
          [2. , 1. , 0. , 0.2, 0. , 2. ]])
```

```
[5]: m.dis.botm.array[:, r, c]
```

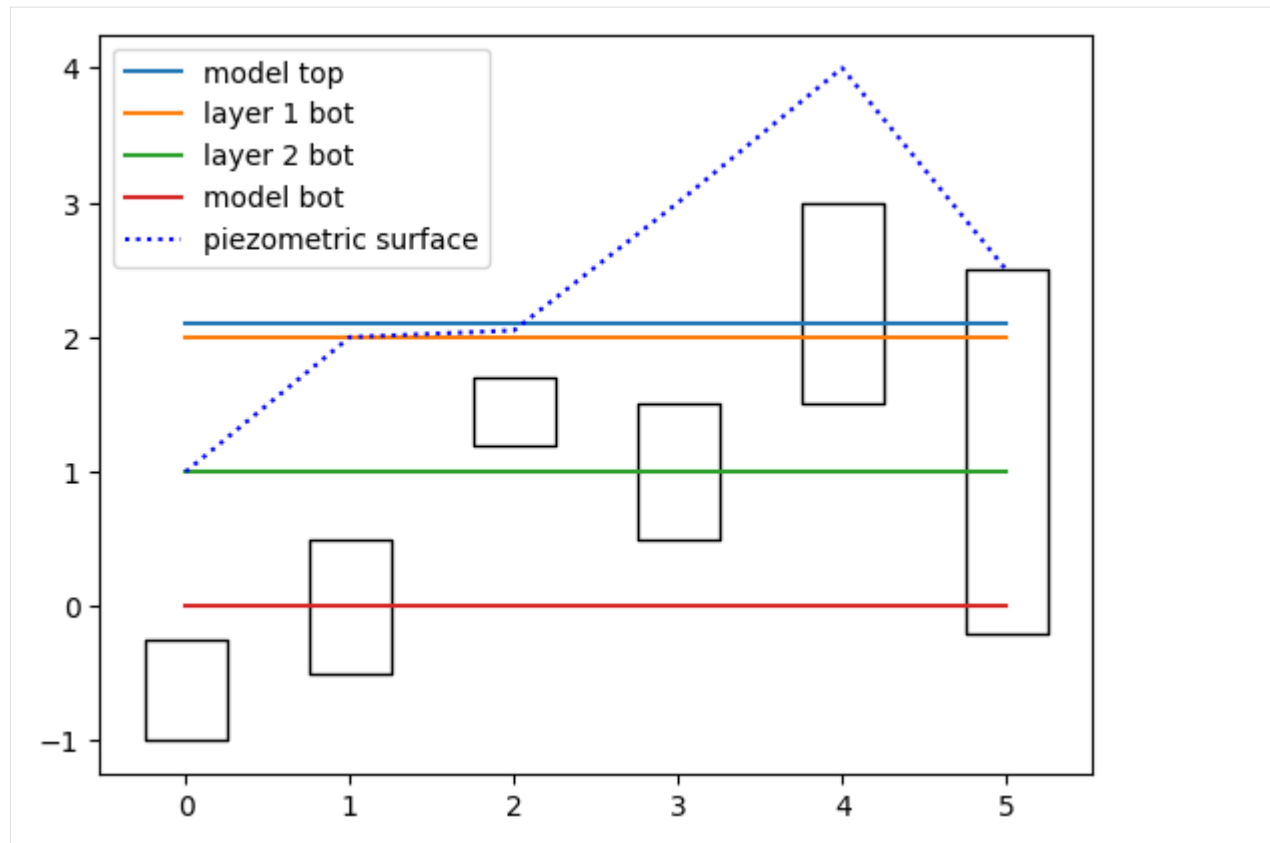
```
[5]: array([[2., 2., 2., 2., 2., 2.],
          [1., 1., 1., 1., 1., 1.],
          [0., 0., 0., 0., 0., 0.]], dtype=float32)
```

Plot the model top and layer bottoms (colors)

open intervals are shown as boxes * well 0 has zero transmissivities for each layer, as it is below the model bottom * well 1 has T values of 0 for layers 1 and 2, and 1 for layer 3 (K=2 x 0.5 thickness)

```
[6]: fig, ax = plt.subplots()
      plt.plot(m.dis.top.array[r, c], label="model top")
      for i, l in enumerate(m.dis.botm.array[:, r, c]):
          label = "layer {} bot".format(i + 1)
          if i == m.nlay - 1:
              label = "model bot"
          plt.plot(l, label=label)
      plt.plot(heads[0], label="piezometric surface", color="b", linestyle=":")
      for iw in range(len(sctop)):
          ax.fill_between(
              [iw - 0.25, iw + 0.25],
              scbot[iw],
              sctop[iw],
              facecolor="None",
              edgecolor="k",
          )
      ax.legend(loc=2)
```

```
[6]: <matplotlib.legend.Legend at 0x7f4b28fe58e0>
```



example of transmissivites without sctop and scbot

- well zero has $T=0$ in layer 1 because it is dry; $T=0.2$ in layer 2 because the sat. thickness there is only 0.1

```
[7]: T = flopy.utils.get_transmissivities(heads, m, r=r, c=c)
      np.round(T, 2)
```

```
[7]: array([[0. , 0. , 0.1, 0.2, 0.2, 0.2],
          [0.2, 2. , 2. , 2. , 2. , 2. ],
          [2. , 2. , 2. , 1.2, 2. , 2. ]])
```

```
[8]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.4.3 Saving array data to MODFLOW-style binary files

```
[1]: import os
import shutil
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))
```

```
3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[2]: nlay, nrow, ncol = 1, 20, 10

# temporary directory
temp_dir = TemporaryDirectory()
model_ws = os.path.join(temp_dir.name, "binary_data")

if os.path.exists(model_ws):
    shutil.rmtree(model_ws)

precision = "single" # or 'double'
dtype = np.float32 # or np.float64

mf = flopy.modflow.Modflow(model_ws=model_ws)
dis = flopy.modflow.ModflowDis(
    mf, nlay=nlay, nrow=nrow, ncol=ncol, delr=20, delc=10
)
```

Create a linear data array

```
[3]: # create the first row of data
b = np.linspace(10, 1, num=ncol, dtype=dtype).reshape(1, ncol)

# extend data to every row
```

(continues on next page)

(continued from previous page)

```
b = np.repeat(b, nrow, axis=0)

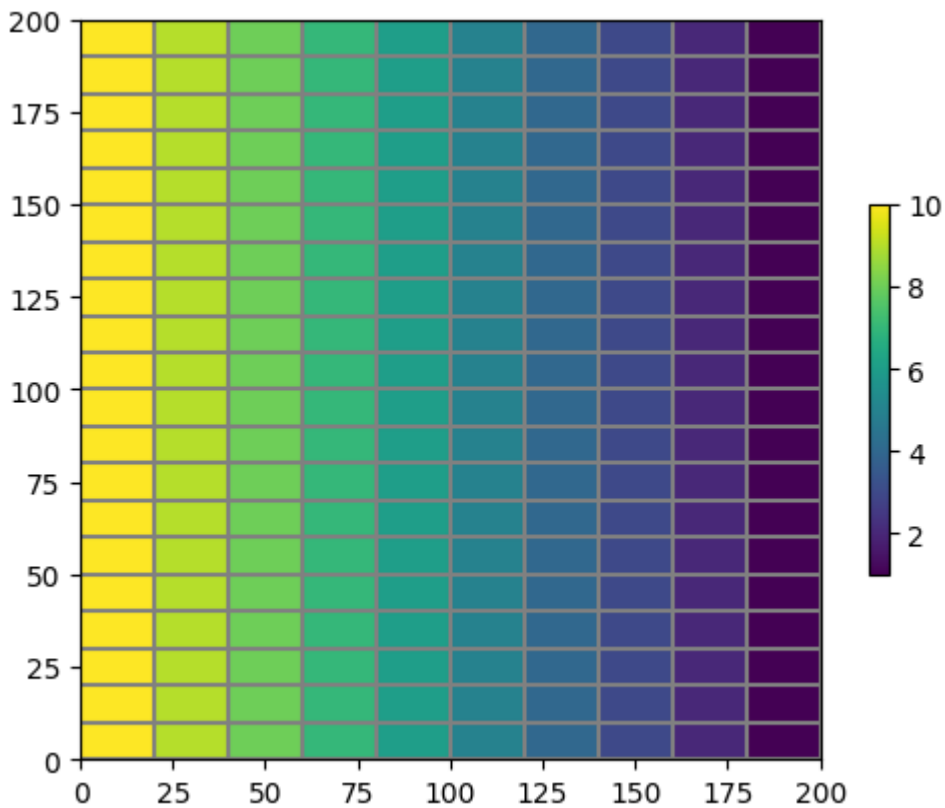
# print the shape and type of the data
print(b.shape)

(20, 10)
```

Plot the data array

```
[4]: pmv = flopy.plot.PlotMapView(model=mf)
      v = pmv.plot_array(b)
      pmv.plot_grid()
      plt.colorbar(v, shrink=0.5)

[4]: <matplotlib.colorbar.Colorbar at 0x7ff311ef7910>
```



Write the linear data array to a binary file

```
[5]: text = "head"

# write a binary data file
pertim = dtype(1.0)
header = flopy.utils.BinaryHeader.create(
    bintype=text,
    precision=precision,
    text=text,
    nrow=nrow,
```

(continues on next page)

(continued from previous page)

```

ncol=ncol,
ilay=1,
pertim=pertim,
totim=pertim,
kstp=1,
kper=1,
)
pth = os.path.join(model_ws, "bottom.bin")
flopy.utils.Util2d.write_bin(b.shape, pth, b, header_data=header)

```

Read the binary data file

```

[6]: bo = flopy.utils.HeadFile(pth, precision=precision)
br = bo.get_data(idx=0)

```

Plot the data that was read from the binary file

```

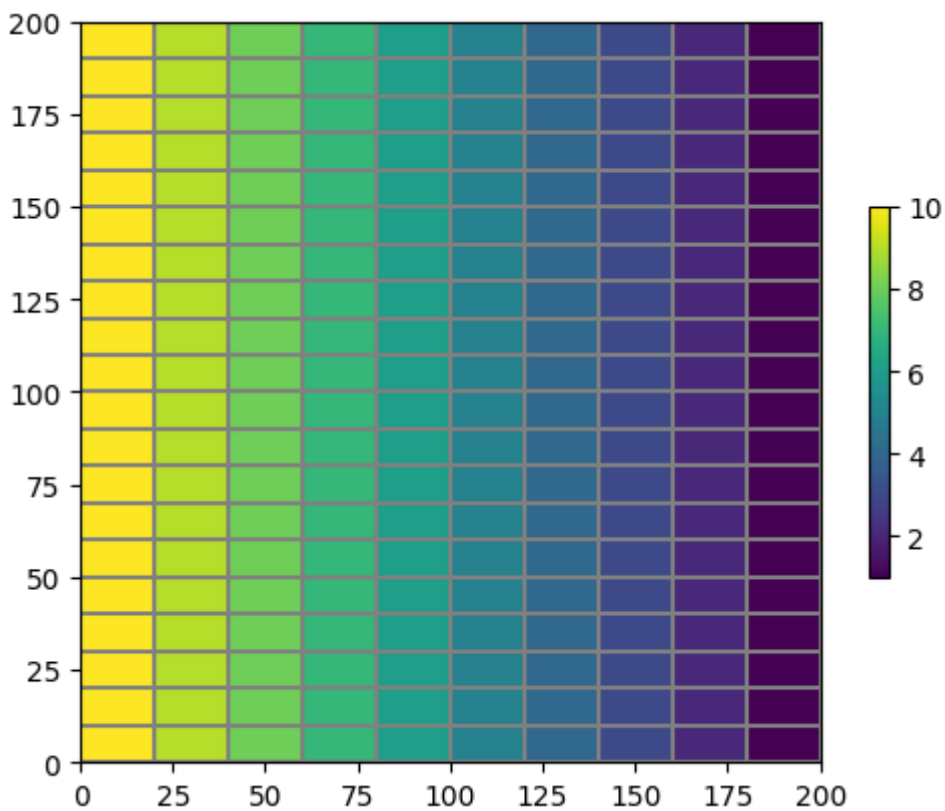
[7]: pmv = flopy.plot.PlotMapView(model=mf)
v = pmv.plot_array(br)
pmv.plot_grid()
plt.colorbar(v, shrink=0.5)

```

```

[7]: <matplotlib.colorbar.Colorbar at 0x7ff30fb3e910>

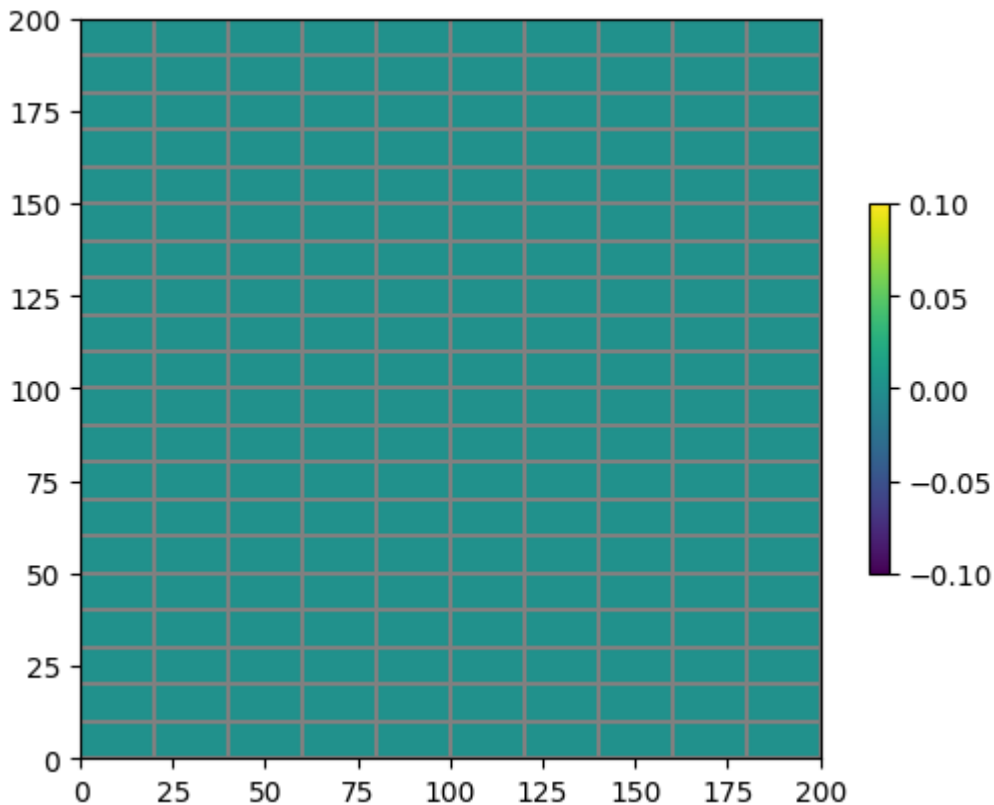
```



Plot the difference in the two values


```
[8]: pmv = flopy.plot.PlotMapView(model=mf)
      v = pmv.plot_array(b - br)
      pmv.plot_grid()
      plt.colorbar(v, shrink=0.5)

[8]: <matplotlib.colorbar.Colorbar at 0x7ff30fa24040>
```



```
[9]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.4.4 Working with shapefiles

This notebook shows some lower-level functionality in flopy for working with shapefiles including: * `recarray2shp` convenience function for writing a numpy record array to a shapefile * `shp2recarray` convenience function for quickly reading a shapefile into a numpy recarray * `utils.geometry` classes for writing shapefiles of model input/output. For example, quickly writing a shapefile of model cells with errors identified by the checker * examples of how the `Point` and `LineString` classes can be used to quickly plot pathlines and endpoints from MODPATH (these are also used by the `PathlineFile` and `EndpointFile` classes to write shapefiles of this output)

```
[1]: import os
      import shutil
      import sys
```

(continues on next page)

(continued from previous page)

```

import warnings
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

import flopy
from flopy.export.shapefile_utils import recarray2shp, shp2recarray
from flopy.utils import geometry
from flopy.utils.geometry import LineString, Point, Polygon
from flopy.utils.modpathfile import EndpointFile, PathlineFile

warnings.simplefilter("ignore", UserWarning)
print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3

```

write a numpy record array to a shapefile

in this case, we want to visualize output from the checker first make a toy model

```

[2]: temp_dir = TemporaryDirectory()
workspace = temp_dir.name

m = flopy.modflow.Modflow("toy_model", model_ws=workspace)
botm = np.zeros((2, 10, 10))
botm[0, :, :] = 1.5
botm[1, 5, 5] = 4 # negative layer thickness!
botm[1, 6, 6] = 4
dis = flopy.modflow.ModflowDis(
    nrow=10, ncol=10, nlay=2, delr=100, delc=100, top=3, botm=botm, model=m
)

```

set coordinate information

```

[3]: grid = m.modelgrid
grid.set_coord_info(xoff=6000000, yoff=5170000, crs="EPSG:26715", angrot=45)

[4]: chk = dis.check()
chk.summary_array

```

```
DIS PACKAGE DATA VALIDATION:
```

```
2 Errors:
```

```
2 instances of zero or negative thickness
```

```
Checks that passed:
```

```
thin cells (less than checker threshold of 1.0)
```

```
nan values in top array
```

```
nan values in bottom array
```

```
[4]: rec.array([('Error', 'DIS', 1, 5, 5, -2.5, 'zero or negative thickness'),
               ('Error', 'DIS', 1, 6, 6, -2.5, 'zero or negative thickness')],
              dtype=[('type', 'O'), ('package', 'O'), ('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('value', '<f8'), ('desc', 'O')])
```

make geometry objects for the cells with errors

- geometry objects allow the shapefile writer to be simpler and agnostic about the kind of geometry

```
[5]: get_vertices = (
      m.modelgrid.get_cell_vertices
    ) # function to get the referenced vertices for a model cell
geoms = [Polygon(get_vertices(i, j)) for i, j in chk.summary_array[["i", "j"]]]
```

```
[6]: geoms[0].type
```

```
[6]: 'Polygon'
```

```
[7]: geoms[0].exterior
```

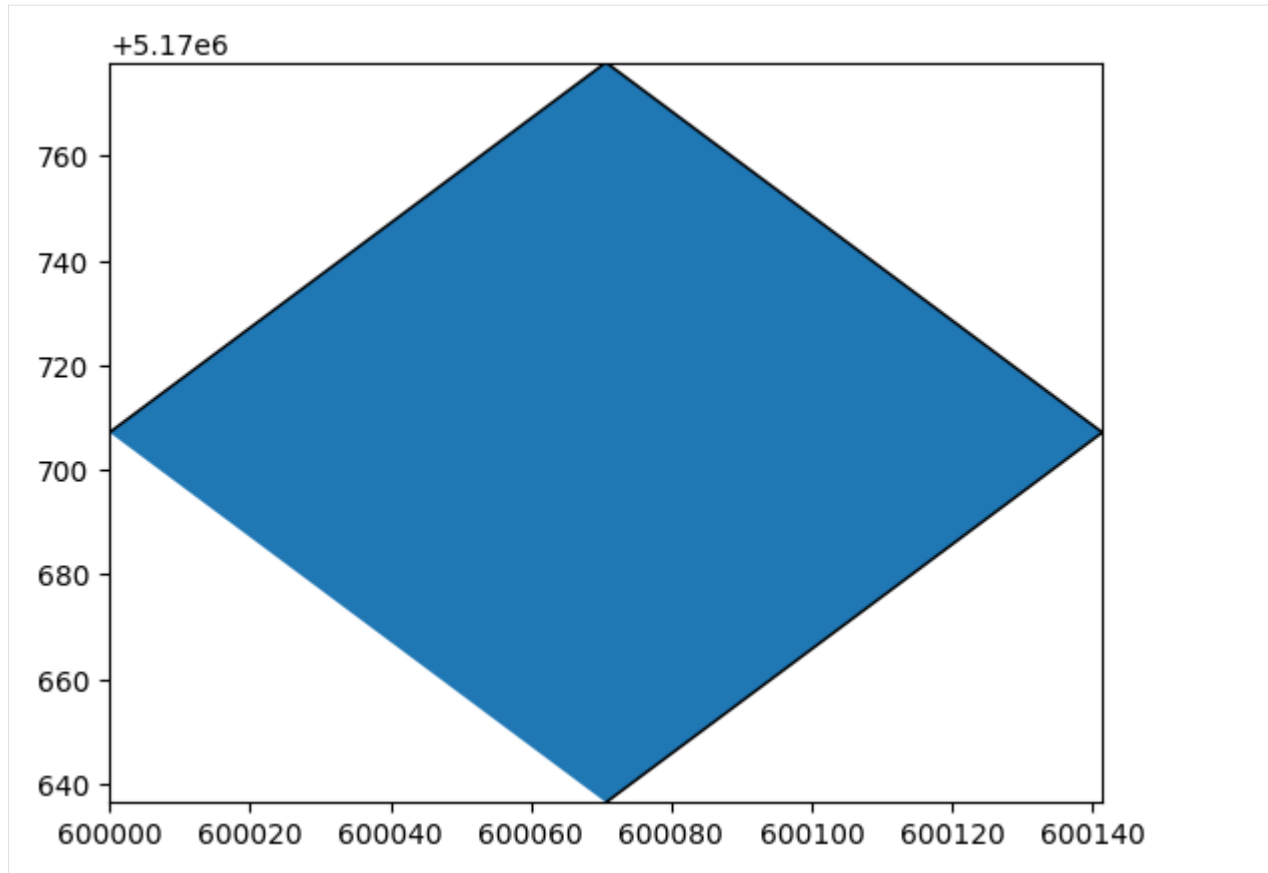
```
[7]: ((6000000.0, 5170707.106781187),
      (600070.7106781186, 5170777.817459305),
      (600141.4213562373, 5170707.106781187),
      (600070.7106781186, 5170636.396103068))
```

```
[8]: geoms[0].bounds
```

```
[8]: (6000000.0, 5170636.396103068, 600141.4213562373, 5170777.817459305)
```

```
[9]: geoms[0].plot() # this feature requires descartes
```

```
[9]: <Axes: >
```



write the shapefile

- the projection (.prj) file can be written using an epsg code
- or copied from an existing .prj file

```
[10]: from pathlib import Path
```

```
recarray2shp(
    chk.summary_array, geoms, os.path.join(workspace, "test.shp"), crs=26715
)
shape_path = os.path.join(workspace, "test.prj")
```

wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks

```
[11]: shutil.copy(shape_path, os.path.join(workspace, "26715.prj"))
```

```
recarray2shp(
    chk.summary_array,
    geoms,
    os.path.join(workspace, "test.shp"),
    prjfile=os.path.join(workspace, "26715.prj"),
)
```

wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks

read it back in

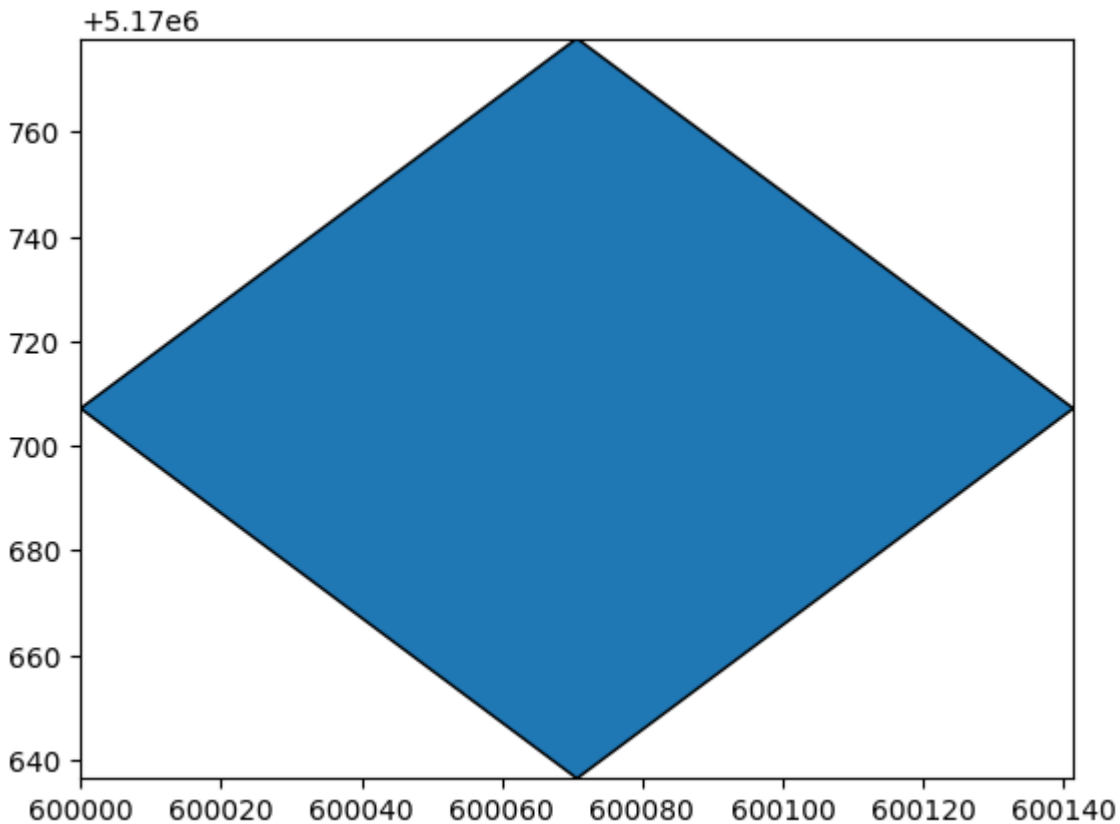
- flopy geometry objects representing the shapes are stored in the ‘geometry’ field

```
[12]: ra = shp2recarray(os.path.join(workspace, "test.shp"))
ra
```

```
[12]: rec.array([('Error', 'DIS', 1, 5, 5, -2.5, 'zero or negative thickness', <flopy.utils.
→geometry.Polygon object at 0x7effa67a8790>),
                ('Error', 'DIS', 1, 6, 6, -2.5, 'zero or negative thickness', <flopy.utils.
→geometry.Polygon object at 0x7effa67a8490>)],
                dtype=[('type', 'O'), ('package', 'O'), ('k', '<i8'), ('i', '<i8'), ('j', '<i8
→'), ('value', '<f8'), ('desc', 'O'), ('geometry', 'O')])
```

```
[13]: ra.geometry[0].plot()
```

```
[13]: <Axes: >
```



Other geometry types

Linestring

- create geometry objects for pathlines from a MODPATH simulation
- plot the paths using the built in plotting method

```
[14]: pthfile = PathlineFile("../examples/data/mp6/EXAMPLE-3.pathline")
      pthdata = pthfile._data.view(np.recarray)

[15]: length_mult = 1.0 # multiplier to convert coordinates from model to real world
      rot = 0 # grid rotation

      particles = np.unique(pthdata.particleid)
      geoms = []
      for pid in particles:
          ra = pthdata[pthdata.particleid == pid]

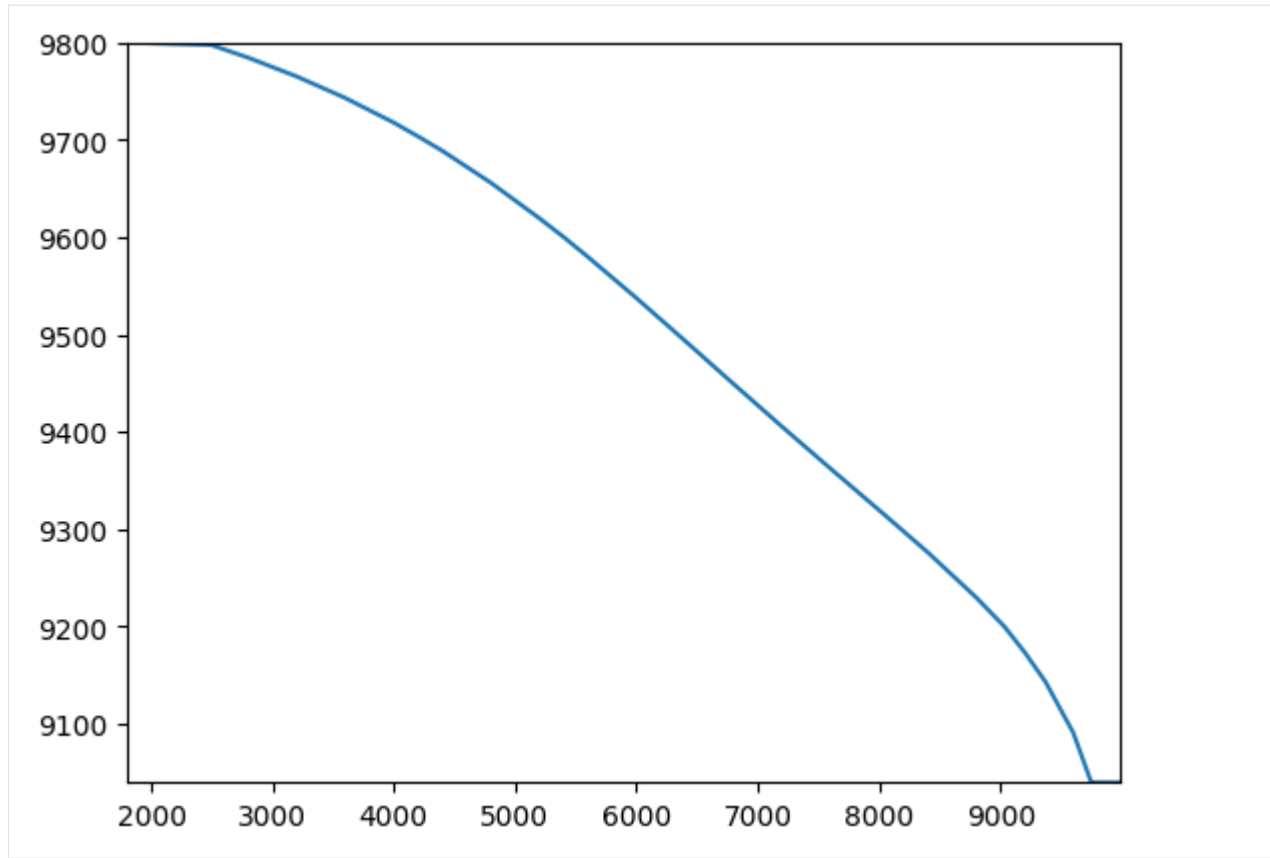
          x, y = geometry.rotate(
              ra.x * length_mult, ra.y * length_mult, grid.xoffset, grid.yoffset, rot
          )
          z = ra.z
          geoms.append(Linestring(list(zip(x, y, z))))

[16]: geoms[0]

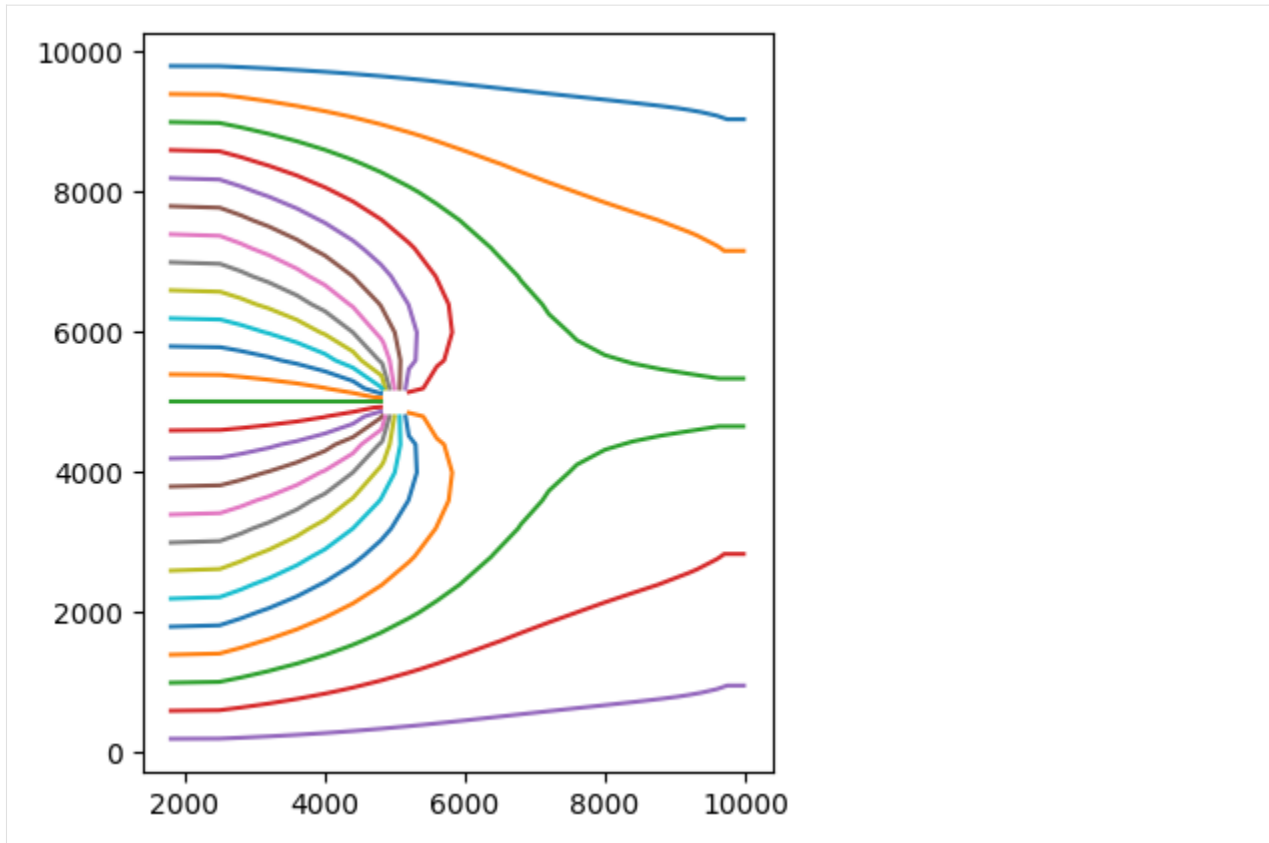
[16]: <flopy.utils.geometry.Linestring at 0x7effa46e3340>

[17]: geoms[0].plot()

[17]: <Axes: >
```



```
[18]: fig, ax = plt.subplots()
      for g in geoms:
          g.plot(ax=ax)
      ax.autoscale()
      ax.set_aspect(1)
```



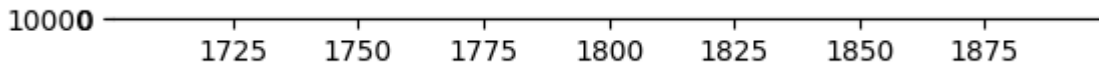
Points

```
[19]: eptfile = EndpointFile("../examples/data/mp6/EXAMPLE-3.endpoint")
      eptdata = eptfile.get_alldata()
```

```
[20]: x, y = geometry.rotate(
      eptdata["x0"] * length_mult,
      eptdata["y0"] * length_mult,
      grid.xoffset,
      grid.yoffset,
      rot,
    )
      z = eptdata["z0"]

      geoms = [Point(x[i], y[i], z[i]) for i in range(len(eptdata))]
```

```
[21]: fig, ax = plt.subplots()
      for g in geoms:
          g.plot(ax=ax)
      ax.autoscale()
      ax.set_aspect(2e-6)
```

```
[22]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass
```

6.5 MODFLOW 6 examples

6.5.1 Creating a Complex MODFLOW 6 Model with Flopy

The purpose of this notebook is to demonstrate the Flopy capabilities for building a more complex MODFLOW 6 model from scratch. This notebook will demonstrate the capabilities by replicating the advgw_tidal model that is distributed with MODFLOW 6.

Setup the Notebook Environment

```
[1]: import os

[2]: import sys
    from tempfile import TemporaryDirectory

    import matplotlib as mpl
    import matplotlib.pyplot as plt
    import numpy as np

    # run installed version of flopy or add local path
    try:
        import flopy
    except:
        fpth = os.path.abspath(os.path.join("..", ".."))
        sys.path.append(fpth)
        import flopy

    print(sys.version)
    print("numpy version: {}".format(np.__version__))
    print("matplotlib version: {}".format(mpl.__version__))
    print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[3]: # For this example, we will set up a temporary workspace.
# Model input files and output files will reside here.
temp_dir = TemporaryDirectory()
model_name = "advgw_tidal"
workspace = os.path.join(temp_dir.name, model_name)
```

```
[4]: data_pth = os.path.join(
    "..",
    "..",
    "examples",
    "data",
    "mf6",
    "create_tests",
    "test005_advgw_tidal",
)
assert os.path.isdir(data_pth)
```

```
[5]: # create simulation
sim = flopy.mf6.MFSimulation(
    sim_name=model_name, version="mf6", exe_name="mf6", sim_ws=workspace
)

# create tdis package
tdis_rc = [(1.0, 1, 1.0), (10.0, 120, 1.0), (10.0, 120, 1.0), (10.0, 120, 1.0)]
tdis = flopy.mf6.ModflowTdis(
    sim, pname="tdis", time_units="DAYS", nper=4, perioddata=tdis_rc
)

# create gwf model
gwf = flopy.mf6.ModflowGwf(
    sim, modelname=model_name, model_nam_file="{}.nam".format(model_name)
)
gwf.name_file.save_flows = True

# create iterative model solution and register the gwf model with it
ims = flopy.mf6.ModflowIms(
    sim,
    pname="ims",
    print_option="SUMMARY",
    complexity="SIMPLE",
    outer_dvclose=0.0001,
    outer_maximum=500,
    under_relaxation="NONE",
    inner_maximum=100,
    inner_dvclose=0.0001,
    rcloserecord=0.001,
    linear_acceleration="CG",
    scaling_method="NONE",
    reordering_method="NONE",
    relaxation_factor=0.97,
)
sim.register_ims_package(ims, [gwf.name])
```

```
[6]: # discretization package
nlay = 3
nrow = 15
ncol = 10
botlay2 = {"factor": 1.0, "data": [-100 for x in range(150)]}
dis = flopy.mf6.ModflowGwfdis(
    gwf,
    pname="dis",
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=500.0,
    delc=500.0,
    top=50.0,
    botm=[5.0, -10.0, botlay2],
    filename="{}.dis".format(model_name),
)

# initial conditions
ic = flopy.mf6.ModflowGwfic(
    gwf, pname="ic", strt=50.0, filename="{}.ic".format(model_name)
)

# node property flow
npf = flopy.mf6.ModflowGwfnpf(
    gwf,
    pname="npf",
    save_flows=True,
    icelltype=[1, 0, 0],
    k=[5.0, 0.1, 4.0],
    k33=[0.5, 0.005, 0.1],
)

# output control
oc = flopy.mf6.ModflowGwfoc(
    gwf,
    pname="oc",
    budget_filerecord="{}.cbb".format(model_name),
    head_filerecord="{}.hds".format(model_name),
    headprintrecord=[("COLUMNS", 10, "WIDTH", 15, "DIGITS", 6, "GENERAL")],
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
    printrecord=[("HEAD", "FIRST"), ("HEAD", "LAST"), ("BUDGET", "LAST")],
)
```

```
[7]: # storage package
sy = flopy.mf6.ModflowGwfsto.sy.empty(gwf, layered=True)
for layer in range(0, 3):
    sy[layer]["data"] = 0.2

ss = flopy.mf6.ModflowGwfsto.ss.empty(
    gwf, layered=True, default_value=0.000001
)
```

(continues on next page)

(continued from previous page)

```
sto = flopy.mf6.ModflowGwfsto(
    gwf,
    pname="sto",
    save_flows=True,
    iconvert=1,
    ss=ss,
    sy=sy,
    steady_state={0: True},
    transient={1: True},
)
```

```
[8]: # well package
# test empty with aux vars, bound names, and time series
period_two = flopy.mf6.ModflowGwfwel.stress_period_data.empty(
    gwf,
    maxbound=3,
    aux_vars=["var1", "var2", "var3"],
    boundnames=True,
    timeseries=True,
)
period_two[0][0] = ((0, 11, 2), -50.0, -1, -2, -3, None)
period_two[0][1] = ((2, 4, 7), "well_1_rate", 1, 2, 3, "well_1")
period_two[0][2] = ((2, 3, 2), "well_2_rate", 4, 5, 6, "well_2")
period_three = flopy.mf6.ModflowGwfwel.stress_period_data.empty(
    gwf,
    maxbound=2,
    aux_vars=["var1", "var2", "var3"],
    boundnames=True,
    timeseries=True,
)
period_three[0][0] = ((2, 3, 2), "well_2_rate", 1, 2, 3, "well_2")
period_three[0][1] = ((2, 4, 7), "well_1_rate", 4, 5, 6, "well_1")
period_four = flopy.mf6.ModflowGwfwel.stress_period_data.empty(
    gwf,
    maxbound=5,
    aux_vars=["var1", "var2", "var3"],
    boundnames=True,
    timeseries=True,
)
period_four[0][0] = ((2, 4, 7), "well_1_rate", 1, 2, 3, "well_1")
period_four[0][1] = ((2, 3, 2), "well_2_rate", 4, 5, 6, "well_2")
period_four[0][2] = ((0, 11, 2), -10.0, 7, 8, 9, None)
period_four[0][3] = ((0, 2, 4), -20.0, 17, 18, 19, None)
period_four[0][4] = ((0, 13, 5), -40.0, 27, 28, 29, None)
stress_period_data = {}
stress_period_data[1] = period_two[0]
stress_period_data[2] = period_three[0]
stress_period_data[3] = period_four[0]
wel = flopy.mf6.ModflowGwfwel(
    gwf,
    pname="wel",
```

(continues on next page)

(continued from previous page)

```

    print_input=True,
    print_flows=True,
    auxiliary=[("var1", "var2", "var3")],
    maxbound=5,
    stress_period_data=stress_period_data,
    boundnames=True,
    save_flows=True,
)

# well ts package
ts_data = [
    (0.0, 0.0, 0.0, 0.0),
    (1.0, -200.0, 0.0, -100.0),
    (11.0, -1800.0, -500.0, -200.0),
    (21.0, -200.0, -400.0, -300.0),
    (31.0, 0.0, -600.0, -400.0),
]
wel.ts.initialize(
    filename="well-rates.ts",
    timeseries=ts_data,
    time_series_namerecord=[("well_1_rate", "well_2_rate", "well_3_rate")],
    interpolation_methodrecord=[("stepwise", "stepwise", "stepwise")],
)

```

```

[9]: # Evapotranspiration
evt_period = flopy.mf6.ModflowGwfevt.stress_period_data.empty(gwf, 150, nseg=3)
for col in range(0, 10):
    for row in range(0, 15):
        evt_period[0][col * 15 + row] = (
            (0, row, col),
            50.0,
            0.0004,
            10.0,
            0.2,
            0.5,
            0.3,
            0.1,
            None,
        )
evt = flopy.mf6.ModflowGwfevt(
    gwf,
    pname="evt",
    print_input=True,
    print_flows=True,
    save_flows=True,
    maxbound=150,
    nseg=3,
    stress_period_data=evt_period,
)

```

```
[10]: # General-Head Boundaries
ghb_period = {}
ghb_period_array = []
for layer, cond in zip(range(1, 3), [15.0, 1500.0]):
    for row in range(0, 15):
        ghb_period_array.append(((layer, row, 9), "tides", cond, "Estuary-L2"))
ghb_period[0] = ghb_period_array
ghb = flopy.mf6.ModflowGwfgbh(
    gwf,
    pname="ghb",
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    maxbound=30,
    stress_period_data=ghb_period,
)
ts_reccarray = []
fd = open(os.path.join(data_pth, "tides.txt"), "r")
for line in fd:
    line_list = line.strip().split(",")
    ts_reccarray.append((float(line_list[0]), float(line_list[1])))
ghb.ts.initialize(
    filename="tides.ts",
    timeseries=ts_reccarray,
    time_series_namerecord="tides",
    interpolation_methodrecord="linear",
)
obs_reccarray = {
    "ghb_obs.csv": [
        ("ghb-2-6-10", "GHB", (1, 5, 9)),
        ("ghb-3-6-10", "GHB", (2, 5, 9)),
    ],
    "ghb_flows.csv": [
        ("Estuary2", "GHB", "Estuary-L2"),
        ("Estuary3", "GHB", "Estuary-L3"),
    ],
}
ghb.obs.initialize(
    filename="{}.ghb.obs".format(model_name),
    print_input=True,
    continuous=obs_reccarray,
)
```

```
[11]: obs_reccarray = {
    "head_obs.csv": [("h1_13_8", "HEAD", (2, 12, 7))],
    "intercell_flow_obs1.csv": [
        ("ICF1_1.0", "FLOW-JA-FACE", (0, 4, 5), (0, 5, 5))
    ],
    "head-hydrographs.csv": [
        ("h3-13-9", "HEAD", (2, 12, 8)),
        ("h3-12-8", "HEAD", (2, 11, 7)),
    ]
}
```

(continues on next page)

(continued from previous page)

```

        ("h1-4-3", "HEAD", (0, 3, 2)),
        ("h1-12-3", "HEAD", (0, 11, 2)),
        ("h1-13-9", "HEAD", (0, 12, 8)),
    ],
}
obs_package = flopy.mf6.ModflowUtlobs(
    gwf,
    pname="head_obs",
    filename="{}.obs".format(model_name),
    print_input=True,
    continuous=obs_recarray,
)

```

```

[12]: # River
riv_period = {}
riv_period_array = [
    ((0, 2, 0), "river_stage_1", 1001.0, 35.9, None),
    ((0, 3, 1), "river_stage_1", 1002.0, 35.8, None),
    ((0, 4, 2), "river_stage_1", 1003.0, 35.7, None),
    ((0, 4, 3), "river_stage_1", 1004.0, 35.6, None),
    ((0, 5, 4), "river_stage_1", 1005.0, 35.5, None),
    ((0, 5, 5), "river_stage_1", 1006.0, 35.4, "riv1_c6"),
    ((0, 5, 6), "river_stage_1", 1007.0, 35.3, "riv1_c7"),
    ((0, 4, 7), "river_stage_1", 1008.0, 35.2, None),
    ((0, 4, 8), "river_stage_1", 1009.0, 35.1, None),
    ((0, 4, 9), "river_stage_1", 1010.0, 35.0, None),
    ((0, 9, 0), "river_stage_2", 1001.0, 36.9, "riv2_upper"),
    ((0, 8, 1), "river_stage_2", 1002.0, 36.8, "riv2_upper"),
    ((0, 7, 2), "river_stage_2", 1003.0, 36.7, "riv2_upper"),
    ((0, 6, 3), "river_stage_2", 1004.0, 36.6, None),
    ((0, 6, 4), "river_stage_2", 1005.0, 36.5, None),
    ((0, 5, 5), "river_stage_2", 1006.0, 36.4, "riv2_c6"),
    ((0, 5, 6), "river_stage_2", 1007.0, 36.3, "riv2_c7"),
    ((0, 6, 7), "river_stage_2", 1008.0, 36.2, None),
    ((0, 6, 8), "river_stage_2", 1009.0, 36.1),
    ((0, 6, 9), "river_stage_2", 1010.0, 36.0),
]
riv_period[0] = riv_period_array
riv = flopy.mf6.ModflowGwfriv(
    gwf,
    pname="riv",
    print_input=True,
    print_flows=True,
    save_flows="{}.cbc".format(model_name),
    boundnames=True,
    maxbound=20,
    stress_period_data=riv_period,
)
ts_recarray = [
    (0.0, 40.0, 41.0),
    (1.0, 41.0, 41.5),
    (2.0, 43.0, 42.0),

```

(continues on next page)

(continued from previous page)

```

(3.0, 45.0, 42.8),
(4.0, 44.0, 43.0),
(6.0, 43.0, 43.1),
(9.0, 42.0, 42.4),
(11.0, 41.0, 41.5),
(31.0, 40.0, 41.0),
]
riv.ts.initialize(
    filename="river_stages.ts",
    timeseries=ts_reccarray,
    time_series_namerecord=[("river_stage_1", "river_stage_2")],
    interpolation_methodrecord=[("linear", "stepwise")],
)
obs_reccarray = {
    "riv_obs.csv": [
        ("rv1-3-1", "RIV", (0, 2, 0)),
        ("rv1-4-2", "RIV", (0, 3, 1)),
        ("rv1-5-3", "RIV", (0, 4, 2)),
        ("rv1-5-4", "RIV", (0, 4, 3)),
        ("rv1-6-5", "RIV", (0, 5, 4)),
        ("rv1-c6", "RIV", "riv1_c6"),
        ("rv1-c7", "RIV", "riv1_c7"),
        ("rv2-upper", "RIV", "riv2_upper"),
        ("rv-2-7-4", "RIV", (0, 6, 3)),
        ("rv2-8-5", "RIV", (0, 6, 4)),
        (
            "rv-2-9-6",
            "RIV",
            (
                0,
                5,
                5,
            ),
        ),
    ],
    "riv_flowsA.csv": [
        ("riv1-3-1", "RIV", (0, 2, 0)),
        ("riv1-4-2", "RIV", (0, 3, 1)),
        ("riv1-5-3", "RIV", (0, 4, 2)),
    ],
    "riv_flowsB.csv": [
        ("riv2-10-1", "RIV", (0, 9, 0)),
        ("riv-2-9-2", "RIV", (0, 8, 1)),
        ("riv2-8-3", "RIV", (0, 7, 2)),
    ],
}
riv.obs.initialize(
    filename="{}/riv.obs".format(model_name),
    print_input=True,
    continuous=obs_reccarray,
)

```



```

[13]: # First recharge package
rch1_period = {}
rch1_period_array = []
col_range = {0: 3, 1: 4, 2: 5}
for row in range(0, 15):
    if row in col_range:
        col_max = col_range[row]
    else:
        col_max = 6
    for col in range(0, col_max):
        if (
            (row == 3 and col == 5)
            or (row == 2 and col == 4)
            or (row == 1 and col == 3)
            or (row == 0 and col == 2)
        ):
            mult = 0.5
        else:
            mult = 1.0
        if row == 0 and col == 0:
            bnd = "rch-1-1"
        elif row == 0 and col == 1:
            bnd = "rch-1-2"
        elif row == 1 and col == 2:
            bnd = "rch-2-3"
        else:
            bnd = None
        rch1_period_array.append(((0, row, col), "rch_1", mult, bnd))
rch1_period[0] = rch1_period_array
rch1 = flopy.mf6.ModflowGwfrch(
    gwf,
    filename="{_1}.rch".format(model_name),
    pname="rch_1",
    fixed_cell=True,
    auxiliary="MULTIPLIER",
    auxmultname="MULTIPLIER",
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    maxbound=84,
    stress_period_data=rch1_period,
)
ts_data = [
    (0.0, 0.0015),
    (1.0, 0.0010),
    (11.0, 0.0015),
    (21.0, 0.0025),
    (31.0, 0.0015),
]
rch1.ts.initialize(
    filename="recharge_rates_1.ts",
    timeseries=ts_data,

```

(continues on next page)

(continued from previous page)

```

    time_series_namerecord="rch_1",
    interpolation_methodrecord="stepwise",
)

```

```

[14]: # Second recharge package
rch2_period = {}
rch2_period_array = [
    ((0, 0, 2), "rch_2", 0.5),
    ((0, 0, 3), "rch_2", 1.0),
    ((0, 0, 4), "rch_2", 1.0),
    ((0, 0, 5), "rch_2", 1.0),
    ((0, 0, 6), "rch_2", 1.0),
    ((0, 0, 7), "rch_2", 1.0),
    ((0, 0, 8), "rch_2", 1.0),
    ((0, 0, 9), "rch_2", 0.5),
    ((0, 1, 3), "rch_2", 0.5),
    ((0, 1, 4), "rch_2", 1.0),
    ((0, 1, 5), "rch_2", 1.0),
    ((0, 1, 6), "rch_2", 1.0),
    ((0, 1, 7), "rch_2", 1.0),
    ((0, 1, 8), "rch_2", 0.5),
    ((0, 2, 4), "rch_2", 0.5),
    ((0, 2, 5), "rch_2", 1.0),
    ((0, 2, 6), "rch_2", 1.0),
    ((0, 2, 7), "rch_2", 0.5),
    ((0, 3, 5), "rch_2", 0.5),
    ((0, 3, 6), "rch_2", 0.5),
]
rch2_period[0] = rch2_period_array
rch2 = flopy.mf6.ModflowGwfrch(
    gwf,
    filename="{_2}.rch".format(model_name),
    pname="rch_2",
    fixed_cell=True,
    auxiliary="MULTIPLIER",
    auxmultname="MULTIPLIER",
    print_input=True,
    print_flows=True,
    save_flows=True,
    maxbound=20,
    stress_period_data=rch2_period,
)
ts_data = [
    (0.0, 0.0016),
    (1.0, 0.0018),
    (11.0, 0.0019),
    (21.0, 0.0016),
    (31.0, 0.0018),
]
rch2.ts.initialize(
    filename="recharge_rates_2.ts",
    timeseries=ts_data,
)

```

(continues on next page)

(continued from previous page)

```

    time_series_namerecord="rch_2",
    interpolation_methodrecord="linear",
)

```

```

[15]: # Third recharge package
rch3_period = {}
rch3_period_array = []
col_range = {0: 9, 1: 8, 2: 7}
for row in range(0, 15):
    if row in col_range:
        col_min = col_range[row]
    else:
        col_min = 6
    for col in range(col_min, 10):
        if (
            (row == 0 and col == 9)
            or (row == 1 and col == 8)
            or (row == 2 and col == 7)
            or (row == 3 and col == 6)
        ):
            mult = 0.5
        else:
            mult = 1.0
        rch3_period_array.append(((0, row, col), "rch_3", mult))
rch3_period[0] = rch3_period_array
rch3 = flopy.mf6.ModflowGwfrch(
    gwf,
    filename="{_3}.rch".format(model_name),
    pname="rch_3",
    fixed_cell=True,
    auxiliary="MULTIPLIER",
    auxmultname="MULTIPLIER",
    print_input=True,
    print_flows=True,
    save_flows=True,
    maxbound=54,
    stress_period_data=rch3_period,
)
ts_data = [
    (0.0, 0.0017),
    (1.0, 0.0020),
    (11.0, 0.0017),
    (21.0, 0.0018),
    (31.0, 0.0020),
]
rch3.ts.initialize(
    filename="recharge_rates_3.ts",
    timeseries=ts_data,
    time_series_namerecord="rch_3",
    interpolation_methodrecord="linear",
)

```

Create the MODFLOW 6 Input Files and Run the Model

Once all the flopy objects are created, it is very easy to create all of the input files and run the model.

```
[16]: # write simulation to new location
sim.write_simulation()
```

```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims...
  writing model advgw_tidal...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
    writing package oc...
    writing package sto...
    writing package wel...
    writing package ts_0...
    writing package evt...
    writing package ghb...
    writing package ts_1...
    writing package obs_0...
    writing package head_obs...
    writing package riv...
    writing package ts_2...
    writing package obs_1...
    writing package rch_1...
    writing package ts_3...
    writing package rch_2...
    writing package ts_4...
    writing package rch_3...
    writing package ts_5...
```

```
[17]: # Print a list of the files that were created
# in workspace
print(os.listdir(workspace))
```

```
['advgw_tidal_1.rch', 'advgw_tidal.ims', 'recharge_rates_1.ts', 'advgw_tidal.ic', 'advgw_
↪tidal.dis', 'advgw_tidal.wel', 'advgw_tidal.oc', 'advgw_tidal.nam', 'advgw_tidal.ghb.
↪obs', 'recharge_rates_3.ts', 'advgw_tidal.npf', 'well-rates.ts', 'advgw_tidal.evt',
↪'advgw_tidal.tdis', 'recharge_rates_2.ts', 'advgw_tidal_3.rch', 'advgw_tidal_2.rch',
↪'advgw_tidal.riv', 'tides.ts', 'river_stages.ts', 'advgw_tidal.ghb', 'advgw_tidal.obs',
↪ 'advgw_tidal.riv.obs', 'advgw_tidal.sto', 'mfsim.nam']
```

Run the Simulation

We can also run the simulation from the notebook, but only if the MODFLOW 6 executable is available. The executable can be made available by putting the executable in a folder that is listed in the system path variable. Another option is to just put a copy of the executable in the simulation folder, though this should generally be avoided. A final option is to provide a full path to the executable when the simulation is constructed. This would be done by specifying `exe_name` with the full path.

```
[18]: # Run the simulation
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")
```

```

                                MODFLOW 6
                U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
                        VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
                        Build 20220726_000000

This software has been approved for release by the U.S. Geological
Survey (USGS). Although the software has been subjected to rigorous
review, the USGS reserves the right to update the software as needed
pursuant to further analysis and review. No warranty, expressed or
implied, is made by the USGS or the U.S. Government as to the
functionality of the software and related material nor shall the
fact of release constitute any such warranty. Furthermore, the
software is released on condition that neither the USGS nor the U.S.
Government shall be held liable for any damages resulting from its
authorized or unauthorized use. Also refer to the USGS Water
Resources Software User Rights Notice for complete use, copyright,
and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:53

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

Solving: Stress period:      1      Time step:      1
Solving: Stress period:      2      Time step:      1
Solving: Stress period:      2      Time step:      2
Solving: Stress period:      2      Time step:      3
Solving: Stress period:      2      Time step:      4
Solving: Stress period:      2      Time step:      5
Solving: Stress period:      2      Time step:      6
Solving: Stress period:      2      Time step:      7
Solving: Stress period:      2      Time step:      8
Solving: Stress period:      2      Time step:      9
Solving: Stress period:      2      Time step:     10
```

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	11
Solving:	Stress period:	2	Time step:	12
Solving:	Stress period:	2	Time step:	13
Solving:	Stress period:	2	Time step:	14
Solving:	Stress period:	2	Time step:	15
Solving:	Stress period:	2	Time step:	16
Solving:	Stress period:	2	Time step:	17
Solving:	Stress period:	2	Time step:	18
Solving:	Stress period:	2	Time step:	19
Solving:	Stress period:	2	Time step:	20
Solving:	Stress period:	2	Time step:	21
Solving:	Stress period:	2	Time step:	22
Solving:	Stress period:	2	Time step:	23
Solving:	Stress period:	2	Time step:	24
Solving:	Stress period:	2	Time step:	25
Solving:	Stress period:	2	Time step:	26
Solving:	Stress period:	2	Time step:	27
Solving:	Stress period:	2	Time step:	28
Solving:	Stress period:	2	Time step:	29
Solving:	Stress period:	2	Time step:	30
Solving:	Stress period:	2	Time step:	31
Solving:	Stress period:	2	Time step:	32
Solving:	Stress period:	2	Time step:	33
Solving:	Stress period:	2	Time step:	34
Solving:	Stress period:	2	Time step:	35
Solving:	Stress period:	2	Time step:	36
Solving:	Stress period:	2	Time step:	37
Solving:	Stress period:	2	Time step:	38
Solving:	Stress period:	2	Time step:	39
Solving:	Stress period:	2	Time step:	40
Solving:	Stress period:	2	Time step:	41
Solving:	Stress period:	2	Time step:	42
Solving:	Stress period:	2	Time step:	43
Solving:	Stress period:	2	Time step:	44
Solving:	Stress period:	2	Time step:	45
Solving:	Stress period:	2	Time step:	46
Solving:	Stress period:	2	Time step:	47
Solving:	Stress period:	2	Time step:	48
Solving:	Stress period:	2	Time step:	49
Solving:	Stress period:	2	Time step:	50
Solving:	Stress period:	2	Time step:	51
Solving:	Stress period:	2	Time step:	52
Solving:	Stress period:	2	Time step:	53
Solving:	Stress period:	2	Time step:	54
Solving:	Stress period:	2	Time step:	55
Solving:	Stress period:	2	Time step:	56
Solving:	Stress period:	2	Time step:	57
Solving:	Stress period:	2	Time step:	58
Solving:	Stress period:	2	Time step:	59
Solving:	Stress period:	2	Time step:	60
Solving:	Stress period:	2	Time step:	61
Solving:	Stress period:	2	Time step:	62

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	63
Solving:	Stress period:	2	Time step:	64
Solving:	Stress period:	2	Time step:	65
Solving:	Stress period:	2	Time step:	66
Solving:	Stress period:	2	Time step:	67
Solving:	Stress period:	2	Time step:	68
Solving:	Stress period:	2	Time step:	69
Solving:	Stress period:	2	Time step:	70
Solving:	Stress period:	2	Time step:	71
Solving:	Stress period:	2	Time step:	72
Solving:	Stress period:	2	Time step:	73
Solving:	Stress period:	2	Time step:	74
Solving:	Stress period:	2	Time step:	75
Solving:	Stress period:	2	Time step:	76
Solving:	Stress period:	2	Time step:	77
Solving:	Stress period:	2	Time step:	78
Solving:	Stress period:	2	Time step:	79
Solving:	Stress period:	2	Time step:	80
Solving:	Stress period:	2	Time step:	81
Solving:	Stress period:	2	Time step:	82
Solving:	Stress period:	2	Time step:	83
Solving:	Stress period:	2	Time step:	84
Solving:	Stress period:	2	Time step:	85
Solving:	Stress period:	2	Time step:	86
Solving:	Stress period:	2	Time step:	87
Solving:	Stress period:	2	Time step:	88
Solving:	Stress period:	2	Time step:	89
Solving:	Stress period:	2	Time step:	90
Solving:	Stress period:	2	Time step:	91
Solving:	Stress period:	2	Time step:	92
Solving:	Stress period:	2	Time step:	93
Solving:	Stress period:	2	Time step:	94
Solving:	Stress period:	2	Time step:	95
Solving:	Stress period:	2	Time step:	96
Solving:	Stress period:	2	Time step:	97
Solving:	Stress period:	2	Time step:	98
Solving:	Stress period:	2	Time step:	99
Solving:	Stress period:	2	Time step:	100
Solving:	Stress period:	2	Time step:	101
Solving:	Stress period:	2	Time step:	102
Solving:	Stress period:	2	Time step:	103
Solving:	Stress period:	2	Time step:	104
Solving:	Stress period:	2	Time step:	105
Solving:	Stress period:	2	Time step:	106
Solving:	Stress period:	2	Time step:	107
Solving:	Stress period:	2	Time step:	108
Solving:	Stress period:	2	Time step:	109
Solving:	Stress period:	2	Time step:	110
Solving:	Stress period:	2	Time step:	111
Solving:	Stress period:	2	Time step:	112
Solving:	Stress period:	2	Time step:	113
Solving:	Stress period:	2	Time step:	114

(continues on next page)

(continued from previous page)

Solving:	Stress period:	2	Time step:	115
Solving:	Stress period:	2	Time step:	116
Solving:	Stress period:	2	Time step:	117
Solving:	Stress period:	2	Time step:	118
Solving:	Stress period:	2	Time step:	119
Solving:	Stress period:	2	Time step:	120
Solving:	Stress period:	3	Time step:	1
Solving:	Stress period:	3	Time step:	2
Solving:	Stress period:	3	Time step:	3
Solving:	Stress period:	3	Time step:	4
Solving:	Stress period:	3	Time step:	5
Solving:	Stress period:	3	Time step:	6
Solving:	Stress period:	3	Time step:	7
Solving:	Stress period:	3	Time step:	8
Solving:	Stress period:	3	Time step:	9
Solving:	Stress period:	3	Time step:	10
Solving:	Stress period:	3	Time step:	11
Solving:	Stress period:	3	Time step:	12
Solving:	Stress period:	3	Time step:	13
Solving:	Stress period:	3	Time step:	14
Solving:	Stress period:	3	Time step:	15
Solving:	Stress period:	3	Time step:	16
Solving:	Stress period:	3	Time step:	17
Solving:	Stress period:	3	Time step:	18
Solving:	Stress period:	3	Time step:	19
Solving:	Stress period:	3	Time step:	20
Solving:	Stress period:	3	Time step:	21
Solving:	Stress period:	3	Time step:	22
Solving:	Stress period:	3	Time step:	23
Solving:	Stress period:	3	Time step:	24
Solving:	Stress period:	3	Time step:	25
Solving:	Stress period:	3	Time step:	26
Solving:	Stress period:	3	Time step:	27
Solving:	Stress period:	3	Time step:	28
Solving:	Stress period:	3	Time step:	29
Solving:	Stress period:	3	Time step:	30
Solving:	Stress period:	3	Time step:	31
Solving:	Stress period:	3	Time step:	32
Solving:	Stress period:	3	Time step:	33
Solving:	Stress period:	3	Time step:	34
Solving:	Stress period:	3	Time step:	35
Solving:	Stress period:	3	Time step:	36
Solving:	Stress period:	3	Time step:	37
Solving:	Stress period:	3	Time step:	38
Solving:	Stress period:	3	Time step:	39
Solving:	Stress period:	3	Time step:	40
Solving:	Stress period:	3	Time step:	41
Solving:	Stress period:	3	Time step:	42
Solving:	Stress period:	3	Time step:	43
Solving:	Stress period:	3	Time step:	44
Solving:	Stress period:	3	Time step:	45
Solving:	Stress period:	3	Time step:	46

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	47
Solving:	Stress period:	3	Time step:	48
Solving:	Stress period:	3	Time step:	49
Solving:	Stress period:	3	Time step:	50
Solving:	Stress period:	3	Time step:	51
Solving:	Stress period:	3	Time step:	52
Solving:	Stress period:	3	Time step:	53
Solving:	Stress period:	3	Time step:	54
Solving:	Stress period:	3	Time step:	55
Solving:	Stress period:	3	Time step:	56
Solving:	Stress period:	3	Time step:	57
Solving:	Stress period:	3	Time step:	58
Solving:	Stress period:	3	Time step:	59
Solving:	Stress period:	3	Time step:	60
Solving:	Stress period:	3	Time step:	61
Solving:	Stress period:	3	Time step:	62
Solving:	Stress period:	3	Time step:	63
Solving:	Stress period:	3	Time step:	64
Solving:	Stress period:	3	Time step:	65
Solving:	Stress period:	3	Time step:	66
Solving:	Stress period:	3	Time step:	67
Solving:	Stress period:	3	Time step:	68
Solving:	Stress period:	3	Time step:	69
Solving:	Stress period:	3	Time step:	70
Solving:	Stress period:	3	Time step:	71
Solving:	Stress period:	3	Time step:	72
Solving:	Stress period:	3	Time step:	73
Solving:	Stress period:	3	Time step:	74
Solving:	Stress period:	3	Time step:	75
Solving:	Stress period:	3	Time step:	76
Solving:	Stress period:	3	Time step:	77
Solving:	Stress period:	3	Time step:	78
Solving:	Stress period:	3	Time step:	79
Solving:	Stress period:	3	Time step:	80
Solving:	Stress period:	3	Time step:	81
Solving:	Stress period:	3	Time step:	82
Solving:	Stress period:	3	Time step:	83
Solving:	Stress period:	3	Time step:	84
Solving:	Stress period:	3	Time step:	85
Solving:	Stress period:	3	Time step:	86
Solving:	Stress period:	3	Time step:	87
Solving:	Stress period:	3	Time step:	88
Solving:	Stress period:	3	Time step:	89
Solving:	Stress period:	3	Time step:	90
Solving:	Stress period:	3	Time step:	91
Solving:	Stress period:	3	Time step:	92
Solving:	Stress period:	3	Time step:	93
Solving:	Stress period:	3	Time step:	94
Solving:	Stress period:	3	Time step:	95
Solving:	Stress period:	3	Time step:	96
Solving:	Stress period:	3	Time step:	97
Solving:	Stress period:	3	Time step:	98

(continues on next page)

(continued from previous page)

Solving:	Stress period:	3	Time step:	99
Solving:	Stress period:	3	Time step:	100
Solving:	Stress period:	3	Time step:	101
Solving:	Stress period:	3	Time step:	102
Solving:	Stress period:	3	Time step:	103
Solving:	Stress period:	3	Time step:	104
Solving:	Stress period:	3	Time step:	105
Solving:	Stress period:	3	Time step:	106
Solving:	Stress period:	3	Time step:	107
Solving:	Stress period:	3	Time step:	108
Solving:	Stress period:	3	Time step:	109
Solving:	Stress period:	3	Time step:	110
Solving:	Stress period:	3	Time step:	111
Solving:	Stress period:	3	Time step:	112
Solving:	Stress period:	3	Time step:	113
Solving:	Stress period:	3	Time step:	114
Solving:	Stress period:	3	Time step:	115
Solving:	Stress period:	3	Time step:	116
Solving:	Stress period:	3	Time step:	117
Solving:	Stress period:	3	Time step:	118
Solving:	Stress period:	3	Time step:	119
Solving:	Stress period:	3	Time step:	120
Solving:	Stress period:	4	Time step:	1
Solving:	Stress period:	4	Time step:	2
Solving:	Stress period:	4	Time step:	3
Solving:	Stress period:	4	Time step:	4
Solving:	Stress period:	4	Time step:	5
Solving:	Stress period:	4	Time step:	6
Solving:	Stress period:	4	Time step:	7
Solving:	Stress period:	4	Time step:	8
Solving:	Stress period:	4	Time step:	9
Solving:	Stress period:	4	Time step:	10
Solving:	Stress period:	4	Time step:	11
Solving:	Stress period:	4	Time step:	12
Solving:	Stress period:	4	Time step:	13
Solving:	Stress period:	4	Time step:	14
Solving:	Stress period:	4	Time step:	15
Solving:	Stress period:	4	Time step:	16
Solving:	Stress period:	4	Time step:	17
Solving:	Stress period:	4	Time step:	18
Solving:	Stress period:	4	Time step:	19
Solving:	Stress period:	4	Time step:	20
Solving:	Stress period:	4	Time step:	21
Solving:	Stress period:	4	Time step:	22
Solving:	Stress period:	4	Time step:	23
Solving:	Stress period:	4	Time step:	24
Solving:	Stress period:	4	Time step:	25
Solving:	Stress period:	4	Time step:	26
Solving:	Stress period:	4	Time step:	27
Solving:	Stress period:	4	Time step:	28
Solving:	Stress period:	4	Time step:	29
Solving:	Stress period:	4	Time step:	30

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	31
Solving:	Stress period:	4	Time step:	32
Solving:	Stress period:	4	Time step:	33
Solving:	Stress period:	4	Time step:	34
Solving:	Stress period:	4	Time step:	35
Solving:	Stress period:	4	Time step:	36
Solving:	Stress period:	4	Time step:	37
Solving:	Stress period:	4	Time step:	38
Solving:	Stress period:	4	Time step:	39
Solving:	Stress period:	4	Time step:	40
Solving:	Stress period:	4	Time step:	41
Solving:	Stress period:	4	Time step:	42
Solving:	Stress period:	4	Time step:	43
Solving:	Stress period:	4	Time step:	44
Solving:	Stress period:	4	Time step:	45
Solving:	Stress period:	4	Time step:	46
Solving:	Stress period:	4	Time step:	47
Solving:	Stress period:	4	Time step:	48
Solving:	Stress period:	4	Time step:	49
Solving:	Stress period:	4	Time step:	50
Solving:	Stress period:	4	Time step:	51
Solving:	Stress period:	4	Time step:	52
Solving:	Stress period:	4	Time step:	53
Solving:	Stress period:	4	Time step:	54
Solving:	Stress period:	4	Time step:	55
Solving:	Stress period:	4	Time step:	56
Solving:	Stress period:	4	Time step:	57
Solving:	Stress period:	4	Time step:	58
Solving:	Stress period:	4	Time step:	59
Solving:	Stress period:	4	Time step:	60
Solving:	Stress period:	4	Time step:	61
Solving:	Stress period:	4	Time step:	62
Solving:	Stress period:	4	Time step:	63
Solving:	Stress period:	4	Time step:	64
Solving:	Stress period:	4	Time step:	65
Solving:	Stress period:	4	Time step:	66
Solving:	Stress period:	4	Time step:	67
Solving:	Stress period:	4	Time step:	68
Solving:	Stress period:	4	Time step:	69
Solving:	Stress period:	4	Time step:	70
Solving:	Stress period:	4	Time step:	71
Solving:	Stress period:	4	Time step:	72
Solving:	Stress period:	4	Time step:	73
Solving:	Stress period:	4	Time step:	74
Solving:	Stress period:	4	Time step:	75
Solving:	Stress period:	4	Time step:	76
Solving:	Stress period:	4	Time step:	77
Solving:	Stress period:	4	Time step:	78
Solving:	Stress period:	4	Time step:	79
Solving:	Stress period:	4	Time step:	80
Solving:	Stress period:	4	Time step:	81
Solving:	Stress period:	4	Time step:	82

(continues on next page)

(continued from previous page)

Solving:	Stress period:	4	Time step:	83
Solving:	Stress period:	4	Time step:	84
Solving:	Stress period:	4	Time step:	85
Solving:	Stress period:	4	Time step:	86
Solving:	Stress period:	4	Time step:	87
Solving:	Stress period:	4	Time step:	88
Solving:	Stress period:	4	Time step:	89
Solving:	Stress period:	4	Time step:	90
Solving:	Stress period:	4	Time step:	91
Solving:	Stress period:	4	Time step:	92
Solving:	Stress period:	4	Time step:	93
Solving:	Stress period:	4	Time step:	94
Solving:	Stress period:	4	Time step:	95
Solving:	Stress period:	4	Time step:	96
Solving:	Stress period:	4	Time step:	97
Solving:	Stress period:	4	Time step:	98
Solving:	Stress period:	4	Time step:	99
Solving:	Stress period:	4	Time step:	100
Solving:	Stress period:	4	Time step:	101
Solving:	Stress period:	4	Time step:	102
Solving:	Stress period:	4	Time step:	103
Solving:	Stress period:	4	Time step:	104
Solving:	Stress period:	4	Time step:	105
Solving:	Stress period:	4	Time step:	106
Solving:	Stress period:	4	Time step:	107
Solving:	Stress period:	4	Time step:	108
Solving:	Stress period:	4	Time step:	109
Solving:	Stress period:	4	Time step:	110
Solving:	Stress period:	4	Time step:	111
Solving:	Stress period:	4	Time step:	112
Solving:	Stress period:	4	Time step:	113
Solving:	Stress period:	4	Time step:	114
Solving:	Stress period:	4	Time step:	115
Solving:	Stress period:	4	Time step:	116
Solving:	Stress period:	4	Time step:	117
Solving:	Stress period:	4	Time step:	118
Solving:	Stress period:	4	Time step:	119
Solving:	Stress period:	4	Time step:	120

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:55

Elapsed run time: 1.389 Seconds

Normal termination of simulation.

Post-Process Head Results

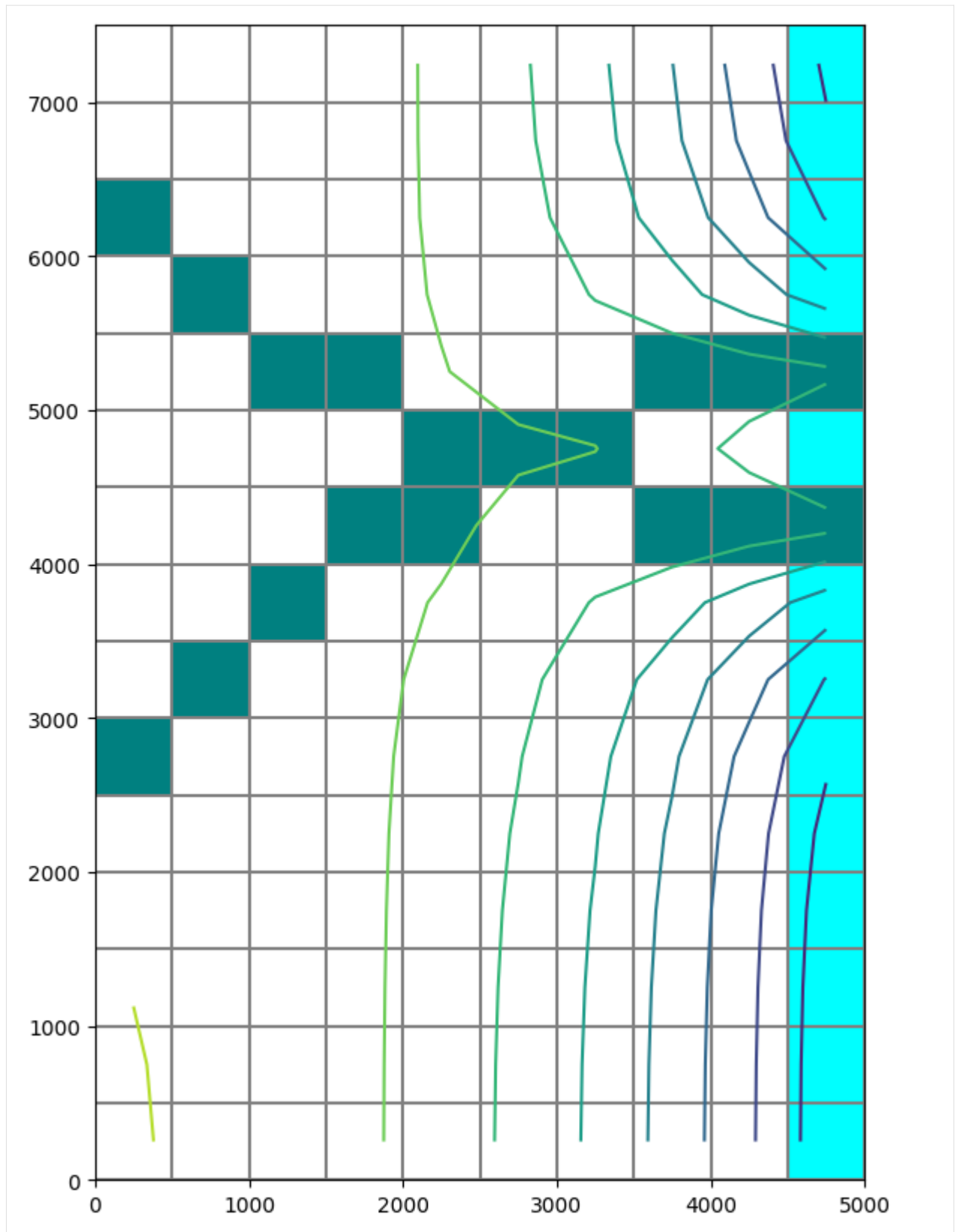
First, we get the simulated head data using the `.output.head()` method and the `get_data` function, by specifying, in this case, the step number and period number for which we want to retrieve data. A three-dimensional array is returned of size `nlay`, `nrow`, `ncol`. FloPy plotting methods are used to make contours of the head in a specific layer (in this case, layer 1). FloPy plotting methods are also used to plot the model grid and the location of GHB cells in the model domain.

```
[19]: # Retrieve the head data using the .output() method
      h = gwf.output.head().get_data(kstpkper=(0, 0))

[20]: fig = plt.figure(figsize=(10, 10))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")

      # Next we create an instance of the ModelMap class
      modelmap = flopy.plot.PlotMapView(model=gwf, ax=ax)

      ghb_quadmesh = modelmap.plot_bc(name="ghb", plotAll=True)
      riv_quadmesh = modelmap.plot_bc(name="riv", plotAll=True)
      linecollection = modelmap.plot_grid()
      contours = modelmap.contour_array(h[0])
```



Post-Process Flows

MODFLOW 6 writes a binary grid file, which contains information about the model grid. MODFLOW 6 also writes a binary budget file, which contains flow information. Both of these files can be read using FloPy methods. The `MfGrdFile` class in FloPy can be used to read the binary grid file, which contains the cell connectivity (`ia` and `ja`). The `output.budget()` method in FloPy can be used to read the binary budget file written by MODFLOW 6.

```
[21]: fname = os.path.join(workspace, "{}.dis.grb".format(model_name))
      bgf = flopy.mf6.utils.MfGrdFile(fname)
      ia, ja = bgf.ia, bgf.ja
```

```
[22]: flowja = gwf.output.budget().get_data(text="FLOW-JA-FACE")[0].squeeze()
```

```
[23]: # By having the ia and ja arrays and the flow-ja-face we can look at
      # the flows for any cell and process them in the follow manner. Note
      # layer, row, column locations are zero-based.
```

```
k = 2
i = 11
j = 2
cell_nodes = gwf.modelgrid.get_node([(k, i, j)])

for celln in cell_nodes:
    print("Printing flows for cell {}".format(celln))
    for ipos in range(ia[celln] + 1, ia[celln + 1]):
        cellm = ja[ipos]
        print(
            "Cell {} flow with cell {} is {}".format(
                celln, cellm, flowja[ipos]
            )
        )
```

```
Printing flows for cell 412
Cell 412 flow with cell 262 is 251.46262091207623
Cell 412 flow with cell 402 is 0.7176346498656017
Cell 412 flow with cell 411 is 439.8629968543804
Cell 412 flow with cell 413 is -693.4212447574185
Cell 412 flow with cell 422 is 1.3779378787739915
```

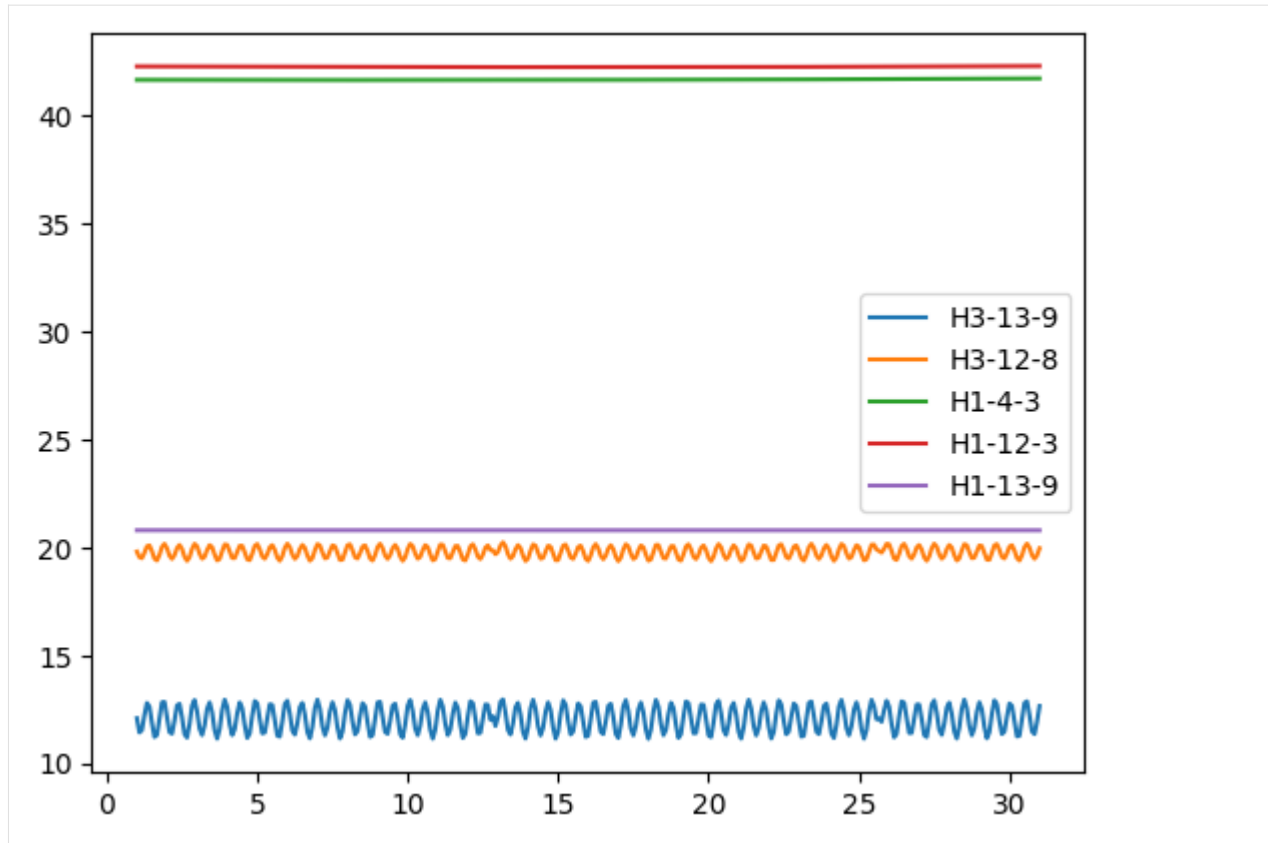
Post-Process Head Observations

MODFLOW 6 observations can be read using the `output.obs()` method in FloPy.

```
[24]: csv = gwf.head_obs.output.obs(f="head-hydrographs.csv").get_data()

      for name in csv.dtype.names[1:]:
          plt.plot(csv["totim"], csv[name], label=name)
      plt.legend()
```

```
[24]: <matplotlib.legend.Legend at 0x7f17f051c130>
```



```
[25]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass
```

6.5.2 Model splitting for parallel and serial MODFLOW 6

The model splitting functionality for MODFLOW 6 is shown in this notebook. Model splitting via the `Mf6Splitter()` class can be performed on groundwater flow models as well as combined groundwater flow and transport models. The `Mf6Splitter()` class maps a model's connectivity and then builds new models, with exchanges and movers between the new models, based on a user defined array of model numbers.

The `Mf6Splitter()` class supports Structured, Vertex, and Unstructured Grid models.

```
[1]: import os
      import sys
```

```
[2]: try:
        import flopy
    except:
        fpth = os.path.abspath(os.path.join("..", ".."))
        sys.path.append(fpth)
        import flopy
```



```
[3]: from pathlib import Path
    from tempfile import TemporaryDirectory

[4]: import matplotlib.pyplot as plt
    import numpy as np

[5]: from flopy.mf6.utils import Mf6Splitter
    from flopy.plot import styles
    from flopy.utils.geometry import LineString, Polygon

[6]: sys.path.append("../common")
    from notebook_utils import geometries, string2geom
```

Example 1: splitting a simple structured grid model

This example shows the basics of using the `Mf6Splitter()` class and applies the method to the Freyberg (1988) model.

```
[7]: simulation_ws = Path("../examples/data/mf6-freyberg")
    sim = flopy.mf6.MFSimulation.load(sim_ws=simulation_ws)

loading simulation...
  loading simulation name file...
  loading tdis package...
  loading model gw6...
    loading package dis...
    loading package ic...
WARNING: Block "options" is not a valid block name for file type ic.
    loading package oc...
    loading package npf...
    loading package sto...
    loading package chd...
    loading package riv...
    loading package wel...
    loading package rch...
  loading solution package freyberg...
```

Create a temporary directory for this example and run the Freyberg (1988) model.

```
[8]: temp_dir = TemporaryDirectory()
    workspace = Path(temp_dir.name)

[9]: sim.set_sim_path(workspace)
    sim.write_simulation()
    success, buff = sim.run_simulation(silent=True)
    assert success

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package freyberg...
  writing model freyberg...
    writing model name file...
```

(continues on next page)

(continued from previous page)

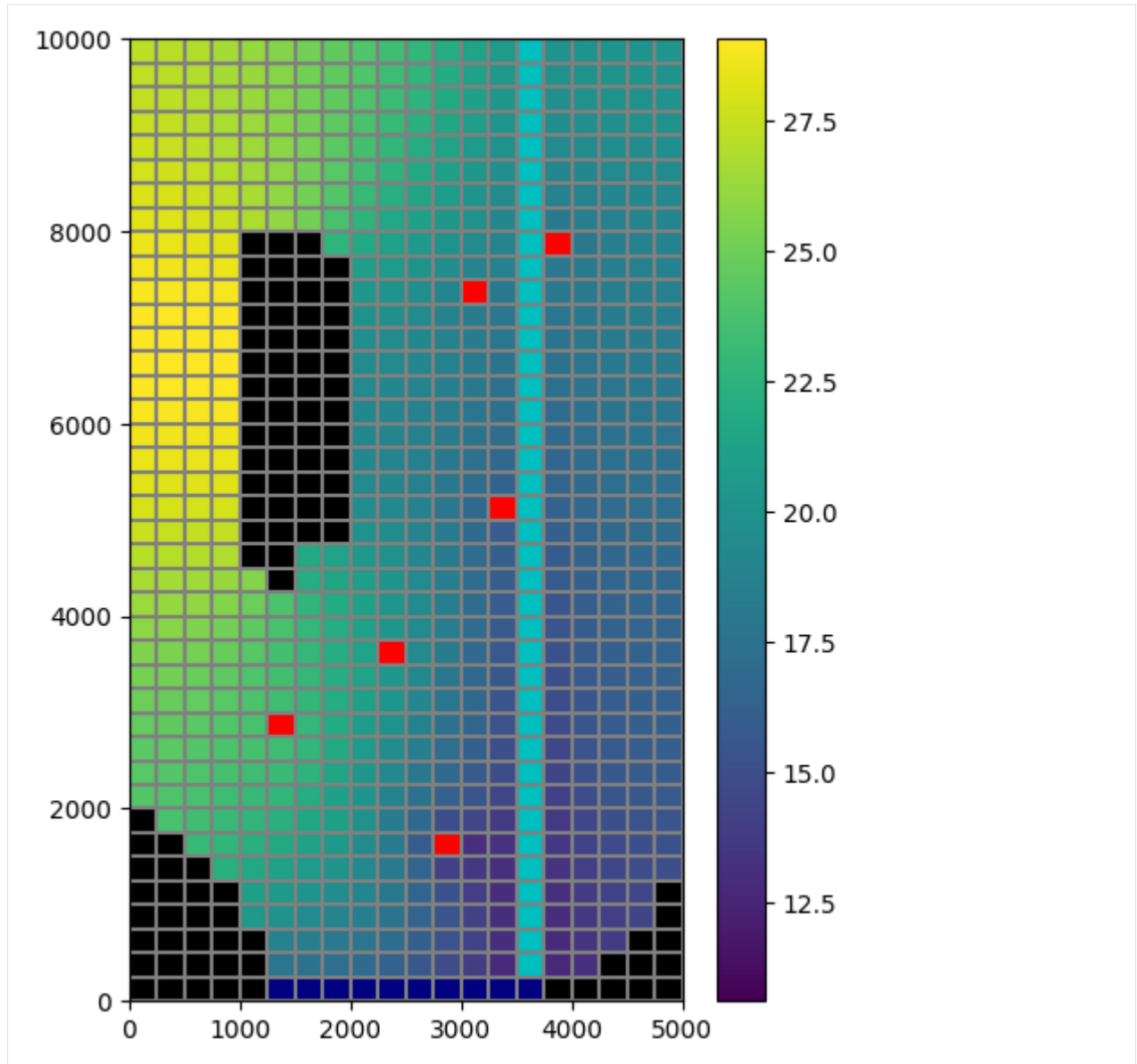
```
writing package dis...
writing package ic...
writing package oc...
writing package npf...
writing package sto...
writing package chd-1...
writing package riv-1...
writing package wel-1...
writing package rch-1...
```

Visualize the head results and boundary conditions from this model.

```
[10]: gwf = sim.get_model()
      head = gwf.output.head().get_alldata()[-1]
```

```
[11]: fig, ax = plt.subplots(figsize=(5, 7))
      pmv = flopy.plot.PlotMapView(gwf, ax=ax)
      heads = gwf.output.head().get_alldata()[-1]
      heads = np.where(heads == 1e30, np.nan, heads)
      vmin = np.nanmin(heads)
      vmax = np.nanmax(heads)
      pc = pmv.plot_array(heads, vmin=vmin, vmax=vmax)
      pmv.plot_bc("WEL")
      pmv.plot_bc("RIV", color="c")
      pmv.plot_bc("CHD")
      pmv.plot_grid()
      pmv.plot_ibound()
      plt.colorbar(pc)
```

```
[11]: <matplotlib.colorbar.Colorbar at 0x7f4cb6f4ff40>
```



Creating an array that defines the new models

In order to split models, the model domain must be discretized using unique model numbers. Any number of models can be created, however all of the cells within each model must be contiguous.

The `Mf6Splitter()` class accept arrays that are equal in size to the number of cells per layer (`StructuredGrid` and `VertexGrid`) or the number of model nodes (`UnstructuredGrid`).

In this example, the model is split diagonally into two model domains.

```
[12]: modelgrid = gwf.modelgrid
```

```
[13]: array = np.ones((modelgrid.nrow, modelgrid.ncol), dtype=int)
      ncol = 1
```

(continues on next page)

(continued from previous page)

```

for row in range(modelgrid.nrow):
    if row != 0 and row % 2 == 0:
        ncol += 1
    array[row, ncol:] = 2

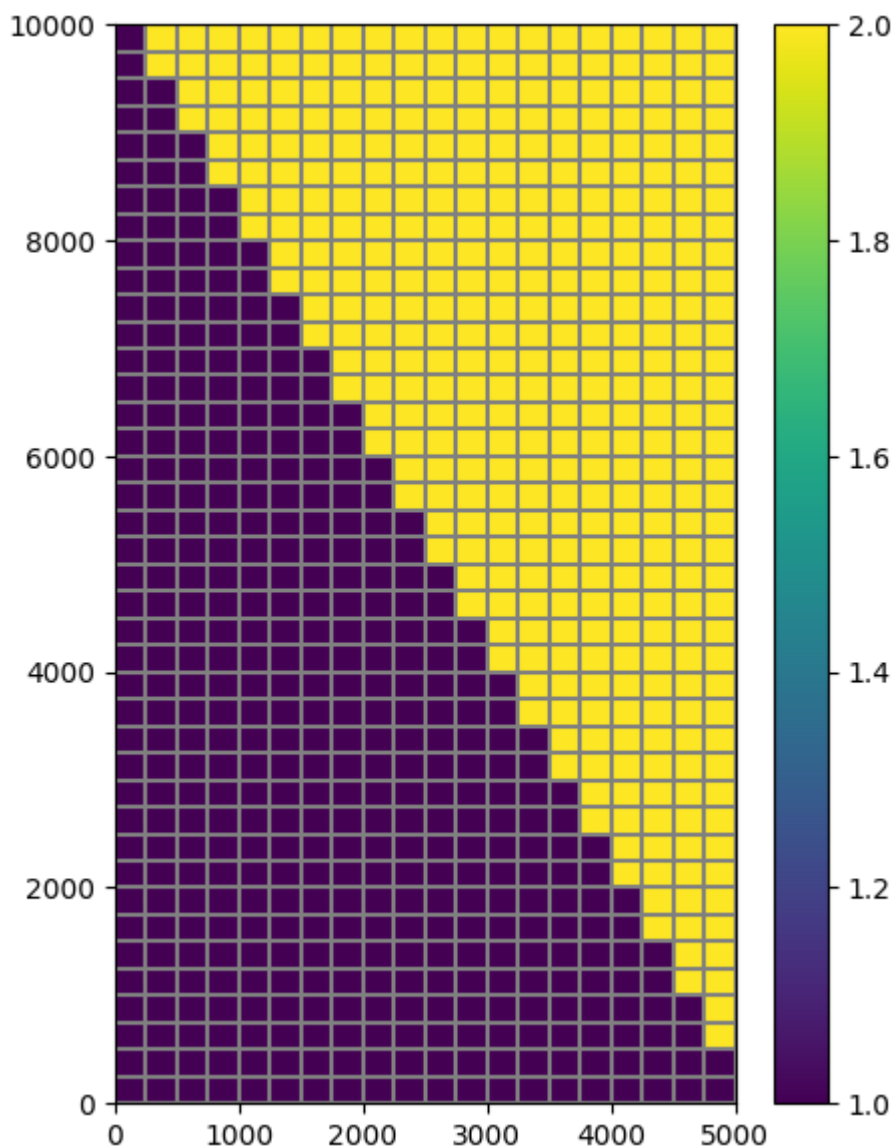
```

Plot the two domains that the model will be split into

```

[14]: fig, ax = plt.subplots(figsize=(5, 7))
      pmv = flopy.plot.PlotMapView(gwf, ax=ax)
      pc = pmv.plot_array(array)
      lc = pmv.plot_grid()
      plt.colorbar(pc)
      plt.show()

```



Splitting the model using Mf6Splitter()

The Mf6Splitter() class accepts one required parameter and one optional parameter. These parameters are: - sim: A flopy.mf6.MFSimulation object - modelname: optional, the name of the model being split. If omitted Mf6Splitter grabs the first groundwater flow model listed in the simulation

```
[15]: mfsplit = Mf6Splitter(sim)
```

The model splitting is then performed by calling the split_model() function. split_model() accepts an array that is either the same size as the number of cells per layer (StructuredGrid and VertexGrid) model or the number of nodes in the model (UnstructuredGrid).

This function returns a new MFSimulation object that contains the split models and exchanges between them

```
[16]: new_sim = mfsplit.split_model(array)
```

```
[17]: # now to write and run the simulation
new_sim.set_sim_path(workspace / "split_model")
new_sim.write_simulation()
success, buff = new_sim.run_simulation(silent=True)
assert success
```

```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims-1...
  writing package sim_1_2.gwfgwf...
  writing model freyberg_1...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package oc...
    writing package npf...
    writing package sto...
    writing package chd-1...
    writing package riv-1...
    writing package wel-1...
    writing package rch-1...
  writing model freyberg_2...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package oc...
    writing package npf...
    writing package sto...
    writing package chd-1...
    writing package riv-1...
    writing package wel-1...
    writing package rch-1...
```

Visualize and reassemble model output

Both models are visualized side by side

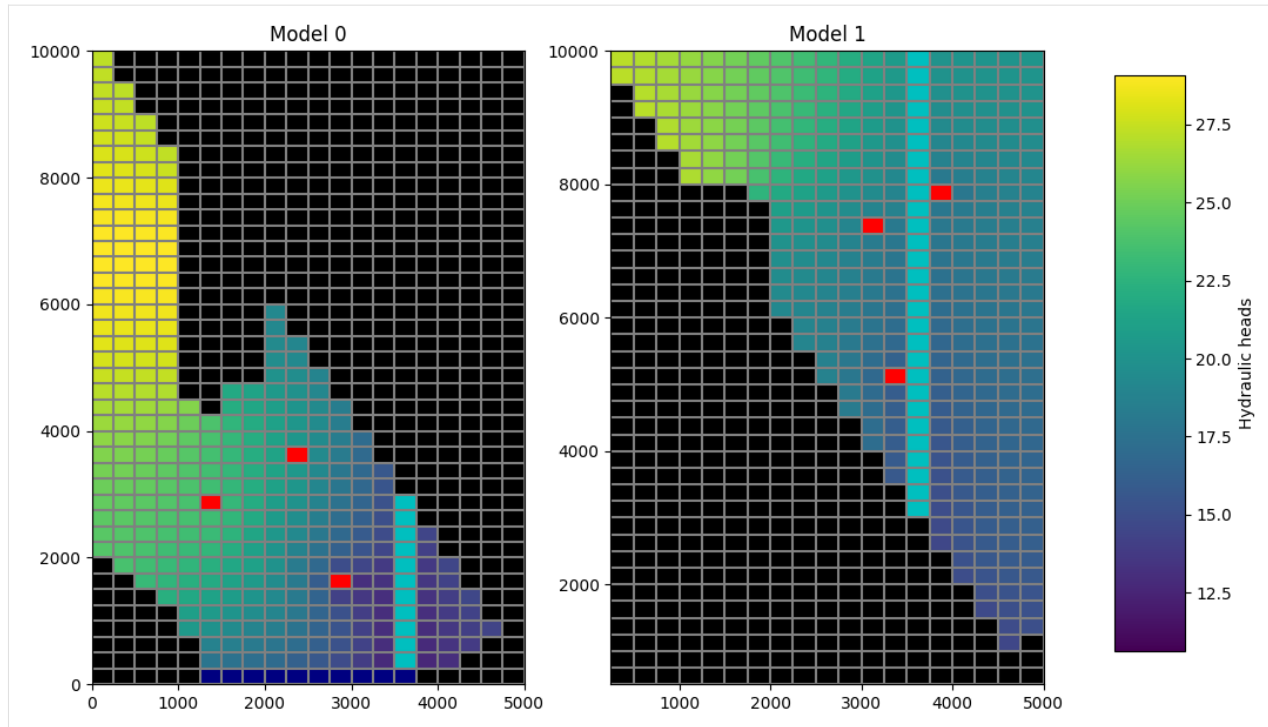
```
[18]: # visualizing both models side by side
ml0 = new_sim.get_model("freyberg_1")
ml1 = new_sim.get_model("freyberg_2")

[19]: heads0 = ml0.output.head().get_alldata()[-1]
heads1 = ml1.output.head().get_alldata()[-1]

[20]: fig, (ax0, ax1) = plt.subplots(1, 2, figsize=(12, 7))
pmv = flopy.plot.PlotMapView(ml0, ax=ax0)
pmv.plot_array(heads0, vmin=vmin, vmax=vmax)
pmv.plot_ibound()
pmv.plot_grid()
pmv.plot_bc("WEL")
pmv.plot_bc("RIV", color="c")
pmv.plot_bc("CHD")
ax0.set_title("Model 0")

pmv = flopy.plot.PlotMapView(ml1, ax=ax1)
pc = pmv.plot_array(heads1, vmin=vmin, vmax=vmax)
pmv.plot_ibound()
pmv.plot_bc("WEL")
pmv.plot_bc("RIV", color="c")
pmv.plot_grid()
ax1.set_title("Model 1")

fig.subplots_adjust(right=0.8)
cbar_ax = fig.add_axes([0.85, 0.15, 0.95, 0.7])
cbar = fig.colorbar(pc, cax=cbar_ax, label="Hydraulic heads")
```



Array based model output can be assembled into the original model's shape by using the `reconstruct_array()` method

`reconstruct_array` accepts a dictionary of array data. This data is assembled as `{model_number: array_from_model}`.

```
[21]: array_dict = {1: heads0, 2: heads1}
```

```
[22]: new_head_array = mfsplit.reconstruct_array(array_dict)
```

Recarray based model inputs and outputs can also be assembled into the original model's shape by using the `reconstruct_recarray()` method

The code below demonstrates how to join the input recarrays for the WEL, RIV, and CHD package and plot them as boundary condition arrays.

```
[23]: models = [m10, m11]
```

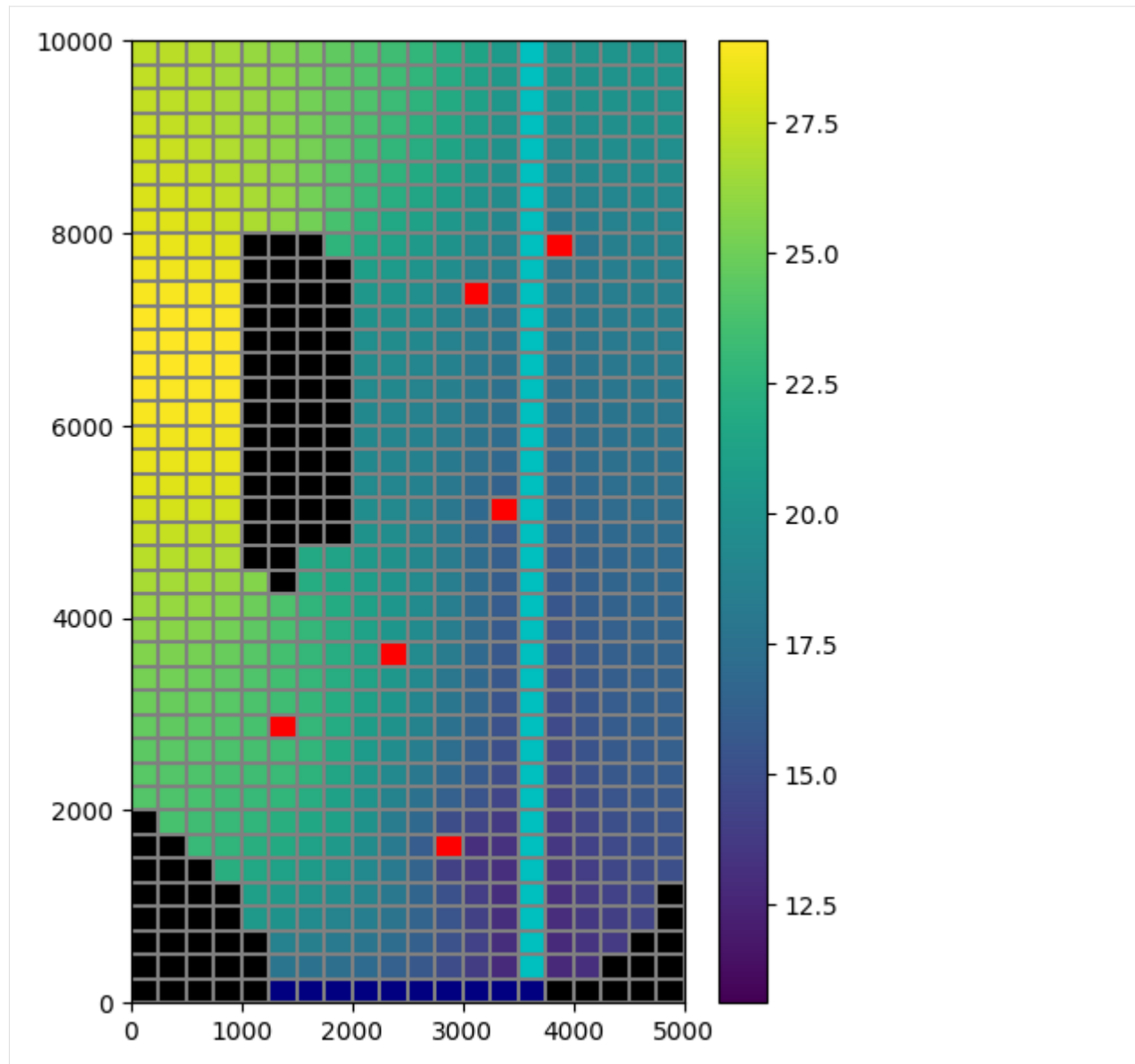
```
[24]: pkgs = ["wel", "riv", "chd"]
d = {}
for pkg in pkgs:
    rarrays = {}
    for ix, model in enumerate(models):
        pak = model.get_package(pkg)
        try:
            rarrays[ix + 1] = pak.stress_period_data.data[0]
```

(continues on next page)

(continued from previous page)

```
    except TypeError:
        pass
    recarray = mfsplit.reconstruct_recarray(rarrays)
    if pkg == "riv":
        color = "c"
        bc_array, kwargs = mfsplit.recarray_bc_array(recarray, color="c")
    else:
        bc_array, kwargs = mfsplit.recarray_bc_array(recarray, pkgtype=pkg)
    d[pkg] = {"bc_array": bc_array, "kwargs": kwargs}
```

```
[25]: fig, ax = plt.subplots(figsize=(5, 7))
      pmv = flopy.plot.PlotMapView(gwf, ax=ax)
      pc = pmv.plot_array(new_head_array, vmin=vmin, vmax=vmax)
      pmv.plot_ibound()
      pmv.plot_grid()
      pmv.plot_array(d["wel"]["bc_array"], **d["wel"]["kwargs"])
      pmv.plot_array(d["riv"]["bc_array"], **d["riv"]["kwargs"])
      pmv.plot_array(d["chd"]["bc_array"], **d["chd"]["kwargs"])
      plt.colorbar(pc)
      plt.show()
```

Example 2: a more comprehensive example with the watershed model from Hughes and others 2023

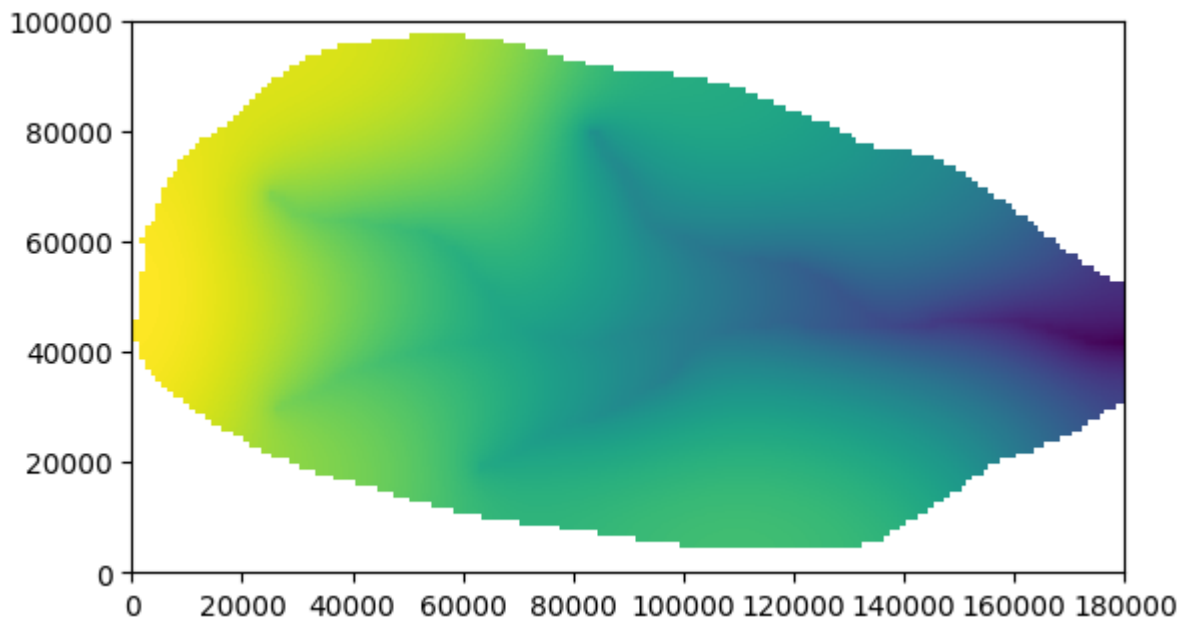
In this example, a basin model is created and is split into many models. From Hughes, Joseph D., Langevin, Christian D., Paulinski, Scott R., Larsen, Joshua D., and Brakenhoff, David, 2023, FloPy Workflows for Creating Structured and Unstructured MODFLOW Models: Groundwater, <https://doi.org/10.1111/gwat.13327>

Create the model

Load an ASCII raster file

```
[26]: ascii_file = Path("../examples/data/geospatial/fine_topo.asc")
```

```
[27]: fine_topo = flopy.utils.Raster.load(ascii_file)
fine_topo.plot()
```



```
[27]: <Axes: >
```

```
[28]: Lx = 180000
Ly = 100000
extent = (0, Lx, 0, Ly)
levels = np.arange(10, 110, 10)
vmin, vmax = 0.0, 100.0
```

```
[29]: temp_dir = TemporaryDirectory()
workspace = Path(temp_dir.name)
```

```
[30]: boundary_polygon = string2geom geometries["boundary"])
boundary_polygon.append(boundary_polygon[0])
bp = np.array(boundary_polygon)
```

```
[31]: # define stream segment locations
segs = [string2geom(geometries[f"streamseg{i}")) for i in range(1, 5)]
```

Plot the model boundary and the individual stream segments for the RIV package

```
[32]: fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot()
```

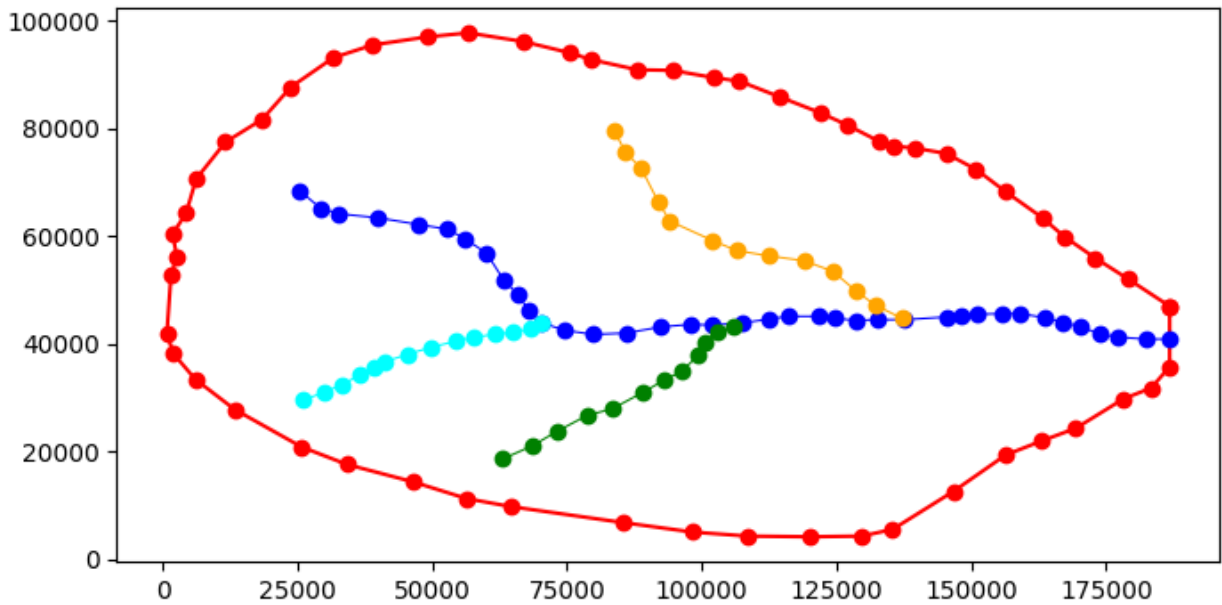
(continues on next page)

(continued from previous page)

```

ax.set_aspect("equal")
riv_colors = ("blue", "cyan", "green", "orange", "red")
ax.plot(bp[:, 0], bp[:, 1], "ro-")
for idx, seg in enumerate(segs):
    sa = np.array(seg)
    ax.plot(sa[:, 0], sa[:, 1], color=riv_colors[idx], lw=0.75, marker="o")

```



Create a MODFLOW model grid

```

[33]: dx = dy = 5000
      dv0 = 5.0
      nlay = 1
      nrow = int(Ly / dy) + 1
      ncol = int(Lx / dx) + 1
      delr = np.array(ncol * [dx])
      delc = np.array(nrow * [dy])
      top = np.ones((nrow, ncol)) * 1000.0
      botm = np.ones((nlay, nrow, ncol)) * -100.0
      modelgrid = flopy.discretization.StructuredGrid(
          nlay=nlay, delr=delr, delc=delc, xoff=0, yoff=0, top=top, botm=botm
      )

```

Crop the raster, resample it for the top elevation, and create an ibound array

```

[34]: new_top = fine_topo.resample_to_grid(
      modelgrid, band=fine_topo.bands[0], method="min", extrapolate_edges=True
  )

```

```

[35]: # calculate and set idomain
      ix = flopy.utils.GridIntersect(modelgrid, method="vertex", rtree=True)
      result = ix.intersect(Polygon(boundary_polygon))
      idxs = tuple(zip(*result.cellids))

```

(continues on next page)

(continued from previous page)

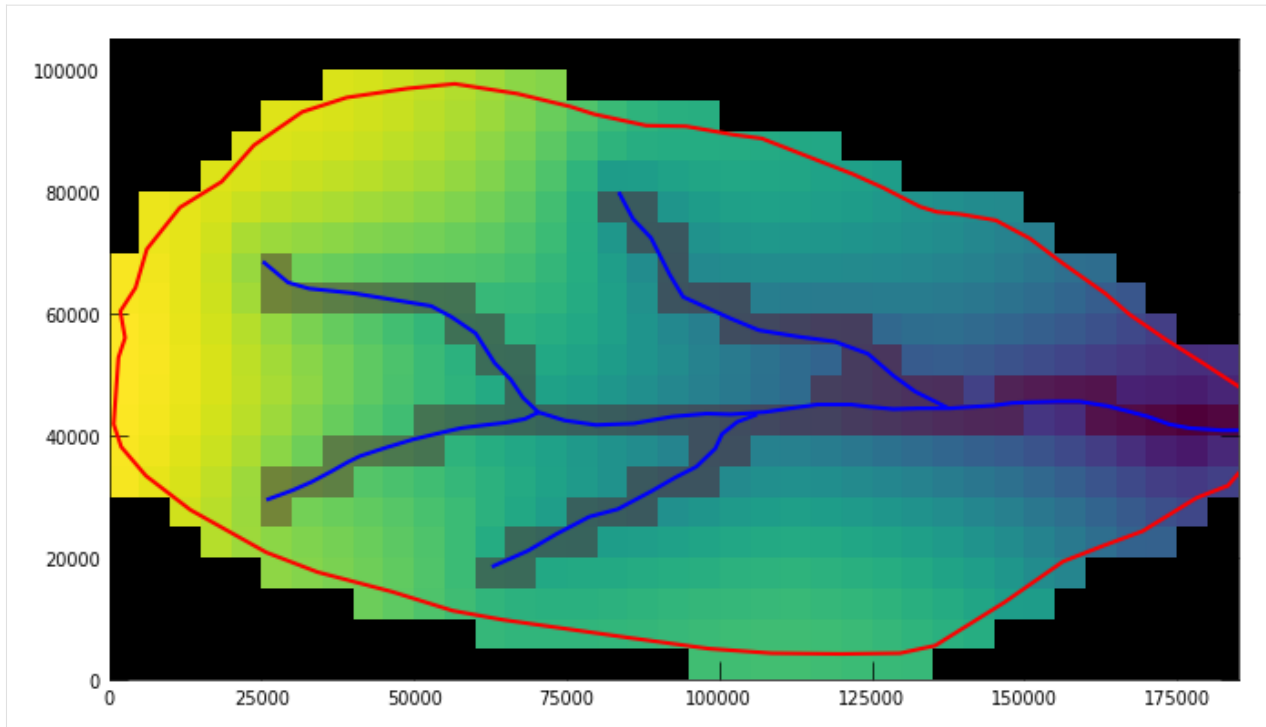
```
idomain = np.zeros((nrow, ncol), dtype=int)
idomain[idxs] = 1
```

```
[36]: # set this idomain and top to the modelgrid
modelgrid._idomain = idomain
modelgrid._top = new_top
```

Intersect the stream segments with the modelgrid

```
[37]: ix = flopy.utils.GridIntersect(modelgrid, method="structured")
cellids = []
for seg in segs:
    v = ix.intersect(LineString(seg), sort_by_cellid=True)
    cellids += v["cellids"].tolist()
intersection_rg = np.zeros(modelgrid.shape[1:])
for loc in cellids:
    intersection_rg[loc] = 1
```

```
[38]: with styles.USGSMap():
    fig, ax = plt.subplots(figsize=(8, 8))
    pmv = flopy.plot.PlotMapView(modelgrid=modelgrid)
    ax.set_aspect("equal")
    pmv.plot_array(modelgrid.top)
    pmv.plot_array(
        intersection_rg,
        masked_values=[
            0,
        ],
        alpha=0.2,
        cmap="Reds_r",
    )
    pmv.plot_inactive()
    ax.plot(bp[:, 0], bp[:, 1], "r-")
    for seg in segs:
        sa = np.array(seg)
        ax.plot(sa[:, 0], sa[:, 1], "b-")
```



Calculate drain conductance, set simulation options, and begin building model arrays

```
[39]: # Set number of model layers to 2
nlay = 2

[40]: # intersect stream segs to simulate as drains
ixs = flopy.utils.GridIntersect(modelgrid, method="structured")
drn_cellids = []
drn_lengths = []
for seg in segs:
    v = ixs.intersect(LineString(seg), sort_by_cellid=True)
    drn_cellids += v["cellids"].tolist()
    drn_lengths += v["lengths"].tolist()

[41]: leakance = 1.0 / (0.5 * dv0) # kv / b
drn_data = []
for (r, c), length in zip(drn_cellids, drn_lengths):
    x = modelgrid.xcellcenters[r, c]
    width = 5.0 + (14.0 / Lx) * (Lx - x)
    conductance = leakance * length * width
    drn_data.append((0, r, c, modelgrid.top[r, c], conductance))
drn_data[:10]

[41]: [(0, 12, 36, 3.6404473781585693, 9622.185546471488),
(0, 12, 35, 1.111111044883728, 10478.844160380193),
(0, 12, 34, 3.8888888359069824, 11731.97414523631),
(0, 12, 33, 6.666666507720947, 12346.648887868123),
(0, 12, 32, 9.44444465637207, 6006.996161832478),
(0, 11, 32, 10.555556297302246, 6942.175535223257),
```

(continues on next page)

(continued from previous page)

```
(0, 11, 31, 12.222222328186035, 13534.73438485134),
(0, 11, 30, 15.0, 14284.510735821312),
(0, 11, 29, 17.77777862548828, 13441.64541297737),
(0, 12, 29, 20.0, 1699.7820641189026)]
```

```
[42]: # groundwater discharge to surface
idomain = modelgrid.idomain.copy()
index = tuple(zip(*drn_cellids))
idomain[index] = -1

gw_discharge_data = []
for r in range(nrow):
    for c in range(ncol):
        if idomain[r, c] < 1:
            continue
        conductance = leakance * dx * dy
        gw_discharge_data.append(
            (0, r, c, modelgrid.top[r, c] - 0.5, conductance, 1.0)
        )
gw_discharge_data[:10]
```

```
[42]: [(0, 1, 7, 99.99099731445312, 100000000.0, 1.0),
(0, 1, 8, 98.9864730834961, 100000000.0, 1.0),
(0, 1, 9, 97.68162536621094, 100000000.0, 1.0),
(0, 1, 10, 96.02616882324219, 100000000.0, 1.0),
(0, 1, 11, 93.93696594238281, 100000000.0, 1.0),
(0, 1, 12, 91.06060028076172, 100000000.0, 1.0),
(0, 1, 13, 87.65284729003906, 100000000.0, 1.0),
(0, 1, 14, 86.53125, 100000000.0, 1.0),
(0, 2, 5, 100.72657012939453, 100000000.0, 1.0),
(0, 2, 6, 100.33457946777344, 100000000.0, 1.0)]
```

```
[43]: botm = np.zeros((nlay, nrow, ncol))
botm[0] = modelgrid.top - dv0
for ix in range(1, nlay):
    dv0 *= 1.5
    botm[ix] = botm[ix - 1] - dv0
```

```
[44]: idomain = np.zeros((nlay, nrow, ncol), dtype=int)
idomain[:] = modelgrid.idomain

strt = np.zeros((nlay, nrow, ncol))
strt[:] = modelgrid.top
```

Create the watershed model using FloPy

```
[45]: temp_dir = TemporaryDirectory()
workspace = Path(temp_dir.name) / "basin"
```

```
[46]: sim = flopy.mf6.MFSimulation(
    sim_name="basin",
    sim_ws=workspace,
```

(continues on next page)

(continued from previous page)

```

    exe_name="mf6",
)

tdis = flopy.mf6.ModflowTdis(sim)
ims = flopy.mf6.ModflowIms(
    sim,
    complexity="simple",
    print_option="SUMMARY",
    linear_acceleration="bicgstab",
    outer_maximum=1000,
    inner_maximum=100,
    outer_dvclose=1e-5,
    inner_dvclose=1e-6,
)
gwf = flopy.mf6.ModflowGwf(
    sim,
    save_flows=True,
    newtonoptions="NEWTON UNDER_RELAXATION",
)

dis = flopy.mf6.ModflowGwfdis(
    gwf,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=dx,
    delc=dy,
    idomain=idomain,
    top=modelgrid.top,
    botm=botm,
    xorigin=0.0,
    yorigin=0.0,
)

ic = flopy.mf6.ModflowGwfic(gwf, strt=strt)
npf = flopy.mf6.ModflowGwfnpf(
    gwf,
    save_specific_discharge=True,
    icelltype=1,
    k=1.0,
)
sto = flopy.mf6.ModflowGwfsto(
    gwf,
    iconvert=1,
    ss=1e-5,
    sy=0.2,
    steady_state=True,
)
rch = flopy.mf6.ModflowGwfrcha(
    gwf,
    recharge=0.000001,
)

```

(continues on next page)

(continued from previous page)

```

drn = flopy.mf6.ModflowGwdrn(
    gwf,
    stress_period_data=drn_data,
    pname="river",
)
drn_gwd = flopy.mf6.ModflowGwdrn(
    gwf,
    auxiliary=["depth"],
    auxdepthname="depth",
    stress_period_data=gw_discharge_data,
    pname="gwd",
)
oc = flopy.mf6.ModflowGwfoc(
    gwf,
    head_filerecord=f"{gwf.name}.hds",
    budget_filerecord=f"{gwf.name}.cbc",
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
    printrecord=[("BUDGET", "ALL")],
)

```

```

[47]: sim.write_simulation()
success, buff = sim.run_simulation(silent=True)
assert success

```

```

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims_1...
  writing model model...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
    writing package sto...
    writing package rcha_0...
    writing package river...
INFORMATION: maxbound in ('gwf6', 'drn', 'dimensions') changed to 88 based on size of
↳ stress_period_data
  writing package gwd...
INFORMATION: maxbound in ('gwf6', 'drn', 'dimensions') changed to 470 based on size of
↳ stress_period_data
  writing package oc...

```

Plot the model results

```

[48]: water_table = flopy.utils.postprocessing.get_water_table(
    gwf.output.head().get_data()
)
heads = gwf.output.head().get_data()
hmin, hmax = water_table.min(), water_table.max()
contours = np.arange(0, 100, 10)
hmin, hmax

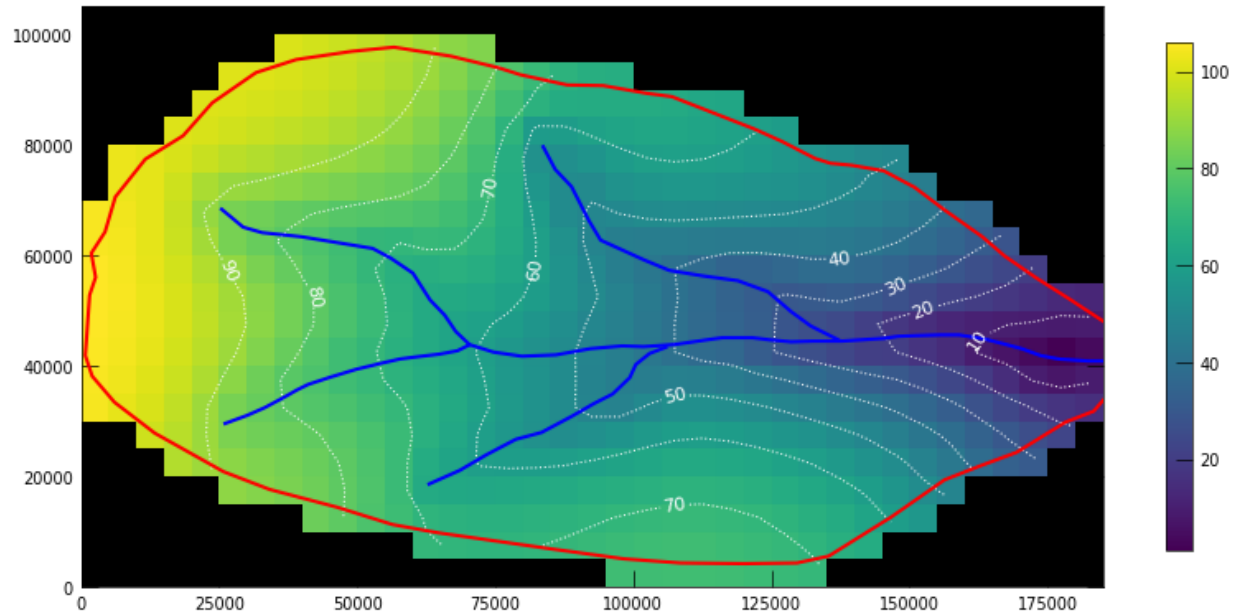
```



```
[48]: (1.1351828298028546, 105.9680367518995)
```

```
[49]: with styles.USGSMap():
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot()
    ax.set_xlim(0, Lx)
    ax.set_ylim(0, Ly)
    ax.set_aspect("equal")
    pmv = flopy.plot.PlotMapView(modelgrid=gwf.modelgrid, ax=ax)
    h = pmv.plot_array(heads, vmin=hmin, vmax=hmax)
    c = pmv.contour_array(
        water_table,
        levels=contours,
        colors="white",
        linewidths=0.75,
        linestyle=":",
    )
    plt.clabel(c, fontsize=8)
    pmv.plot_inactive()
    plt.colorbar(h, ax=ax, shrink=0.5)

    ax.plot(bp[:, 0], bp[:, 1], "r-")
    for seg in segs:
        sa = np.array(seg)
        ax.plot(sa[:, 0], sa[:, 1], "b-")
```



Split the watershed model

Build a splitting array and split this model into many models for parallel modflow runs

```
[50]: nrow_blocks, ncol_blocks = 2, 4
      row_inc, col_inc = int(nrow / nrow_blocks), int(ncol / ncol_blocks)
      row_inc, col_inc
```

```
[50]: (10, 9)
```

```
[51]: icnt = 0
      row_blocks = [icnt]
      for i in range(nrow_blocks):
          icnt += row_inc
          row_blocks.append(icnt)
      if row_blocks[-1] < nrow:
          row_blocks[-1] = nrow
      row_blocks
```

```
[51]: [0, 10, 21]
```

```
[52]: icnt = 0
      col_blocks = [icnt]
      for i in range(ncol_blocks):
          icnt += col_inc
          col_blocks.append(icnt)
      if col_blocks[-1] < ncol:
          col_blocks[-1] = ncol
      col_blocks
```

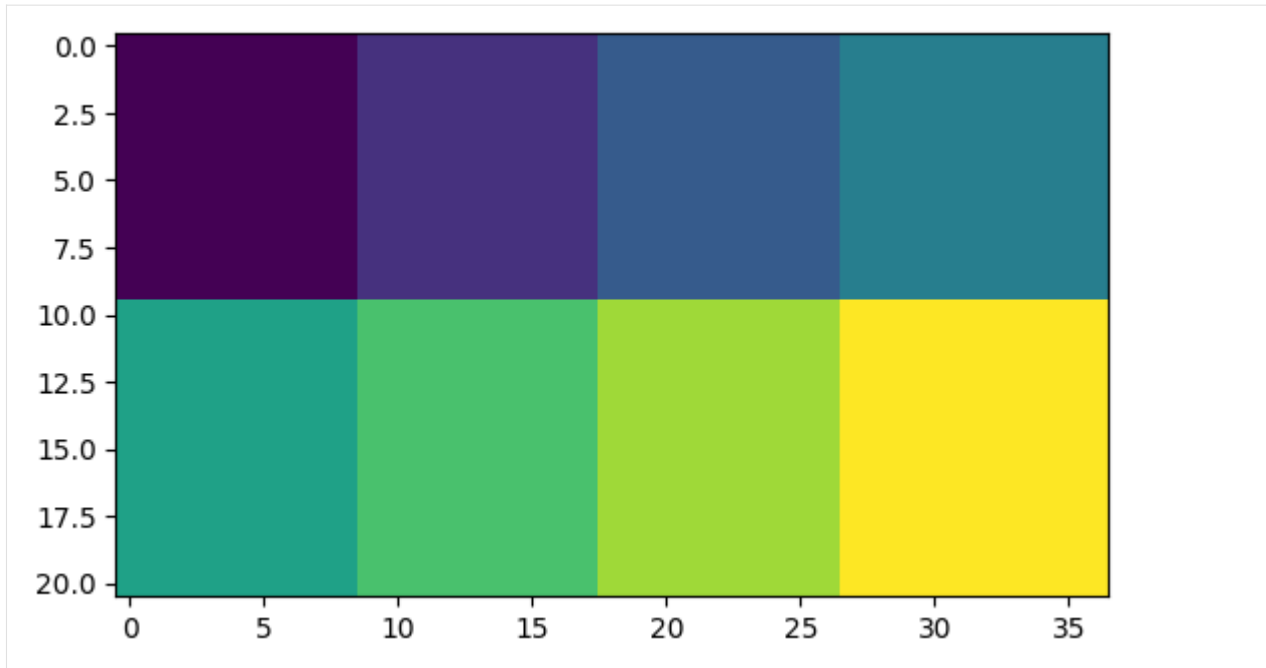
```
[52]: [0, 9, 18, 27, 37]
```

```
[53]: mask = np.zeros((nrow, ncol), dtype=int)
```

```
[54]: # create masking array
      ival = 1
      model_row_col_offset = {}
      for idx in range(len(row_blocks) - 1):
          for jdx in range(len(col_blocks) - 1):
              mask[
                  row_blocks[idx] : row_blocks[idx + 1],
                  col_blocks[jdx] : col_blocks[jdx + 1],
              ] = ival
              model_row_col_offset[ival - 1] = (row_blocks[idx], col_blocks[jdx])
              # increment model number
              ival += 1
```

```
[55]: plt.imshow(mask)
```

```
[55]: <matplotlib.image.AxesImage at 0x7f4ca10e4370>
```



Now split the model into many models using `Mf6Splitter()`

```
[56]: mfsplit = Mf6Splitter(sim)
      new_sim = mfsplit.split_model(mask)
```

```
[57]: new_ws = workspace / "split_models"
      new_sim.set_sim_path(new_ws)
      new_sim.write_simulation()
      success, buff = new_sim.run_simulation(silent=True)
      assert success
```

```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims_-1...
  writing package sim_1_2.gwfgwf...
  writing package sim_1_5.gwfgwf...
  writing package sim_2_3.gwfgwf...
  writing package sim_2_6.gwfgwf...
  writing package sim_3_4.gwfgwf...
  writing package sim_3_7.gwfgwf...
  writing package sim_4_8.gwfgwf...
  writing package sim_5_6.gwfgwf...
  writing package sim_6_7.gwfgwf...
  writing package sim_7_8.gwfgwf...
  writing model model_1...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
```

(continues on next page)

(continued from previous page)

```
writing package sto...
writing package rcha_0...
writing package river...
writing package gwd...
writing package oc...
writing model model_2...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
  writing package river...
  writing package gwd...
  writing package oc...
writing model model_3...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
  writing package river...
  writing package gwd...
  writing package oc...
writing model model_4...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
  writing package river...
  writing package gwd...
  writing package oc...
writing model model_5...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
  writing package river...
  writing package gwd...
  writing package oc...
writing model model_6...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
```

(continues on next page)

(continued from previous page)

```

writing package river...
writing package gwd...
writing package oc...
writing model model_7...
writing model name file...
writing package dis...
writing package ic...
writing package npf...
writing package sto...
writing package rcha_0...
writing package river...
writing package gwd...
writing package oc...
writing model model_8...
writing model name file...
writing package dis...
writing package ic...
writing package npf...
writing package sto...
writing package rcha_0...
writing package river...
writing package gwd...
writing package oc...

```

Reassemble the heads to the original model shape for plotting

Create a dictionary of model number : heads and use the `reconstruct_array()` method to get a numpy array that is the original shape of the unsplit model.

```

[58]: model_names = list(new_sim.model_names)
      head_dict = {}
      for modelname in model_names:
          mnum = int(modelname.split("_")[-1])
          head = new_sim.get_model(modelname).output.head().get_alldata()[-1]
          head_dict[mnum] = head

```

```

[59]: ra_heads = mfsplit.reconstruct_array(head_dict)
      ra_watertable = flopy.utils.postprocessing.get_water_table(ra_heads)

```

```

[60]: with styles.USGSMap():
      fig, axs = plt.subplots(nrows=3, figsize=(8, 12))
      diff = ra_heads - heads
      hv = [ra_heads, heads, diff]
      titles = ["Multiple models", "Single model", "Multiple - single"]
      for idx, ax in enumerate(axs):
          ax.set_aspect("equal")
          ax.set_title(titles[idx])

      if idx < 2:
          levels = contours

```

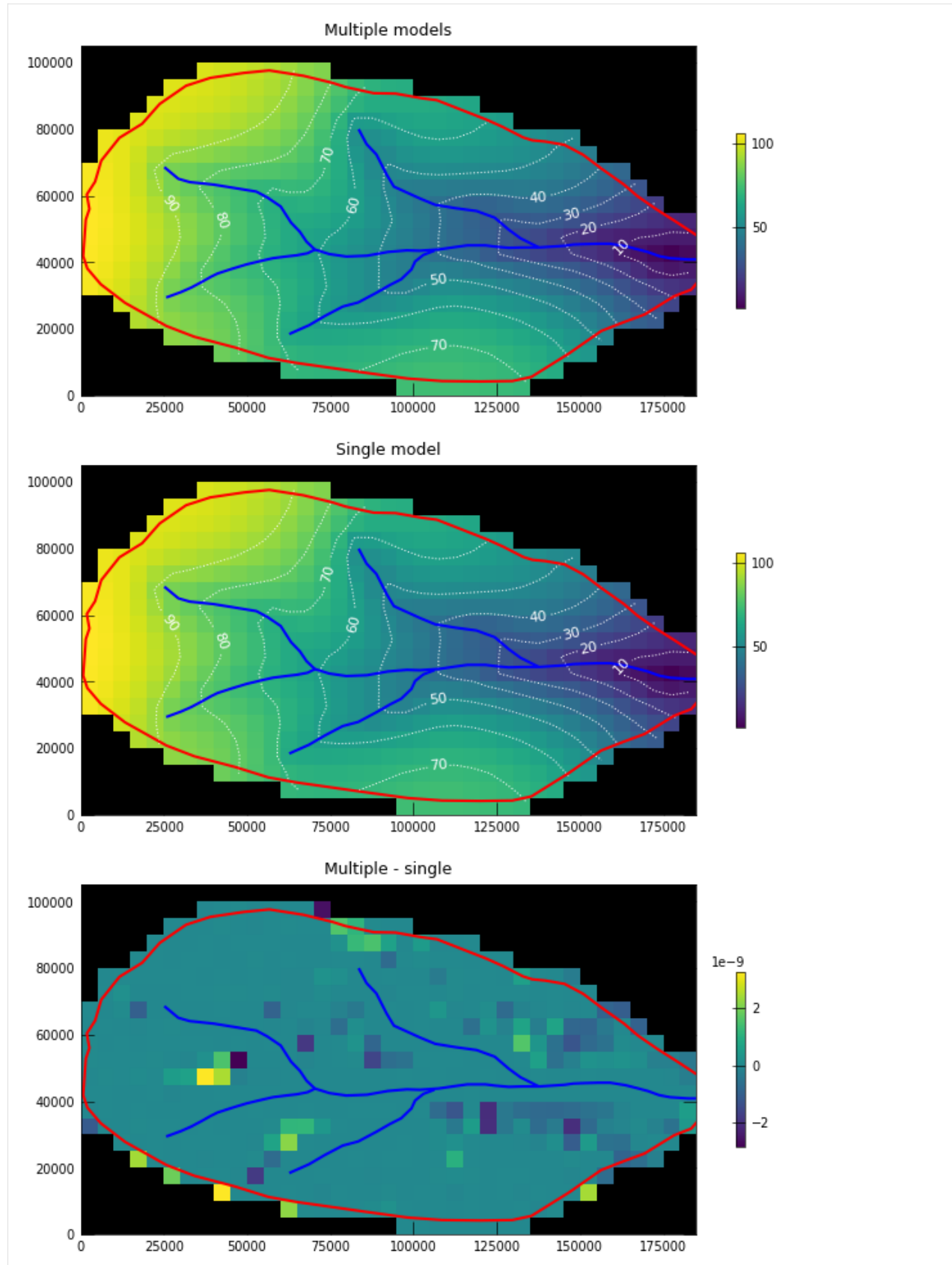
(continues on next page)

(continued from previous page)

```
vmin = hmin
vmax = hmax
else:
    levels = None
    vmin = None
    vmax = None

pmv = flopy.plot.PlotMapView(modelgrid=gwf.modelgrid, ax=ax, layer=0)
h = pmv.plot_array(hv[idx], vmin=vmin, vmax=vmax)
if levels is not None:
    c = pmv.contour_array(
        hv[idx],
        levels=levels,
        colors="white",
        linewidths=0.75,
        linestyle=":",
    )
    plt.clabel(c, fontsize=8)
pmv.plot_inactive()
plt.colorbar(h, ax=ax, shrink=0.5)

ax.plot(bp[:, 0], bp[:, 1], "r-")
for seg in segs:
    sa = np.array(seg)
    ax.plot(sa[:, 0], sa[:, 1], "b-")
```



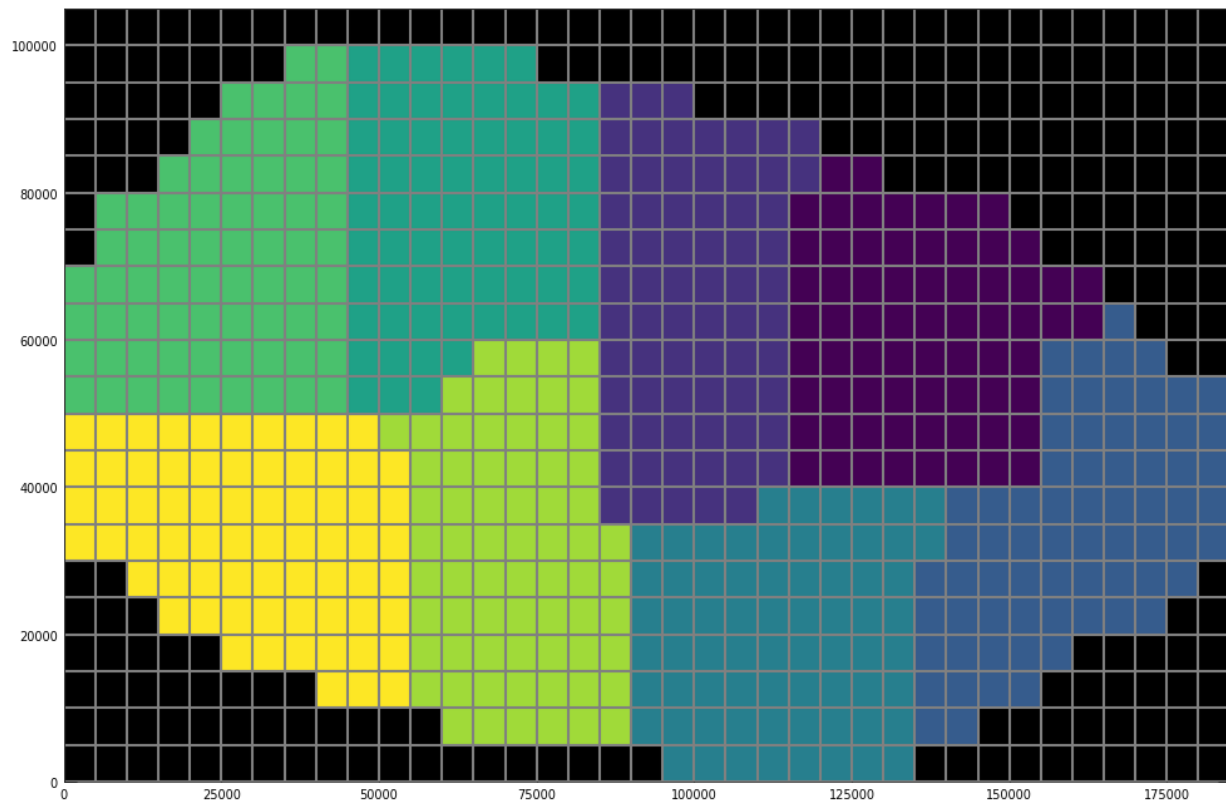
Example 3: create an optimized splitting mask for a model

In the previous examples, the watershed model splitting mask was defined by the user. `Mf6Splitter` also has a method called `optimize_splitting_mask` that creates a mask based on the number of models the user would like to generate.

The `optimize_splitting_mask()` method generates a vertex weighted adjacency graph, based on the number active and inactive nodes in all layers of the model. This adjacency graph is then provided to `pymetis` which does the work for us and returns a membership array for each node.

```
[61]: # Split the watershed model into many models
mfsplit = Mf6Splitter(sim)
split_array = mfsplit.optimize_splitting_mask(nparts=8)
```

```
with styles.USGSMap():
    fig, ax = plt.subplots(figsize=(12, 8))
    pmv = flopy.plot.PlotMapView(gwf, ax=ax)
    pmv.plot_array(split_array)
    pmv.plot_inactive()
    pmv.plot_grid()
```



```
[62]: new_sim = mfsplit.split_model(split_array)

temp_dir = TemporaryDirectory()
workspace = Path("temp")

new_ws = workspace / "opt_split_models"
new_sim.set_sim_path(new_ws)
```

(continues on next page)

(continued from previous page)

```

new_sim.write_simulation()
success, buff = new_sim.run_simulation(silent=True)
assert success

```

```

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims_-1...
  writing package sim_0_1.gwfgwf...
  writing package sim_0_2.gwfgwf...
  writing package sim_0_3.gwfgwf...
  writing package sim_1_3.gwfgwf...
  writing package sim_1_4.gwfgwf...
  writing package sim_1_6.gwfgwf...
  writing package sim_2_3.gwfgwf...
  writing package sim_3_6.gwfgwf...
  writing package sim_4_5.gwfgwf...
  writing package sim_4_6.gwfgwf...
  writing package sim_4_7.gwfgwf...
  writing package sim_5_7.gwfgwf...
  writing package sim_6_7.gwfgwf...
writing model model_0...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
  writing package river...
  writing package gwd...
  writing package oc...
writing model model_1...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
  writing package river...
  writing package gwd...
  writing package oc...
writing model model_2...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
  writing package river...
  writing package gwd...
  writing package oc...
writing model model_3...

```

(continues on next page)

(continued from previous page)

```
writing model name file...
writing package dis...
writing package ic...
writing package npf...
writing package sto...
writing package rcha_0...
writing package river...
writing package gwd...
writing package oc...
writing model model_4...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
  writing package river...
  writing package gwd...
  writing package oc...
writing model model_5...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
  writing package river...
  writing package gwd...
  writing package oc...
writing model model_6...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
  writing package river...
  writing package gwd...
  writing package oc...
writing model model_7...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package sto...
  writing package rcha_0...
  writing package river...
  writing package gwd...
  writing package oc...
```

Reassemble the heads and plot results

```
[63]: model_names = list(new_sim.model_names)
      head_dict = {}
      for modelname in model_names:
          mnum = int(modelname.split("_")[-1])
          head = new_sim.get_model(modelname).output.head().get_alldata()[-1]
          head_dict[mnum] = head

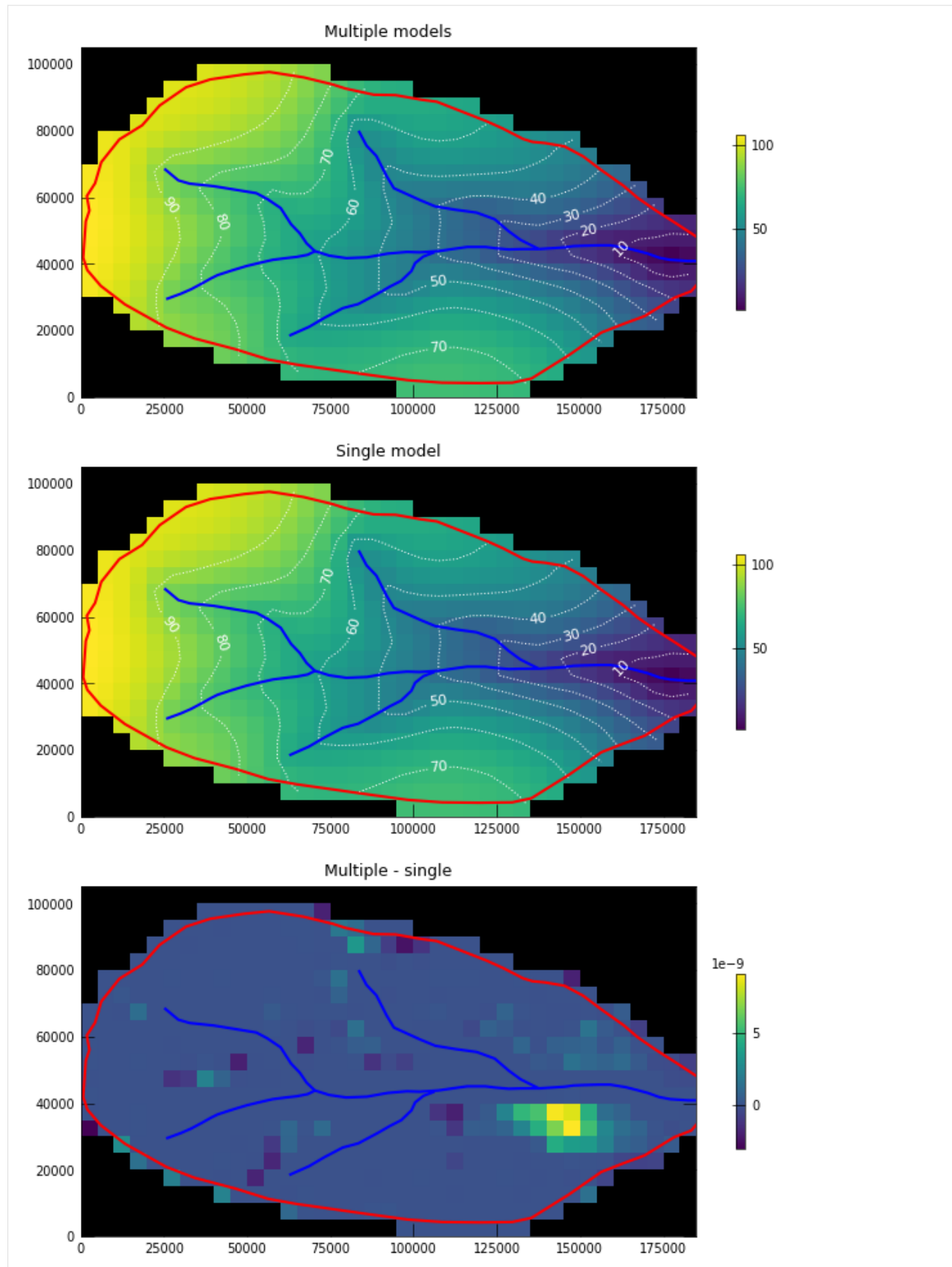
[64]: ra_heads = mfsplit.reconstruct_array(head_dict)
      ra_watertable = flopy.utils.postprocessing.get_water_table(ra_heads)

[65]: with styles.USGSMap():
      fig, axs = plt.subplots(nrows=3, figsize=(8, 12))
      diff = ra_heads - heads
      hv = [ra_heads, heads, diff]
      titles = ["Multiple models", "Single model", "Multiple - single"]
      for idx, ax in enumerate(axs):
          ax.set_aspect("equal")
          ax.set_title(titles[idx])

          if idx < 2:
              levels = contours
              vmin = hmin
              vmax = hmax
          else:
              levels = None
              vmin = None
              vmax = None

          pmv = flopy.plot.PlotMapView(modelgrid=gwf.modelgrid, ax=ax, layer=0)
          h = pmv.plot_array(hv[idx], vmin=vmin, vmax=vmax)
          if levels is not None:
              c = pmv.contour_array(
                  hv[idx],
                  levels=levels,
                  colors="white",
                  linewidths=0.75,
                  linestyle=":",
              )
              plt.clabel(c, fontsize=8)
          pmv.plot_inactive()
          plt.colorbar(h, ax=ax, shrink=0.5)

          ax.plot(bp[:, 0], bp[:, 1], "r-")
          for seg in segs:
              sa = np.array(seg)
              ax.plot(sa[:, 0], sa[:, 1], "b-")
```



6.5.3 Creating a Simple MODFLOW 6 Model with Flopy

The purpose of this notebook is to demonstrate the Flopy capabilities for building a simple MODFLOW 6 model from scratch, running the model, and viewing the results. This notebook will demonstrate the capabilities using a simple lake example. A separate notebook is also available in which the same lake example is created for MODFLOW-2005 (flopy3_lake_example.ipynb).

Setup the Notebook Environment

```
[1]: import os

[2]: import sys
    from tempfile import TemporaryDirectory

    import matplotlib as mpl
    import matplotlib.pyplot as plt
    import numpy as np

    # run installed version of flopy or add local path
    try:
        import flopy
    except:
        fpth = os.path.abspath(os.path.join("..", ".."))
        sys.path.append(fpth)
        import flopy

    print(sys.version)
    print("numpy version: {}".format(np.__version__))
    print("matplotlib version: {}".format(mpl.__version__))
    print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3

[3]: # For this example, we will set up a temporary workspace.
    # Model input files and output files will reside here.
    temp_dir = TemporaryDirectory()
    workspace = os.path.join(temp_dir.name, "mf6lake")
```

Create the Flopy Model Objects

We are creating a square model with a specified head equal to h_1 along all boundaries. The head at the cell in the center in the top layer is fixed to h_2 . First, set the name of the model and the parameters of the model: the number of layers N_{lay} , the number of rows and columns N , lengths of the sides of the model L , aquifer thickness H , hydraulic conductivity k

```
[4]: name = "mf6lake"
    h1 = 100
```

(continues on next page)

(continued from previous page)

```
h2 = 90
Nlay = 10
N = 101
L = 400.0
H = 50.0
k = 1.0
```

One big difference between MODFLOW 6 and previous MODFLOW versions is that MODFLOW 6 is based on the concept of a simulation. A simulation consists of the following:

- Temporal discretization (TDIS)
- One or more models (GWF is the only model supported at present)
- Zero or more exchanges (instructions for how models are coupled)
- Solutions

For this simple lake example, the simulation consists of the temporal discretization (TDIS) package (TDIS), a ground-water flow (GWF) model, and an iterative model solution (IMS), which controls how the GWF model is solved.

```
[5]: # Create the Flopy simulation object
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws=workspace
)

# Create the Flopy temporal discretization object
tdis = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim, pname="tdis", time_units="DAYS", nper=1, perioddata=[(1.0, 1, 1.0)]
)

# Create the Flopy groundwater flow (gwf) model object
model_name_file = "{}.nam".format(name)
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, model_name_file=model_name_file)

# Create the Flopy iterative model solver (ims) Package object
ims = flopy.mf6.modflow.mfims.ModflowIms(sim, pname="ims", complexity="SIMPLE")
```

Now that the overall simulation is set up, we can focus on building the groundwater flow model. The groundwater flow model will be built by adding packages to it that describe the model characteristics.

Define the discretization of the model. All layers are given equal thickness. The bot array is build from H and the Nlay values to indicate top and bottom of each layer, and delrow and delcol are computed from model size L and number of cells N. Once these are all computed, the Discretization file is built.

```
[6]: # Create the discretization package
bot = np.linspace(-H / Nlay, -H, Nlay)
delrow = delcol = L / (N - 1)
dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf,
    pname="dis",
    nlay=Nlay,
    nrow=N,
    ncol=N,
    delr=delrow,
```

(continues on next page)

(continued from previous page)

```

    delc=delcol,
    top=0.0,
    botm=bot,
)

```

```

[7]: # Create the initial conditions package
start = h1 * np.ones((Nlay, N, N))
ic = flopy.mf6.modflow.mfgwfic.ModflowGwfic(gwf, pname="ic", strt=start)

```

```

[8]: # Create the node property flow package
npf = flopy.mf6.modflow.mfgwnpf.ModflowGwnpf(
    gwf, pname="npf", icelltype=1, k=k, save_flows=True
)

```

```

[9]: # Create the constant head package.
# List information is created a bit differently for
# MODFLOW 6 than for other MODFLOW versions. The
# cellid (layer, row, column, for a regular grid)
# must be entered as a tuple as the first entry.
# Remember that these must be zero-based indices!
chd_rec = []
chd_rec.append(((0, int(N / 4), int(N / 4)), h2))
for layer in range(0, Nlay):
    for row_col in range(0, N):
        chd_rec.append(((layer, row_col, 0), h1))
        chd_rec.append(((layer, row_col, N - 1), h1))
        if row_col != 0 and row_col != N - 1:
            chd_rec.append(((layer, 0, row_col), h1))
            chd_rec.append(((layer, N - 1, row_col), h1))
chd = flopy.mf6.modflow.mfgwfchd.ModflowGwfchd(
    gwf,
    pname="chd",
    maxbound=len(chd_rec),
    stress_period_data=chd_rec,
    save_flows=True,
)

```

```

[10]: # The chd package stored the constant heads in a structured
# array, also called a recarray. We can get a pointer to the
# recarray for the first stress period (iper = 0) as follows.
iper = 0
ra = chd.stress_period_data.get_data(key=iper)
ra

```

```

[10]: rec.array([(0, 25, 25), 90.), ((0, 0, 0), 100.), ((0, 0, 100), 100.),
..., ((9, 100, 99), 100.), ((9, 100, 0), 100.),
((9, 100, 100), 100.)],
dtype=[('cellid', '0'), ('head', '<f8')])

```

```

[11]: # We can make a quick plot to show where our constant
# heads are located by creating an integer array

```

(continues on next page)

(continued from previous page)

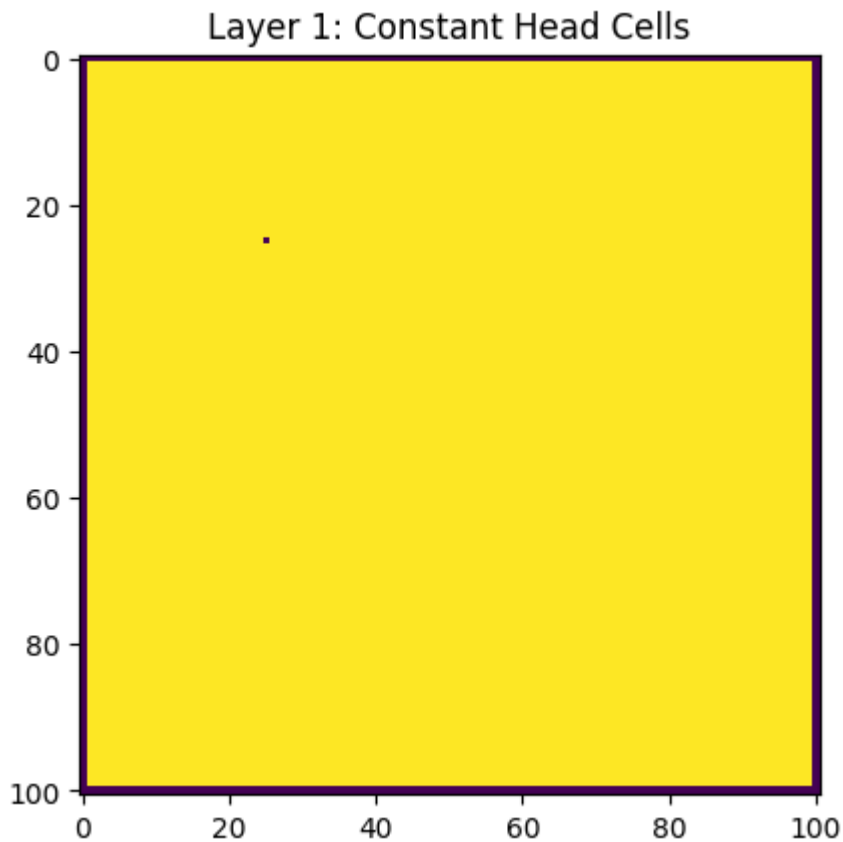
```

# that starts with ones everywhere, but is assigned
# a -1 where chds are located
ibd = np.ones((Nlay, N, N), dtype=int)
for k, i, j in ra["cellid"]:
    ibd[k, i, j] = -1

ilay = 0
plt.imshow(ibd[ilay, :, :], interpolation="none")
plt.title("Layer {}: Constant Head Cells".format(ilay + 1))

```

```
[11]: Text(0.5, 1.0, 'Layer 1: Constant Head Cells')
```



```

[12]: # Create the output control package
headfile = "{}.hds".format(name)
head_filerecord = [headfile]
budgetfile = "{}.cbb".format(name)
budget_filerecord = [budgetfile]
saverecord = [("HEAD", "ALL"), ("BUDGET", "ALL")]
printrecord = [("HEAD", "LAST")]
oc = flopy.mf6.modflow.mfgwfoc.ModflowGwfoc(
    gwf,
    pname="oc",
    saverecord=saverecord,
    head_filerecord=head_filerecord,

```

(continues on next page)

(continued from previous page)

```

    budget_filerecord=budget_filerecord,
    printrecord=printrecord,
)

```

```

[13]: # Note that help can always be found for a package
      # using either forms of the following syntax
      help(oc)
      # help(flopy.mf6.modflow.mfgwfoc.ModflowGwfoc)

```

Help on ModflowGwfoc in module flopy.mf6.modflow.mfgwfoc object:

```

class ModflowGwfoc(flopy.mf6.mfpackage.MFPackage)
|   ModflowGwfoc(model, loading_package=False, budget_filerecord=None, budgetcsv_
|   ↪ filerecord=None, head_filerecord=None, headprintrecord=None, saverecord=None, ↪
|   ↪ printrecord=None, filename=None, pname=None, **kwargs)
|
|   ModflowGwfoc defines a oc package within a gw6 model.
|
|   Parameters
|   -----
|   model : MFModel
|       Model that this package is a part of. Package is automatically
|       added to model when it is initialized.
|   loading_package : bool
|       Do not set this parameter. It is intended for debugging and internal
|       processing purposes only.
|   budget_filerecord : [budgetfile]
|       * budgetfile (string) name of the output file to write budget
|       information.
|   budgetcsv_filerecord : [budgetcsvfile]
|       * budgetcsvfile (string) name of the comma-separated value (CSV) output
|       file to write budget summary information. A budget summary record
|       will be written to this file for each time step of the simulation.
|   head_filerecord : [headfile]
|       * headfile (string) name of the output file to write head information.
|   headprintrecord : [columns, width, digits, format]
|       * columns (integer) number of columns for writing data.
|       * width (integer) width for writing each number.
|       * digits (integer) number of digits to use for writing a number.
|       * format (string) write format can be EXPONENTIAL, FIXED, GENERAL, or
|       SCIENTIFIC.
|   saverecord : [rtype, ocsetting]
|       * rtype (string) type of information to save or print. Can be BUDGET or
|       HEAD.
|       * ocsetting (keystring) specifies the steps for which the data will be
|       saved.
|           all : [keyword]
|               * all (keyword) keyword to indicate save for all time steps in
|               period.
|           first : [keyword]
|               * first (keyword) keyword to indicate save for first step in
|               period. This keyword may be used in conjunction with other

```

(continues on next page)

(continued from previous page)

```

|         keywords to print or save results for multiple time steps.
|     last : [keyword]
|         * last (keyword) keyword to indicate save for last step in
|         period. This keyword may be used in conjunction with other
|         keywords to print or save results for multiple time steps.
|     frequency : [integer]
|         * frequency (integer) save at the specified time step
|         frequency. This keyword may be used in conjunction with other
|         keywords to print or save results for multiple time steps.
|     steps : [integer]
|         * steps (integer) save for each step specified in STEPS. This
|         keyword may be used in conjunction with other keywords to
|         print or save results for multiple time steps.
| printrecord : [rtype, ocsetting]
|     * rtype (string) type of information to save or print. Can be BUDGET or
|     HEAD.
|     * ocsetting (keystring) specifies the steps for which the data will be
|     saved.
|     all : [keyword]
|         * all (keyword) keyword to indicate save for all time steps in
|         period.
|     first : [keyword]
|         * first (keyword) keyword to indicate save for first step in
|         period. This keyword may be used in conjunction with other
|         keywords to print or save results for multiple time steps.
|     last : [keyword]
|         * last (keyword) keyword to indicate save for last step in
|         period. This keyword may be used in conjunction with other
|         keywords to print or save results for multiple time steps.
|     frequency : [integer]
|         * frequency (integer) save at the specified time step
|         frequency. This keyword may be used in conjunction with other
|         keywords to print or save results for multiple time steps.
|     steps : [integer]
|         * steps (integer) save for each step specified in STEPS. This
|         keyword may be used in conjunction with other keywords to
|         print or save results for multiple time steps.
| filename : String
|     File name for this package.
| pname : String
|     Package name for this package.
| parent_file : MFPackage
|     Parent package file that references this package. Only needed for
|     utility packages (mfutl*). For example, mfutllaktab package must have
|     a mfgwflak package parent_file.
|
| Method resolution order:
|     ModflowGwfoc
|     flopy.mf6.mfpackage.MFPackage
|     flopy.mf6.mfbase.PackageContainer
|     flopy.pakbase.PackageInterface
|     builtins.object

```

(continues on next page)

(continued from previous page)

```

|
| Methods defined here:
|
| __init__(self, model, loading_package=False, budget_filerecord=None, budgetcsv_
↪ filerecord=None, head_filerecord=None, headprintrecord=None, saverecord=None,
↪ printrecord=None, filename=None, pname=None, **kwargs)
|     Initialize self.  See help(type(self)) for accurate signature.
|
| -----
| Data and other attributes defined here:
|
| budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator o...
|
| budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerato...
|
| dfn = [['header'], ['block options', 'name budget_filerecord', 'type r...
|
| dfn_file_name = 'gwf-oc.dfn'
|
| head_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator obj...
|
| headprintrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator obj...
|
| package_abbr = 'gwfoc'
|
| printrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
|
| saverecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
|
| -----
| Methods inherited from flopy.mf6.mfpackage.MFPackage:
|
| __repr__(self)
|     Return repr(self).
|
| __setattr__(self, name, value)
|     Implement setattr(self, name, value).
|
| __str__(self)
|     Return str(self).
|
| build_child_package(self, pkg_type, data, parameter_name, filerecord)
|     Builds a child package.  This method is only intended for FloPy
|     internal use.
|
| build_child_packages_container(self, pkg_type, filerecord)
|     Builds a container object for any child packages.  This method is
|     only intended for FloPy internal use.
|
| build_mfdata(self, var_name, data=None)
|     Returns the appropriate data type object (mfdataalist, mfdataarray,
|     or mfdatascalar) given that object the appropriate structure (looked

```

(continues on next page)

(continued from previous page)

```

|         up based on var_name) and any data supplied. This method is for
|         internal FloPy library use only.
|
|         Parameters
|         -----
|         var_name : str
|             Variable name
|
|         data : many supported types
|             Data contained in this object
|
|         Returns
|         -----
|         data object : MFDData subclass
|
|     check(self, f=None, verbose=True, level=1, checktype=None)
|         Data check, returns True on success.
|
|     create_package_dimensions(self)
|         Creates a package dimensions object. For internal FloPy library
|         use.
|
|         Returns
|         -----
|         package dimensions : PackageDimensions
|
|     export(self, f, **kwargs)
|         Method to export a package to netcdf or shapefile based on the
|         extension of the file name (.shp for shapefile, .nc for netcdf)
|
|         Parameters
|         -----
|         f : str
|             Filename
|         kwargs : keyword arguments
|             modelgrid : flopy.discretization.Grid instance
|                 User supplied modelgrid which can be used for exporting
|                 in lieu of the modelgrid associated with the model object
|
|         Returns
|         -----
|         None or Netcdf object
|
|     get_file_path(self)
|         Returns the package file's path.
|
|         Returns
|         -----
|         file path : str
|
|     inspect_cells(self, cell_list, stress_period=None)
|         Inspect model cells. Returns package data associated with cells.

```

(continues on next page)

(continued from previous page)

```

|
| Parameters
| -----
| cell_list : list of tuples
|     List of model cells. Each model cell is a tuple of integers.
|     ex: [(1,1,1), (2,4,3)]
| stress_period : int
|     For transient data, only return data from this stress period. If
|     not specified or None, all stress period data will be returned.
|
| Returns
| -----
| output : array
|     Array containing inspection results
|
| is_valid(self)
|     Returns whether or not this package is valid.
|
| Returns
| -----
| is valid : bool
|
| load(self, strict=True)
|     Loads the package from file.
|
| Parameters
| -----
| strict : bool
|     Enforce strict checking of data.
|
| Returns
| -----
| success : bool
|
| plot(self, **kwargs)
|     Plot 2-D, 3-D, transient 2-D, and stress period list (MfList)
|     package input data
|
| Parameters
| -----
| **kwargs : dict
|     filename_base : str
|         Base file name that will be used to automatically generate
|         file names for output image files. Plots will be exported as
|         image files if file_name_base is not None. (default is None)
|     file_extension : str
|         Valid matplotlib.pyplot file extension for savefig(). Only
|         used if filename_base is not None. (default is 'png')
|     mflay : int
|         MODFLOW zero-based layer number to return. If None, then all
|         all layers will be included. (default is None)
|     kper : int

```

(continues on next page)

(continued from previous page)

```

|         MODFLOW zero-based stress period number to return. (default is
|         zero)
|     key : str
|         MfList dictionary key. (default is None)
|
|     Returns
|     -----
|     axes : list
|         Empty list is returned if filename_base is not None. Otherwise
|         a list of matplotlib.pyplot.axis are returned.
|
| remove(self)
|     Removes this package from the simulation/model it is currently a
|     part of.
|
| set_all_data_external(self, check_data=True, external_data_folder=None)
|     Sets the package's list and array data to be stored externally.
|
|     Parameters
|     -----
|         check_data : bool
|             Determine if data error checking is enabled
|         external_data_folder
|             Folder where external data will be stored
|
| set_all_data_internal(self, check_data=True)
|     Sets the package's list and array data to be stored internally.
|
|     Parameters
|     -----
|         check_data : bool
|             Determine if data error checking is enabled
|
| set_model_relative_path(self, model_ws)
|     Sets the model path relative to the simulation's path.
|
|     Parameters
|     -----
|     model_ws : str
|         Model path relative to the simulation's path.
|
| write(self, ext_file_action=<ExtFileAction.copy_relative_paths: 3>)
|     Writes the package to a file.
|
|     Parameters
|     -----
|     ext_file_action : ExtFileAction
|         How to handle pathing of external data files.
|
| -----
| Class methods inherited from flopy.mf6.mfpackage.MFPackage:

```

(continues on next page)

(continued from previous page)

```

|  __init_subclass__() from builtins.type
|      Register package type
|
|  -----
|  Readonly properties inherited from flopy.mf6.mfpackage.MFPackage:
|
|  data_list
|      List of data in this package.
|
|  output
|      Method to get output associated with a specific package
|
|      Returns
|      -----
|          MF6Output object
|
|  package_type
|      String describing type of package
|
|  plottable
|      If package is plottable
|
|  quoted_filename
|      Package's file name with quotes if there is a space.
|
|  -----
|  Data descriptors inherited from flopy.mf6.mfpackage.MFPackage:
|
|  filename
|      Package's file name.
|
|  name
|      Name of package
|
|  parent
|      Parent package
|
|  -----
|  Methods inherited from flopy.mf6.mfbase.PackageContainer:
|
|  get_package(self, name=None)
|      Finds a package by package name, package key, package type, or partial
|      package name. returns either a single package, a list of packages,
|      or None.
|
|      Parameters
|      -----
|      name : str
|          Name of the package, 'RIV', 'LPF', etc.
|
|      Returns
|      -----

```

(continues on next page)

(continued from previous page)

```

|     pp : Package object
|
| register_package(self, package)
|     Base method for registering a package.  Should be overridden.
|
| -----
| Static methods inherited from flopy.mf6.mfbase.PackageContainer:
|
| get_module_val(module, item, attrb)
|     Static method that returns a python class module value.  For
|     internal FloPy use only, not intended for end users.
|
| model_factory(model_type)
|     Static method that returns the appropriate model type object based
|     on the model_type string. For internal FloPy use only, not intended
|     for end users.
|
|     Parameters
|     -----
|         model_type : str
|             Type of model that package is a part of
|
|     Returns
|     -----
|         model : MFModel subclass
|
| package_factory(package_type: str, model_type: str)
|     Static method that returns the appropriate package type object based
|     on the package_type and model_type strings.  For internal FloPy use
|     only, not intended for end users.
|
|     Parameters
|     -----
|         package_type : str
|             Type of package to create
|         model_type : str
|             Type of model that package is a part of
|
|     Returns
|     -----
|         package : MFPackage subclass
|
| package_list()
|     Static method that returns the list of available packages.
|     For internal FloPy use only, not intended for end users.
|
|     Returns a list of MFPackage subclasses
|
| -----
| Readonly properties inherited from flopy.mf6.mfbase.PackageContainer:
|
| package_dict

```

(continues on next page)

(continued from previous page)

```

|     Returns a copy of the package name dictionary.
|
| package_key_dict
|
| package_names
|     Returns a list of package names.
|
| -----
| Data descriptors inherited from flopy.mf6.mfbase.PackageContainer:
|
| __dict__
|     dictionary for instance variables (if defined)
|
| __weakref__
|     list of weak references to the object (if defined)
|
| -----
| Data and other attributes inherited from flopy.mf6.mfbase.PackageContainer:
|
| models_by_type = {'gwf': <class 'flopy.mf6.modflow.mfgwf.ModflowGwf'>,...
|
| modflow_models = [<class 'flopy.mf6.modflow.mfgwf.ModflowGwf'>, <class...
|
| modflow_packages = [<class 'flopy.mf6.modflow.mfnam.ModflowNam'>, <cla...
|
| packages_by_abbr = {'ems': <class 'flopy.mf6.modflow.mfems.ModflowEms'...
|
| -----
| Readonly properties inherited from flopy.pakbase.PackageInterface:
|
| has_stress_period_data

```

Create the MODFLOW 6 Input Files and Run the Model

Once all the flopy objects are created, it is very easy to create all of the input files and run the model.

```

[14]: # Write the datasets
sim.write_simulation()

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims...
  writing model mf6lake...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
    writing package chd...
    writing package oc...

```

```
[15]: # Print a list of the files that were created
# in workspace
print(os.listdir(workspace))

['mf6lake.ims', 'mf6lake.oc', 'mf6lake.npf', 'mf6lake.ic', 'mf6lake.nam', 'mf6lake.dis',
↪ 'mf6lake.tdis', 'mf6lake.chd', 'mfsim.nam']
```

Run the Simulation

We can also run the simulation from the notebook, but only if the MODFLOW 6 executable is available. The executable can be made available by putting the executable in a folder that is listed in the system path variable. Another option is to just put a copy of the executable in the simulation folder, though this should generally be avoided. A final option is to provide a full path to the executable when the simulation is constructed. This would be done by specifying `exe_name` with the full path.

```
[16]: # Run the simulation
success, buff = sim.run_simulation(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")
```

```
MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023
```

```
MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000
```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

```
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:17
```

```
Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam
```

```
Solving: Stress period:      1      Time step:      1
```

```
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:18
```

(continues on next page)

(continued from previous page)

Elapsed run time: 0.636 Seconds

Normal termination of simulation.

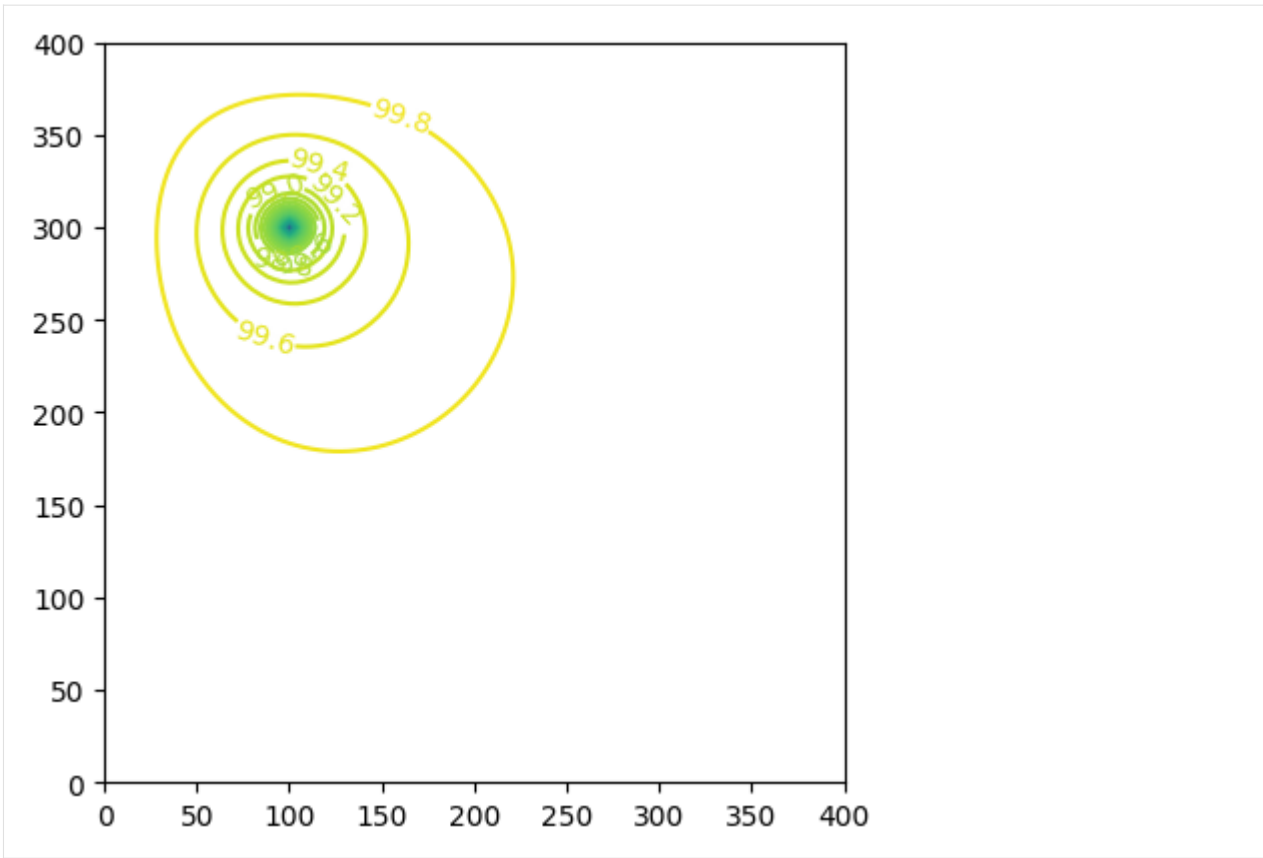
Post-Process Head Results

Post-processing MODFLOW 6 results is still a work in progress. There aren't any Flopy plotting functions built in yet, like they are for other MODFLOW versions. So we need to plot the results using general Flopy capabilities. We can also use some of the Flopy ModelMap capabilities for MODFLOW 6, but in order to do so, we need to manually create a SpatialReference object, that is needed for the plotting. Examples of both approaches are shown below.

First, a link to the heads file is created with HeadFile. The link can then be accessed with the `get_data` function, by specifying, in this case, the step number and period number for which we want to retrieve data. A three-dimensional array is returned of size `nlay`, `nrow`, `ncol`. Matplotlib contouring functions are used to make contours of the layers or a cross-section.

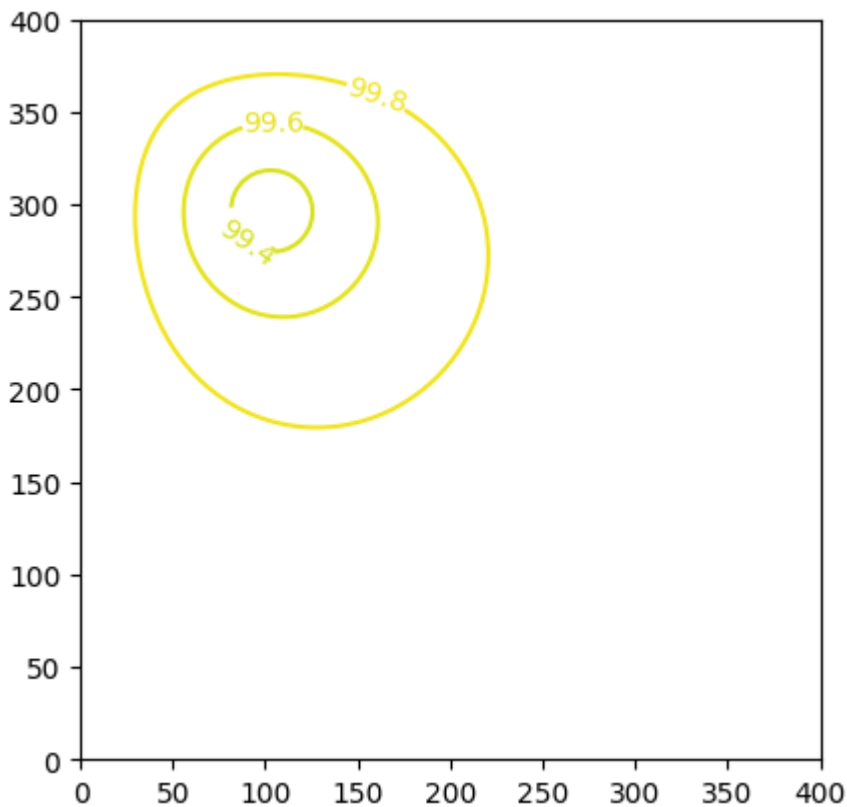
```
[17]: # Read the binary head file and plot the results
# We can use the existing Flopy HeadFile class because
# the format of the headfile for MODFLOW 6 is the same
# as for previous MODFLOW versions
fname = os.path.join(workspace, headfile)
hds = flopy.utils.binaryfile.HeadFile(fname)
h = hds.get_data(kstpker=(0, 0))
x = y = np.linspace(0, L, N)
y = y[::-1]
c = plt.contour(x, y, h[0], np.arange(90, 100.1, 0.2))
plt.clabel(c, fmt="%2.1f")
plt.axis("scaled")

[17]: (0.0, 400.0, 0.0, 400.0)
```



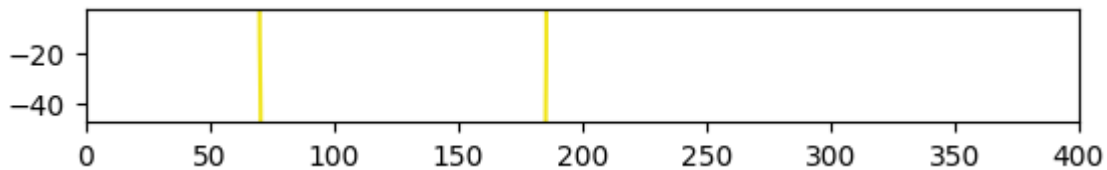
```
[18]: x = y = np.linspace(0, L, N)
      y = y[::-1]
      c = plt.contour(x, y, h[-1], np.arange(90, 100.1, 0.2))
      plt.clabel(c, fmt="%1.1f")
      plt.axis("scaled")
```

```
[18]: (0.0, 400.0, 0.0, 400.0)
```



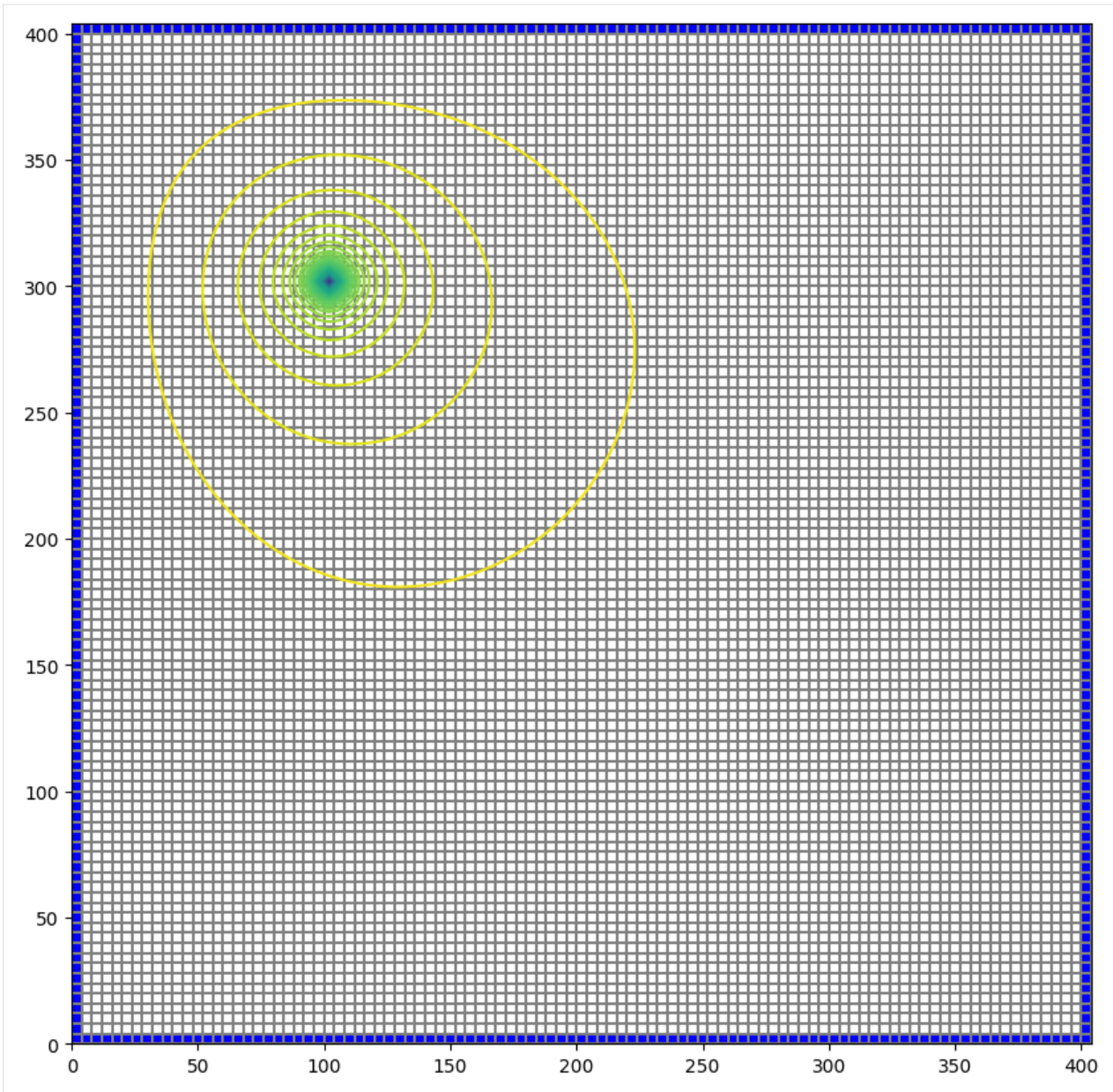
```
[19]: z = np.linspace(-H / Nlay / 2, -H + H / Nlay / 2, Nlay)
c = plt.contour(x, z, h[:, 50, :], np.arange(90, 100.1, 0.2))
plt.axis("scaled")
```

```
[19]: (0.0, 400.0, -47.5, -2.5)
```



```
[20]: # We can also use the Flopy PlotMapView capabilities for MODFLOW 6
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
modelmap = flopy.plot.PlotMapView(model=gwf, ax=ax)

# Then we can use the plot_grid() method to draw the grid
# The return value for this function is a matplotlib LineCollection object,
# which could be manipulated (or used) later if necessary.
quadmesh = modelmap.plot_ibound(ibound=ibd)
linecollection = modelmap.plot_grid()
contours = modelmap.contour_array(h[0], levels=np.arange(90, 100.1, 0.2))
```



```
[21]: # We can also use the Flopy PlotMapView capabilities for MODFLOW 6
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 1, 1, aspect="equal")

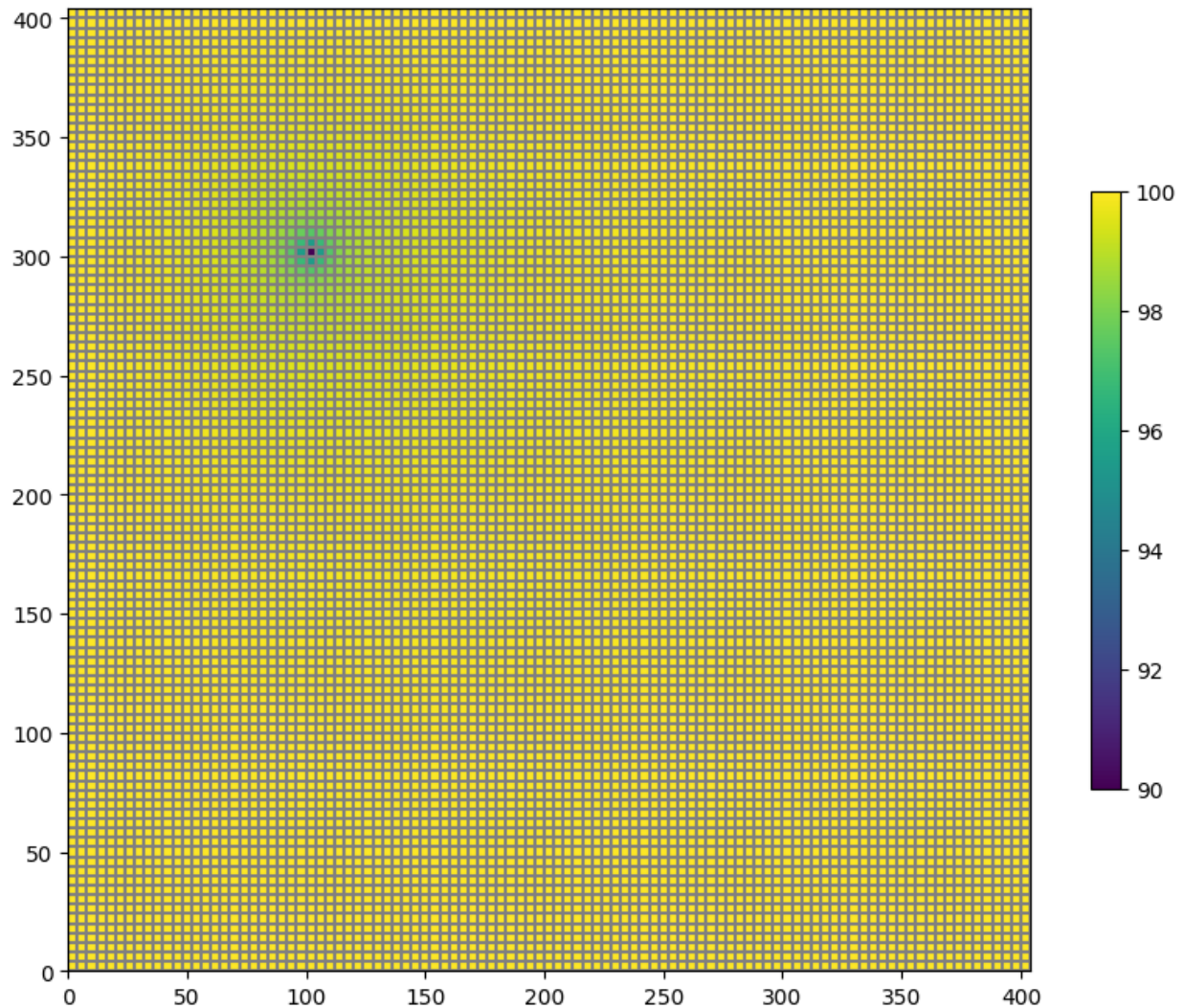
# Next we create an instance of the ModelMap class
modelmap = flopy.plot.PlotMapView(model=gwf, ax=ax)

# Then we can use the plot_grid() method to draw the grid
# The return value for this function is a matplotlib LineCollection object,
# which could be manipulated (or used) later if necessary.
quadmesh = modelmap.plot_ibound(ibound=ibd)
linecollection = modelmap.plot_grid()
pa = modelmap.plot_array(h[0])
```

(continues on next page)

(continued from previous page)

```
cb = plt.colorbar(pa, shrink=0.5)
```



Post-Process Flows

MODFLOW 6 writes a binary grid file, which contains information about the model grid. MODFLOW 6 also writes a binary budget file, which contains flow information. Both of these files can be read using Flopy capabilities. The `MfGrdFile` class in Flopy can be used to read the binary grid file. The `CellBudgetFile` class in Flopy can be used to read the binary budget file written by MODFLOW 6.

```
[22]: # read the binary grid file
fname = os.path.join(workspace, "{}.dis.grb".format(name))
bgf = flopy.mf6.utils.MfGrdFile(fname)

# data read from the binary grid file is stored in a dictionary
bgf._datadict
```

```
[22]: {'NCELLS': 102010,  
      'NLAY': 10,  
      'NROW': 101,  
      'NCOL': 101,  
      'NJA': 689628,  
      'XORIGIN': 0.0,  
      'YORIGIN': 0.0,  
      'ANGROT': 0.0,  
      'DELR': array([4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.,  
                    4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.,  
                    4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.,  
                    4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.,  
                    4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.] ),  
      'DELCL': array([4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.,  
                     4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.,  
                     4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.,  
                     4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.,  
                     4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4., 4.] ),  
      'TOP': array([0., 0., 0., ..., 0., 0., 0.] ),  
      'BOTM': array([-5., -5., -5., ..., -50., -50., -50.] ),  
      'IA': array([    1,     5,    10, ..., 689620, 689625, 689629], dtype=int32),  
      'JA': array([    1,     2,   102, ..., 91809, 101909, 102009], dtype=int32),  
      'IDOMAIN': array([1, 1, 1, ..., 1, 1, 1], dtype=int32),  
      'ICELLTYPE': array([1, 1, 1, ..., 1, 1, 1], dtype=int32)}
```

```
[23]: # read the cell budget file
fname = os.path.join(workspace, "{}.cbb".format(name))
cbb = flopy.utils.CellBudgetFile(fname, precision="double")
cbb.list_records()

flowja = cbb.get_data(text="FLOW-JA-FACE")[0][0, 0, :]
chdflow = cbb.get_data(text="CHD")[0]

(1, 1, b'      FLOW-JA-FACE', 689628, 1, -1, 1, 1., 1., 1., b'', b'', b'', b'')
(1, 1, b'              CHD', 101, 101, -10, 6, 1., 1., 1., b'MF6LAKE          ', b'MF6LAKE_
↪       ', b'MF6LAKE           ', b'CHD            ')

```

```
[24]: # By having the ia and ja arrays and the flow-ja-face we can look at
# the flows for any cell and process them in the follow manner.
k = 5
i = 50
j = 50
celln = k * N * N + i * N + j
ia, ja = bgf.ia, bgf.ja
print("Printing flows for cell {}".format(celln))
for ipos in range(ia[celln] + 1, ia[celln + 1]):
    cellm = ja[ipos]
    print("Cell {} flow with cell {} is {}".format(celln, cellm, flowja[ipos]))

Printing flows for cell 56105
Cell 56105 flow with cell 45904 is -0.0002858449337963975
```

(continues on next page)

(continued from previous page)

```

Cell 56105 flow with cell 56004 is -0.025019694309449392
Cell 56105 flow with cell 56104 is -0.025019694309449392
Cell 56105 flow with cell 56106 is 0.025058524820593675
Cell 56105 flow with cell 56206 is 0.02505852482066473
Cell 56105 flow with cell 66306 is 0.00033448625827077195

```

```

[25]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
except:
    pass

```

6.5.4 Flopy MODFLOW 6 (MF6) Support

The Flopy library contains classes for creating, saving, running, loading, and modifying MF6 simulations. The MF6 portion of the flopy library is located in the `flopy.mf6` module.

Conceptual model

While there are a number of classes in `flopy.mf6`, to get started you only need to use the main classes summarized below:

`flopy.mf6.MFSimulation`

MODFLOW Simulation Class. Entry point into any MODFLOW simulation.

`flopy.mf6.ModflowGwf`

MODFLOW Groundwater Flow Model Class. Represents a single model in a simulation.

`flopy.mf6.Modflow[pc]`

MODFLOW package classes where [pc] is the abbreviation of the package name. Each package is a separate class.

For packages that are part of a groundwater flow model, the abbreviation begins with Gwf. For example, `flopy.mf6.ModflowGwfdis` is the Discretization package.

Creating a simulation

A MF6 simulation is created by first creating a simulation object “MFSimulation”. When you create the simulation object you can define the simulation’s name, version, executable name, workspace path, and the name of the tdis file. All of these are optional parameters, and if not defined each one will default to the following:

- `sim_name`: ‘modflowtest’
- `version`: ‘mf6’
- `exe_name`: ‘mf6’

- `sim_ws: os.curdir('.')`
- `sim_tdis_file: 'modflow6.tdis'`

The `sim_ws` parameter accepts `str` or `os.PathLike` arguments.

```
[1]: import os
import sys
from pathlib import Path
from shutil import copyfile
from tempfile import TemporaryDirectory

proj_root = Path.cwd().parent.parent

# run installed version of flopy or add local path
try:
    import flopy
except:
    sys.path.append(proj_root)
    import flopy

# temporary directory
temp_dir = TemporaryDirectory()
sim_name = "example_sim"
sim_path = Path(temp_dir.name) / "example_project"
sim = flopy.mf6.MFSimulation(
    sim_name=sim_name, version="mf6", exe_name="mf6", sim_ws=sim_path
)
```

The next step is to create a `tdis` package object “`ModflowTdis`”. The first parameter of the `ModflowTdis` class is a simulation object, which ties a `ModflowTdis` object to a specific simulation. The other parameters and their definitions can be found in the docstrings.

```
[2]: tdis = flopy.mf6.ModflowTdis(
    sim,
    pname="tdis",
    time_units="DAYS",
    nper=2,
    perioddata=[(1.0, 1, 1.0), (10.0, 5, 1.0)],
)
```

Next one or more models are created using the `ModflowGwf` class. The first parameter of the `ModflowGwf` class is the simulation object that the model will be a part of.

```
[3]: model_name = "example_model"
model = flopy.mf6.ModflowGwf(
    sim, modelname=model_name, model_nam_file="{}.nam".format(model_name)
)
```

Next create one or more Iterative Model Solution (IMS) files.

```
[4]: ims_package = flopy.mf6.ModflowIms(
    sim,
    pname="ims",
    print_option="ALL",
```

(continues on next page)

(continued from previous page)

```

complexity="SIMPLE",
outer_dvclose=0.00001,
outer_maximum=50,
under_relaxation="NONE",
inner_maximum=30,
inner_dvclose=0.00001,
linear_acceleration="CG",
preconditioner_levels=7,
preconditioner_drop_tolerance=0.01,
number_orthogonalizations=2,
)

```

Each ModflowGwf object needs to be associated with an ModflowIms object. This is done by calling the MFSimulation object's "register_ims_package" method. The first parameter in this method is the ModflowIms object and the second parameter is a list of model names (strings) for the models to be associated with the ModflowIms object.

```
[5]: sim.register_ims_package(ims_package, [model_name])
```

Next add packages to each model. The first package added needs to be a spatial discretization package since flopy uses information from the spatial discretization package to help you build other packages. There are three spatial discretization packages to choose from:

- DIS (ModflowGwfDis): Structured discretization
- DISV (ModflowGwfdisv): Discretization with vertices
- DISU (ModflowGwfdisu): Unstructured discretization

```
[6]: dis_package = flopy.mf6.ModflowGwfdis(
    model,
    pname="dis",
    length_units="FEET",
    nlay=2,
    nrow=2,
    ncol=5,
    delr=500.0,
    delc=500.0,
    top=100.0,
    botm=[50.0, 20.0],
    filename="{}.dis".format(model_name),
)

```

Accessing namefiles

Namefiles are automatically built for you by flopy. However, there are some options contained in the namefiles that you may want to set. To get the namefile object access the name_file attribute in either a simulation or model object to get the simulation or model namefile.

```
[7]: # set the nocheck property in the simulation namefile
sim.name_file.nocheck = True
# set the print_input option in the model namefile
model.name_file.print_input = True

```

Specifying options

Option that appear alone are assigned a boolean value, like the `print_input` option above. Options that have additional optional parameters are assigned using a tuple, with the entries containing the names of the optional parameters to turn on. Use a tuple with an empty string to indicate no optional parameters and use a tuple with `None` to turn the option off.

```
[8]: # Turn Newton option on with under relaxation
model.name_file.newtonoptions = "UNDER_RELAXATION"
# Turn Newton option on without under relaxation
model.name_file.newtonoptions = ""
# Turn off Newton option
model.name_file.newtonoptions = None
```

MFArrray templates

Lastly define all other packages needed.

Note that flopy supports a number of ways to specify data for a package. A template, which defines the data array shape for you, can be used to specify the data. Templates are built by calling the empty of the data type you are building. For example, to build a template for `k` in the `npf` package you would call:

```
ModflowGwfnpf.k.empty()
```

The empty method for MFArrray data templates (data templates whose size is based on the structure of the model grid) take up to four parameters:

- `model`: The model object that the data is a part of. A valid model object with a discretization package is required in order to build the proper array dimensions. This parameter is required.
- `layered`: True or false whether the data is layered or not.
- `data_storage_type_list`: List of data storage types, one for each model layer. If the template is not layered, only one data storage type needs to be specified. There are three data storage types supported, `internal_array`, `internal_constant`, and `external_file`.
- `default_value`: The initial value for the array.

```
[9]: # build a data template for k that stores the first layer as an internal array and the
      ↪second
      # layer as a constant with the default value of k for all layers set to 100.0
layer_storage_types = [
    flopy.mf6.data.mfdatastorage.DataStorageType.internal_array,
    flopy.mf6.data.mfdatastorage.DataStorageType.internal_constant,
]
k_template = flopy.mf6.ModflowGwfnpf.k.empty(
    model, True, layer_storage_types, 100.0
)
# change the value of the second layer to 50.0
k_template[0]["data"] = [
    65.0,
    60.0,
    55.0,
    50.0,
    45.0,
```

(continues on next page)

(continued from previous page)

```

    40.0,
    35.0,
    30.0,
    25.0,
    20.0,
]
k_template[0]["factor"] = 1.5
print(k_template)
# create npf package using the k template to define k
npf_package = flopy.mf6.ModflowGwfnpf(
    model, pname="npf", save_flows=True, icelltype=1, k=k_template
)

[{'factor': 1.5, 'iprn': 1, 'data': [65.0, 60.0, 55.0, 50.0, 45.0, 40.0, 35.0, 30.0, 25.0, 20.0]}, 100.0]

```

Specifying MFArray Data

MFArray data can also be specified as a numpy array, a list of values, or a single value. Below strt (starting heads) are defined as a single value, 100.0, which is interpreted as an internal constant storage type of value 100.0. Strt could also be defined as a list defining a value for every model cell:

```

strt=[100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0,
      90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0]

```

Or as a list defining a value or values for each model layer:

```

strt=[100.0, 90.0]

```

or:

```

strt=[[100.0], [90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0]]

```

MFArray data can also be stored in an external file by using a dictionary using the keys 'filename' to specify the file name relative to the model folder and 'data' to specify the data. The optional 'factor', 'iprn', and 'binary' keys may also be used.

```

strt={'filename': 'strt.txt', 'factor':1.0, 'data':[100.0, 100.0, 100.0, 100.0, 100.0,
100.0, 100.0, 100.0, 100.0, 100.0,
90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0, 90.0], 'binary': 'True'}

```

If the 'data' key is omitted from the dictionary flopy will try to read the data from an existing file 'filename'. Any relative paths for loading data from a file should be specified relative to the MF6 simulation folder.

```

[10]: strt = {
    "filename": "strt.txt",
    "factor": 1.0,
    "data": [
        100.0,
        100.0,
        100.0,
        100.0,

```

(continues on next page)

(continued from previous page)

```

        100.0,
        100.0,
        100.0,
        100.0,
        100.0,
        100.0,
        90.0,
        90.0,
        90.0,
        90.0,
        90.0,
        90.0,
        90.0,
        90.0,
        90.0,
        90.0,
    ],
    "binary": "True",
}
ic_package = flopy.mf6.ModflowGwfic(
    model, pname="ic", strt=strt, filename="{}.ic".format(model_name)
)
# move external file data into model folder
icv_data_path = os.path.join(
    "..", "..", "examples", "data", "mf6", "notebooks", "iconvert.txt"
)
copyfile(icv_data_path, os.path.join(sim_path, "iconvert.txt"))
# create storage package
sto_package = flopy.mf6.ModflowGwfsto(
    model,
    pname="sto",
    save_flows=True,
    iconvert={"filename": "iconvert.txt"},
    ss=[0.000001, 0.000002],
    sy=[
        0.15,
        0.14,
        0.13,
        0.12,
        0.11,
        0.11,
        0.12,
        0.13,
        0.14,
        0.15,
        0.15,
        0.14,
        0.13,
        0.12,
        0.11,
        0.11,
        0.12,
    ]

```

(continues on next page)

(continued from previous page)

```

    0.13,
    0.14,
    0.15,
],
)

```

MFList Templates

FloPy supports specifying record and recarray MFList data in a number of ways. Templates can be created that define the shape of the data. The empty method for MFList data templates take up to 7 parameters.

- **model**: The model object that the data is a part of. A valid model object with a discretization package is required in order to build the proper array dimensions. This parameter is required.
- **maxbound**: The number of rows in the recarray. If not specified one row is returned.
- **aux_vars**: List of auxiliary variable names. If not specified auxiliary variables are not used.
- **boundnames**: True/False if boundnames is to be used.
- **nseg**: Number of segments (only relevant for a few data types)
- **timeseries**: True/False indicates that time series data will be used.
- **stress_periods**: List of integer stress periods to be used (transient MFList data only). If not specified for transient data, template will only be defined for stress period 1.

MFList transient data templates are numpy recarrays stored in a dictionary with the dictionary key an integer zero based stress period value (stress period - 1).

In the code below the well package is set up using a transient MFList template to help build the well's stress_periods.

```

[11]: maxbound = 2
# build a stress_period_data template with 2 wells over stress periods 1 and 2 with
↳ boundnames
# and three aux variables
wel_periodrec = flopy.mf6.ModflowGwfwel.stress_period_data.empty(
    model,
    maxbound=maxbound,
    boundnames=True,
    aux_vars=["var1", "var2", "var3"],
    stress_periods=[0, 1],
)
# define the two wells for stress period one
wel_periodrec[0][0] = ((0, 1, 2), -50.0, -1, -2, -3, "First Well")
wel_periodrec[0][1] = ((1, 1, 4), -25.0, 2, 3, 4, "Second Well")
# define the two wells for stress period two
wel_periodrec[1][0] = ((0, 1, 2), -200.0, -1, -2, -3, "First Well")
wel_periodrec[1][1] = ((1, 1, 4), -4000.0, 2, 3, 4, "Second Well")
# build the well package
wel_package = flopy.mf6.ModflowGwfwel(
    model,
    pname="wel",
    print_input=True,
    print_flows=True,

```

(continues on next page)

(continued from previous page)

```
    auxiliary=["var1", "var2", "var3"],
    maxbound=maxbound,
    stress_period_data=wel_periodrec,
    boundnames=True,
    save_flows=True,
)
```

Cell IDs

Cell IDs always appear as tuples in an MFList. For a structured grid cell IDs appear as:

(<layer>, <row>, <column>)

For vertice based grid cells IDs appear as:

(<layer>, <intralayer_cell_id>)

Unstructured grid cell IDs appear as:

(<cell_id>)

Specifying MFList Data

MFList data can also be defined as a list of tuples, with each tuple being a row of the recarray. For transient data the list of tuples can be stored in a dictionary with the dictionary key an integer zero based stress period value. If only a list of tuples is specified for transient data, the data is assumed to apply to stress period 1. Additional stress periods can be added with the `add_transient_key` method. The code below defines `saverecord` and `printrecord` as a list of tuples.

```
[12]: # printrecord data as a list of tuples. since no stress
# period is specified it will default to stress period 1
printrec_tuple_list = [("HEAD", "ALL"), ("BUDGET", "ALL")]
# saverecord data as a dictionary of lists of tuples for
# stress periods 1 and 2.
saverec_dict = {
    0: [("HEAD", "ALL"), ("BUDGET", "ALL")],
    1: [("HEAD", "ALL"), ("BUDGET", "ALL")],
}
# create oc package
oc_package = flopy.mf6.ModflowGwfoc(
    model,
    pname="oc",
    budget_filerecord=[("{} .cbc".format(model_name),)],
    head_filerecord=[("{} .hds".format(model_name),)],
    saverecord=saverec_dict,
    printrecord=printrec_tuple_list,
)
# add stress period two to the print record
oc_package.printrecord.add_transient_key(1)
# set the data for stress period two in the print record
oc_package.printrecord.set_data([("HEAD", "ALL"), ("BUDGET", "ALL")], 1)
```


Specifying MFLIST Data in an External File

MFLIST data can be specified in an external file using a dictionary with the ‘filename’ key. If the ‘data’ key is also included in the dictionary and is not None, flopy will create the file with the data contained in the ‘data’ key. The ‘binary’ key can be used to save data to a binary file (‘binary’: True). The code below creates a chd package which creates and references an external file containing data for stress period 1 and stores the data internally in the chd package file for stress period 2.

```
[13]: stress_period_data = {
        0: {"filename": "chd_sp1.dat", "data": [[(0, 0, 0), 70.0]]},
        1: [[(0, 0, 0), 60.0]],
    }
    chd = flopy.mf6.ModflowGwfchd(
        model, maxbound=1, stress_period_data=stress_period_data
    )
```

Packages that Support both List-based and Array-based Data

The recharge and evapotranspiration packages can be specified using list-based or array-based input. The array packages have an “a” on the end of their name:

- ModflowGwfrch: list based recharge package
- ModflowGwfrcha: array based recharge package
- ModflowGwfevt: list based evapotranspiration package
- ModflowGwfevta: array based evapotranspiration package

```
[14]: rch_reccarray = {
        0: [(0, 0, 0), "rch_1"), ((1, 1, 1), "rch_2")],
        1: [(0, 0, 0), "rch_1"), ((1, 1, 1), "rch_2")],
    }
    rch_package = flopy.mf6.ModflowGwfrch(
        model,
        pname="rch",
        fixed_cell=True,
        print_input=True,
        maxbound=2,
        stress_period_data=rch_reccarray,
    )
```

Utility Files (TS, TAS, OBS, TAB)

Utility files, MF6 formatted files that reference by packages, include time series, time array series, observation, and tab files. The file names for utility files are specified using the package that references them. The utility files can be created in several ways. A simple case is demonstrated below. More detail is given in the flopy3_mf6_obs_ts_tas notebook.

```
[15]: # build a time series array for the recharge package
    ts_data = [
        (0.0, 0.015, 0.0017),
        (1.0, 0.016, 0.0019),
        (2.0, 0.012, 0.0015),
```

(continues on next page)

(continued from previous page)

```

    (3.0, 0.020, 0.0014),
    (4.0, 0.015, 0.0021),
    (5.0, 0.013, 0.0012),
    (6.0, 0.022, 0.0012),
    (7.0, 0.016, 0.0014),
    (8.0, 0.013, 0.0011),
    (9.0, 0.021, 0.0011),
    (10.0, 0.017, 0.0016),
    (11.0, 0.012, 0.0015),
]
rch_package.ts.initialize(
    time_series_namerecord=["rch_1", "rch_2"],
    timeseries=ts_data,
    filename="recharge_rates.ts",
    interpolation_methodrecord=["stepwise", "stepwise"],
)

# build an recharge observation package that outputs the western recharge to a binary_
↪ file and the eastern
# recharge to a text file
obs_data = {
    ("rch_west.csv", "binary"): [
        ("rch_1_1_1", "RCH", (0, 0, 0)),
        ("rch_1_2_1", "RCH", (0, 1, 0)),
    ],
    "rch_east.csv": [
        ("rch_1_1_5", "RCH", (0, 0, 4)),
        ("rch_1_2_5", "RCH", (0, 1, 4)),
    ],
}
rch_package.obs.initialize(
    filename="example_model.rch.obs",
    digits=10,
    print_input=True,
    continuous=obs_data,
)

```

Saving and Running a MF6 Simulation

Saving and running a simulation are done with the MFSimulation class's `write_simulation` and `run_simulation` methods.

```

[16]: # write simulation to new location
sim.write_simulation()

# run simulation
success, buff = sim.run_simulation(silent=True, report=True)
assert success, "Failed to run"
for line in buff:
    print(line)

```

```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims...
  writing model example_model...
    writing model name file...
    writing package dis...
    writing package npf...
    writing package ic...
    writing package sto...
    writing package wel...
    writing package oc...
    writing package chd_0...
    writing package rch...
    writing package ts_0...
    writing package obs_0...
```

```
MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023
```

```
MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000
```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

```
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:23
```

```
Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam
```

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5

```
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:23
Elapsed run time: 0.042 Seconds
```

(continues on next page)

(continued from previous page)

Normal termination of simulation.

Exporting a MF6 Model

Exporting a MF6 model to a shapefile or netcdf is the same as exporting a MF2005 model.

```
[17]: # make directory
pth = Path(temp_dir.name) / "netCDF_export"
pth.mkdir(exist_ok=True)

# export the dis package to a netcdf file
model.dis.export(pth / "dis.nc")

# export the botm array to a shapefile
model.dis.botm.export(str(pth / "botm.shp")) # supports os.PathLike or str

initialize_geometry::
model.crs: None
initialize_geometry::nc_crs = EPSG:4326
No CRS information for writing a .prj file.
Supply an valid coordinate system reference to the attached modelgrid object or .
↪ export() method.
```

Loading an Existing MF6 Simulation

Loading a simulation can be done with the flopy.mf6.MFSimulation.load static method.

```
[18]: # load the simulation
loaded_sim = flopy.mf6.MFSimulation.load(sim_name, "mf6", "mf6", sim_path)

loading simulation...
  loading simulation name file...
  loading tdis package...
  loading model gwf6...
    loading package dis...
    loading package npf...
    loading package ic...
    loading package sto...
    loading package wel...
    loading package oc...
    loading package chd...
    loading package rch...
  loading solution package example_model...
```

Retrieving Data and Modifying an Existing MF6 Simulation

Data can be easily retrieved from a simulation. Data can be retrieved using two methods. One method is to retrieve the data object from a master simulation dictionary that keeps track of all the data. The master simulation dictionary is accessed by accessing a simulation's "simulation_data" property and then the "mfdata" property:

```
sim.simulation_data.mfdata[(model_name, package_name, block_name, variable_name)]
```

The data path is the path to the data stored as a tuple containing the model name, package name, block name, and data name.

The second method is to get the data from the package object. If you do not already have the package object, you can work your way down the simulation structure, from the simulation to the correct model, to the correct package, and finally to the data object.

These methods are demonstrated in the code below.

```
[19]: # get hydraulic conductivity data object from the data dictionary
hk = sim.simulation_data.mfdata[(model_name, "npf", "griddata", "k")]

# get specific yield data object from the storage package
sy = sto_package.sy

# get the model object from the simulation object using the get_model method,
# which takes a string with the model's name and returns the model object
mdl = sim.get_model(model_name)
# get the package object from the model object using the get_package method,
# which takes a string with the package's name or type
ic = mdl.get_package("ic")
# get the data object from the initial condition package object
strt = ic.strt
```

Once you have the appropriate data object there are a number of methods to retrieve data from that object. Data retrieved can either be the data as it appears in the model file or the data with any factor specified in the model file applied to it. To get the raw data without applying a factor use the `get_data` method. To get the data with the factor already applied use `.array`.

Note that `MFArray` data is always a copy of the data stored by flopy. Modifying the copy of the flopy data will have no effect on the data stored in flopy. Non-constant internal `MFLIST` data is returned as a reference to a numpy recarray. Modifying this recarray will modify the data stored in flopy.

```
[20]: # get the data without applying any factor
hk_data_no_factor = hk.get_data()
print("Data without factor:\n{}\n".format(hk_data_no_factor))

# get data with factor applied
hk_data_factor = hk.array
print("Data with factor:\n{}\n".format(hk_data_factor))

Data without factor:
[[[ 65.  60.  55.  50.  45.]
  [ 40.  35.  30.  25.  20.]]

 [[100. 100. 100. 100. 100.]
  [100. 100. 100. 100. 100.]]]
```

(continues on next page)

(continued from previous page)

```
Data with factor:
[[[ 97.5  90.   82.5  75.   67.5]
  [ 60.   52.5  45.   37.5  30. ]]

 [[100.  100.  100.  100.  100. ]
  [100.  100.  100.  100.  100. ]]]
```

Data can also be retrieved from the data object using []. For unlayered data the [] can be used to slice the data.

```
[21]: # slice layer one row two
print("SY slice of layer on row two\n{}\n".format(sy[0, :, 2]))
```

```
SY slice of layer on row two
[0.13 0.13]
```

For layered data specify the layer number within the brackets. This will return a “LayerStorage” object which let’s you change attributes of an individual layer.

```
[22]: # get layer one LayerStorage object
hk_layer_one = hk[0]
# change the print code and factor for layer one
hk_layer_one.iprn = "2"
hk_layer_one.factor = 1.1
print("Layer one data without factor:\n{}\n".format(hk_layer_one.get_data()))
print("Data with new factor:\n{}\n".format(hk.array))
```

```
Layer one data without factor:
[[65. 60. 55. 50. 45.]
 [40. 35. 30. 25. 20.]]
```

```
Data with new factor:
[[[ 71.5  66.   60.5  55.   49.5]
  [ 44.   38.5  33.   27.5  22. ]]

 [[100.  100.  100.  100.  100. ]
  [100.  100.  100.  100.  100. ]]]
```

Modifying Data

Data can be modified in several ways. One way is to set data for a given layer within a LayerStorage object, like the one accessed in the code above. Another way is to set the data attribute to the new data. Yet another way is to call the data object’s set_data method.

```
[23]: # set data within a LayerStorage object
hk_layer_one.set_data(
    [120.0, 100.0, 80.0, 70.0, 60.0, 50.0, 40.0, 30.0, 25.0, 20.0]
)
print("New HK data no factor:\n{}\n".format(hk.get_data()))
# set data attribute to new data
```

(continues on next page)

(continued from previous page)

```
ic_package.strt = 150.0
print("New strt values:\n{}\n".format(ic_package.strt.array))
# call set_data
sto_package.ss.set_data([0.000003, 0.000004])
print("New ss values:\n{}\n".format(sto_package.ss.array))
```

```
New HK data no factor:
[[[120. 100. 80. 70. 60.]
  [ 50. 40. 30. 25. 20.]
```

```

[[100. 100. 100. 100. 100.]
 [100. 100. 100. 100. 100.]]]
```

```
New strt values:
[[[150. 150. 150. 150. 150.]
  [150. 150. 150. 150. 150.]
```

```

[[150. 150. 150. 150. 150.]
 [150. 150. 150. 150. 150.]]]
```

```
New ss values:
[[[3.e-06 3.e-06 3.e-06 3.e-06 3.e-06]
  [3.e-06 3.e-06 3.e-06 3.e-06 3.e-06]]
```

```

[[[4.e-06 4.e-06 4.e-06 4.e-06 4.e-06]
  [4.e-06 4.e-06 4.e-06 4.e-06 4.e-06]]]
```

Modifying the Simulation Path

The simulation path folder can be changed by using the `set_sim_path` method in the `MFFileMgmt` object. The `MFFileMgmt` object can be obtained from the simulation object through properties:

```
sim.simulation_data.mfpath
```

```
[24]: save_folder = sim_path / "sim_modified" # define path
      save_folder.mkdir(exist_ok=True) # ensure path exists
      sim.set_sim_path(save_folder) # change simulation path
```

The `sim_path` property is a shortcut for `simulation_data.mfpath.get_sim_path()`:

```
[25]: assert sim.sim_path == sim.simulation_data.mfpath.get_sim_path()
```

Adding a Model Relative Path

A model relative path lets you put all of the files associated with a model in a folder relative to the simulation folder. Warning, this will override all of your file paths to model package files and will also override any relative file paths to external model data files.

[26]: *# Change path of model files relative to the simulation folder*

```
model.set_model_relative_path("model_folder")
model_folder = save_folder / "model_folder"
model_folder.mkdir(exist_ok=True)

# write simulation to new folder
sim.write_simulation()

# run simulation from new folder
success, buff = sim.run_simulation(silent=True, report=True)
assert success, "Failed to run"
for line in buff:
    print(line)
```

writing simulation...

```
writing simulation name file...
writing simulation tdis package...
writing solution package ims...
writing model example_model...
  writing model name file...
  writing package dis...
  writing package npf...
  writing package ic...
  writing package sto...
  writing package wel...
  writing package oc...
  writing package chd_0...
  writing package rch...
  writing package ts_0...
  writing package obs_0...
```

MODFLOW 6

U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_0000000

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water

(continues on next page)

(continued from previous page)

Resources Software User Rights Notice for complete use, copyright,
and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:24

Writing simulation list file: mfsim.lst

Using Simulation name file: mfsim.nam

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:24

Elapsed run time: 0.047 Seconds

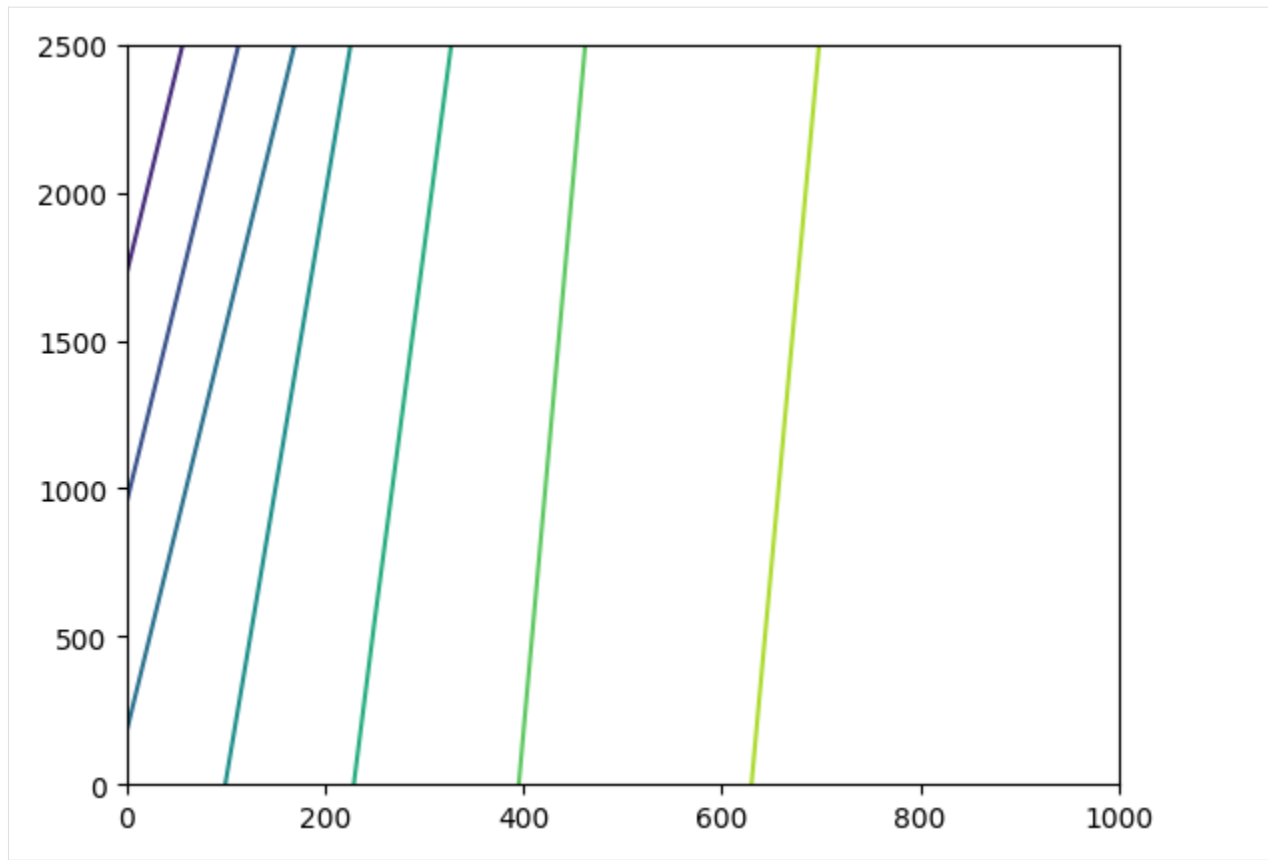
Normal termination of simulation.

Post-Processing the Results

Results can be retrieved using the output method available on each groundwater flow and transport model.

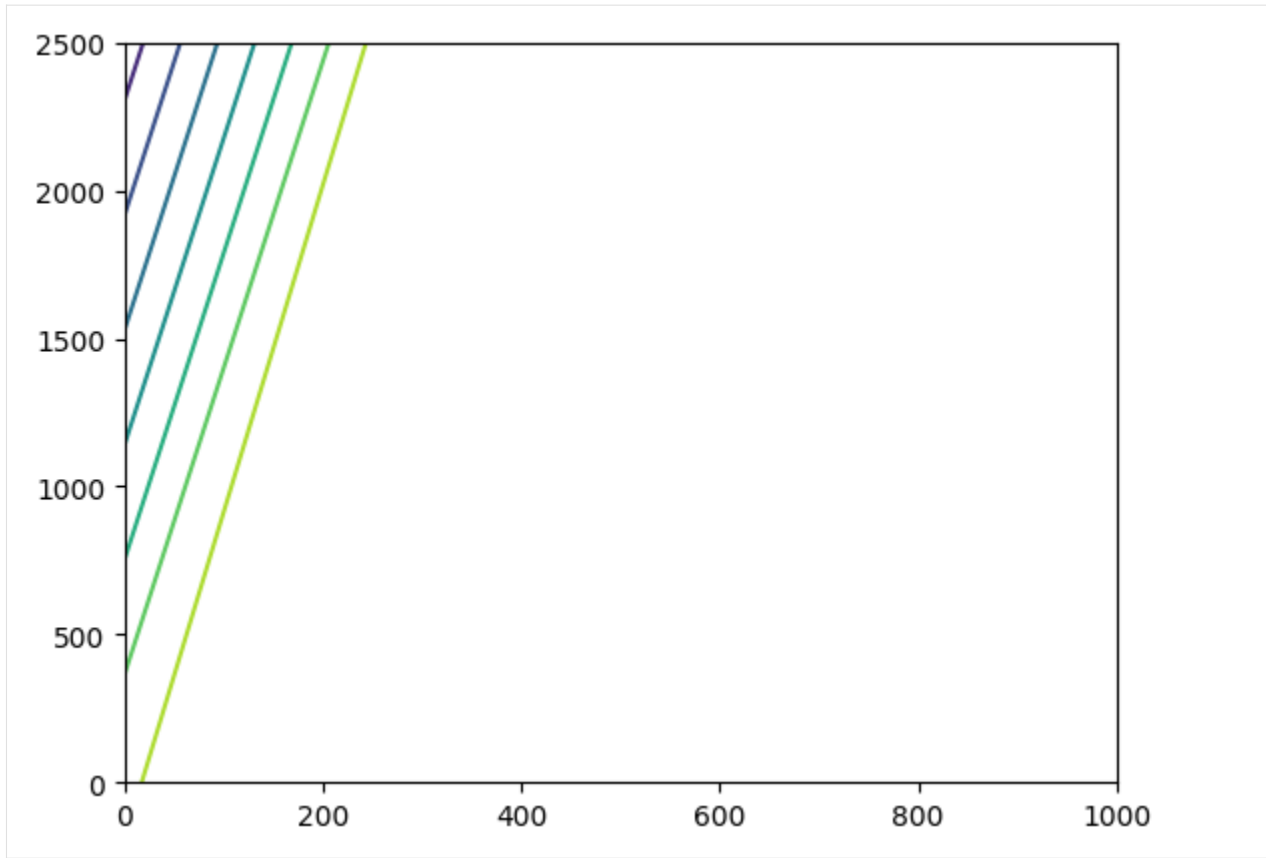
```
[27]: import matplotlib.pyplot as plt
import numpy as np

# get all head data
head = model.output.head().get_alldata()
# get the head data from the end of the model run
head_end = head[-1]
# plot the head data from the end of the model run
levels = np.arange(160, 162, 1)
extent = (0.0, 1000.0, 2500.0, 0.0)
plt.contour(head_end[0, :, :], extent=extent)
plt.show()
```



Results can also be retrieved using the existing `binaryfile` method.

```
[28]: # get head data using old flopy method
hds_path = sim_path / f"{model_name}.hds"
hds = flopy.utils.HeadFile(hds_path)
# get heads after 1.0 days
head = hds.get_data(totim=1.0)
# plot head data
plt.contour(head[0, :, :], extent=extent)
plt.show()
```



Clean up the temporary workspace.

```
[29]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.6 MODFLOW USG examples

6.6.1 MODFLOW-USG CLN package demo

This example problem demonstrates use of the CLN process for simulating flow to a well pumping from two aquifers separated by an impermeable confining unit. A structured finite-difference grid with 100 rows and 100 columns was used. Each square cells is 470m by 470 m in extent. Initial heads are 10m in aquifer 1 and 30m in aquifer 2.

```
[1]: import os
import shutil
from tempfile import TemporaryDirectory

import matplotlib.pyplot as plt
import numpy as np
```

(continues on next page)

(continued from previous page)

```
import flopy
```

```
[2]: # temporary directory
temp_dir = TemporaryDirectory()
cln_ws = temp_dir.name
```

Loading Example 03_conduit_confined

The elevation of the top of layer 1 is -100 m, the bottom of layer 1 is -110 m, the top of layer 2 is -120 m and the bottom of layer 2 is -130 m. The confining unit between layers 1 and 2 is impermeable and is represented using a quasi-three-dimensional approach. The only way a stress from one aquifer can be propagated to another is through the cln well that penetrates both aquifers.

The hydraulic conductivity values of the upper and lower aquifers are 100 and 400 m/d, respectively. Both aquifers have a primary storage coefficient of 0.0001 and a specific yield of 0.01.

A vertical conduit well is located at the center of the domain and has a radius of 0.5 m. The well pumps 62,840 m³/d and is open fully to both aquifers from top to bottom. The CLN Process was used with a circular conduit geometry type to discretize the well bore with two conduit cells, one in each layer. The WEL Package was used to pump from the bottom CLN cell.

```
[3]: model_ws = os.path.join(
    "../examples/data/mfusg_test", "03_conduit_confined"
)
mf = flopy.mfusg.MfUsrg.load(
    "ex3.nam", model_ws=model_ws, exe_name="mfusg", check=False, verbose=True
)
```

Creating new model with name: ex3

```
-----
Parsing the namefile --> ../examples/data/mfusg_test/03_conduit_confined/ex3.nam
could not set filehandle to ex3.lst
could not set filehandle to ex3.cbb
could not set filehandle to ex3.hds
could not set filehandle to ex3.ddn
could not set filehandle to ex3.clnhds
could not set filehandle to ex3.clnhds
-----
```

External unit dictionary:

```
{7: filename:../examples/data/mfusg_test/03_conduit_confined/ex3.lst, filetype:LIST,
→ 1: filename:../examples/data/mfusg_test/03_conduit_confined/ex3.bas, filetype:BAS6,
→ 14: filename:../examples/data/mfusg_test/03_conduit_confined/ex3.wel, filetype:WEL,
→ 12: filename:../examples/data/mfusg_test/03_conduit_confined/ex3.dis, filetype:DIS,
→ 11: filename:../examples/data/mfusg_test/03_conduit_confined/ex3.bcf, filetype:BCF6,
→ 13: filename:../examples/data/mfusg_test/03_conduit_confined/ex3.cln, filetype:CLN,
→ 22: filename:../examples/data/mfusg_test/03_conduit_confined/ex3.oc, filetype:OC,
→ 19: filename:../examples/data/mfusg_test/03_conduit_confined/ex3.sms, filetype:SMS,
→ 50: filename:../examples/data/mfusg_test/03_conduit_confined/ex3.cbb, filetype:
```

(continues on next page)

(continued from previous page)

```

→DATA(BINARY), 30: filename:../../examples/data/mfusg_test/03_conduit_confined/ex3.hds,
→ filetype:DATA(BINARY), 31: filename:../../examples/data/mfusg_test/03_conduit_confined/
→ex3.ddn, filetype:DATA(BINARY), 35: filename:../../examples/data/mfusg_test/03_conduit_
→confined/ex3.clnccb, filetype:DATA(BINARY), 36: filename:../../examples/data/mfusg_
→test/03_conduit_confined/ex3.clnhds, filetype:DATA(BINARY)}
-----

```

ModflowBas6 free format:True

loading dis package file...

Loading dis package with:

2 layers, 100 rows, 100 columns, and 1 stress periods

loading laycbd...

loading delr...

loading delc...

loading top...

loading botm...

for 2 layers and 1 confining beds

loading stress period data...

for 1 stress periods

adding Package: DIS

DIS package load...success

LIST package load...skipped

loading bas6 package file...

adding Package: BAS6

BAS6 package load...success

loading wel package file...

loading <class 'flopy.mfusg.mfusgwel.MfUsgWel'> for kper 1

implicit itmp_cln of 0 in ../../examples/data/mfusg_test/03_conduit_confined/ex3.wel

Adding ex3.cbc (unit=50) to the output list.

adding Package: WEL

WEL package load...success

loading bcf package file...

loading ipakcb, HDRY, IWDFLG, WETFCT, IWETIT, IHDWET...

loading LAYCON...

loading TRPY...

loading sf1 layer 1...

loading hy layer 1...

loading vcont layer 1...

loading sf2 layer 1...

loading sf1 layer 2...

loading hy layer 2...

loading sf2 layer 2...

adding Package: BCF6

BCF6 package load...success

loading CLN package file...

ncln 1

iclnnds -1

iclncb 35

iclnhd 36

iclndd 0

(continues on next page)

(continued from previous page)

```

iclnib 0
  nclngwc 2
TRANSIENT False
  PRINTIAJA False
RECTANGULAR 0
  BHEDETAIL False
SAVECLNCON 0
  SAVECLNMAS 0
GRAVITY None
  VISCOSITY None
Reading nndcln...
Reading node_prop...
Reading cln_gwc...
Reading cln_circ...
Reading ibound...
Reading strt...
Adding ex3.clnbbb (unit=35) to the output list.
Adding ex3.clnhds (unit=36) to the output list.
adding Package: CLN
  CLN package load...success
loading oc package file...
Adding ex3.hds (unit=30) to the output list.
Adding ex3.ddn (unit=31) to the output list.
adding Package: OC
  OC package load...success
loading sms package file...
  loading HCLOSE HICLOSE MXITER ITER1 IPRSMS NONLINMETH LINMETH...
  HCLOSE 0.001
  HICLOSE 1e-05
  MXITER 220
  ITER1 600
  IPRSMS 1
  NONLINMETH 2
  LINMETH 1
  loading THETA AKAPPA GAMMA AMOMENTUM NUMTRACK BTOL BREDUC RESLIM...
  THETA 0.9
  AKAPPA 0.07
  GAMMA 0.1
  AMOMENTUM 0.0
  NUMTRACK 200
  BTOL 1.1
  BREDUC 0.2
  RESLIM 1.0
  loading IACL NORDER LEVEL NORTH IREDSYS RRCTOL IDROPTOL EPSRN
  IACL 2
  NORDER 1
  LEVEL 3
  NORTH 14
  IREDSYS 0
  RRCTOL 0.0
  IDROPTOL 0
  EPSRN 0.001

```

(continues on next page)

(continued from previous page)

```

adding Package:  SMS
  SMS  package load...success
  DATA(BINARY) package load...skipped
    ex3.cbb
  DATA(BINARY) package load...skipped
    ex3.hds
  DATA(BINARY) package load...skipped
    ex3.ddn
  DATA(BINARY) package load...skipped
    ex3.clncbb
  DATA(BINARY) package load...skipped
    ex3.clnhds

WARNING:
  External file unit 0 does not exist in ext_unit_dict.

  The following 7 packages were successfully loaded.
    ex3.dis
    ex3.bas
    ex3.wel
    ex3.bcf
    ex3.cln
    ex3.oc
    ex3.sms
  The following 1 packages were not loaded.
    ex3.lst

```

```

[4]: # output control
mf.remove_package("OC")

spd = {}
for i in range(mf.nper):
    for j in range(mf.dis.nstp[i]):
        spd[(i, j)] = ["save head", "save budget"]

oc = flopy.modflow.ModflowOc(
    mf, stress_period_data=spd, unitnumber=[22, 30, 31, 50]
)

removing Package:  ['OC']
adding Package:  OC

```

```

[5]: model_ws = os.path.join(cln_ws, "ex03")

if os.path.exists(model_ws):
    shutil.rmtree(model_ws)
os.mkdir(model_ws)

mf.model_ws = model_ws

changing model workspace...
../..../tmp/tmpuq6p___p/ex03

```

```
[6]: mf.write_input()
      success, buff = mf.run_model(silent=True, report=True)
      if success:
          for line in buff:
              print(line)
      else:
          raise ValueError("Failed to run.")
```

Writing packages:

```
Package: DIS
Package: BAS6
Package: WEL
Package: BCF6
Package: CLN
Package: SMS
Package: OC
```

MODFLOW-USG

U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER FLOW MODEL
Version 1.5.00 02/27/2019

Using NAME file: ex3.nam

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:47

Solving:	Stress period:	1	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	2	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	3	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	4	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	5	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	6	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	7	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	8	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	9	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	10	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	11	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	12	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	13	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	14	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	15	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	16	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	17	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	18	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	19	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	20	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	21	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	22	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	23	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	24	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	25	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	26	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	27	Groundwater Flow Eqn.

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 1 Time step: 28 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 29 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 30 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 31 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 32 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 33 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 34 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 35 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 36 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 37 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 38 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 39 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 40 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 41 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 42 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 43 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 44 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 45 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 46 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 47 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 48 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 49 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 50 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 51 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 52 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 53 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 54 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 55 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 56 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 57 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 58 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 59 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 60 Groundwater Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:49
Elapsed run time: 2.007 Seconds

Normal termination of simulation

```

```

[7]: head_file = os.path.join(mf.model_ws, "ex3.clnhds")
headobj = flopy.utils.HeadFile(head_file)

```

```

[8]: simtimes = headobj.get_times()

nper = len(simtimes)

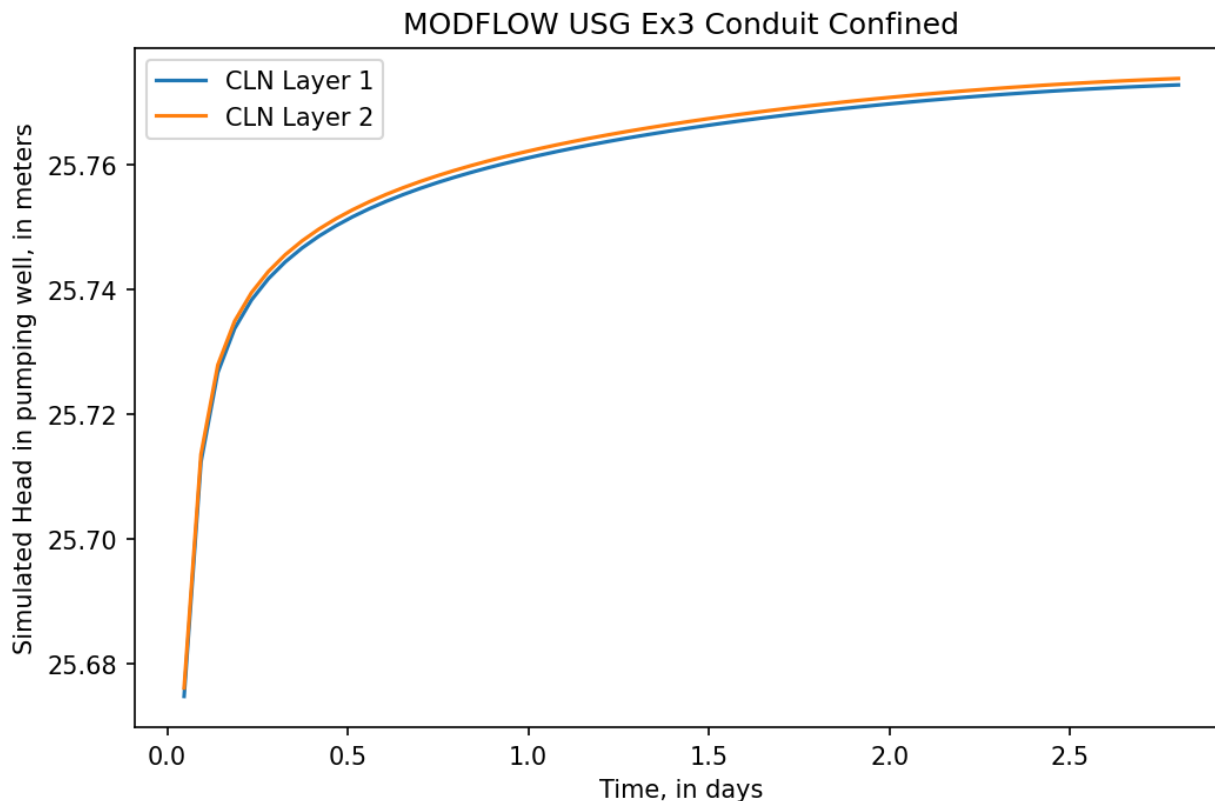
nnode = mf.cln.nclnnds

simhead = np.zeros((1, 1, nnode, nper))
for i in range(nper):
    simhead[:, :, :, i] = headobj.get_data(kstpiper=(i, 0))
simhead = np.squeeze(simhead)

```

```
[9]: fig = plt.figure(figsize=(8, 5), dpi=150)
ax = fig.add_subplot(111)
ax.plot(simtimes, simhead[0], label="CLN Layer 1")
ax.plot(simtimes, simhead[1], label="CLN Layer 2")
ax.set_xlabel("Time, in days")
ax.set_ylabel("Simulated Head in pumping well, in meters")
ax.set_title("MODFLOW USG Ex3 Conduit Confined")
ax.legend()
```

```
[9]: <matplotlib.legend.Legend at 0x7fa422f83580>
```



```
[10]: cbb_file = os.path.join(mf.model_ws, "ex3.clncbb")
cbb = flopy.utils.CellBudgetFile(cbb_file)
# cbb.list_records()

simflow = cbb.get_data(kstpker=(0, 0), text="GWF")[0]
for i in range(nper - 1):
    simflow = np.append(
        simflow, cbb.get_data(kstpker=(i + 1, 0), text="GWF")[0]
    )
simflow1 = simflow[simflow["node"] == 1]["q"]
simflow2 = simflow[simflow["node"] == 2]["q"]
```

```
[11]: fig = plt.figure(figsize=(8, 11), dpi=150)
ax1 = fig.add_subplot(211)
ax1.plot(simtimes, simflow1, label="CLN Layer 1")
```

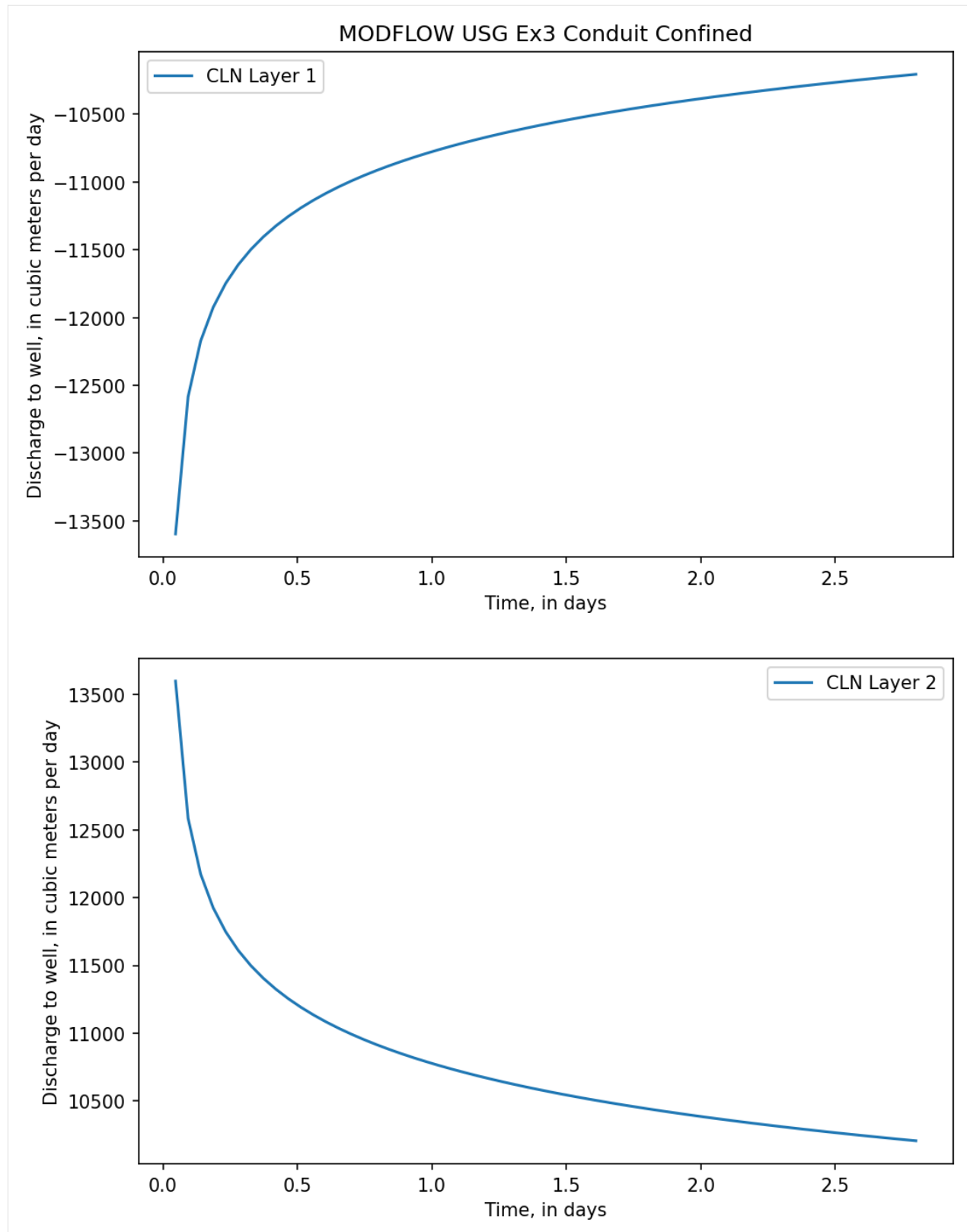
(continues on next page)

(continued from previous page)

```
ax1.set_xlabel("Time, in days")
ax1.set_ylabel("Discharge to well, in cubic meters per day")
ax1.set_title("MODFLOW USG Ex3 Conduit Confined")
ax1.legend()

ax2 = fig.add_subplot(212)
ax2.plot(simtimes, simflow2, label="CLN Layer 2")
ax2.set_xlabel("Time, in days")
ax2.set_ylabel("Discharge to well, in cubic meters per day")
ax2.legend()
```

```
[11]: <matplotlib.legend.Legend at 0x7fa422b87790>
```



Create example 03A_conduit_unconfined of mfusg 1.5

An unconfined example was simulated with this problem setup. The same simulation setup of previous example is used. However, the elevation of the top of layer 1 is 10 m, the bottom of layer 1 is 0 m, the top of layer 2 is -10 m, and the bottom of layer 2 is -20 m. The CLN domain is discretized using two vertical conduit cells. It depicts the behavior of unconfined flow in the conduit when the CLN cell in layer 1 becomes dry.

```
[12]: modelname = "ex03a"
      model_ws = os.path.join(cln_ws, modelname)
      mf = flopy.mfusg.MfUsg(
          modelname=modelname, model_ws=model_ws, exe_name="mfusg", verbose=True
      )
```

```
[13]: ipakcb = 50

      nlay = 2
      nrow = 100
      ncol = 100
      laycbd = [1, 0]

      delr = 470.000
      delc = 470.000

      ztop = 10.0
      botm = [0.0, -10.0, -20.0]

      perlen = 160
      nstp = 160

      dis = flopy.modflow.ModflowDis(
          mf,
          nlay,
          nrow,
          ncol,
          delr=delr,
          delc=delc,
          laycbd=laycbd,
          top=ztop,
          botm=botm,
          perlen=perlen,
          nstp=nstp,
          steady=False,
          lenuni=0,
      )

      bas = flopy.modflow.ModflowBas(mf, ibound=1, strt=[10.0, 30.0])

      bcf = flopy.mfusg.MfUsgBcf(
          mf,
          ipakcb=ipakcb,
          laycon=4,
          wetfct=1.0,
          iwetit=5,
```

(continues on next page)

(continued from previous page)

```

    hy=[100.0, 400.0],
    vcont=0.0,
    sf1=1e-4,
    sf2=0.01,
)

sms = flopy.mfusg.MfUsgSms(
    mf,
    hclose=1.0e-3,
    hiclose=1.0e-5,
    mxiter=220,
    iter1=600,
    iprsms=1,
    nonlinmeth=2,
    linmeth=1,
    theta=0.9,
    akappa=0.07,
    gamma=0.1,
    amomentum=0.0,
    numtrack=200,
    btol=1.1,
    breduc=0.2,
    reslim=1.0,
    iacl=2,
    norder=1,
    level=3,
    north=14,
)

```

```

adding Package: DIS
adding Package: BAS6
Adding ex03a.cbc (unit=50) to the output list.
adding Package: BCF6
adding Package: SMS

```

```

[14]: # output control
spd = {}
for i in range(mf.nper):
    for j in range(mf.dis.nstp[i]):
        spd[(i, j)] = ["save head", "save budget"]

oc = flopy.modflow.ModflowOc(mf, stress_period_data=spd)

Adding ex03a.hds (unit=51) to the output list.
adding Package: OC

```

```

[15]: unitnumber = [71, 35, 36, 0, 0, 0, 0]

node_prop = [
    [1, 1, 0, 10.0, 0.0, 1.57, 0, 0],
    [2, 1, 0, 10.0, -20.0, 1.57, 0, 0],
]

```

(continues on next page)

(continued from previous page)

```

cln_gwc = [
    [1, 1, 50, 50, 0, 0, 10.0, 1.0, 0],
    [2, 2, 50, 50, 0, 0, 10.0, 1.0, 0],
]

```

```

nconduityp = 1
cln_circ = [[1, 0.5, 3.23e10]]

```

```

strt = [10.0, 30.0]
cln = flopy.mfusg.MfUsgCln(
    mf,
    ncln=1,
    iclnnds=-1,
    nndcln=2,
    nclngwc=2,
    node_prop=node_prop,
    cln_gwc=cln_gwc,
    cln_circ=cln_circ,
    strt=strt,
    unitnumber=unitnumber,
)

```

Adding ex03a.clncb (unit=35) to the output list.
 Adding ex03a.clnhd (unit=36) to the output list.
 adding Package: CLN

```

[16]: options = []
options.append("autoflowreduce")
cln_stress_period_data = {0: [[1, -62840.0]]}

wel = flopy.mfusg.MfUsgWel(
    mf,
    ipakcb=ipakcb,
    options=options,
    cln_stress_period_data=cln_stress_period_data,
)

```

adding Package: WEL

```

[17]: wel.cln_stress_period_data.data

```

```

[17]: {0: rec.array([(1, -62840.)],
    dtype=[('node', '<i8'), ('flux', '<f4')])}

```

```

[18]: mf.write_input()
success, buff = mf.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

```

Writing packages:

Package: DIS
 Package: BAS6
 Package: BCF6
 Package: SMS
 Package: OC
 Package: CLN
 Package: WEL

MODFLOW-USG

U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER FLOW MODEL
 Version 1.5.00 02/27/2019

Using NAME file: ex03a.nam

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:50

Solving:	Stress period:	1	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	2	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	3	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	4	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	5	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	6	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	7	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	8	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	9	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	10	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	11	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	12	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	13	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	14	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	15	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	16	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	17	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	18	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	19	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	20	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	21	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	22	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	23	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	24	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	25	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	26	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	27	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	28	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	29	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	30	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	31	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	32	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	33	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	34	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	35	Groundwater Flow Eqn.

(continues on next page)

[illegible]

(continued from previous page)

[illegible]

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 1 Time step: 140 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 141 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 142 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 143 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 144 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 145 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 146 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 147 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 148 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 149 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 150 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 151 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 152 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 153 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 154 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 155 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 156 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 157 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 158 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 159 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 160 Groundwater Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:05
Elapsed run time: 15.561 Seconds

```

Normal termination of simulation

```

[19]: head_file = os.path.join(mf.model_ws, modelname + ".clnhd")
      headobj = flopy.utils.HeadFile(head_file)

      simtimes = headobj.get_times()
      nper = len(simtimes)
      nnode = mf.cln.nclnnds

      simhead = np.zeros((1, 1, nnode, nper))
      for i in range(nper):
          simhead[:, :, :, i] = headobj.get_data(kstpker=(i, 0))

      head_case1 = np.squeeze(simhead)

```

```

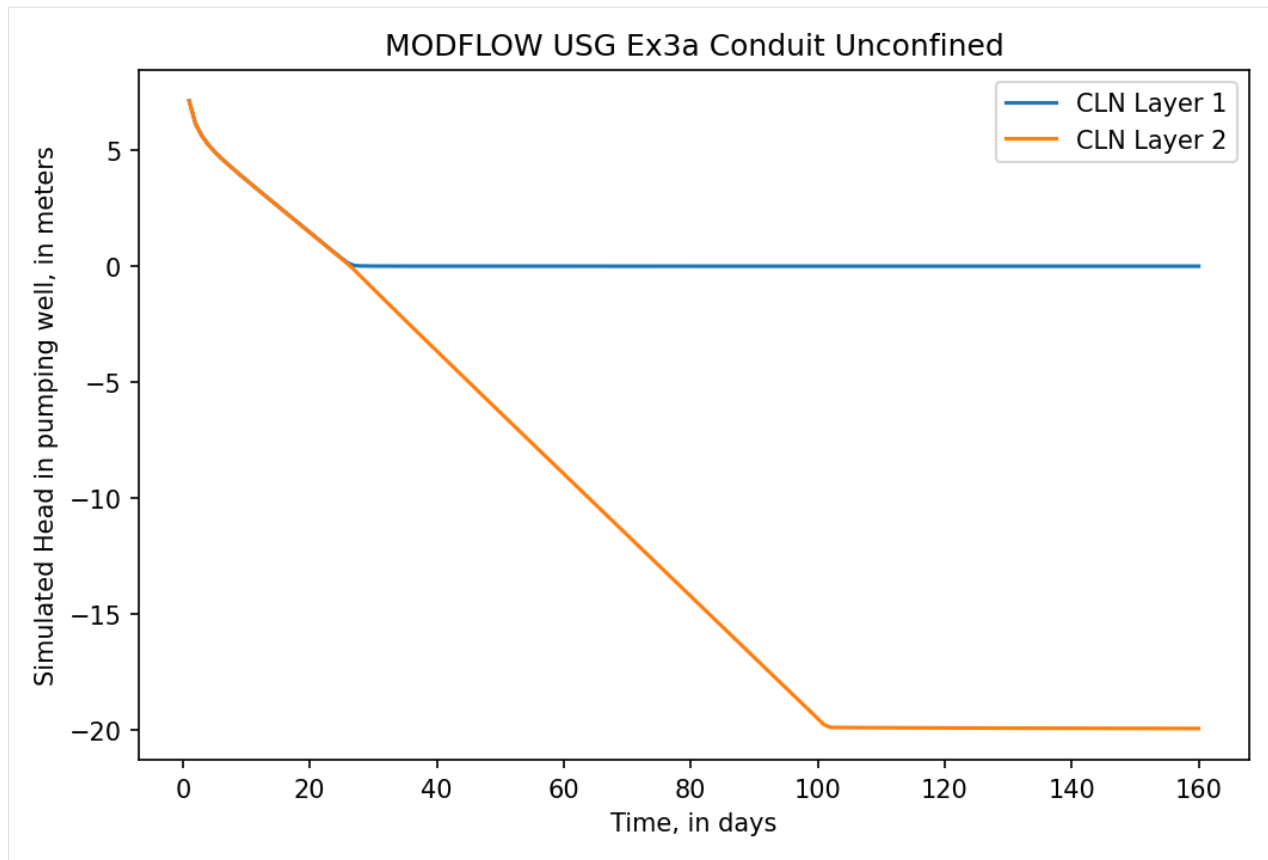
[20]: fig = plt.figure(figsize=(8, 5), dpi=150)
      ax = fig.add_subplot(111)
      ax.plot(simtimes, head_case1[0], label="CLN Layer 1")
      ax.plot(simtimes, head_case1[1], label="CLN Layer 2")
      ax.set_xlabel("Time, in days")
      ax.set_ylabel("Simulated Head in pumping well, in meters")
      ax.set_title("MODFLOW USG Ex3a Conduit Unconfined")
      ax.legend()

```

```

[20]: <matplotlib.legend.Legend at 0x7fa422ad7820>

```



```
[21]: cbb_file = os.path.join(mf.model_ws, modelname + ".clncb")
      cbb = flopy.utils.CellBudgetFile(cbb_file)
      # cbb.list_records()

      simflow = cbb.get_data(kstpkper=(0, 0), text="GWF")[0]
      for i in range(nper - 1):
          simflow = np.append(
              simflow, cbb.get_data(kstpkper=(i + 1, 0), text="GWF")[0]
          )

      flow_case1 = simflow
```

Modify CLN and WEL package to example create 03B_conduit_unconfined of mfusg 1.5

The problem is solved using only one CLN conduit cell to represent the well connecting both aquifer layers. This is conceptually equivalent to the MNW methodology and does not solve for flow within the well. The behavior of unconfined flow between the well and layer 1 neglects the dry-cell condition whereby head in the well is below the bottom of layer 1.

```
[22]: modelname = "ex03b"
      model_ws = os.path.join(cln_ws, modelname)

      mf.model_ws = model_ws
      mf._set_name(modelname)
```

(continues on next page)

(continued from previous page)

```
for i, fname in enumerate(mf.output_fnames):
    mf.output_fnames[i] = modelname + os.path.splitext(fname)[1]
```

```
creating model workspace...
.././.././.././../././tmp/tmpuq6p____p/ex03b

changing model workspace...
.././.././.././../././tmp/tmpuq6p____p/ex03b
```

```
[23]: mf.remove_package("CLN")

node_prop = [[1, 1, 0, 30.0, -20.0, 1.57]]
cln_gwc = [
    [1, 1, 50, 50, 0, 0, 10.0, 1.0, 0],
    [1, 2, 50, 50, 0, 0, 10.0, 1.0, 0],
]

strt = 20.0
cln = flopy.mfusg.MfUsgCln(
    mf,
    ncln=1,
    iclnnds=-1,
    nndcln=1,
    nclngwc=2,
    node_prop=node_prop,
    cln_gwc=cln_gwc,
    cln_circ=cln_circ,
    strt=strt,
    unitnumber=unitnumber,
)
```

```
removing Package: ['CLN']
adding Package: CLN
```

```
[24]: mf.remove_package("WEL")

options = []
options.append("autoflowreduce")
options.append("iunitafr 55")
cln_stress_period_data = {0: [[0, -62840.0]]}

wel = flopy.mfusg.MfUsgWel(
    mf,
    ipakcb=ipakcb,
    options=options,
    cln_stress_period_data=cln_stress_period_data,
)
```

```
removing Package: ['WEL']
Adding ex03b.afr (unit=55) to the output list.
adding Package: WEL
```

```
[25]: mf.write_input()
      success, buff = mf.run_model(silent=True, report=True)
      if success:
          for line in buff:
              print(line)
      else:
          raise ValueError("Failed to run.")
```

Writing packages:

```
Package: DIS
Package: BAS6
Package: BCF6
Package: SMS
Package: OC
Package: CLN
Package: WEL
```

MODFLOW-USG

U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER FLOW MODEL
Version 1.5.00 02/27/2019

Using NAME file: ex03b.nam

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:06

Solving:	Stress period:	1	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	2	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	3	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	4	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	5	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	6	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	7	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	8	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	9	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	10	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	11	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	12	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	13	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	14	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	15	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	16	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	17	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	18	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	19	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	20	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	21	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	22	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	23	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	24	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	25	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	26	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	27	Groundwater Flow Eqn.

(continues on next page)

[illegible]

(continues on next page)

(continued from previous page)

[illegible]

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 1 Time step: 132 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 133 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 134 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 135 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 136 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 137 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 138 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 139 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 140 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 141 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 142 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 143 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 144 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 145 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 146 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 147 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 148 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 149 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 150 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 151 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 152 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 153 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 154 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 155 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 156 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 157 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 158 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 159 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 160 Groundwater Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:17
Elapsed run time: 11.180 Seconds

```

Normal termination of simulation

```

[26]: head_file = os.path.join(mf.model_ws, modelname + ".clnhd")
      headobj = flopy.utils.HeadFile(head_file)

      simtimes = headobj.get_times()
      nper = len(simtimes)
      nnode = mf.cln.nclnnds

      simhead = np.zeros((1, 1, nnode, nper))
      for i in range(nper):
          simhead[:, :, :, i] = headobj.get_data(kstpker=(i, 0))

      head_case2 = np.squeeze(simhead)

```

```

[27]: fig = plt.figure(figsize=(8, 5), dpi=150)
      ax = fig.add_subplot(111)
      ax.plot(simtimes, head_case2)
      ax.set_xlabel("Time, in days")

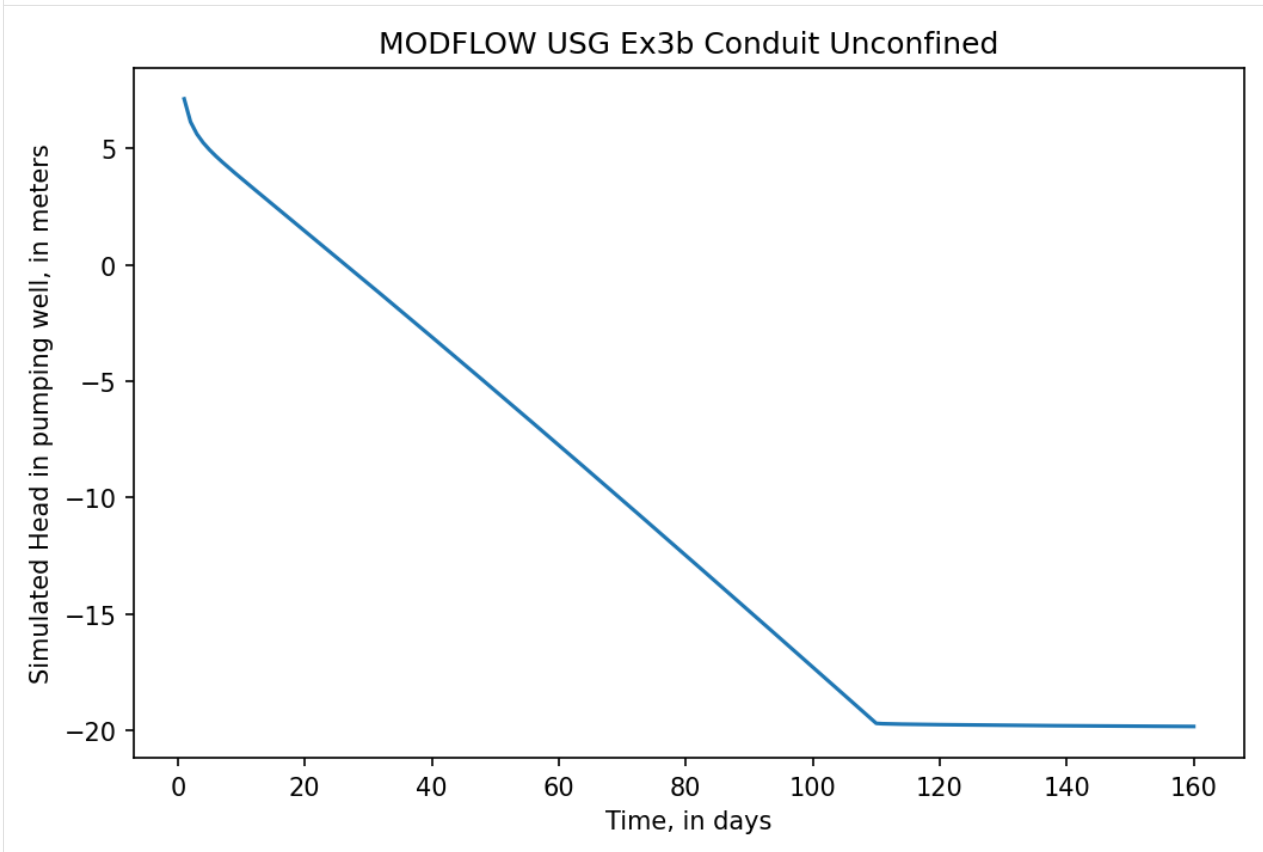
```

(continues on next page)

(continued from previous page)

```
ax.set_ylabel("Simulated Head in pumping well, in meters")
ax.set_title("MODFLOW USG Ex3b Conduit Unconfined")
```

```
[27]: Text(0.5, 1.0, 'MODFLOW USG Ex3b Conduit Unconfined')
```



```
[28]: cbb_file = os.path.join(mf.model_ws, modelname + ".clncb")
cbb = flopy.utils.CellBudgetFile(cbb_file)
# cbb.list_records()

simflow = cbb.get_data(kstpkper=(0, 0), text="GWF")[0]
for i in range(nper - 1):
    simflow = np.append(
        simflow, cbb.get_data(kstpkper=(i + 1, 0), text="GWF")[0]
    )

flow_case2 = simflow
```

Modify CLN amd WEL package to example create 03C_conduit_unconfined of mfusg 1.5

The CLN well is discretized using two conduit geometry CLN cells but with the confined option for flow within the conduit and between CLN and GWF domains.

```
[29]: modelname = "ex03c"
      model_ws = os.path.join(cln_ws, modelname)

      mf.model_ws = model_ws
      mf._set_name(modelname)
      for i, fname in enumerate(mf.output_fnames):
          mf.output_fnames[i] = modelname + os.path.splitext(fname)[1]
```

```
creating model workspace...
  ../../../../../../tmp/tmpuq6p__p/ex03c
changing model workspace...
  ../../../../../../tmp/tmpuq6p__p/ex03c
```

```
[30]: mf.remove_package("CLN")

node_prop = [
    [1, 1, 0, 10.0, 0.0, 1.57, 1, 0],
    [2, 1, 0, 10.0, -20.0, 1.57, 1, 0],
]

cln_gwc = [
    [1, 1, 50, 50, 0, 0, 10.0, 1.0, 0],
    [2, 2, 50, 50, 0, 0, 10.0, 1.0, 0],
]

strt = [10.0, 30.0]

cln = flopy.mfug.MfUsgCln(
    mf,
    ncln=1,
    iclnnds=-1,
    nndcln=2,
    nclngwc=2,
    node_prop=node_prop,
    cln_gwc=cln_gwc,
    cln_circ=cln_circ,
    strt=strt,
    unitnumber=unitnumber,
)
```

```
removing Package:  ['CLN']
adding Package:    CLN
```

```
[31]: mf.remove_package("WEL")

      cln_stress_period_data = {0: [[1, -62840.0]]}

      wel = floppy.mfusg.MfUsgWel(
```

(continues on next page)

(continued from previous page)

```

mf,
ipakcb=ipakcb,
options=options,
cln_stress_period_data=cln_stress_period_data,
)

```

```
removing Package: ['WEL']
```

```
adding Package: WEL
```

```

[32]: mf.write_input()
success, buff = mf.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

```

Writing packages:

```

Package: DIS
Package: BAS6
Package: BCF6
Package: SMS
Package: OC
Package: CLN
Package: WEL

```

MODFLOW-USG

U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER FLOW MODEL
Version 1.5.00 02/27/2019

Using NAME file: ex03c.nam

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:18

Solving:	Stress period:	1	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	2	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	3	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	4	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	5	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	6	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	7	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	8	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	9	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	10	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	11	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	12	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	13	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	14	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	15	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	16	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	17	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	18	Groundwater Flow Eqn.

(continues on next page)

[illegible]

(continued from previous page)

[illegible]

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 1 Time step: 123 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 124 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 125 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 126 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 127 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 128 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 129 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 130 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 131 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 132 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 133 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 134 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 135 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 136 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 137 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 138 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 139 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 140 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 141 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 142 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 143 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 144 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 145 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 146 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 147 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 148 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 149 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 150 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 151 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 152 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 153 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 154 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 155 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 156 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 157 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 158 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 159 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 160 Groundwater Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:31
Elapsed run time: 13.103 Seconds

Normal termination of simulation

```

```

[33]: head_file = os.path.join(mf.model_ws, modelname + ".clnhd")
      headobj = flopy.utils.HeadFile(head_file)

      simtimes = headobj.get_times()
      nper = len(simtimes)
      nnode = mf.cln.nclnnds

      simhead = np.zeros((1, 1, nnode, nper))
      for i in range(nper):

```

(continues on next page)

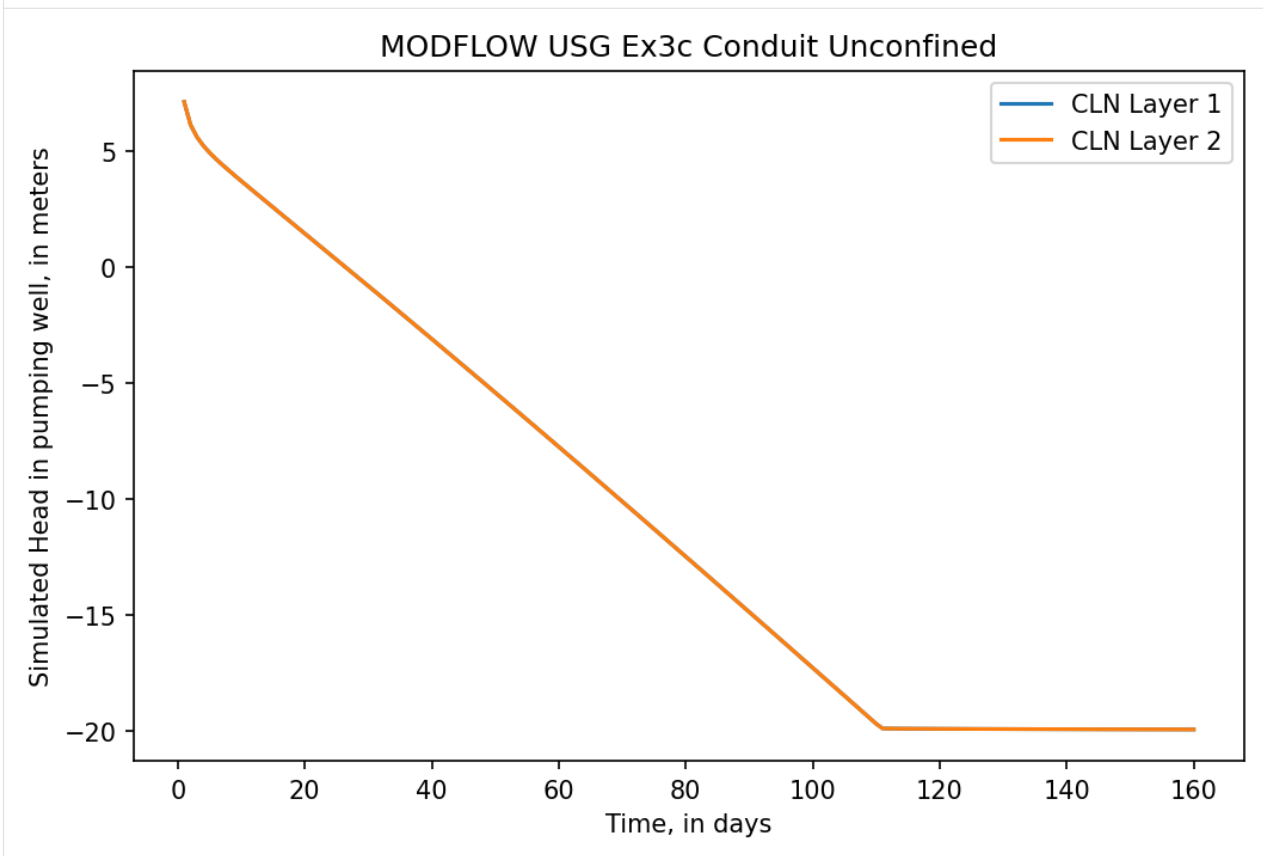
(continued from previous page)

```
simhead[:, :, :, i] = headobj.get_data(kstpkper=(i, 0))

head_case3 = np.squeeze(simhead)
```

```
[34]: fig = plt.figure(figsize=(8, 5), dpi=150)
ax = fig.add_subplot(111)
ax.plot(simtimes, head_case3[0], label="CLN Layer 1")
ax.plot(simtimes, head_case3[1], label="CLN Layer 2")
ax.set_xlabel("Time, in days")
ax.set_ylabel("Simulated Head in pumping well, in meters")
ax.set_title("MODFLOW USG Ex3c Conduit Unconfined")
ax.legend()
```

```
[34]: <matplotlib.legend.Legend at 0x7fa4230af850>
```



```
[35]: cbb_file = os.path.join(mf.model_ws, modelname + ".clncb")
cbb = flopy.utils.CellBudgetFile(cbb_file)
# cbb.list_records()

simflow = cbb.get_data(kstpkper=(0, 0), text="GWF")[0]
for i in range(nper - 1):
    simflow = np.append(
        simflow, cbb.get_data(kstpkper=(i + 1, 0), text="GWF")[0]
    )
```

(continues on next page)

(continued from previous page)

```
flow_case3 = simflow
```

Modify CLN amd WEL package to example create 03D_conduit_unconfined of mfusg 1.5

Only one CLN cell to discretize the well but includes the “flow-to-dry-cell” option to limit flow in layer 1 when the head in the CLN cell is below the bottom of the layer.

```
[36]: modelname = "ex03d"
model_ws = os.path.join(cln_ws, modelname)

mf.model_ws = model_ws
mf._set_name(modelname)
for i, fname in enumerate(mf.output_fnames):
    mf.output_fnames[i] = modelname + os.path.splitext(fname)[1]

creating model workspace...
../..../tmp/tmpuq6p____p/ex03d

changing model workspace...
../..../tmp/tmpuq6p____p/ex03d
```

```
[37]: mf.remove_package("CLN")

node_prop = [[1, 1, 0, 30.0, -20.0, 1.57]]
cln_gwc = [
    [1, 1, 50, 50, 0, 0, 10.0, 1.0, 1],
    [1, 2, 50, 50, 0, 0, 10.0, 1.0, 1],
]

strt = 20.0

cln = flopy.mfusg.MfUsgCln(
    mf,
    ncln=1,
    iclnnds=-1,
    nndcln=1,
    nclngwc=2,
    node_prop=node_prop,
    cln_gwc=cln_gwc,
    cln_circ=cln_circ,
    strt=strt,
    unitnumber=unitnumber,
)
```

```
removing Package: ['CLN']
adding Package: CLN
```

```
[38]: mf.remove_package("WEL")

cln_stress_period_data = {0: [[0, -62840.0]]}
```

(continues on next page)

(continued from previous page)

```
wel = flopy.mfusg.MfUsgWel(
    mf,
    ipakcb=ipakcb,
    options=options,
    cln_stress_period_data=cln_stress_period_data,
)
```

```
removing Package:  ['WEL']
adding Package:  WEL
```

```
[39]: mf.write_input()
success, buff = mf.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")
```

Writing packages:

```
Package:  DIS
Package:  BAS6
Package:  BCF6
Package:  SMS
Package:  OC
Package:  CLN
Package:  WEL
```

MODFLOW-USG
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER FLOW MODEL
Version 1.5.00 02/27/2019

Using NAME file: ex03d.nam

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:32

Solving:	Stress period:	1	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	2	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	3	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	4	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	5	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	6	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	7	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	8	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	9	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	10	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	11	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	12	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	13	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	14	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	15	Groundwater Flow Eqn.
Solving:	Stress period:	1	Time step:	16	Groundwater Flow Eqn.

(continues on next page)

[illegible]

(continued from previous page)

[illegible]

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 1 Time step: 121 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 122 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 123 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 124 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 125 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 126 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 127 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 128 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 129 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 130 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 131 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 132 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 133 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 134 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 135 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 136 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 137 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 138 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 139 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 140 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 141 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 142 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 143 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 144 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 145 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 146 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 147 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 148 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 149 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 150 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 151 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 152 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 153 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 154 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 155 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 156 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 157 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 158 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 159 Groundwater Flow Eqn.
Solving: Stress period: 1 Time step: 160 Groundwater Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:43
Elapsed run time: 11.224 Seconds

```

Normal termination of simulation

```

[40]: head_file = os.path.join(mf.model_ws, modelname + ".clnhd")
      headobj = flopy.utils.HeadFile(head_file)

      simtimes = headobj.get_times()
      nper = len(simtimes)
      nnode = mf.cln.nclnnds

```

(continues on next page)

(continued from previous page)

```
simhead = np.zeros((1, 1, nnode, nper))
for i in range(nper):
    simhead[:, :, :, i] = headobj.get_data(kstpkper=(i, 0))

head_case4 = np.squeeze(simhead)
```

```
[41]: cbb_file = os.path.join(mf.model_ws, modelname + ".clncb")
      cbb = flopy.utils.CellBudgetFile(cbb_file)
      # cbb.list_records()

      simflow = cbb.get_data(kstpkper=(0, 0), text="GWF")[0]
      for i in range(nper - 1):
          simflow = np.append(
              simflow, cbb.get_data(kstpkper=(i + 1, 0), text="GWF")[0]
          )

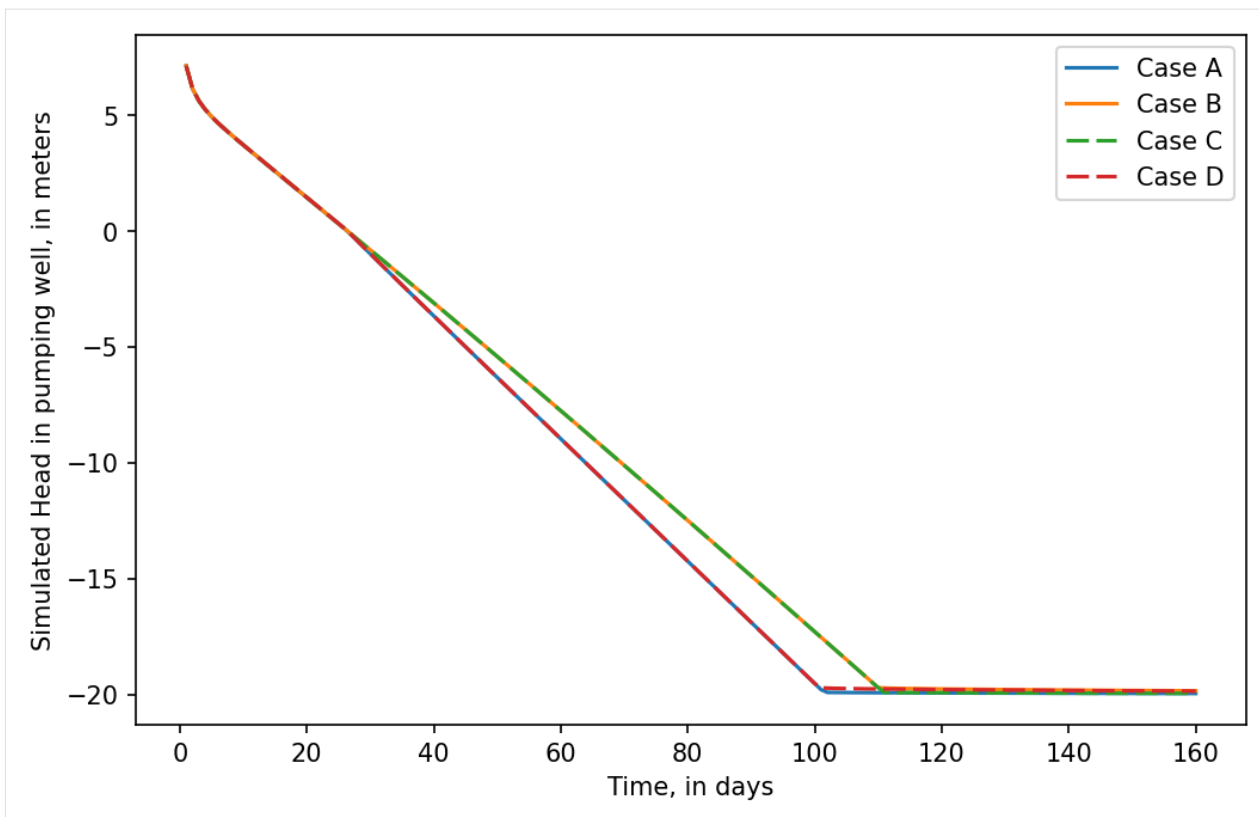
      flow_case4 = simflow
```

Comparing four cases

```
[42]: fig = plt.figure(figsize=(8, 5), dpi=150)
      ax = fig.add_subplot(111)
      ax.plot(simtimes, head_case1[1], label="Case A")
      ax.plot(simtimes, head_case2, label="Case B")
      ax.plot(simtimes, head_case3[1], dashes=[6, 2], label="Case C")
      ax.plot(simtimes, head_case4, dashes=[6, 2], label="Case D")

      ax.set_xlabel("Time, in days")
      ax.set_ylabel("Simulated Head in pumping well, in meters")
      ax.legend()

[42]: <matplotlib.legend.Legend at 0x7fa41d459070>
```



```
[43]: fig = plt.figure(figsize=(8, 11), dpi=150)
ax1 = fig.add_subplot(211)
ax1.plot(
    simtimes,
    flow_case1[:,2,]["q"],
    label="Case A",
)
ax1.plot(
    simtimes,
    flow_case2[:,2,]["q"],
    label="Case B",
)
ax1.plot(
    simtimes,
    flow_case3[:,2,]["q"],
    dashes=[6, 2],
    label="Case C",
)
ax1.plot(
    simtimes,
    flow_case4[:,2,]["q"],
    dashes=[6, 2],
    label="Case D",
)
ax1.set_xlabel("Time, in days")
ax1.set_ylabel("Layer 1 flow to well")
```

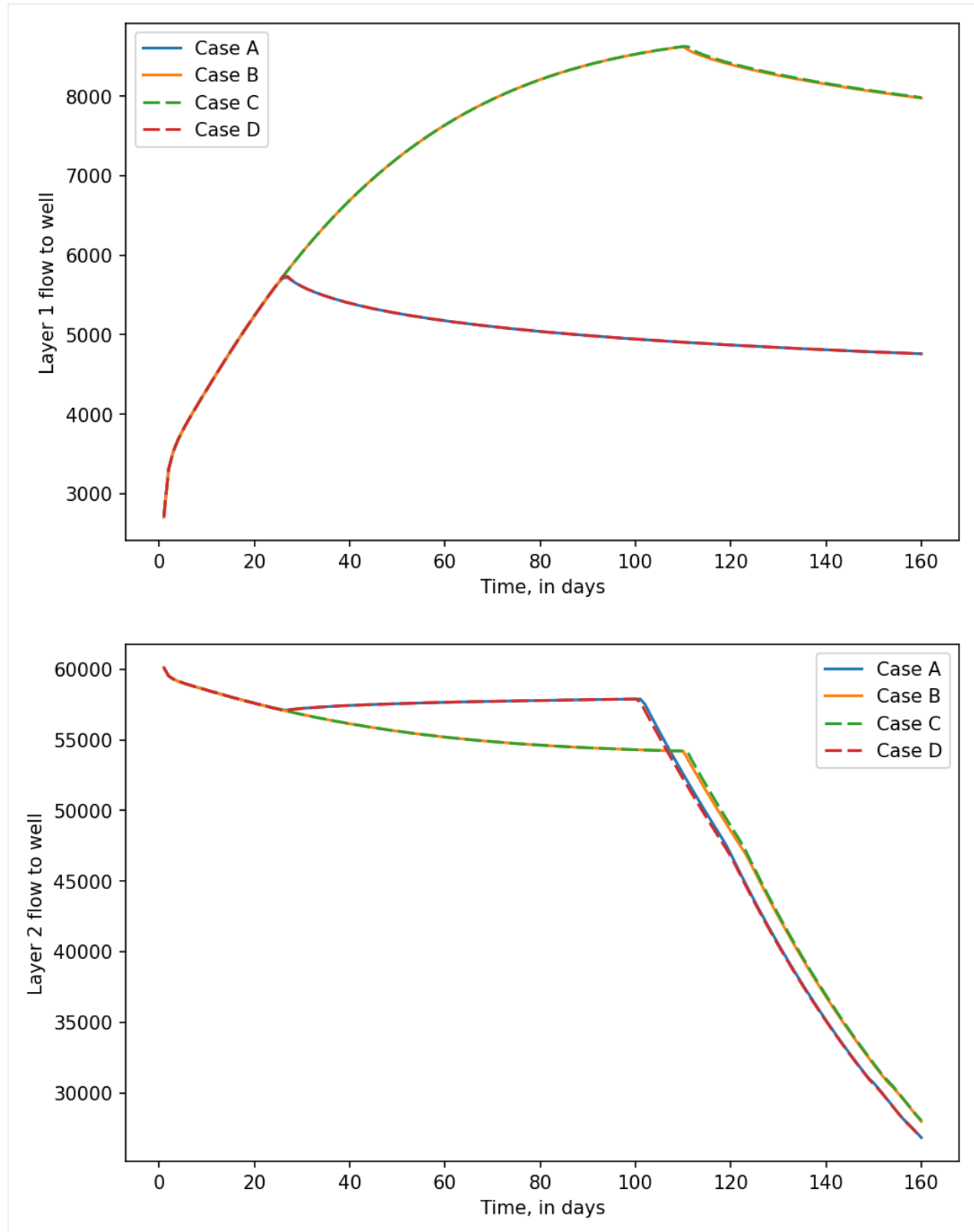
(continues on next page)

(continued from previous page)

```
ax1.legend()

ax2 = fig.add_subplot(212)
ax2.plot(
    simtimes,
    flow_case1[1::2,]["q"],
    label="Case A",
)
ax2.plot(
    simtimes,
    flow_case2[1::2,]["q"],
    label="Case B",
)
ax2.plot(
    simtimes,
    flow_case3[1::2,]["q"],
    dashes=[6, 2],
    label="Case C",
)
ax2.plot(
    simtimes,
    flow_case4[1::2,]["q"],
    dashes=[6, 2],
    label="Case D",
)
ax2.set_xlabel("Time, in days")
ax2.set_ylabel("Layer 2 flow to well")
ax2.legend()
```

```
[43]: <matplotlib.legend.Legend at 0x7fa41d4b5c70>
```

6.6.2 MODFLOW-USG Freyberg demo

This example demonstrates a MODFLOW USG Freyberg model, including construction of an `UnstructuredGrid` from a specification file and plotting head data in cross-section.

First we locate the model directory.

```
[1]: from pathlib import Path

import flopy

root_name = "freyberg.usg"
model_ws = Path.cwd().parent / "../examples/data" / root_name.replace(".", "_")
```

Now construct an `UnstructuredGrid` from a grid specification file.

```
[2]: from flopy.discretization import UnstructuredGrid

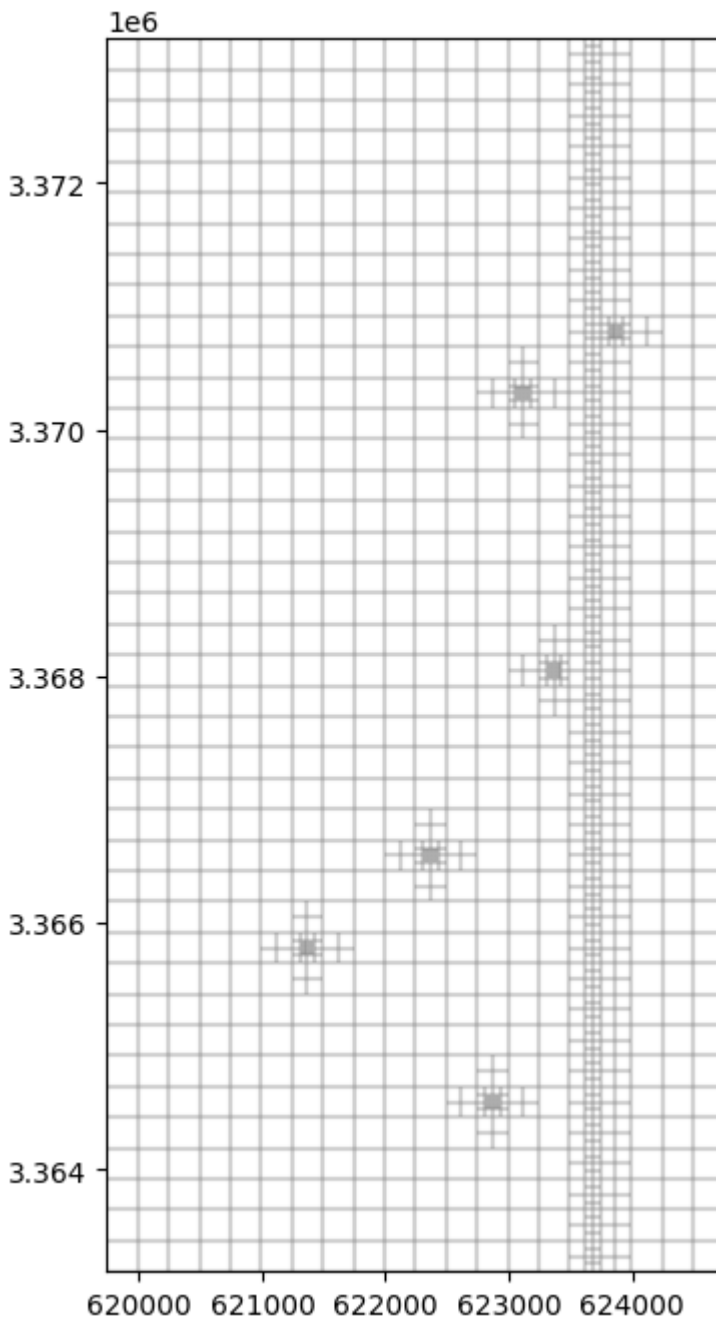
mfgrid = UnstructuredGrid.from_gridspec(str(model_ws / f"{root_name}.gsf"))
```

Plot the grid in map view.

```
[3]: import matplotlib.pyplot as plt

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
pmv = flopy.plot.PlotMapView(modelgrid=mfgrid, ax=ax)
pmv.plot_grid(alpha=0.1)

[3]: <matplotlib.collections.LineCollection at 0x7fe270d659d0>
```



Create cross-section lines.

```
[4]: from flopy.utils.geometry import LineString

lines = [
    LineString(ls)
    for ls in [
        [(623000, 3364000), (623000, 3372000)],
        [(623650, 3364000), (623650, 3372000)],
```

(continues on next page)

(continued from previous page)

```
]
]
```

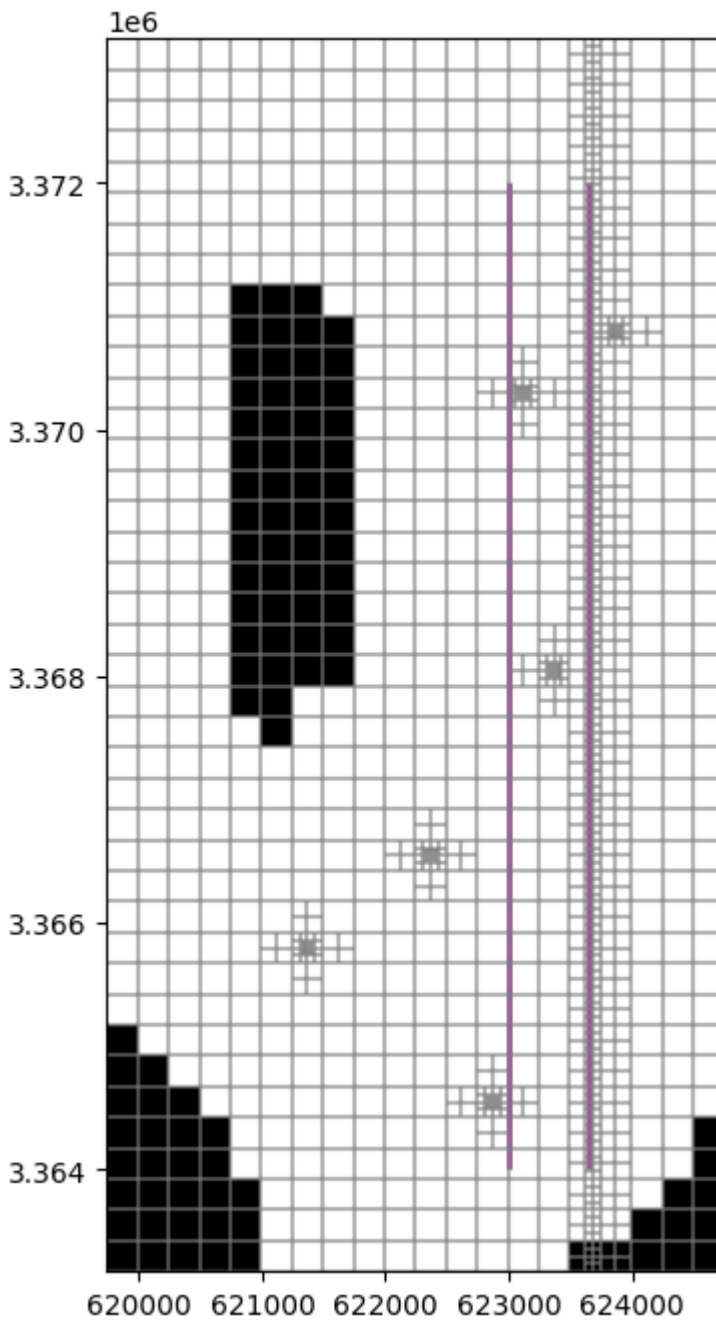
Load the model and retrieve inactive cells.

```
[5]: gwf = flopy.mfusg.MfUsg.load(
      f"{root_name}.nam",
      model_ws=str(model_ws),
      verbose=False,
      check=False,
      exe_name="mfusg",
      )

bas6 = gwf.get_package("bas6")
ibound = bas6.ibound.array
```

Show the map view again with cross-section lines and inactive cells.

```
[6]: fig = plt.figure(figsize=(8, 8))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      pmv = flopy.plot.PlotMapView(modelgrid=mfgrid, ax=ax)
      grid = pmv.plot_grid(alpha=0.2)
      shps = pmv.plot_shapes(lines, edgecolor="purple", lw=2, alpha=0.7)
      inac = pmv.plot_inactive(ibound=ibound)
```



Next, we can run the model and plot cross-sections of the resulting head. We will run the model in a temporary workspace to avoid altering the example data.

```
[7]: from tempfile import TemporaryDirectory

# temporary directory
temp_dir = TemporaryDirectory()
work_dir = Path(temp_dir.name) / "freyberg_usg"
```

(continues on next page)

(continued from previous page)

```

gwf.change_model_ws(str(work_dir))
gwf.write_name_file()
gwf.write_input()
success, buff = gwf.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

```

creating model workspace...

../../../../../tmp/tmp0esr7hlf/freyberg_usg

MODFLOW-USG

U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER FLOW MODEL

Version 1.5.00 02/27/2019

Using NAME file: freyberg.usg.nam

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:51

Solving:	Stress period:	1	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	2	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	3	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	4	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	5	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	6	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	7	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	8	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	9	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	10	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	11	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	12	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	13	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	14	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	15	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	16	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	17	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	18	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	19	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	20	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	21	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	22	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	23	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	24	Time step:	1	Groundwater Flow Eqn.
Solving:	Stress period:	25	Time step:	1	Groundwater Flow Eqn.

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:51

Elapsed run time: 0.386 Seconds

Normal termination of simulation

Load head data from the output files:

```
[8]: import numpy as np

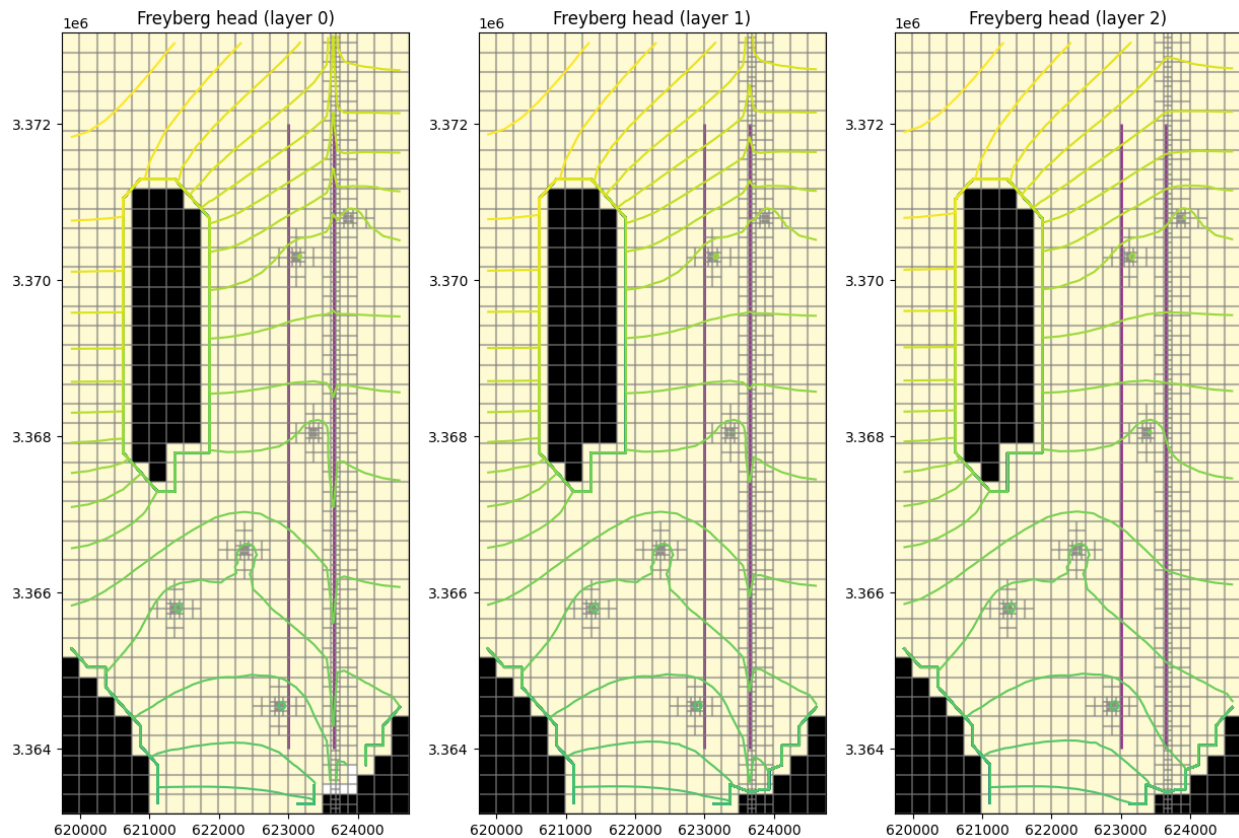
hds = flopy.utils.HeadUFile(str(work_dir / f"{root_name}.hds"), model=gwf)
times = hds.get_times()
head = np.array(hds.get_data())
print(head.shape)

(3, 1499)
```

Plot the head colormap and contours for each layer.

```
[9]: levels = np.arange(30, 35.4, 0.1)
fig = plt.figure(figsize=(15, 10))

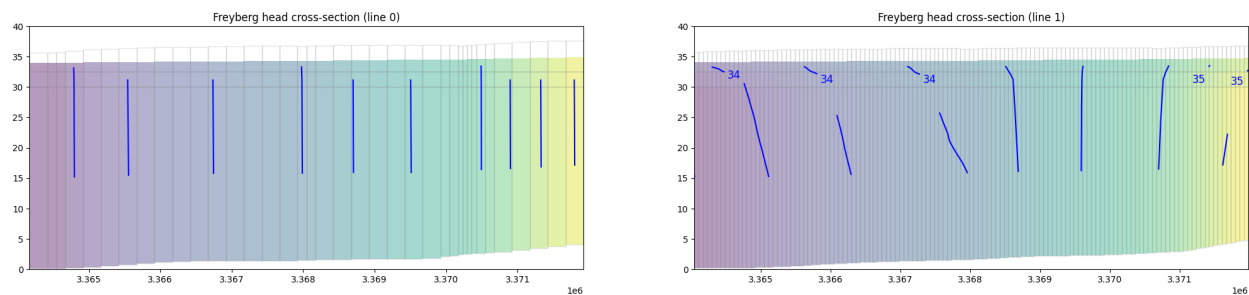
for layer, h in enumerate(head):
    ax = fig.add_subplot(1, len(head), layer + 1)
    ax.set_title(f"Freyberg head (layer {layer})")
    pmv = flopy.plot.PlotMapView(modelgrid=mgfgrid, ax=ax)
    mesh = pmv.plot_array(h, alpha=0.2)
    grid = pmv.plot_grid(alpha=0.2)
    shps = pmv.plot_shapes(lines, edgecolor="purple", lw=2, alpha=0.8)
    inac = pmv.plot_inactive(ibound=ibound)
    ctrs = pmv.contour_array(h, levels=levels)
```



A head argument can be provided to `CrossSectionPlot.contour_array()` to show the phreatic surface.

```
[10]: fig = plt.figure(figsize=(25, 5))

for i, line in enumerate(lines):
    ax = fig.add_subplot(1, len(lines), i + 1)
    ax.set_title(f"Freyberg head cross-section (line {i})")
    xssect = flopy.plot.PlotCrossSection(
        modelgrid=mfgrid,
        ax=ax,
        line={"line": lines[i]},
        geographic_coords=True,
    )
    xssect.plot_array(head, head=head, alpha=0.4)
    xssect.plot_ibound(ibound=ibound, head=head)
    xssect.plot_inactive(ibound=ibound)
    contours = xssect.contour_array(
        head,
        masked_values=[999.0],
        head=head,
        levels=levels,
        alpha=1.0,
        colors="blue",
    )
    plt.clabel(contours, fmt="%.0f", colors="blue", fontsize=12)
    xssect.plot_grid(alpha=0.2)
    ax.set_ylim([0, 40]) # set y axis range to ignore low elevations
```



The head argument can be a 1D array or a 2D array matching the shape of the grid (i.e., `head.shape == (layer count, ncpl)`).

```
[11]: line = lines[0]

for time in times[0:3]:
    head = np.array(hds.get_data(totim=time))
    head2 = np.hstack(head)

    fig = plt.figure(figsize=(25, 5))
    ax = fig.add_subplot(1, 3, 1)
    ax.set_title(f"Freyberg cross-section (t = {int(time)}, no head)")
    xssect = flopy.plot.PlotCrossSection(
        modelgrid=mfgrid, ax=ax, line={"line": line}, geographic_coords=True
    )
    cmap = xssect.plot_array(
        head2,
```

(continues on next page)

(continued from previous page)

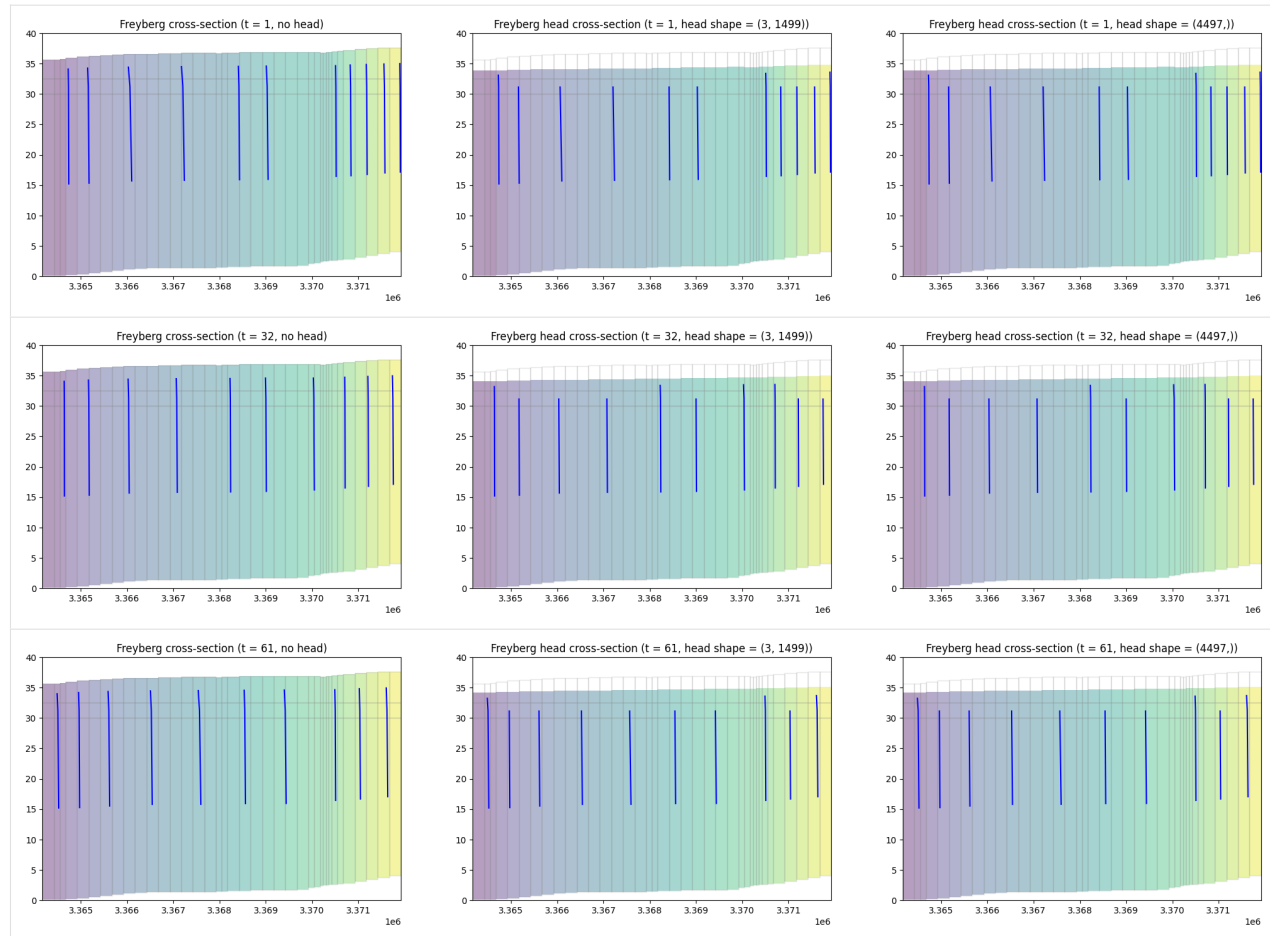
```

        masked_values=[-999.99],
        alpha=0.4,
    )
    contours = xsect.contour_array(
        head2, levels=levels, alpha=1.0, colors="blue"
    )
    xsect.plot_inactive(ibound=ibound, color_noflow=(0.8, 0.8, 0.8))
    xsect.plot_grid(alpha=0.2)
    ax.set_ylim([0, 40]) # set y axis range to ignore low elevations

    ax = fig.add_subplot(1, 3, 2)
    ax.set_title(
        f"Freyberg head cross-section (t = {int(time)}, head shape = {head.shape})"
    )
    xsect = flopy.plot.PlotCrossSection(
        modelgrid=mfgrid, ax=ax, line={"line": line}, geographic_coords=True
    )
    cmap = xsect.plot_array(
        head,
        masked_values=[-999.99],
        head=head,
        alpha=0.4,
    )
    contours = xsect.contour_array(
        head, head=head, levels=levels, alpha=1.0, colors="blue"
    )
    xsect.plot_inactive(ibound=ibound, color_noflow=(0.8, 0.8, 0.8))
    xsect.plot_grid(alpha=0.2)
    ax.set_ylim([0, 40])

    ax = fig.add_subplot(1, 3, 3)
    ax.set_title(
        f"Freyberg head cross-section (t = {int(time)}, head shape = {head2.shape})"
    )
    xsect = flopy.plot.PlotCrossSection(
        modelgrid=mfgrid, ax=ax, line={"line": line}, geographic_coords=True
    )
    cmap = xsect.plot_array(
        head2,
        masked_values=[-999.99],
        head=head2,
        alpha=0.4,
    )
    contours = xsect.contour_array(
        head2, head=head2, levels=levels, alpha=1.0, colors="blue"
    )
    xsect.plot_inactive(ibound=ibound, color_noflow=(0.8, 0.8, 0.8))
    xsect.plot_grid(alpha=0.2)
    ax.set_ylim([0, 40])

```



```
[12]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
except:
    pass
```

6.6.3 MODFLOW-USG: Discontinuous water table configuration over a stairway impervious base

One of the most challenging numerical cases for MODFLOW arises from drying-rewetting problems often associated with abrupt changes in the elevations of impervious base of a thin unconfined aquifer. This problem simulates a discontinuous water table configuration over a stairway impervious base and flow between constant-head boundaries in column 1 and 200. This problem is based on

Zaidel, J. (2013), Discontinuous Steady-State Analytical Solutions of the Boussinesq Equation and Their Numerical Representation by Modflow. Groundwater, 51: 952–959. doi: 10.1111/gwat.12019

The model consists of a grid of 200 columns, 1 row, and 1 layer; a bottom altitude of ranging from 20 to 0 m; constant heads of 23 and 5 m in column 1 and 200, respectively; and a horizontal hydraulic conductivity of 1×10^{-4} m/d. The discretization is 5 m in the row direction for all cells.

In this example results from MODFLOW-USG will be evaluated.

```
[1]: import sys
from pathlib import Path
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[2]: # Set temporary workspace and MODFLOW executable
# (assume executable is in users' path variable)
mfexe = "mfusg"
modelname = "zaidel"
temp_dir = TemporaryDirectory()
workspace = Path(temp_dir.name) / modelname
workspace.mkdir()
```

Model parameters

```
[3]: # model dimensions
nlay, nrow, ncol = 1, 1, 200
delr = 50.0
delc = 1.0

# boundary heads
h1 = 23.0
h2 = 5.0

# cell centroid locations
x = np.arange(0.0, float(ncol) * delr, delr) + delr / 2.0

# ibound
ibound = np.ones((nlay, nrow, ncol), dtype=int)
ibound[:, :, 0] = -1
ibound[:, :, -1] = -1

# bottom of the model
botm = 25 * np.ones((nlay + 1, nrow, ncol), dtype=float)
base = 20.0
```

(continues on next page)

(continued from previous page)

```
for j in range(ncol):
    botm[1, :, j] = base
    # if j > 0 and j % 40 == 0:
    if j + 1 in [40, 80, 120, 160]:
        base -= 5

# starting heads
strt = h1 * np.ones((nlay, nrow, ncol), dtype=float)
strt[:, :, -1] = h2
```

Create and run the MODFLOW-USG model

```
[4]: # make the flopy model
mf = flopy.mfusg.MfUsg(modelname=modelname, exe_name=mfexe, model_ws=workspace)
dis = flopy.modflow.ModflowDis(
    mf,
    nlay,
    nrow,
    ncol,
    delr=delr,
    delc=delc,
    top=botm[0, :, :],
    botm=botm[1:, :, :],
    perlen=1,
    nstp=1,
    steady=True,
)
bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
lpf = flopy.mfusg.MfUsgLpf(mf, hk=0.0001, laytyp=4)
oc = flopy.modflow.ModflowOc(
    mf,
    stress_period_data={
        (0, 0): ["print budget", "print head", "save head", "save budget"]
    },
)
sms = flopy.mfusg.MfUsgSms(
    mf,
    nonlinmeth=1,
    linmeth=1,
    numtrack=50,
    btol=1.1,
    breduc=0.70,
    reslim=0.0,
    theta=0.85,
    akappa=0.0001,
    gamma=0.0,
    amomentum=0.1,
    iacl=2,
    norder=0,
    level=5,
```

(continues on next page)

(continued from previous page)

```

    north=7,
    iredsys=0,
    rrctol=0.0,
    idroptol=1,
    epsrn=1.0e-5,
    mxiter=500,
    hclose=1.0e-3,
    hiclose=1.0e-3,
    iter1=50,
)
mf.write_input()

# remove any existing head files
try:
    (workspace / f"{modelname}.hds").unlink(missing_ok=True)
except:
    pass

# run the model
success, buff = mf.run_model(silent=True, report=True)
assert success, "Failed to run"
for line in buff:
    print(line)

```

```

                                MODFLOW-USG
    U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER FLOW MODEL
                                Version 1.5.00 02/27/2019

Using NAME file: zaidel.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:04

Solving:  Stress period:      1      Time step:      1      Groundwater Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:04
Elapsed run time:  0.005 Seconds

    Normal termination of simulation

```

Read the simulated MODFLOW-USG model results

```

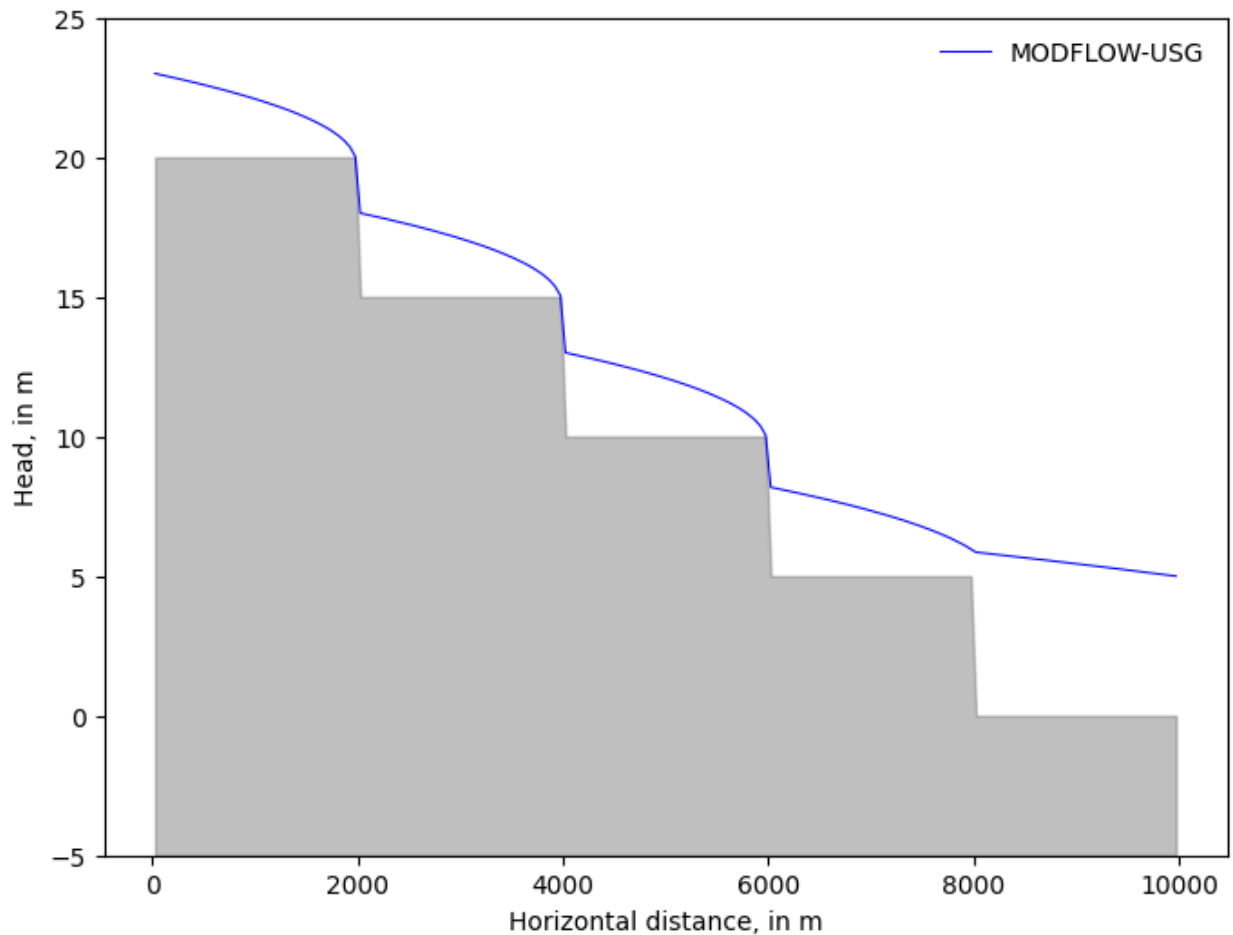
[5]: # Create the mfusg headfile object
headfile = workspace / f"{modelname}.hds"
headobj = flopy.utils.HeadFile(headfile)
times = headobj.get_times()
mfusghead = headobj.get_data(totim=times[-1])

```

Plot MODFLOW-USG results

```
[6]: fig = plt.figure(figsize=(8, 6))
fig.subplots_adjust(
    left=None, bottom=None, right=None, top=None, wspace=0.25, hspace=0.25
)
ax = fig.add_subplot(1, 1, 1)
ax.plot(
    x, mfsughead[0, 0, :], linewidth=0.75, color="blue", label="MODFLOW-USG"
)
ax.fill_between(x, y1=botm[1, 0, :], y2=-5, color="0.5", alpha=0.5)
leg = ax.legend(loc="upper right")
leg.draw_frame(False)
ax.set_xlabel("Horizontal distance, in m")
ax.set_ylabel("Head, in m")
ax.set_ylim(-5, 25)
```

```
[6]: (-5.0, 25.0)
```



```
[7]: try:
    # ignore PermissionError on Windows
    temp_dir.cleanup()
```

(continues on next page)

(continued from previous page)

```
except:
    pass
```

6.7 MODFLOW-2005/MODFLOW-NWT examples

6.7.1 Flopy Drain Return (DRT) capabilities

```
[1]: import os

[2]: import sys
    from tempfile import TemporaryDirectory

    import matplotlib as mpl
    import matplotlib.pyplot as plt
    import numpy as np

    import flopy

    print(sys.version)
    print("numpy version: {}".format(np.__version__))
    print("matplotlib version: {}".format(mpl.__version__))
    print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[3]: # temporary directory
    temp_dir = TemporaryDirectory()
    modelpth = temp_dir.name

    # creat the model package
    m = flopy.modflow.Modflow(
        "drt_test",
        model_ws=modelpth,
        exe_name="mfwt",
        version="mfwt",
    )
    d = flopy.modflow.ModflowDis(
        m,
        nlay=1,
        nrow=10,
        ncol=10,
        nper=1,
        perlen=1,
        top=10,
        botm=0,
```

(continues on next page)

(continued from previous page)

```

    steady=True,
)
b = flopy.modflow.ModflowBas(m, strt=10, ibound=1)
u = flopy.modflow.ModflowUpw(m, hk=10)
n = flopy.modflow.ModflowNwt(m)
o = flopy.modflow.ModflowOc(m)

```

```

[4]: # create the drt package
    spd = []
    for i in range(m.nrow):
        spd.append([0, i, m.ncol - 1, 5.0, 50.0, 1, 1, 1, 1.0])
    d = flopy.modflow.ModflowDrt(m, stress_period_data={0: spd})

```

```

[5]: # run the drt model
    m.write_input()
    success, buff = m.run_model(silent=True, report=True)
    if success:
        for line in buff:
            print(line)
    else:
        raise ValueError("Failed to run.")

```

```

                                MODFLOW-NWT-SWR1
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER-FLOW MODEL
                                WITH NEWTON FORMULATION
                                Version 1.3.0 07/01/2022
                                BASED ON MODFLOW-2005 Version 1.12.0 02/03/2017

                                SWR1 Version 1.05.0 03/10/2022

Using NAME file: drt_test.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:40:33

Solving: Stress period:      1      Time step:      1      Groundwater-Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:40:33
Elapsed run time:  0.002 Seconds

Normal termination of simulation

```

```

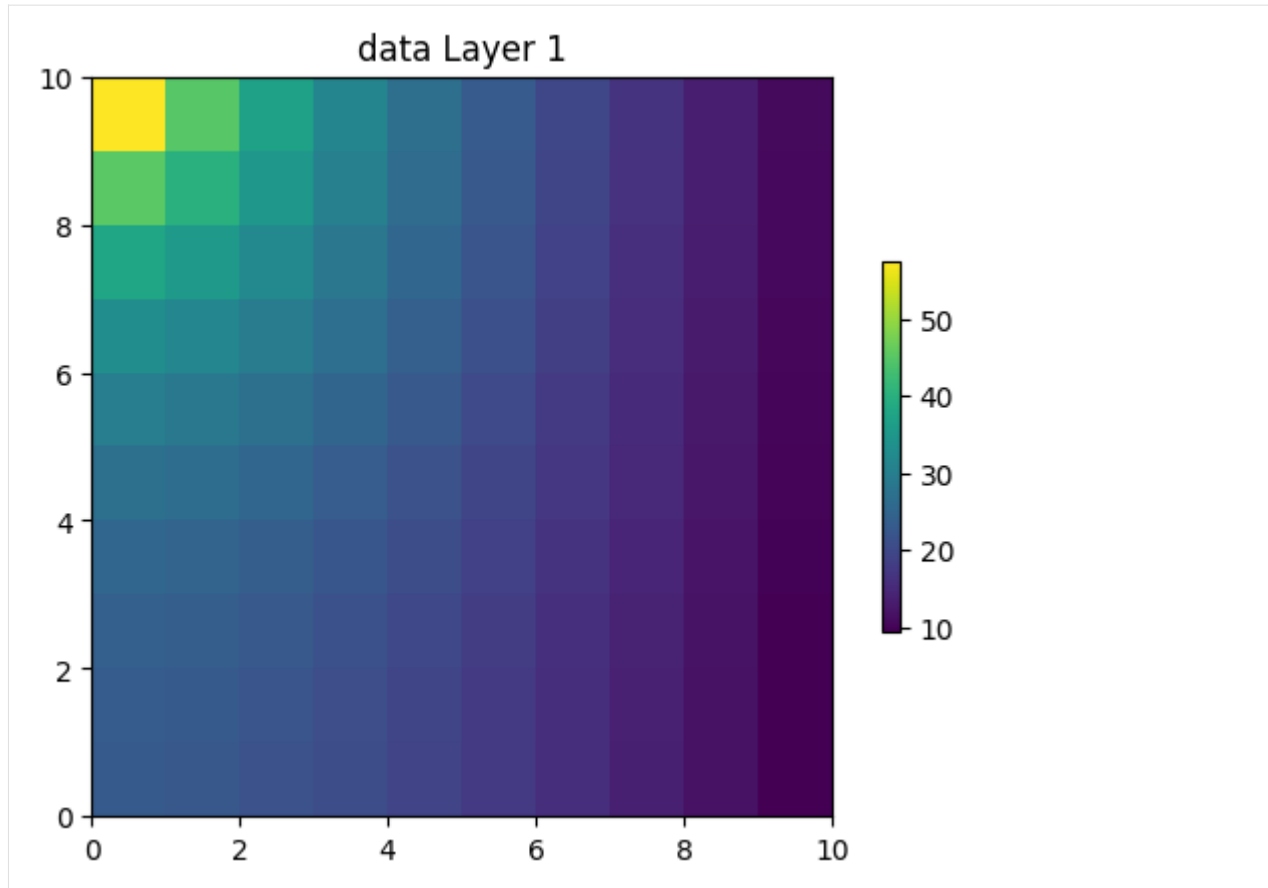
[6]: # plot heads for the drt model
    hds = flopy.utils.HeadFile(os.path.join(m.model_ws, m.name + ".hds"))
    hds.plot(colorbar=True)

```

```

[6]: <Axes: title={'center': 'data Layer 1'}>

```

```
[7]: # remove the drt package and create a standard drain file
spd = []
for i in range(m.nrow):
    spd.append([0, i, m.ncol - 1, 5.0, 1.0])
m.remove_package("DRT")
d = flopy.modflow.ModflowDrn(m, stress_period_data={0: spd})
```

```
[8]: # run the drain model
m.write_input()
success, buff = m.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")
```

```

MODFLOW-NWT-SWR1
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER-FLOW MODEL
WITH NEWTON FORMULATION
Version 1.3.0 07/01/2022
BASED ON MODFLOW-2005 Version 1.12.0 02/03/2017

SWR1 Version 1.05.0 03/10/2022

```

(continues on next page)

(continued from previous page)

```

Using NAME file: drt_test.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:40:33

Solving: Stress period:    1    Time step:    1    Groundwater-Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:40:33
Elapsed run time:  0.002 Seconds

Normal termination of simulation

```

```

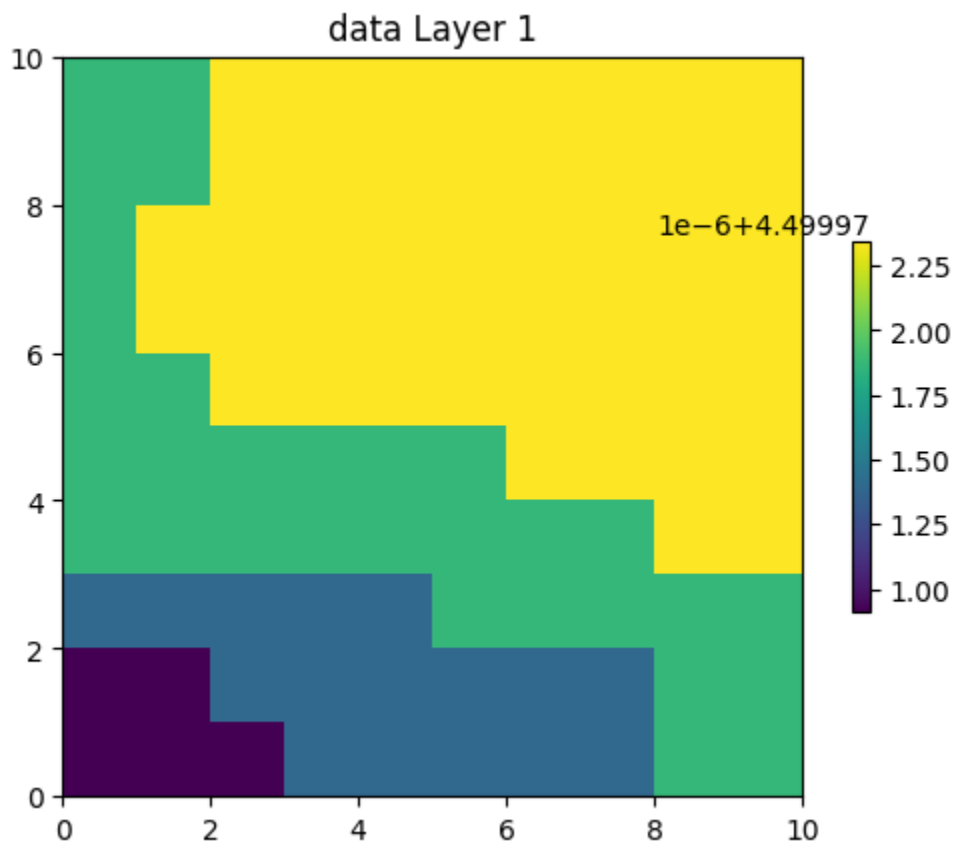
[9]: # plot the heads for the model with the drain
hds = flopy.utils.HeadFile(os.path.join(m.model_ws, m.name + ".hds"))
hds.plot(colorbar=True)

```

```

[9]: <Axes: title={'center': 'data Layer 1'}>

```



```

[10]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass

```

6.7.2 Lake Package Example

First set the path and import the required packages. The flopy path doesn't have to be set if you install flopy from a binary installer. If you want to run this notebook, you have to set the path to your own flopy path.

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

# temporary directory
temp_dir = TemporaryDirectory()
workspace = temp_dir.name

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

We are creating a square model with a specified head equal to h_1 along all boundaries. The head at the cell in the center in the top layer is fixed to h_2 . First, set the name of the model and the parameters of the model: the number of layers N_{lay} , the number of rows and columns N , lengths of the sides of the model L , aquifer thickness H , hydraulic conductivity k

```
[2]: name = "lake_example"
h1 = 100
h2 = 90
Nlay = 10
N = 101
L = 400.0
H = 50.0
k = 1.0
```

Create a MODFLOW model and store it (in this case in the variable `m1`, but you can call it whatever you want). The modelname will be the name given to all MODFLOW files (input and output). The `exe_name` should be the full path to your MODFLOW executable. The version is either 'mf2k' for MODFLOW2000 or 'mf2005' for MODFLOW2005.

```
[3]: ml = flopy.modflow.Modflow(
    modelname=name, exe_name="mf2005", version="mf2005", model_ws=workspace
)
```

Define the discretization of the model. All layers are given equal thickness. The bot array is build from the Hlay values to indicate top and bottom of each layer, and delrow and delcol are computed from model size L and number of cells N. Once these are all computed, the Discretization file is built.

```
[4]: bot = np.linspace(-H / Nlay, -H, Nlay)
delrow = delcol = L / (N - 1)
dis = flopy.modflow.ModflowDis(
    ml,
    nlay=Nlay,
    nrow=N,
    ncol=N,
    delr=delrow,
    delc=delcol,
    top=0.0,
    botm=bot,
    laycbd=0,
)
```

Next we specify the boundary conditions and starting heads with the Basic package. The ibound array will be 1 in all cells in all layers, except for along the boundary and in the cell at the center in the top layer where it is set to -1 to indicate fixed heads. The starting heads are used to define the heads in the fixed head cells (this is a steady simulation, so none of the other starting values matter). So we set the starting heads to h1 everywhere, except for the head at the center of the model in the top layer.

```
[5]: Nhalf = int((N - 1) / 2)
ibound = np.ones((Nlay, N, N), dtype=int)
ibound[:, 0, :] = -1
ibound[:, -1, :] = -1
ibound[:, :, 0] = -1
ibound[:, :, -1] = -1
ibound[0, Nhalf, Nhalf] = -1
start = h1 * np.ones((N, N))
start[Nhalf, Nhalf] = h2
bas = flopy.modflow.ModflowBas(ml, ibound=ibound, strt=start)
```

The aquifer properties (really only the hydraulic conductivity) are defined with the LPF package.

```
[6]: lpf = flopy.modflow.ModflowLpf(ml, hk=k)
```

Finally, we need to specify the solver we want to use (PCG with default values), and the output control (using the default values). Then we are ready to write all MODFLOW input files and run MODFLOW.

```
[7]: pcg = flopy.modflow.ModflowPcg(ml)
oc = flopy.modflow.ModflowOc(ml)
ml.write_input()
success, buff = ml.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
```

(continues on next page)

(continued from previous page)

```
else:
    raise ValueError("Failed to run.")
```

```

                                MODFLOW-2005
      U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
                                Version 1.12.00 2/3/2017

Using NAME file: lake_example.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:40

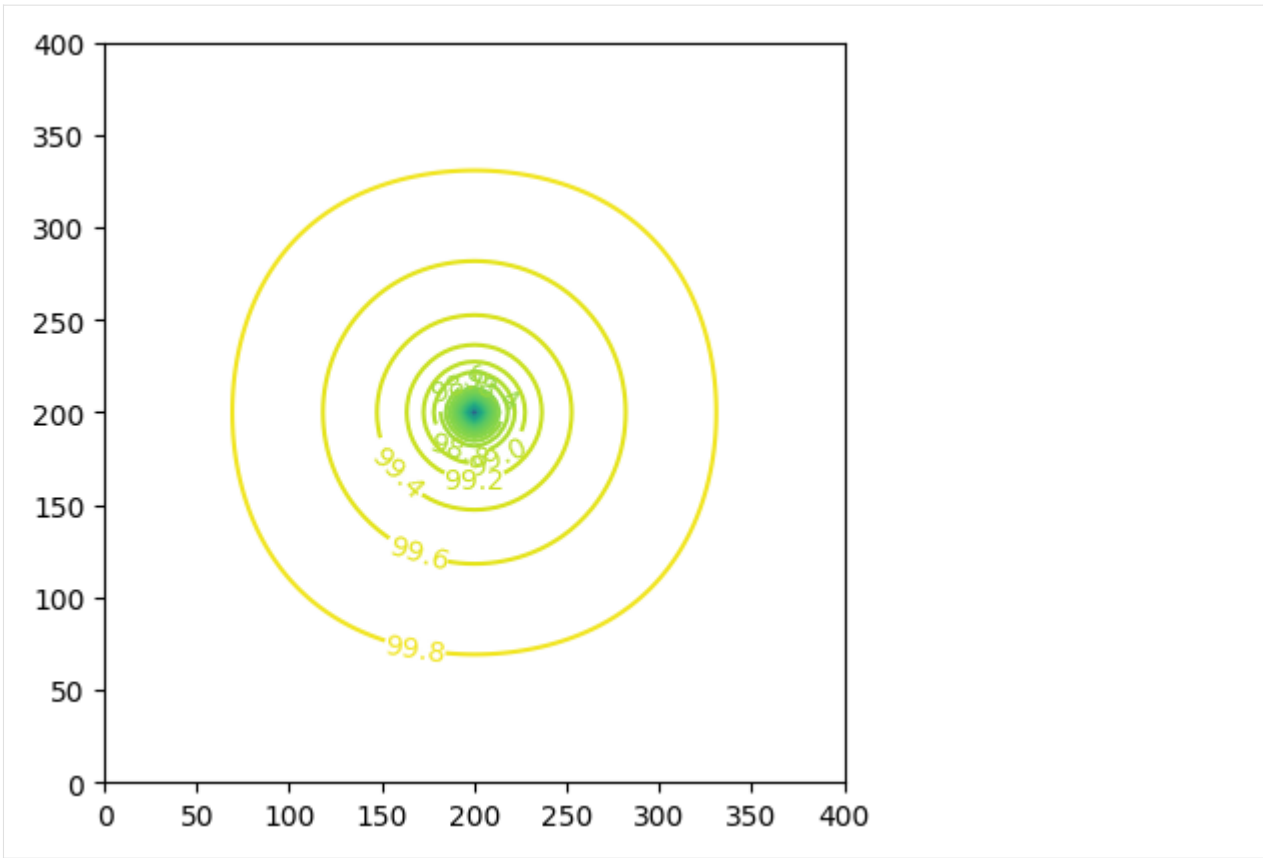
Solving:  Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:43:41
Elapsed run time:  0.383 Seconds

Normal termination of simulation
```

Once the model has terminated normally, we can read the heads file. First, a link to the heads file is created with `HeadFile`. The link can then be accessed with the `get_data` function, by specifying, in this case, the step number and period number for which we want to retrieve data. A three-dimensional array is returned of size `nlay`, `nrow`, `ncol`. Matplotlib contouring functions are used to make contours of the layers or a cross-section.

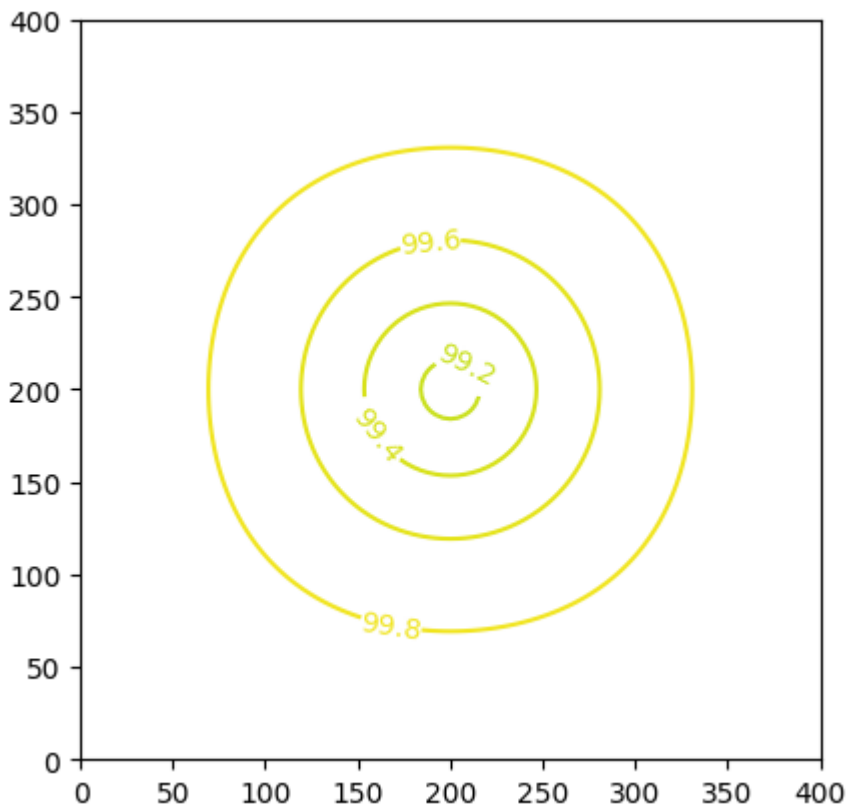
```
[8]: hds = flopy.utils.HeadFile(os.path.join(workspace, name + ".hds"))
      h = hds.get_data(kstpker=(0, 0))
      x = y = np.linspace(0, L, N)
      c = plt.contour(x, y, h[0], np.arange(90, 100.1, 0.2))
      plt.clabel(c, fmt="%2.1f")
      plt.axis("scaled")
```

```
[8]: (0.0, 400.0, 0.0, 400.0)
```



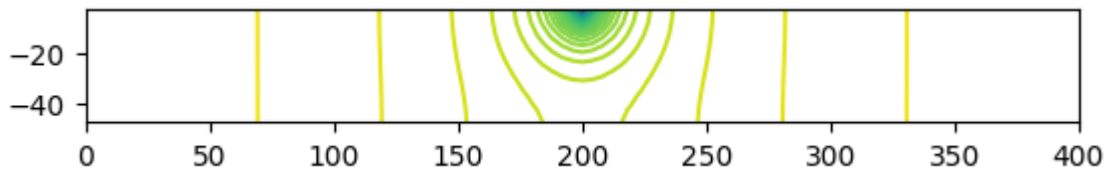
```
[9]: x = y = np.linspace(0, L, N)
      c = plt.contour(x, y, h[-1], np.arange(90, 100.1, 0.2))
      plt.clabel(c, fmt="%1.1f")
      plt.axis("scaled")
```

```
[9]: (0.0, 400.0, 0.0, 400.0)
```



```
[10]: z = np.linspace(-H / Nlay / 2, -H + H / Nlay / 2, Nlay)
      c = plt.contour(x, z, h[:, 50, :], np.arange(90, 100.1, 0.2))
      plt.axis("scaled")
```

```
[10]: (0.0, 400.0, -47.5, -2.5)
```



```
[11]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.7.3 Simple water-table solution with recharge

This problem is an unconfined system with a uniform recharge rate, a horizontal bottom, and flow between constant-head boundaries in column 1 and 100. MODFLOW models cannot match the analytical solution exactly because they do not allow recharge to constant-head cells. Constant-head cells in column 1 and 100 were made very thin (0.1 m) in the direction of flow to minimize the effect of recharge applied to them. The analytical solution for this problem can be written as:

$$h = \sqrt{b_1^2 - \frac{x}{L}(b_1^2 - b_2^2) + \left(\frac{Rx}{K}(L - x)\right)} + z_{bottom}$$

where R is the recharge rate, K is the hydraulic conductivity in the horizontal direction, b_1 is the specified saturated thickness at the left boundary, b_2 is the specified saturated thickness at the right boundary, x is the distance from the left boundary, L is the length of the model domain, and z_{bottom} is the elevation of the bottom of the aquifer.

The model consists of a grid of 100 columns, 1 row, and 1 layer; a bottom altitude of 0 m; constant heads of 20 and 11m in column 1 and 100, respectively; a recharge rate of 0.001 m/d; and a horizontal hydraulic conductivity of 50 m/d. The discretization is 0.1 m in the row direction for the constant-head cells (column 1 and 100) and 50 m for all other cells.

```
[1]: import sys
from pathlib import Path
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[2]: # Set name of MODFLOW exe
# assumes executable is in users path statement
exe_name = "mfnwt"
mfexe = exe_name
```

Create a temporary workspace.

```
[3]: temp_dir = TemporaryDirectory()
workspace = Path(temp_dir.name)
workspace.mkdir(exist_ok=True)
modelname = "watertable"
```

Define a function to calculate the analytical solution at specified points in an aquifer.


```
[4]: def analytical_water_table_solution(h1, h2, z, R, K, L, x):
    h = np.zeros((x.shape[0]), float)
    # dx = x[1] - x[0]
    # x -= dx
    b1 = h1 - z
    b2 = h2 - z
    h = (
        np.sqrt(
            b1**2 - (x / L) * (b1**2 - b2**2) + (R * x / K) * (L - x)
        )
        + z
    )
    return h
```

Define model data required to create input files and calculate the analytical solution.

```
[5]: # model dimensions
nlay, nrow, ncol = 1, 1, 100

# cell spacing
delr = 50.0
delc = 1.0

# domain length
L = 5000.0

# boundary heads
h1 = 20.0
h2 = 11.0

# ibound
ibound = np.ones((nlay, nrow, ncol), dtype=int)

# starting heads
strt = np.zeros((nlay, nrow, ncol), dtype=float)
strt[0, 0, 0] = h1
strt[0, 0, -1] = h2

# top of the aquifer
top = 25.0

# bottom of the aquifer
botm = 0.0

# hydraulic conductivity
hk = 50.0

# location of cell centroids
x = np.arange(0.0, L, delr) + (delr / 2.0)

# location of cell edges
xa = np.arange(0, L + delr, delr)
```

(continues on next page)

(continued from previous page)

```

# recharge rate
rchrate = 0.001

# calculate the head at the cell centroids using the analytical solution function
hac = analytical_water_table_solution(h1, h2, botm, rchrate, hk, L, x)

# calculate the head at the cell edges using the analytical solution function
ha = analytical_water_table_solution(h1, h2, botm, rchrate, hk, L, xa)

# ghbs
# ghb conductance
b1, b2 = 0.5 * (h1 + hac[0]), 0.5 * (h2 + hac[-1])
c1, c2 = hk * b1 * delc / (0.5 * delr), hk * b2 * delc / (0.5 * delr)
# dtype
ghb_dtype = flopy.modflow.ModflowGhb.get_default_dtype()
print(ghb_dtype)
# build ghb recarray
stress_period_data = np.zeros((2), dtype=ghb_dtype)
stress_period_data = stress_period_data.view(np.recarray)
print("stress_period_data: ", stress_period_data)
print("type is: ", type(stress_period_data))
# fill ghb recarray
stress_period_data[0] = (0, 0, 0, h1, c1)
stress_period_data[1] = (0, 0, ncol - 1, h2, c2)

[('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('bhead', '<f4'), ('cond', '<f4')]
stress_period_data: [(0, 0, 0, 0., 0.) (0, 0, 0, 0., 0.)]
type is: <class 'numpy.recarray'>

```

Create and run the MODFLOW-NWT model.

```

[6]: mf = flopy.modflow.Modflow(
    modelname=modelname, exe_name=mfexe, model_ws=workspace, version="mfnwt"
)
dis = flopy.modflow.ModflowDis(
    mf,
    nlay,
    nrow,
    ncol,
    delr=delr,
    delc=delc,
    top=top,
    botm=botm,
    perlen=1,
    nstp=1,
    steady=True,
)
bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
lpf = flopy.modflow.ModflowUpw(mf, hk=hk, laytyp=1)
ghb = flopy.modflow.ModflowGhb(mf, stress_period_data=stress_period_data)
rch = flopy.modflow.ModflowRch(mf, rech=rchrate, nrchop=1)
oc = flopy.modflow.ModflowOc(mf)
nwt = flopy.modflow.ModflowNwt(mf, linmeth=2, iprnwt=1, options="COMPLEX")

```

(continues on next page)

(continued from previous page)

```
mf.write_input()

# remove existing heads results, if necessary
try:
    (workspace / f"{modelname}.hds").unlink(missing_ok=True)
except:
    pass
# run existing model
success, buff = mf.run_model(silent=True, report=True)
assert success, "Failed to run"
for line in buff:
    print(line)
```

```

                                MODFLOW-NWT-SWR1
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER-FLOW MODEL
                                WITH NEWTON FORMULATION
                                Version 1.3.0 07/01/2022
                                BASED ON MODFLOW-2005 Version 1.12.0 02/03/2017

                                SWR1 Version 1.05.0 03/10/2022

Using NAME file: watertable.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:27

Solving:  Stress period:      1      Time step:      1      Groundwater-Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:27
Elapsed run time:  0.010 Seconds

Normal termination of simulation
```

Read the simulation's results.

```
[7]: # Create the headfile object
headfile = workspace / f"{modelname}.hds"
headobj = flopy.utils.HeadFile(headfile, precision="single")
times = headobj.get_times()
head = headobj.get_data(totim=times[-1])
```

Plot the MODFLOW-NWT results and compare to the analytical solution.

```
[8]: fig = plt.figure(figsize=(16, 6))
fig.subplots_adjust(
    left=None, bottom=None, right=None, top=None, wspace=0.25, hspace=0.25
)

ax = fig.add_subplot(1, 3, 1)
ax.plot(xa, ha, linewidth=8, color="0.5", label="analytical solution")
ax.plot(x, head[0, 0, :], color="red", label="MODFLOW-2015")
leg = ax.legend(loc="lower left")
leg.draw_frame(False)
ax.set_xlabel("Horizontal distance, in m")
ax.set_ylabel("Head, in m")
```

(continues on next page)

(continued from previous page)

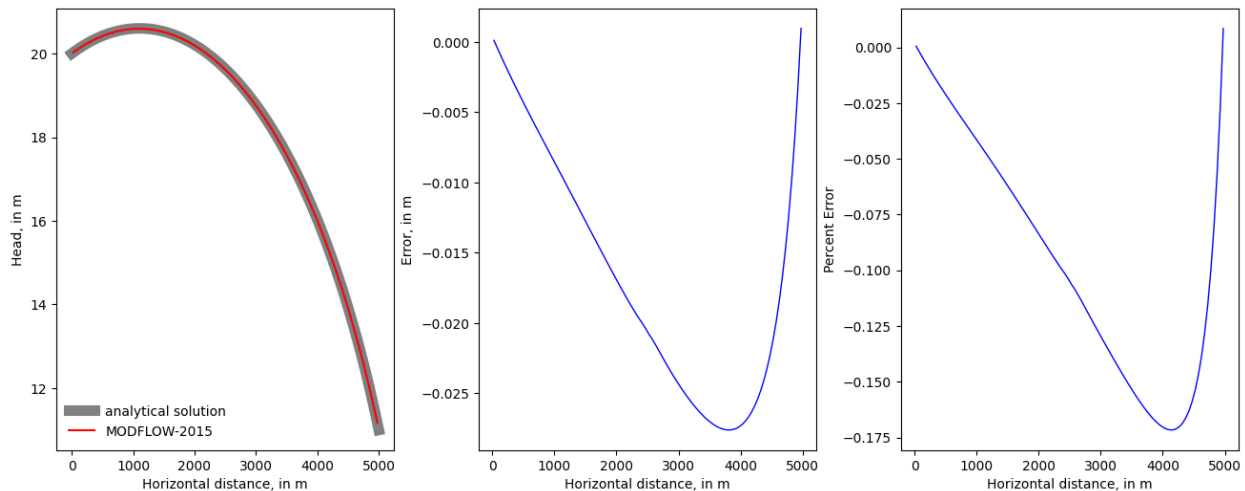
```

ax = fig.add_subplot(1, 3, 2)
ax.plot(x, head[0, 0, :] - hac, linewidth=1, color="blue")
ax.set_xlabel("Horizontal distance, in m")
ax.set_ylabel("Error, in m")

ax = fig.add_subplot(1, 3, 3)
ax.plot(x, 100.0 * (head[0, 0, :] - hac) / hac, linewidth=1, color="blue")
ax.set_xlabel("Horizontal distance, in m")
ax.set_ylabel("Percent Error")

```

[8]: `Text(0, 0.5, 'Percent Error')`



```

[9]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass

```

6.7.4 Postprocessing head results from MODFLOW

```

[1]: import os
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

import flopy
import flopy.utils.binaryfile as bf
from flopy.utils.postprocessing import (
    get_gradients,
    get_transmissivities,

```

(continues on next page)

(continued from previous page)

```

    get_water_table,
)

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3

```

```

[2]: mfnam = "EXAMPLE.nam"
    model_ws = "../examples/data/mp6/"
    heads_file = "EXAMPLE.HED"

    # temporary directory
    temp_dir = TemporaryDirectory()
    workspace = temp_dir.name

```

Load example model and head results

```

[3]: m = flopy.modflow.Modflow.load(mfnam, model_ws=model_ws)

[4]: hdsobj = bf.HeadFile(model_ws + heads_file)
    hds = hdsobj.get_data(kstpker=(0, 2))
    hds.shape

[4]: (5, 25, 25)

```

Plot heads in each layer; export the heads and head contours for viewing in a GIS

for more information about GIS export, type `help(export_array)`, for example

```

[5]: fig, axes = plt.subplots(2, 3, figsize=(11, 8.5))
    axes = axes.flat
    grid = m.modelgrid
    for i, hdslayer in enumerate(hds):
        im = axes[i].imshow(hdslayer, vmin=hds.min(), vmax=hds.max())
        axes[i].set_title("Layer {}".format(i + 1))
        ctr = axes[i].contour(hdslayer, colors="k", linewidths=0.5)

        # export head rasters
        # (GeoTiff export requires the rasterio package; for ascii grids, just change the
        ↪ extension to *.asc)
        flopy.export.utils.export_array(
            grid, os.path.join(workspace, "heads{}.tif".format(i + 1)), hdslayer
        )

```

(continues on next page)

(continued from previous page)

```

# export head contours to a shapefile
flopy.export.utils.export_array_contours(
    grid, os.path.join(workspace, "heads{}.shp".format(i + 1)), hdslayer
)

fig.delaxes(axes[-1])
fig.subplots_adjust(right=0.8)
cbar_ax = fig.add_axes([0.85, 0.15, 0.03, 0.7])
fig.colorbar(im, cax=cbar_ax, label="Head")

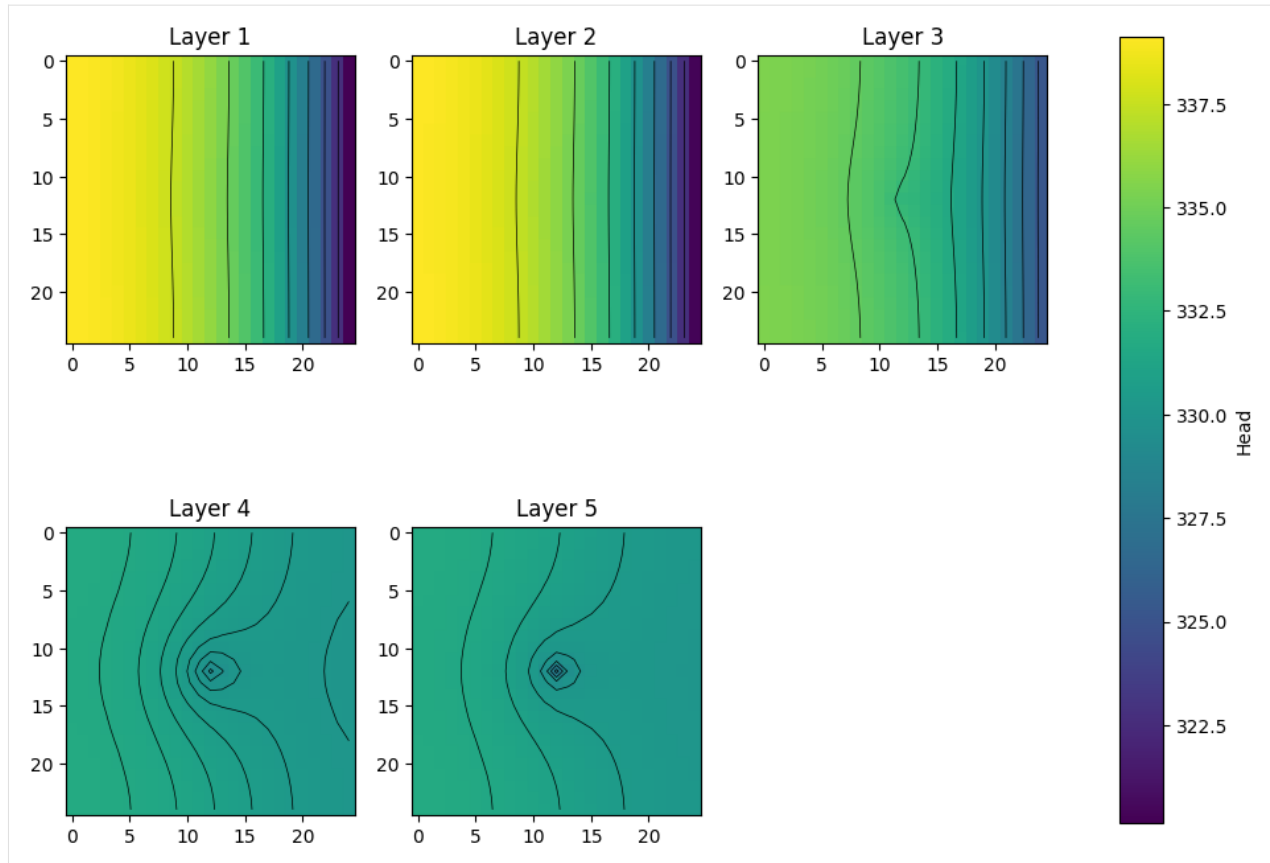
```

```

No CRS information for writing a .prj file.
Supply an valid coordinate system reference to the attached modelgrid object or .
↳export() method.
wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks
No CRS information for writing a .prj file.
Supply an valid coordinate system reference to the attached modelgrid object or .
↳export() method.
wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks
No CRS information for writing a .prj file.
Supply an valid coordinate system reference to the attached modelgrid object or .
↳export() method.
wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks
No CRS information for writing a .prj file.
Supply an valid coordinate system reference to the attached modelgrid object or .
↳export() method.
wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks

```

```
[5]: <matplotlib.colorbar.Colorbar at 0x7f5daa92de80>
```

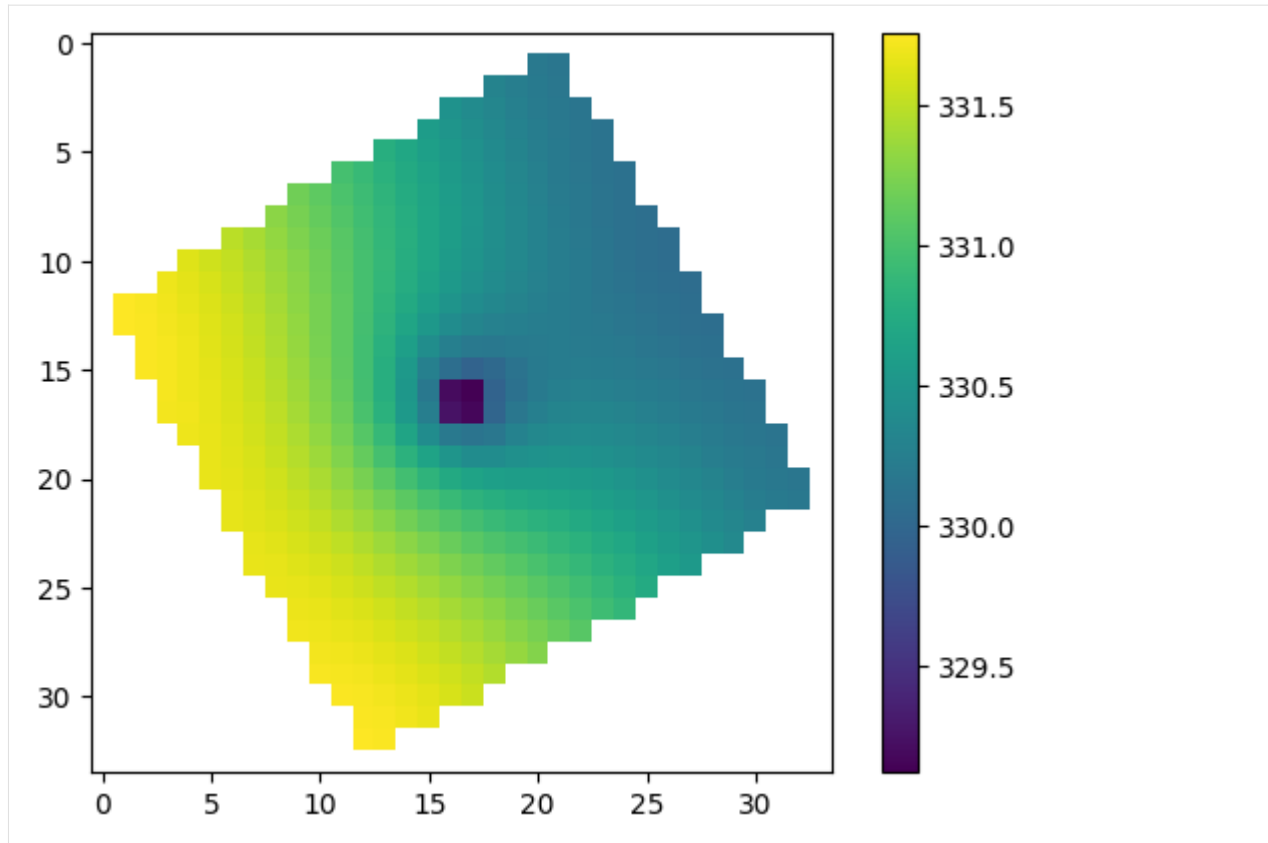


Compare rotated arc-ascii and GeoTiff output

```
[6]: grid.set_coord_info(angrot=30.0)
      nodata = 0.0
      flopy.export.utils.export_array(
          grid, os.path.join(workspace, "heads5_rot.asc"), hdslayer, nodata=nodata
      )
      flopy.export.utils.export_array(
          grid, os.path.join(workspace, "heads5_rot.tif"), hdslayer, nodata=nodata
      )

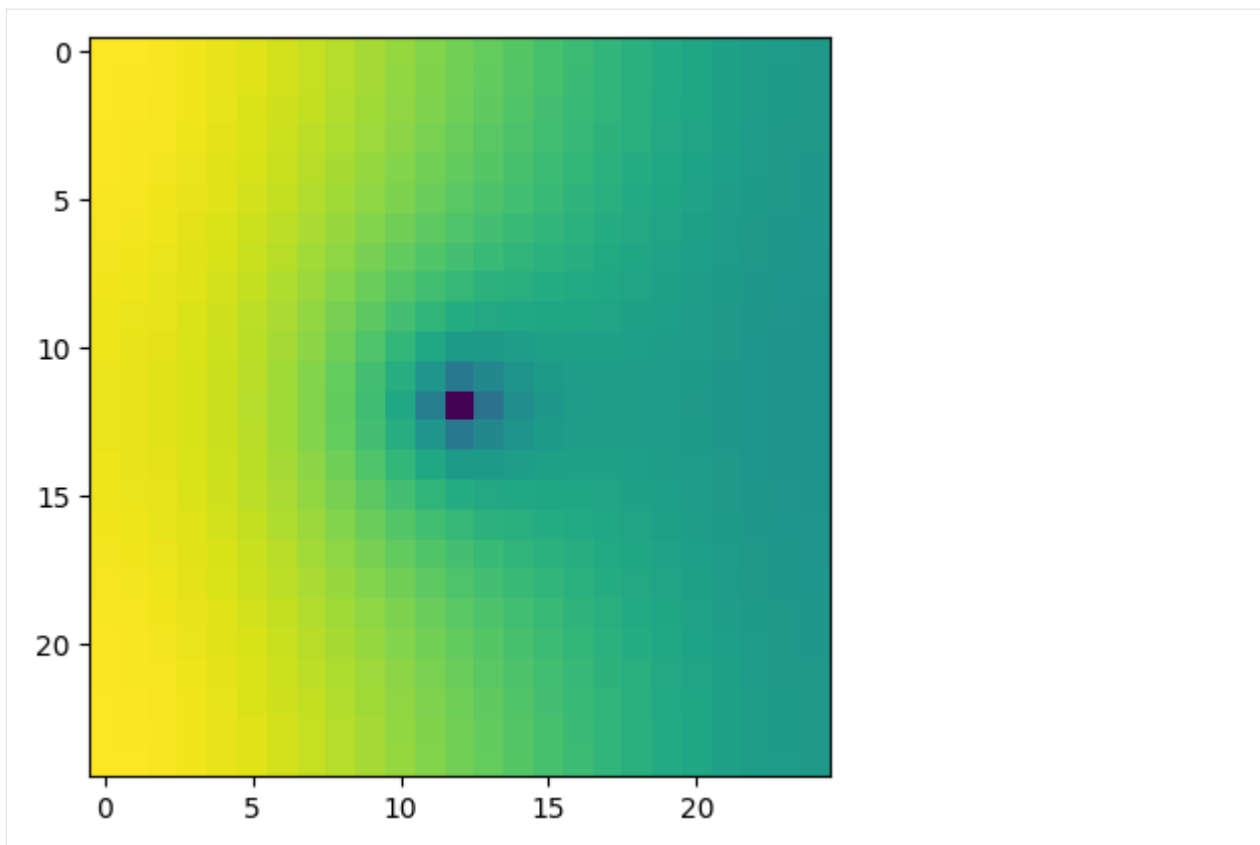
      results = np.loadtxt(
          os.path.join(workspace, "heads5_rot.asc".format(i + 1)), skiprows=6
      )
      results[results == nodata] = np.nan
      plt.imshow(results)
      plt.colorbar()
```

```
[6]: <matplotlib.colorbar.Colorbar at 0x7f5da43eaac0>
```



```
[7]: try:
      import rasterio
    except:
        rasterio = None
        print("install rasterio to create GeoTiff output")
    if rasterio is not None:
        with rasterio.open(os.path.join(workspace, "heads5_rot.tif")) as src:
            print(src.meta)
            plt.imshow(src.read(1))

{'driver': 'GTiff', 'dtype': 'float32', 'nodata': 0.0, 'width': 25, 'height': 25, 'count': 1, 'crs': None, 'transform': Affine(346.4101615137755, 199.99999999999997, -4999.999999999999, 199.99999999999997, -346.4101615137755, 8660.254037844386)}
```

Get the vertical head gradients between layers

```
[8]: grad = get_gradients(hds, m, nodata=-999)

fig, axes = plt.subplots(2, 3, figsize=(11, 8.5))
axes = axes.flat

for i, vertical_gradient in enumerate(grad):
    im = axes[i].imshow(vertical_gradient, vmin=grad.min(), vmax=grad.max())
    axes[i].set_title(
        "Vertical gradient\nbetween Layers {} and {}".format(i + 1, i + 2)
    )
    ctr = axes[i].contour(
        vertical_gradient,
        levels=[-0.1, -0.05, 0.0, 0.05, 0.1],
        colors="k",
        linewidths=0.5,
    )
    plt.clabel(ctr, fontsize=8, inline=1)

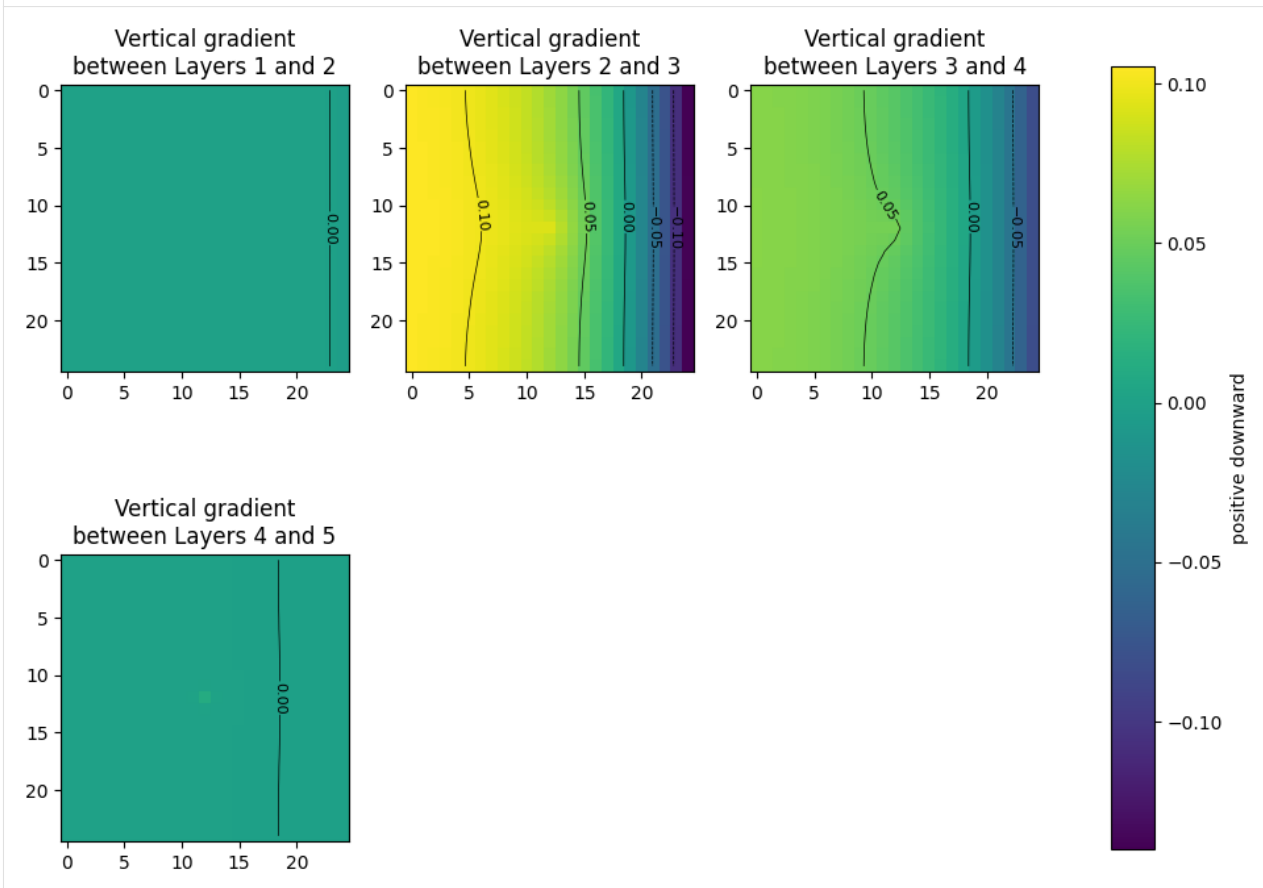
fig.delaxes(axes[-2])
fig.delaxes(axes[-1])
fig.subplots_adjust(right=0.8)
cbar_ax = fig.add_axes([0.85, 0.15, 0.03, 0.7])
```

(continues on next page)

(continued from previous page)

```
fig.colorbar(im, cax=cbar_ax, label="positive downward")
```

```
[8]: <matplotlib.colorbar.Colorbar at 0x7f5da40f9c40>
```



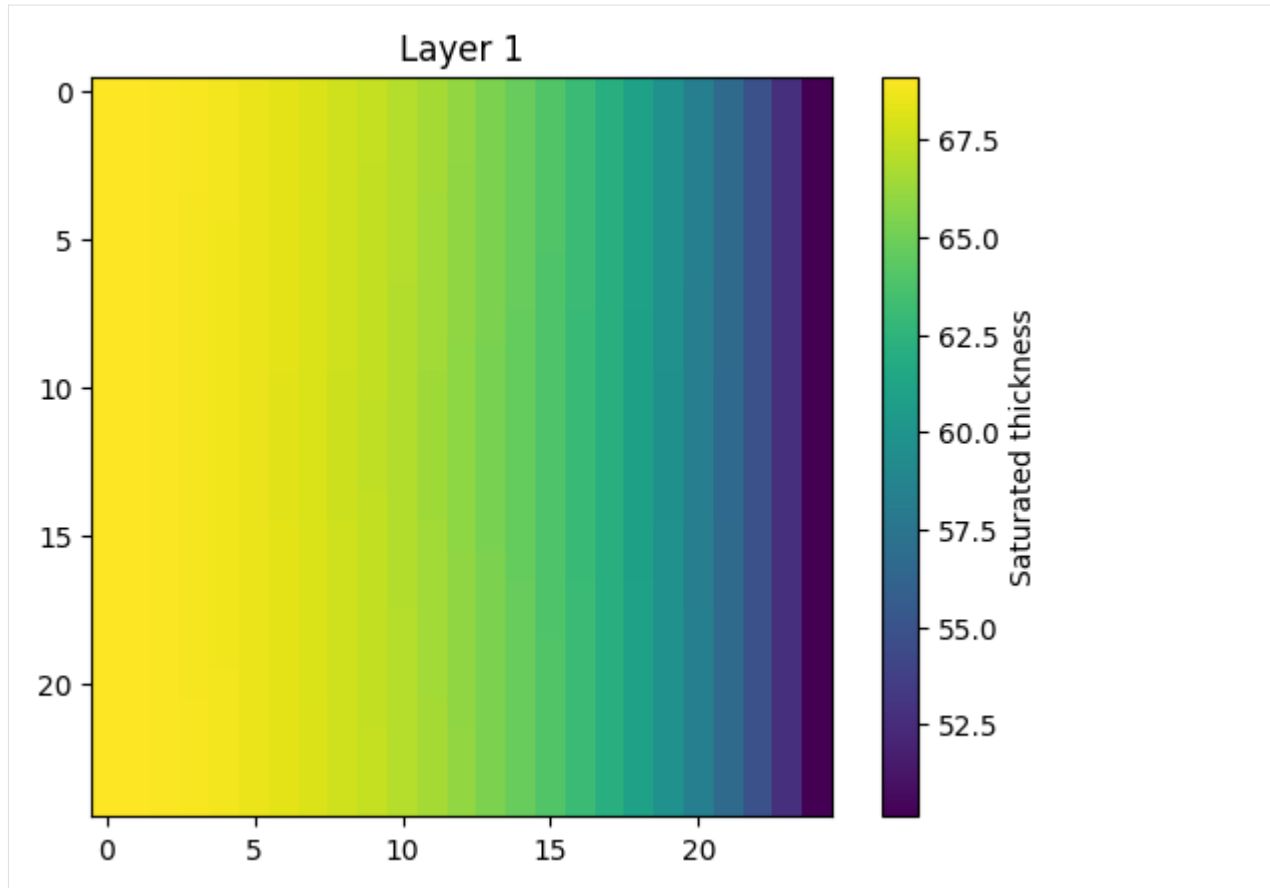
Get the saturated thickness of a layer

`m.modelgrid.saturated_thickness()` returns an `nlay, nrow, ncol` array of saturated thicknesses.

```
[9]: st = m.modelgrid.saturated_thickness(hds, mask=-9999.0)
```

```
plt.imshow(st[0])
plt.colorbar(label="Saturated thickness")
plt.title("Layer 1")
```

```
[9]: Text(0.5, 1.0, 'Layer 1')
```



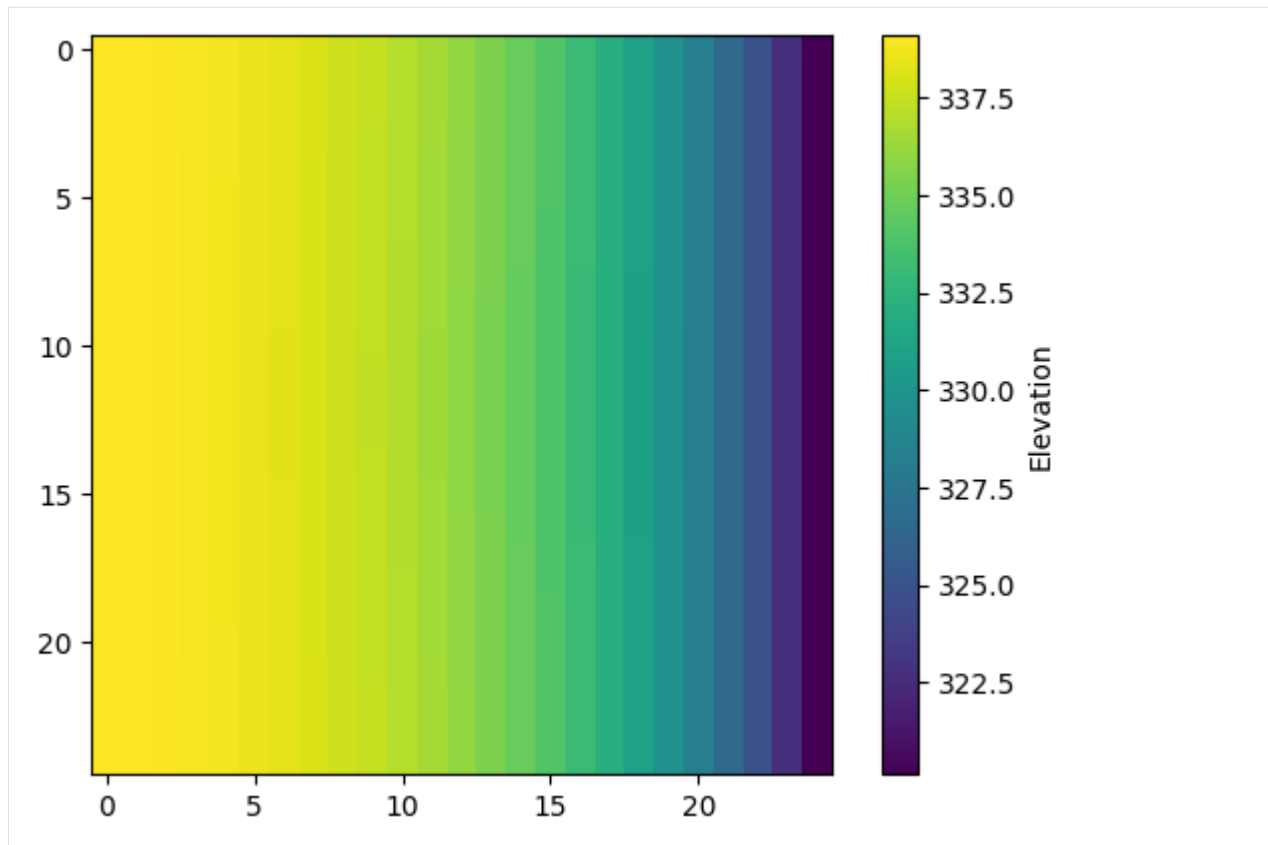
Get the water table

`get_water_table()` returns an `nrow, ncol` array of the water table elevation. This method can be useful when HDRY is turned on and the water table is in multiple layers.

```
[10]: wt = get_water_table(heads=hds, hdry=-9999)

plt.imshow(wt)
plt.colorbar(label="Elevation")

[10]: <matplotlib.colorbar.Colorbar at 0x7f5da435ceb0>
```



Get layer transmissivities at arbitrary locations, accounting for the position of the water table

- for this method, the heads input is an nlay x nobs array of head results, which could be constructed using the Hydmod package with an observation in each layer at each observation location, for example .
- x, y values in real-world coordinates can be used in lieu of row, column, provided a correct coordinate information is supplied to the flopy model object's grid.
- open interval tops and bottoms can be supplied at each location for computing transmissivity-weighted average heads
- this method can also be used for apportioning boundary fluxes for an inset from a 2-D regional model
- see `**flopy3_get_transmissivities_example.ipynb**` for more details on how this method works

```
[11]: r = [20, 5]
      c = [5, 20]
      headresults = hds[:, r, c]
      get_transmissivities(headresults, m, r=r, c=c)
```

```
[11]: array([[3.42867432e+03, 2.91529083e+03],
            [2.50000000e+03, 2.50000000e+03],
            [1.99999996e-01, 1.99999996e-01],
            [2.00000000e+04, 2.00000000e+04],
            [2.00000000e+04, 2.00000000e+04]])
```

```
[12]: r = [20, 5]
      c = [5, 20]
      sctop = [340, 320] # top of open interval at each location
      scbot = [210, 150] # top of bottom interval at each location
      headresults = hds[:, r, c]
      tr = get_transmissivities(headresults, m, r=r, c=c, sctop=sctop, scbot=scbot)
      tr

[12]: array([[3.42867432e+03, 2.50000000e+03],
            [2.50000000e+03, 2.50000000e+03],
            [9.99999978e-02, 1.99999996e-01],
            [0.00000000e+00, 1.00000000e+04],
            [0.00000000e+00, 0.00000000e+00]])
```

convert to transmissivity fractions

```
[13]: trfrac = tr / tr.sum(axis=0)
      trfrac

[13]: array([[5.78310817e-01, 1.66664444e-01],
            [4.21672316e-01, 1.66664444e-01],
            [1.68668923e-05, 1.33331553e-05],
            [0.00000000e+00, 6.6657778e-01],
            [0.00000000e+00, 0.00000000e+00]])
```

Layer 3 contributes almost no transmissivity because of its K-value

```
[14]: m.lpf.hk.array[:, r, c]

[14]: array([[5.e+01, 5.e+01],
            [5.e+01, 5.e+01],
            [1.e-02, 1.e-02],
            [2.e+02, 2.e+02],
            [2.e+02, 2.e+02]], dtype=float32)
```

```
[15]: m.modelgrid.cell_thickness[:, r, c]

[15]: array([[130., 130.],
            [ 50.,  50.],
            [ 20.,  20.],
            [100., 100.],
            [100., 100.]])
```

```
[16]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.7.5 SFR package Prudic and others (2004) example

Demonstrates functionality of FloPy SFR module using the example documented by [Prudic and others \(2004\)](#):

Problem description:

- Grid dimensions: 1 Layer, 15 Rows, 10 Columns
- Stress periods: 1 steady
- Flow package: LPF
- Stress packages: SFR, GHB, EVT, RCH
- Solver: SIP

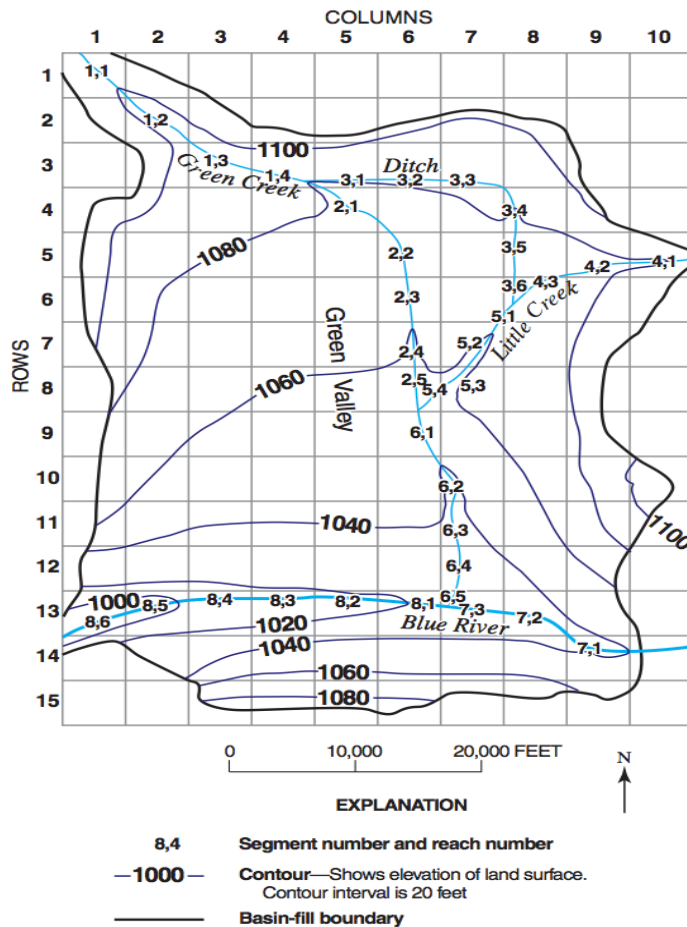


Figure 6. Hypothetical basin-fill aquifer with model grid, land-surface contours, and stream segment and reach numbering scheme for test simulation 1.

```
[1]: import glob
import os
import shutil
```

```
[2]: import sys
from tempfile import TemporaryDirectory
```

(continues on next page)

(continued from previous page)

```

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

import flopy
import flopy.utils.binaryfile as bf
from flopy.utils.sfroutputfile import SfrFile

mpl.rcParams["figure.figsize"] = (11, 8.5)

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("pandas version: {}".format(pd.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
pandas version: 2.0.3
matplotlib version: 3.7.3
flopy version: 3.4.3

```

```

[3]: # Set name of MODFLOW exe
# assumes executable is in users path statement
exe_name = "mf2005"

```

copy over the example files to the working directory

```

[4]: # temporary directory
temp_dir = TemporaryDirectory()
path = temp_dir.name

gpth = os.path.join("../", "../", "examples", "data", "mf2005_test", "test1ss.*")
for f in glob.glob(gpth):
    shutil.copy(f, path)
gpth = os.path.join("../", "../", "examples", "data", "mf2005_test", "test1tr.*")
for f in glob.glob(gpth):
    shutil.copy(f, path)

```

Load example dataset, skipping the SFR package

```
[5]: m = flopy.modflow.Modflow.load(
    "test1ss.nam",
    version="mf2005",
    exe_name=exe_name,
    model_ws=path,
    load_only=["ghb", "evt", "rch", "dis", "bas6", "oc", "sip", "lpf"],
)
```

```
[6]: oc = m.oc
    oc.stress_period_data
```

```
[6]: {(-1, -1): [],
      (0, 0): ['print budget', 'print head', 'save head', 'save budget']}
```

Read pre-prepared reach and segment data into numpy recarrays using numpy.genfromtxt()

Reach data (Item 2 in the SFR input instructions), are input and stored in a numpy record array <https://numpy.org/doc/stable/reference/generated/numpy.recarray.html>. This allows for reach data to be indexed by their variable names, as described in the SFR input instructions.

For more information on Item 2, see the Online Guide to MODFLOW: <https://water.usgs.gov/nrp/gwsoftware/modflow2000/MFDOC/sfr.html>

```
[7]: rpth = os.path.join(
    "..", "..", "examples", "data", "sfr_examples", "test1ss_reach_data.csv"
)
reach_data = np.genfromtxt(rpth, delimiter=",", names=True)
reach_data
```

```
[7]: array([(0., 0., 0., 1., 1., 4500.), (0., 1., 1., 1., 2., 7000.),
          (0., 2., 2., 1., 3., 6000.), (0., 2., 3., 1., 4., 5550.),
          (0., 3., 4., 2., 1., 6500.), (0., 4., 5., 2., 2., 5000.),
          (0., 5., 5., 2., 3., 5000.), (0., 6., 5., 2., 4., 5000.),
          (0., 7., 5., 2., 5., 5000.), (0., 2., 4., 3., 1., 5000.),
          (0., 2., 5., 3., 2., 5000.), (0., 2., 6., 3., 3., 4500.),
          (0., 3., 7., 3., 4., 6000.), (0., 4., 7., 3., 5., 5000.),
          (0., 5., 7., 3., 6., 2000.), (0., 4., 9., 4., 1., 2500.),
          (0., 4., 8., 4., 2., 5000.), (0., 5., 7., 4., 3., 3500.),
          (0., 5., 7., 5., 1., 4000.), (0., 6., 6., 5., 2., 5000.),
          (0., 7., 6., 5., 3., 3500.), (0., 7., 5., 5., 4., 2500.),
          (0., 8., 5., 6., 1., 5000.), (0., 9., 6., 6., 2., 5000.),
          (0., 10., 6., 6., 3., 5000.), (0., 11., 6., 6., 4., 5000.),
          (0., 12., 6., 6., 5., 2000.), (0., 13., 8., 7., 1., 5000.),
          (0., 12., 7., 7., 2., 5500.), (0., 12., 6., 7., 3., 5000.),
          (0., 12., 5., 8., 1., 5000.), (0., 12., 4., 8., 2., 5000.),
          (0., 12., 3., 8., 3., 5000.), (0., 12., 2., 8., 4., 5000.),
          (0., 12., 1., 8., 5., 5000.), (0., 12., 0., 8., 6., 3000.)],
      dtype=[('k', '<f8'), ('i', '<f8'), ('j', '<f8'), ('iseg', '<f8'), ('ireach', '<f8'
→), ('rchlen', '<f8')])
```


Segment Data structure

Segment data are input and stored in a dictionary of record arrays, which

```
[8]: spth = os.path.join(
    "..", "..", "examples", "data", "sfr_examples", "test1ss_segment_data.csv"
)
ss_segment_data = np.genfromtxt(spth, delimiter=",", names=True)
segment_data = {0: ss_segment_data}
segment_data[0][0:1]["width1"]
[8]: array([0.])
```

define dataset 6e (channel flow data) for segment 1

dataset 6e is stored in a nested dictionary keyed by stress period and segment, with a list of the following lists defined for each segment with icalc == 4 FLOWTAB(1) FLOWTAB(2) ... FLOWTAB(NSTRPTS) DPTHTAB(1) DPTHTAB(2) ... DPTHTAB(NSTRPTS) WDTHTAB(1) WDTHTAB(2) ... WDTHTAB(NSTRPTS)

```
[9]: channel_flow_data = {
    0: {
        1: [
            [0.5, 1.0, 2.0, 4.0, 7.0, 10.0, 20.0, 30.0, 50.0, 75.0, 100.0],
            [0.25, 0.4, 0.55, 0.7, 0.8, 0.9, 1.1, 1.25, 1.4, 1.7, 2.6],
            [3.0, 3.5, 4.2, 5.3, 7.0, 8.5, 12.0, 14.0, 17.0, 20.0, 22.0],
        ]
    }
}
```

define dataset 6d (channel geometry data) for segments 7 and 8

dataset 6d is stored in a nested dictionary keyed by stress period and segment, with a list of the following lists defined for each segment with icalc == 4 FLOWTAB(1) FLOWTAB(2) ... FLOWTAB(NSTRPTS) DPTHTAB(1) DPTHTAB(2) ... DPTHTAB(NSTRPTS) WDTHTAB(1) WDTHTAB(2) ... WDTHTAB(NSTRPTS)

```
[10]: channel_geometry_data = {
    0: {
        7: [
            [0.0, 10.0, 80.0, 100.0, 150.0, 170.0, 240.0, 250.0],
            [20.0, 13.0, 10.0, 2.0, 0.0, 10.0, 13.0, 20.0],
        ],
        8: [
            [0.0, 10.0, 80.0, 100.0, 150.0, 170.0, 240.0, 250.0],
            [25.0, 17.0, 13.0, 4.0, 0.0, 10.0, 16.0, 20.0],
        ],
    }
}
```

Define SFR package variables

```
[11]: nstrm = len(reach_data) # number of reaches
      nss = len(segment_data[0]) # number of segments
      nsfrpar = 0 # number of parameters (not supported)
      nparseg = 0
      const = 1.486 # constant for manning's equation, units of cfs
      dleak = 0.0001 # closure tolerance for stream stage computation
      ipakcb = 53 # flag for writing SFR output to cell-by-cell budget (on unit 53)
      istcb2 = 81 # flag for writing SFR output to text file
      dataset_5 = {0: [nss, 0, 0]} # dataset 5 (see online guide)
```

Instantiate SFR package

Input arguments generally follow the variable names defined in the Online Guide to MODFLOW

```
[12]: sfr = flopy.modflow.ModflowSfr2(
      m,
      nstrm=nstrm,
      nss=nss,
      const=const,
      dleak=dleak,
      ipakcb=ipakcb,
      istcb2=istcb2,
      reach_data=reach_data,
      segment_data=segment_data,
      channel_geometry_data=channel_geometry_data,
      channel_flow_data=channel_flow_data,
      dataset_5=dataset_5,
      unit_number=15,
      )
```

```
[13]: sfr.reach_data[0:1]
```

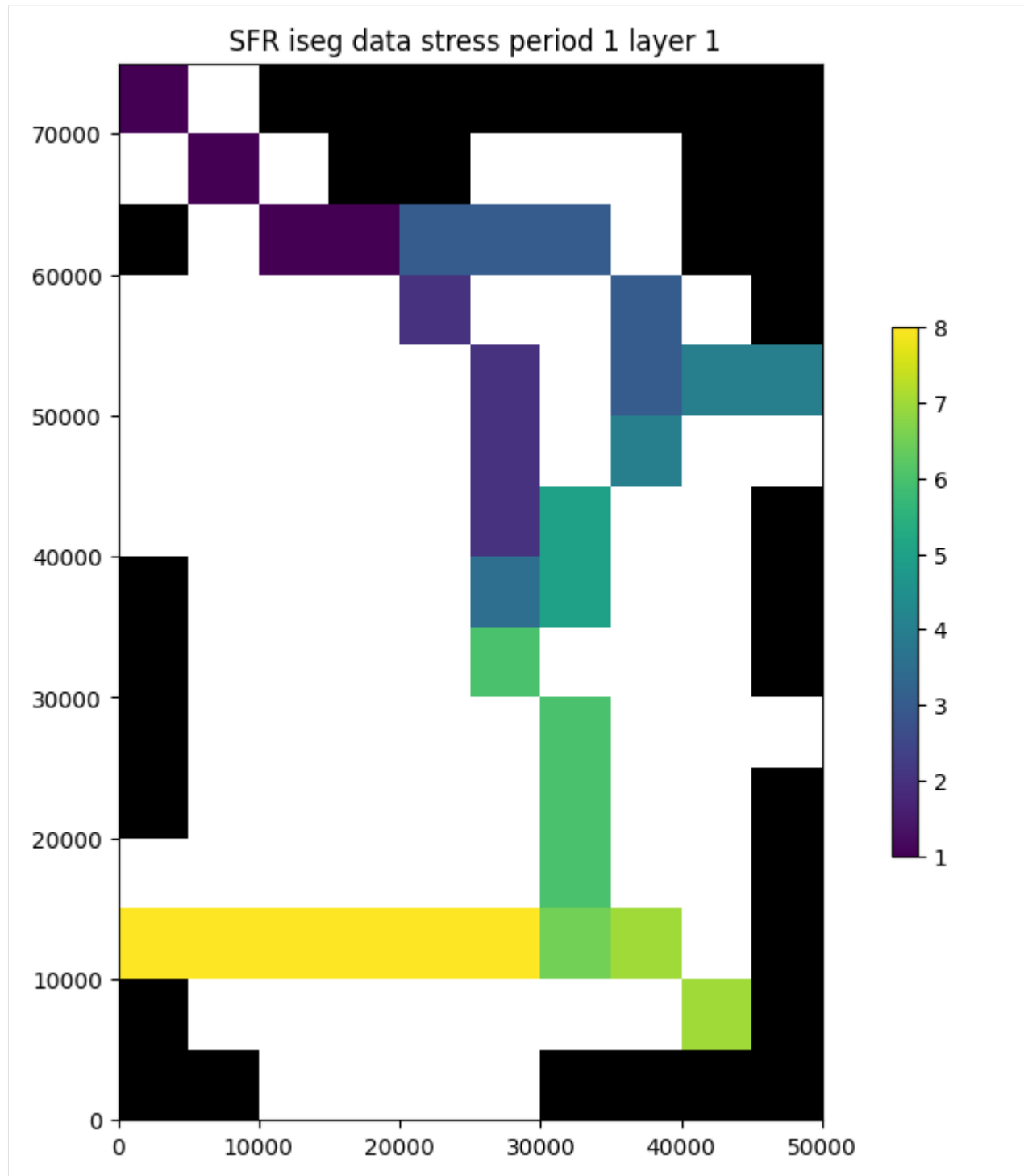
```
[13]: rec.array([(0, 0, 0, 0, 1, 1, 4500., 0., 0., 0., 0., 0., 0., 1, 2)],
      dtype=[('node', '<i8'), ('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('iseg', '<i8'
      ↪'), ('ireach', '<i8'), ('rchlen', '<f4'), ('strtop', '<f4'), ('slope', '<f4'), (
      ↪), ('strthick', '<f4'), ('strhc1', '<f4'), ('thts', '<f4'), ('thti', '<f4'), ('eps', '<f4'
      ↪'), ('uhc', '<f4'), ('reachID', '<i8'), ('outreach', '<i8')])]
```

Plot the SFR segments

any column in the reach_data array can be plotted using the key argument

```
[14]: sfr.plot(key="iseg")
```

```
[14]: [<Axes: title={'center': ' SFR iseg data stress period 1 layer 1'}>]
```



Check the SFR dataset for errors

```
[15]: chk = sfr.check()

passed.

Checking for continuity in segment and reach numbering...
passed.

Checking for increasing segment numbers in downstream direction...
passed.

Checking for circular routing...
passed.

Checking reach connections for proximity...
0 segments with non-adjacent reaches found.
At segments:

0 segments with non-adjacent reaches found.
At segments:

Checking for model cells with multiple non-zero SFR conductances...
3 model cells with multiple non-zero SFR conductances found.
This may lead to circular routing between collocated reaches.
Nodes with overlapping conductances:
k      i      j      iseg   ireach  rchlen  strthick      strhc1
0       7      5       2      5      5000.0  3.0      2.9999999242136255e-05
0       5      7       3      6      2000.0  2.0      2.9999999242136255e-05
0       5      7       4      3      3500.0  3.0      2.9999999242136255e-05
0       5      7       5      1      4000.0  3.0      2.9999999242136255e-05
0       7      5       5      4      2500.0  3.0      2.9999999242136255e-05
0      12      6       6      5      2000.0  3.0      2.9999999242136255e-05
0      12      6       7      3      5000.0  3.0      5.999999848427251e-05

Checking for streambed tops of less than -10...
strtop not specified for isfropt=0
passed.

Checking for streambed tops of greater than 15000...
strtop not specified for isfropt=0
passed.

Checking segment_data for downstream rises in streambed elevation...
passed.

Checking reach_data for downstream rises in streambed elevation...
Reach strtop not specified for nstrm=36, reachinput=False and isfropt=0
passed.
```

(continues on next page)

(continued from previous page)

```

Checking reach_data for inconsistencies between streambed elevations and the model.
↳grid...
Reach strtot, strthick not specified for nstrm=36, reachinput=False and isfropt=0
passed.

Checking segment_data for inconsistencies between segment end elevations and the model.
↳grid...
passed.

Checking for streambed slopes of less than 0.0001...
slope not specified for isfropt=0
passed.

Checking for streambed slopes of greater than 1.0...
slope not specified for isfropt=0
passed.

```

```
[16]: m.external_fnames = [os.path.split(f)[1] for f in m.external_fnames]
m.external_fnames
```

```
[16]: ['test1ss.sg1',
      'test1ss.sg2',
      'test1ss.sg3',
      'test1ss.sg4',
      'test1ss.sg5',
      'test1ss.sg6',
      'test1ss.sg7',
      'test1ss.sg8',
      'test1ss.dvsg9']
```

```
[17]: m.write_input()
```

```
[18]: success, buff = m.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")
```

```

                                MODFLOW-2005
      U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
                                Version 1.12.00 2/3/2017

Using NAME file: test1ss.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:49:51

Solving:  Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:49:51
Elapsed run time:  0.013 Seconds

```

(continues on next page)

(continued from previous page)

Normal termination of simulation

Load SFR formatted water balance output into pandas dataframe using the SfrFile class

- requires the **pandas** library

```
[19]: sfr_outfile = os.path.join(
        "..", "..", "examples", "data", "sfr_examples", "test1ss.flw"
    )
    sfrout = SfrFile(sfr_outfile)
    df = sfrout.get_dataframe()
    df.head()
```

```
[19]:
```

	layer	row	column	segment	reach	Qin	Qaquifer	Qout	Qovr	\
0	1	1	1	1	1	25.0000	0.7923	24.2080	0.0	
1	1	2	2	1	2	24.2080	2.1408	22.0670	0.0	
2	1	3	3	1	3	22.0670	2.9909	19.0760	0.0	
3	1	3	4	1	4	19.0760	2.5538	16.5220	0.0	
4	1	4	5	2	1	6.5222	2.7058	3.8163	0.0	

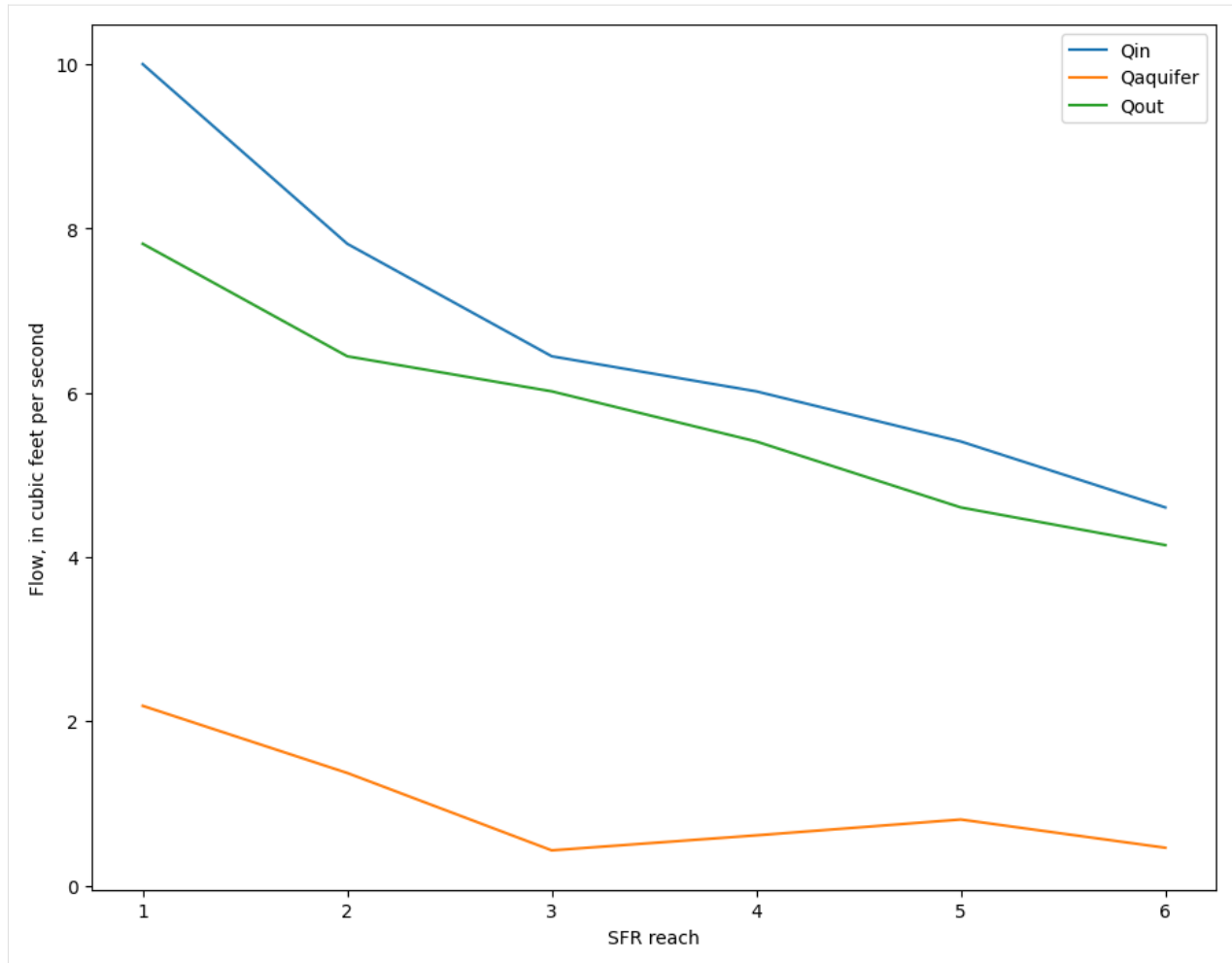
	Qprecip	Qet	stage	depth	width	Cond	gradient	kstpkper	k	i	j
0	0.0	0.0	1094.22	1.174	12.98	0.5843	0.4520	(0, 0)	0	0	0
1	0.0	0.0	1089.21	1.152	12.68	0.8878	0.8038	(0, 0)	0	1	1
2	0.0	0.0	1083.53	1.110	12.13	0.7278	1.3700	(0, 0)	0	2	2
3	0.0	0.0	1078.47	1.064	11.32	0.6285	1.3550	(0, 0)	0	2	3
4	0.0	0.0	1072.40	0.469	12.00	0.7800	1.1560	(0, 0)	0	3	4

Plot streamflow and stream/aquifer interactions for a segment

```
[20]: inds = df.segment == 3
    print(df.reach[inds].astype(str))
    # ax = df.ix[inds, ['Qin', 'Qaquifer', 'Qout']].plot(x=df.reach[inds])
    ax = df.loc[inds, ["reach", "Qin", "Qaquifer", "Qout"]].plot(x="reach")
    ax.set_ylabel("Flow, in cubic feet per second")
    ax.set_xlabel("SFR reach")
```

```
9      1
10     2
11     3
12     4
13     5
14     6
Name: reach, dtype: object
```

```
[20]: Text(0.5, 0, 'SFR reach')
```



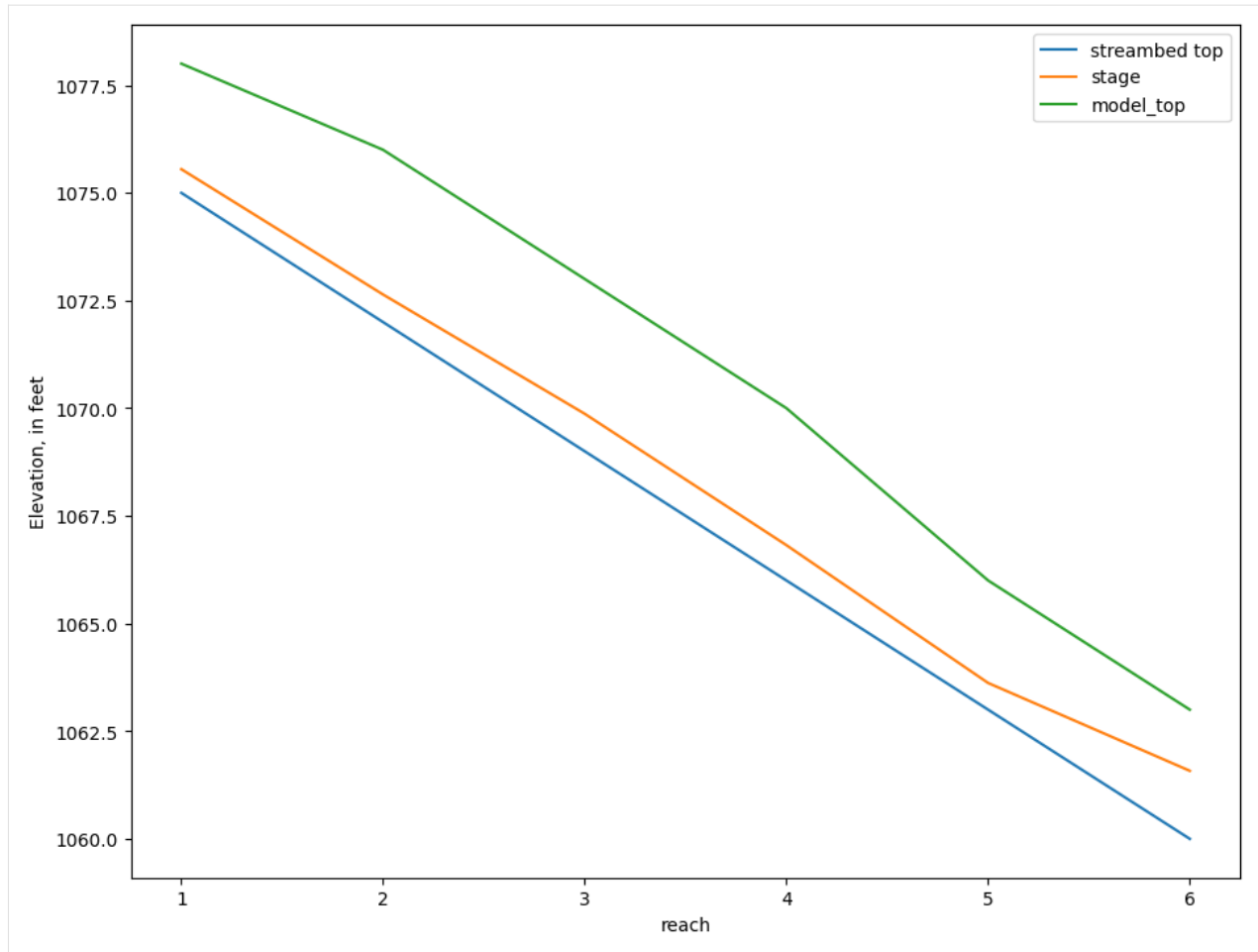
Look at stage, model top, and streambed top

```
[21]: streambed_top = m.sfr.segment_data[0][m.sfr.segment_data[0].nseg == 3][
      ["elevup", "elevdn"]
      ][0]
      streambed_top
```

```
[21]: (1075., 1060.)
```

```
[22]: df["model_top"] = m.dis.top.array[df.row.values - 1, df.column.values - 1]
fig, ax = plt.subplots()
plt.plot([1, 6], list(streambed_top), label="streambed top")
# ax = df.loc[inds, ['stage', 'model_top']].plot(ax=ax, x=df.reach[inds])
ax = df.loc[inds, ["reach", "stage", "model_top"]].plot(ax=ax, x="reach")
ax.set_ylabel("Elevation, in feet")
plt.legend()
```

```
[22]: <matplotlib.legend.Legend at 0x7fc16564e190>
```



Get SFR leakage results from cell budget file

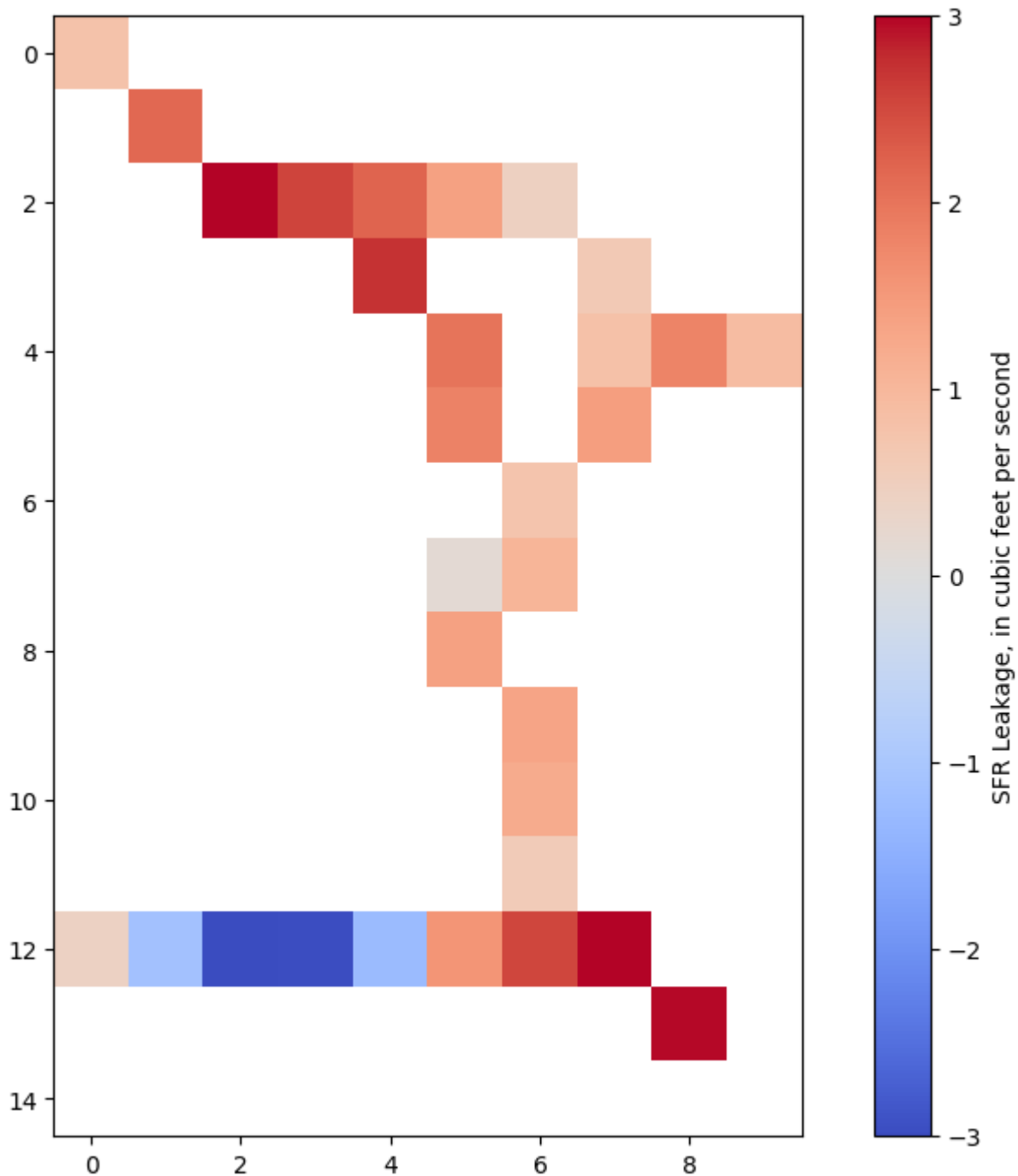
```
[23]: bpth = os.path.join(path, "test1ss.cbc")
      cbbobj = bf.CellBudgetFile(bpth)
      cbbobj.list_records()

(1, 1, b' STREAM LEAKAGE', 10, 15, 1, 0, 0., 0., -1., b'', b'', b'', b'')
```

```
[24]: sfrleak = cbbobj.get_data(text=" STREAM LEAKAGE")[0]
      sfrleak[sfrleak == 0] = np.nan # remove zero values
```


Plot leakage in plan view

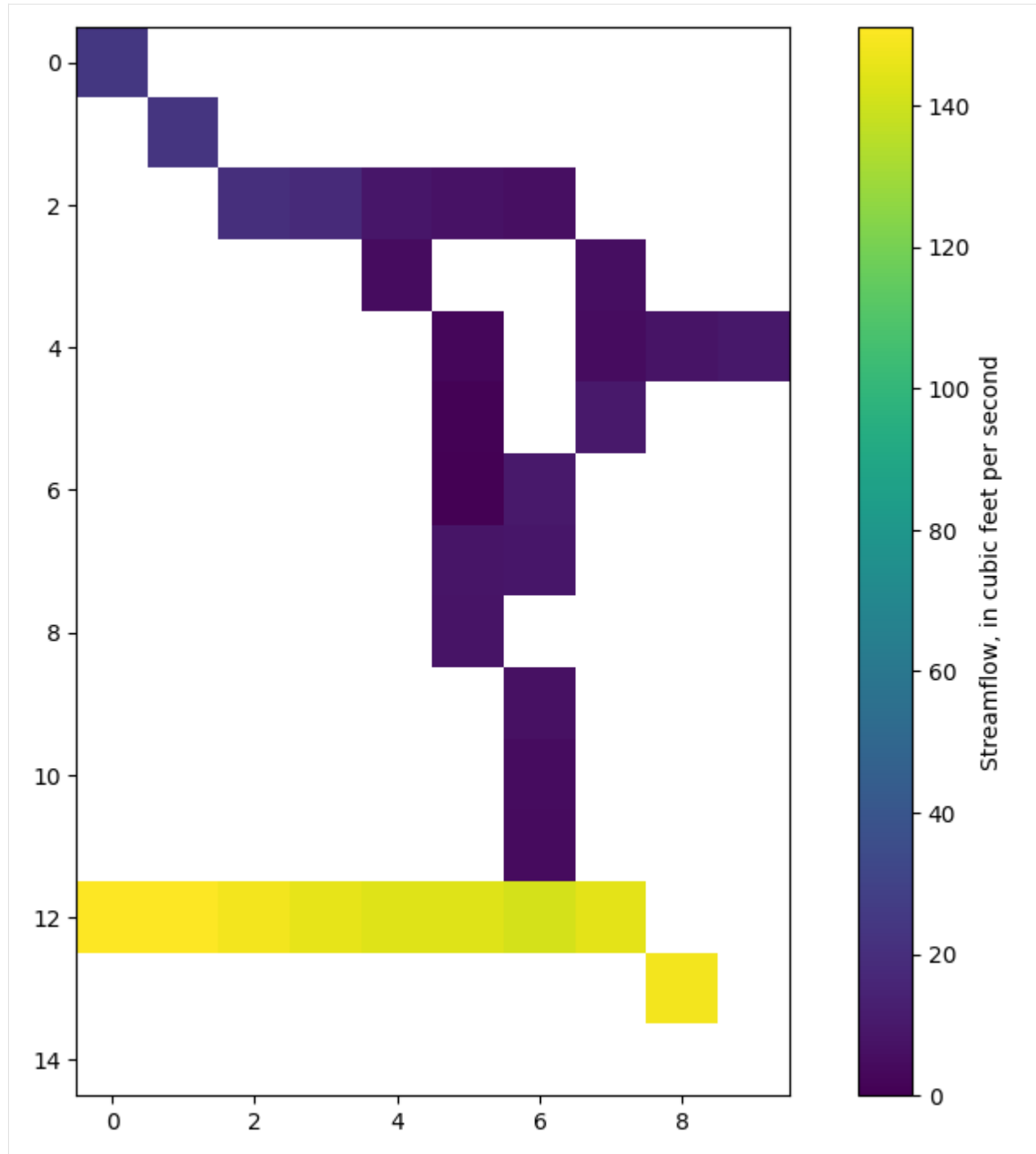
```
[25]: im = plt.imshow(
    sfrleak[0], interpolation="none", cmap="coolwarm", vmin=-3, vmax=3
)
cb = plt.colorbar(im, label="SFR Leakage, in cubic feet per second")
```



Plot total streamflow

```
[26]: sfrQ = sfrleak[0].copy()
sfrQ[sfrQ == 0] = np.nan
sfrQ[df.row.values - 1, df.column.values - 1] = (
    df[["Qin", "Qout"]].mean(axis=1).values
)
im = plt.imshow(sfrQ, interpolation="none")
plt.colorbar(im, label="Streamflow, in cubic feet per second")

[26]: <matplotlib.colorbar.Colorbar at 0x7fc16553ef70>
```



Reading transient SFR formatted output

The SfrFile class handles this the same way

Files for the transient version of the above example were already copied to the data folder in the third cell above. First run the transient model to get the output:

```
>mf2005 test1tr.nam
```

```
[27]: flopy.run_model(exe_name, "test1tr.nam", model_ws=path, silent=True)
```

```
[27]: (True, [])
```

```
[28]: sfrout_tr = SfrFile(os.path.join(path, "test1tr.flw"))
dftr = sfrout_tr.get_dataframe()
dftr.head()
```

```
[28]:
```

	layer	row	column	segment	reach	Qin	Qaquifer	Qout	Qovr	\
0	1	1	1	1	1	25.0000	0.77759	24.2220	0.0	
1	1	2	2	1	2	24.2220	2.21540	22.0070	0.0	
2	1	3	3	1	3	22.0070	2.98700	19.0200	0.0	
3	1	3	4	1	4	19.0200	2.54940	16.4710	0.0	
4	1	4	5	2	1	6.4706	2.70370	3.7669	0.0	

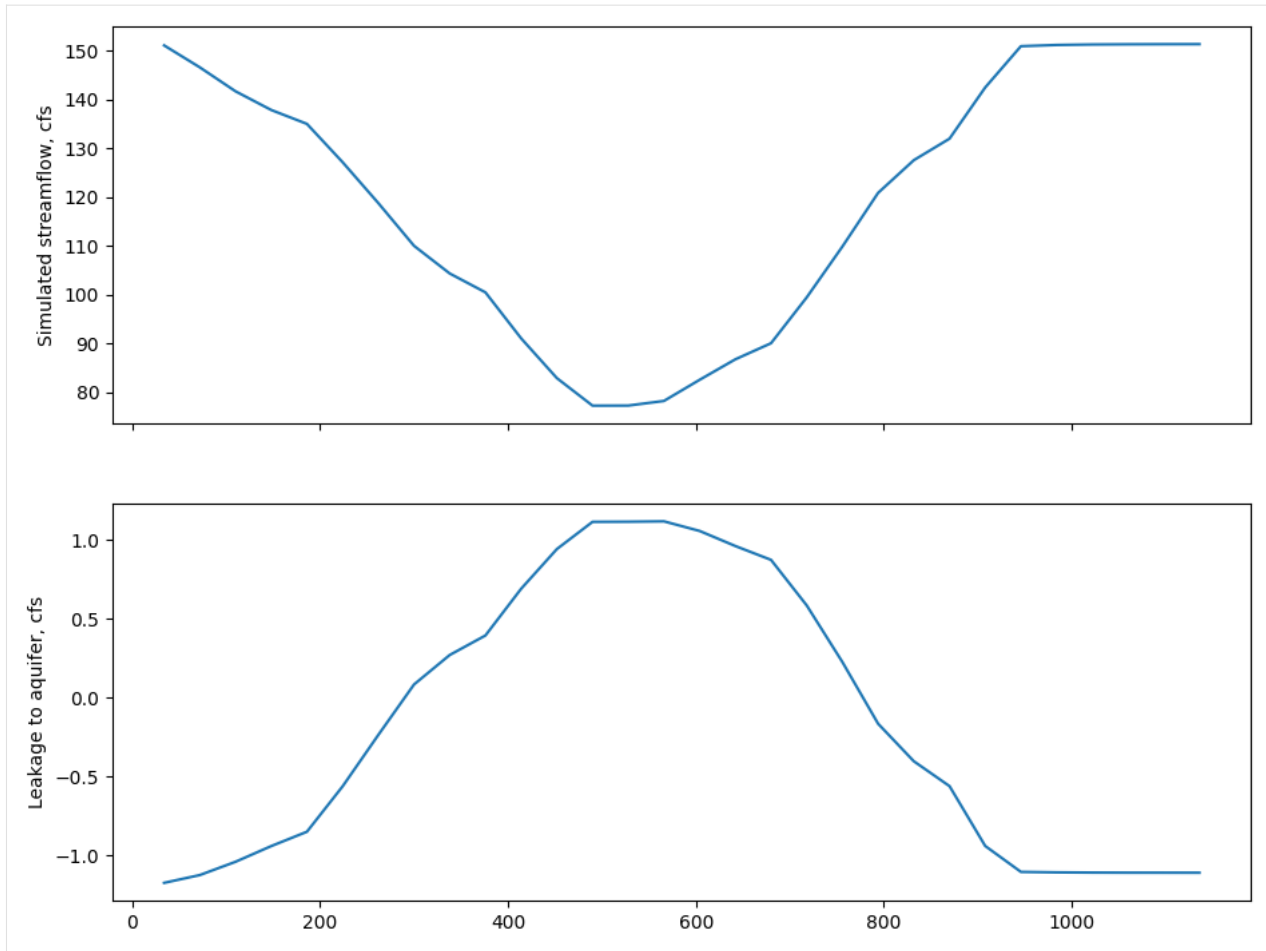
	Qprecip	Qet	stage	depth	width	Cond	gradient	kstpker	k	i	j
0	0.0	0.0	1094.22	1.1740	12.98	0.5843	0.4436	(0, 0)	0	0	0
1	0.0	0.0	1089.21	1.1510	12.68	0.8875	0.8321	(0, 0)	0	1	1
2	0.0	0.0	1083.53	1.1090	12.12	0.7270	1.3700	(0, 0)	0	2	2
3	0.0	0.0	1078.47	1.0630	11.31	0.6275	1.3540	(0, 0)	0	2	3
4	0.0	0.0	1072.40	0.4663	12.00	0.7800	1.1550	(0, 0)	0	3	4

plot a hydrograph

plot Qout (simulated streamflow) and Qaquifer (simulated stream leakage) through time

```
[29]: fig, axes = plt.subplots(2, 1, sharex=True)
dftr8 = dftr.loc[(dftr.segment == 8) & (dftr.reach == 5)]
dftr8.Qout.plot(ax=axes[0])
axes[0].set_ylabel("Simulated streamflow, cfs")
dftr8.Qaquifer.plot(ax=axes[1])
axes[1].set_ylabel("Leakage to aquifer, cfs")
```

```
[29]: Text(0, 0.5, 'Leakage to aquifer, cfs')
```



```
[30]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.7.6 SWI2 Example 1. Rotating Interface

This example problem is the first example problem in the SWI2 documentation (<https://pubs.usgs.gov/tm/6a46/>) and simulates transient movement of a freshwater-seawater interface separating two density zones in a two-dimensional vertical plane. The problem domain is 250 m long, 40 m high, and 1 m wide. The aquifer is confined, storage changes are not considered (all MODFLOW stress periods are steady-state), and the top and bottom of the aquifer are horizontal and impermeable.

The domain is discretized into 50 columns, 1 row, and 1 layer, with respective cell dimensions of 5 m (DELR), 1 m (DELC), and 40 m. A constant head of 0 m is specified for column 50. The hydraulic conductivity is 2 m/d and the effective porosity (SSZ) is 0.2. A flow of 1 m³/d of seawater is specified in column 1 and causes groundwater to flow from left to right in the model domain.

The domain contains one freshwater zone and one seawater zone, separated by an active ZETA surface between the zones (NSRF=1) that approximates the 50-percent seawater salinity contour. A 400-day period is simulated using a constant time step of 2 days. Fluid density is represented using the stratified option (ISTRAT=1) and the elevation of the interface is output every 100 days (every 50 time steps). The densities, ρ , of the freshwater and saltwater are 1,000

and 1,025 kg/m³, respectively. Hence, the dimensionless densities, ν , of the freshwater and saltwater are 0.0 and 0.025, respectively. The maximum slope of the toe and tip is specified as 0.2 (TOESLOPE=TIPSLOPE=0.2), and default tip/toe parameters are used (ALPHA=BETA=0.1). Initially, the interface is at a 45° angle from (x,z) = (80,0) to (x,z) = (120,-40). The source/sink terms (ISOURCE) are specified to be freshwater everywhere (ISOURCE=1) except in cell 1 where saltwater enters the model and ISOURCE equals 2. A comparison of results for SWI2 and the exact Dupuit solution of Wilson and Sa Da Costa (1982) are presented below. The constant flow from left to right results in an average velocity of 0.125 m/d. The exact Dupuit solution is a rotating straight interface of which the center moves to the right with this velocity

Import numpy and matplotlib, set all figures to be inline, import flopy.modflow and flopy.utils.

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))
```

```
3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

Define model name of your model and the location of MODFLOW executable. All MODFLOW files and output will be stored in the subdirectory defined by the workspace. Create a model named m1 and specify that this is a MODFLOW-2005 model.

```
[2]: modelname = "swiex1"

# Set name of MODFLOW exe
# assumes executable is in users path statement
exe_name = "mf2005"

temp_dir = TemporaryDirectory()
workspace = temp_dir.name

m1 = flopy.modflow.Modflow(
    modelname, version="mf2005", exe_name=exe_name, model_ws=workspace
)
```

Define the number of layers, rows and columns, and the cell size along the rows (delr) and along the columns (delc). Then create a discretization file. Specify the top and bottom of the aquifer. The heads are computed quasi-steady state (hence a steady MODFLOW run) while the interface will move. There is one stress period with a length of 400 days and 200 steps (so one step is 2 days).

```
[3]: nlay = 1
      nrow = 1
      ncol = 50
      delr = 5.0
      delc = 1.0
      nper, perlen, nstp = 1, 400.0, 200
      discret = flopy.modflow.ModflowDis(
          ml,
          nlay=nlay,
          nrow=nrow,
          ncol=ncol,
          delr=delr,
          delc=delc,
          top=0,
          botm=-40.0,
          steady=True,
          nper=nper,
          perlen=perlen,
          nstp=nstp,
      )
```

All cells are active (ibound=1), while the last cell is fixed head (ibound=-1). The starting values of the head are not important, as the heads are computed every time with a steady run.

```
[4]: ibound = np.ones((nrow, ncol))
      ibound[0, -1] = -1
      bas = flopy.modflow.ModflowBas(ml, ibound=ibound, strt=0.0)
```

Define the hydraulic conductivity. The aquifer is confined (laytype=0) and the intercell hydraulic conductivity is the harmonic meand (layavg=0).

```
[5]: lpf = flopy.modflow.ModflowLpf(ml, hk=2.0, laytyp=0, layavg=0)
```

Inflow on the right side of the model is 1 m³/d (layer 0, row 0, column 0, discharge 1)

```
[6]: wel = flopy.modflow.ModflowWel(ml, stress_period_data={0: [[0, 0, 0, 1]]})
```

Define the output control to save heads and interface every 50 steps, and define the pcg solver with default arguments.

```
[7]: spd = {}
      for istp in range(49, nstp + 1, 50):
          spd[(0, istp)] = ["save head", "print budget"]
          spd[(0, istp + 1)] = []

      oc = flopy.modflow.ModflowOc(ml, stress_period_data=spd)
      pcg = flopy.modflow.ModflowPcg(ml)
```

The initial interface is straight. The isource is one everywhere (inflow and outflow is fresh (zone 1)) except for the first cell (index=0) which has saltwater inflow (zone 2).

```
[8]: z = np.zeros((nrow, ncol), float)
      z[0, 16:24] = np.arange(-2.5, -40, -5)
      z[0, 24:] = -40
      z = [z]  # zeta needs to be
```

(continues on next page)

(continued from previous page)

```

isource = np.ones((nrow, ncol), int)
isource[0, 0] = 2
#
swi = flopy.modflow.ModflowSwi2(
    ml,
    nsrf=1,
    istrat=1,
    toeslope=0.2,
    tipslope=0.2,
    nu=[0, 0.025],
    zeta=z,
    ssz=0.2,
    isource=isource,
    nsolver=1,
    iswizt=55,
)

```

Write the MODFLOW input files and run the model

```

[9]: ml.write_input()
success, buff = ml.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

```

```

                                MODFLOW-2005
      U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
                                Version 1.12.00 2/3/2017

Using NAME file: swiex1.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:50:10

Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:      2      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:      3      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:      4      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:      5      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:      6      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:      7      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:      8      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:      9      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:     10      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:     11      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:     12      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:     13      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:     14      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:     15      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:     16      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:     17      Ground-Water Flow Eqn.
Solving: Stress period:      1      Time step:     18      Ground-Water Flow Eqn.

```

(continues on next page)

[illegible]

(continued from previous page)

[illegible]

(continues on next page)

[illegible]

(continued from previous page)

```
Solving: Stress period: 1 Time step: 175 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 176 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 177 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 178 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 179 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 180 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 181 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 182 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 183 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 184 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 185 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 186 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 187 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 188 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 189 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 190 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 191 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 192 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 193 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 194 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 195 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 196 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 197 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 198 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 199 Ground-Water Flow Eqn.
Solving: Stress period: 1 Time step: 200 Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:50:10
Elapsed run time: 0.019 Seconds
```

Normal termination of simulation

Load the head and zeta data from the file

```
[10]: # read model heads
hfile = flopy.utils.HeadFile(os.path.join(ml.model_ws, modelname + ".hds"))
head = hfile.get_alldata()
# read model zeta
zfile = flopy.utils.CellBudgetFile(
    os.path.join(ml.model_ws, modelname + ".zta")
)
kstpker = zfile.get_kstpker()
zeta = []
for kk in kstpker:
    zeta.append(zfile.get_data(kstpker=kk, text="ZETASRF 1")[0])
zeta = np.array(zeta)
```

Make a graph and add the solution of Wilson and Sa da Costa

```
[11]: plt.figure(figsize=(16, 6))
# define x-values of xcells and plot interface
x = np.arange(0, ncol * delr, delr) + delr / 2.0
label = [
```

(continues on next page)

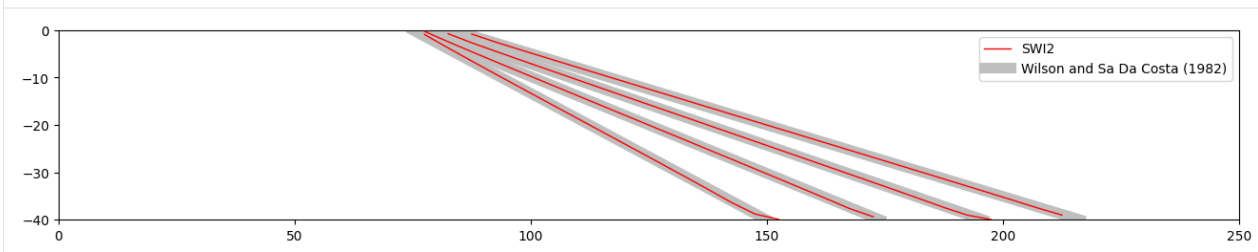
(continued from previous page)

```

    "SWI2",
    "-",
    "-",
    "-",
    "-",
] # labels with an underscore are not added to legend
for i in range(4):
    zt = np.ma.masked_outside(zeta[i, 0, 0, :], -39.99999, -0.00001)
    plt.plot(x, zt, "r-", lw=1, zorder=10, label=label[i])
# Data for the Wilson - Sa da Costa solution
k = 2.0
n = 0.2
nu = 0.025
H = 40.0
tzero = H * n / (k * nu) / 4.0
Ltoe = np.zeros(4)
v = 0.125
t = np.arange(100, 500, 100)
label = [
    "Wilson and Sa Da Costa (1982)",
    "-",
    "-",
    "-",
    "-",
] # labels with an underscore are not added to legend
for i in range(4):
    Ltoe[i] = H * np.sqrt(k * nu * (t[i] + tzero) / n / H)
    plt.plot(
        [100 - Ltoe[i] + v * t[i], 100 + Ltoe[i] + v * t[i]],
        [0, -40],
        "0.75",
        lw=8,
        zorder=0,
        label=label[i],
    )
# Scale figure and add legend
plt.axis("scaled")
plt.xlim(0, 250)
plt.ylim(-40, 0)
plt.legend(loc="best")

```

[11]: <matplotlib.legend.Legend at 0x7f395c1b94f0>



Use ModelCrossSection plotting class and plot_surface() method to plot zeta surfaces.

```

[12]: fig = plt.figure(figsize=(16, 3))
      ax = fig.add_subplot(1, 1, 1)

```

(continues on next page)

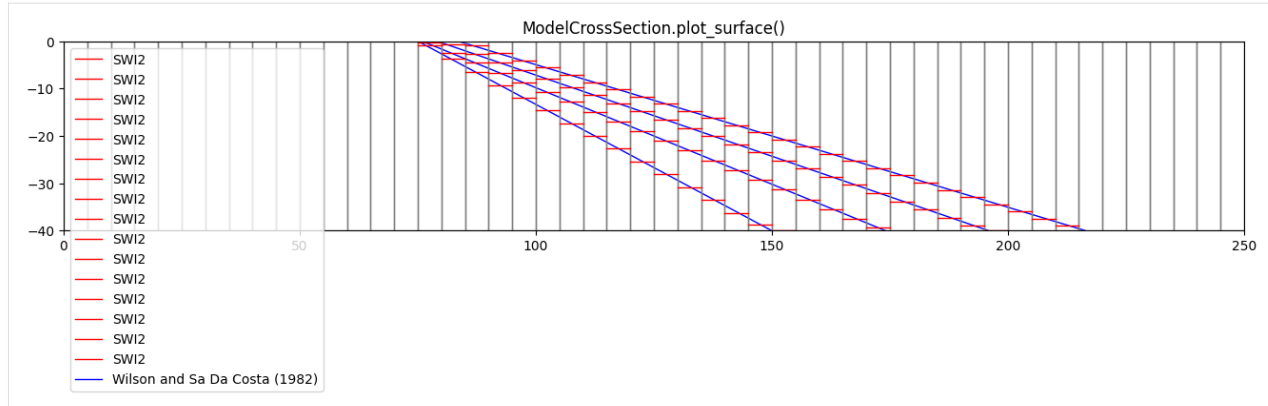
(continued from previous page)

```

modelxsect = flopy.plot.PlotCrossSection(model=ml, line={"Row": 0})
label = ["SWI2", "_", "_", "_"]
for k in range(zeta.shape[0]):
    modelxsect.plot_surface(
        zeta[k, :, :, :],
        masked_values=[0, -40.0],
        color="red",
        lw=1,
        label=label[k],
    )
linecollection = modelxsect.plot_grid()
ax.set_title("ModelCrossSection.plot_surface()")
# Data for the Wilson - Sa da Costa solution
k = 2.0
n = 0.2
nu = 0.025
H = 40.0
tzero = H * n / (k * nu) / 4.0
Ltoe = np.zeros(4)
v = 0.125
t = np.arange(100, 500, 100)
label = [
    "Wilson and Sa Da Costa (1982)",
    "_",
    "_",
    "_",
    "_",
] # labels with an underscore are not added to legend
for i in range(4):
    Ltoe[i] = H * np.sqrt(k * nu * (t[i] + tzero) / n / H)
    ax.plot(
        [100 - Ltoe[i] + v * t[i], 100 + Ltoe[i] + v * t[i]],
        [0, -40],
        "blue",
        lw=1,
        zorder=0,
        label=label[i],
    )
# Scale figure and add legend
ax.axis("scaled")
ax.set_xlim(0, 250)
ax.set_ylim(-40, 0)
ax.legend(loc="best")

```

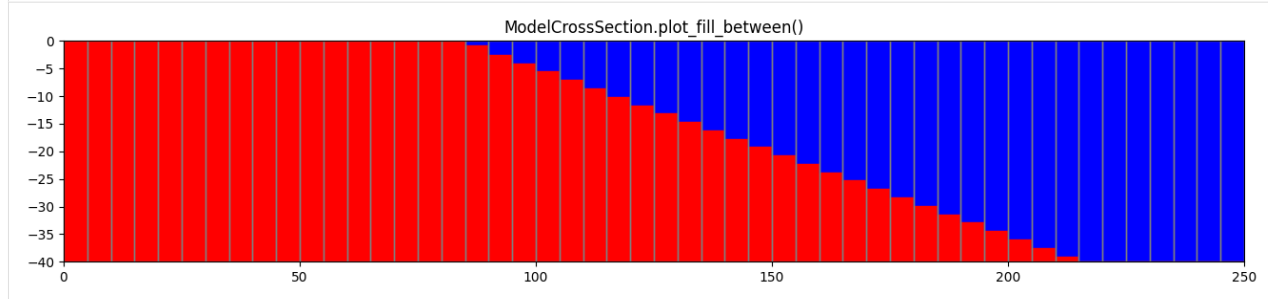
[12]: <matplotlib.legend.Legend at 0x7f39599feb20>



Use ModelCrossSection plotting class and plot_fill_between() method to fill between zeta surfaces.

```
[13]: fig = plt.figure(figsize=(16, 3))
ax = fig.add_subplot(1, 1, 1)
modelxsect = flopy.plot.PlotCrossSection(model=ml, line={"Row": 0})
modelxsect.plot_fill_between(zeta[3, :, :, :])
linecollection = modelxsect.plot_grid()
ax.set_title("ModelCrossSection.plot_fill_between()")
```

```
[13]: Text(0.5, 1.0, 'ModelCrossSection.plot_fill_between()')
```

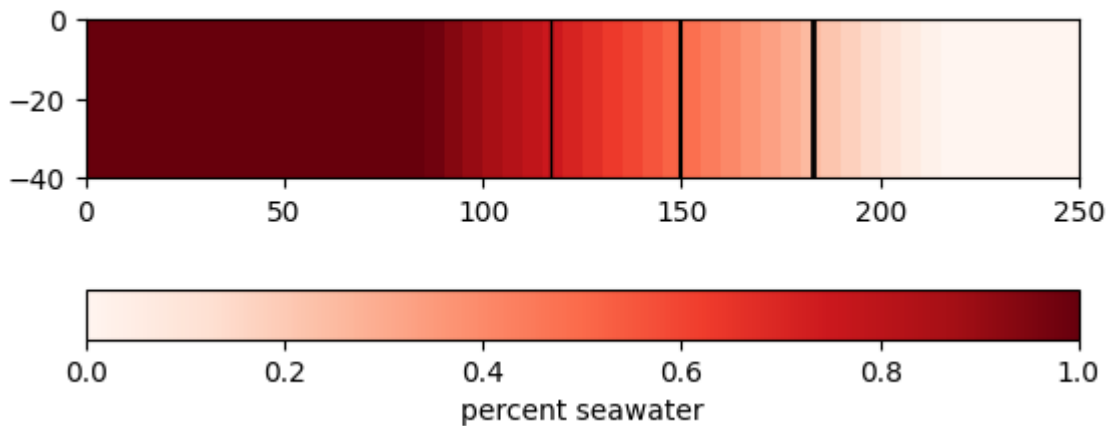


Convert zeta surfaces to relative seawater concentrations

```
[14]: X, Y = np.meshgrid(x, [0, -40])
zc = flopy.plot.SwiConcentration(model=ml)
conc = zc.calc_conc(zeta={0: zeta[3, :, :, :]}) / 0.025
print(conc[0, 0, :])
v = np.vstack((conc[0, 0, :], conc[0, 0, :]))
plt.imshow(v, extent=[0, 250, -40, 0], cmap="Reds")
cb = plt.colorbar(orientation="horizontal")
cb.set_label("percent seawater")
plt.contour(X, Y, v, [0.25, 0.5, 0.75], linewidths=[2, 1.5, 1], colors="black")
```

```
[1.000000001 1.000000001 1.000000001 1.000000001 1.000000001 1.000000001
 1.000000001 1.000000001 1.000000001 1.000000001 1.000000001 1.000000001
 1.000000001 1.000000001 1.000000001 1.000000001 1.000000001 0.97957368
 0.93777831 0.89900599 0.8606122 0.82240776 0.78431243 0.74628681
 0.70830902 0.6703652 0.63244558 0.59454289 0.55665098 0.5187645
 0.4808785 0.44298783 0.40508729 0.36717053 0.32923027 0.29125705
 0.25323816 0.21515508 0.17697897 0.13865538 0.10004826 0.06099987
 0.02483263 0. 0. 0. 0. 0.
 0. 0. ]
```

```
[14]: <matplotlib.contour.QuadContourSet at 0x7f395bb693d0>
```



```
[15]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass
```

6.7.7 SWI2 Example 4. Upconing Below a Pumping Well in a Two-Aquifer Island System

This example problem is the fourth example problem in the SWI2 documentation (<https://pubs.usgs.gov/tm/6a46/>) and simulates transient movement of the freshwater-seawater interface beneath an island in response to recharge and groundwater withdrawals. The island is 2,050 m and consists of two 20-m thick aquifers that extend below sea level. The aquifers are confined, storage changes are not considered (all MODFLOW stress periods are steady-state), and the top and bottom of each aquifer is horizontal. The top of the upper aquifer and the bottom of the lower aquifer are impermeable.

The domain is discretized into 61 columns, 61 rows, and 2 layers, with respective cell dimensions of 50 m (DELR), 50 m (DELC), and 20 m. A total of 230 years is simulated using three stress periods with lengths of 200, 12, and 18 years, with constant time steps of 0.2, 0.1, and 0.1 years, respectively.

The horizontal and vertical hydraulic conductivity of both aquifers are 10 m/d and 0.2 m/d, respectively. The effective porosity is 0.2 for both aquifers. The model is extended 500 m offshore along all sides and the ocean boundary is represented as a general head boundary condition (GHB) in model layer 1. A freshwater head of 0 m is specified at the ocean bottom in all general head boundaries. The GHB conductance that controls outflow from the aquifer into the ocean is 62.5 m²/d and corresponds to a leakance of 0.025 d⁻¹ (or a resistance of 40 days).

The groundwater is divided into a freshwater zone and a seawater zone, separated by an active ZETA surface between the zones (NSRF=1) that approximates the 50-percent seawater salinity contour. Fluid density is represented using the stratified density option (ISTRAT=1). The dimensionless density difference (ν) between freshwater and saltwater is 0.025. The tip and toe tracking parameters are a TOESLOPE and TIPSLOPE of 0.005, a default ALPHA of 0.1, and a default BETA of 0.1. Initially, the interface between freshwater and saltwater is 1 m below land surface on the island and at the top of the upper aquifer offshore. The SWI2 ISOURCE parameter is set to -2 in cells having GHBs so that water that infiltrates into the aquifer from the GHB cells is saltwater (zone 2), whereas water that flows out of the model at the GHB cells is identical to water at the top of the aquifer. ISOURCE in layer 2, row 31, column 36 is set to 2 so that a saltwater well may be simulated in the third stress period of simulation 2. In all other cells, the SWI2 ISOURCE parameter is set to 0, indicating boundary conditions have water that is identical to water at the top of the aquifer and

can be either freshwater or saltwater, depending on the elevation of the active ZETA surface in the cell.

A constant recharge rate of 0.4 millimeters per day (mm/d) is used in all three stress periods. The development of the freshwater lens is simulated for 200 years, after which a pumping well having a withdrawal rate of 250 m³/d is started in layer 1, row 31, column 36. For the first simulation (simulation 1), the well pumps for 30 years, after which the interface almost reaches the top of the upper aquifer layer. In the second simulation (simulation 2), an additional well withdrawing saltwater at a rate of 25 m³/d is simulated below the freshwater well in layer 2, row 31, column 36, 12 years after the freshwater groundwater withdrawal begins in the well in layer 1. The saltwater well is intended to prevent the interface from upconing into the upper aquifer (model layer).

Import numpy and matplotlib, set all figures to be inline, import flopy.modflow and flopy.utils.

```
[1]: import os
import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

Define model name of your model and the location of MODFLOW executable. All MODFLOW files and output will be stored in the subdirectory defined by the workspace. Create a model named m1 and specify that this is a MODFLOW-2005 model.

```
[2]: # Set name of MODFLOW exe
# assumes executable is in users path statement
exe_name = "mf2005"

temp_dir = TemporaryDirectory()
workspace = temp_dir.name
```

Define the number of layers, rows and columns. The heads are computed quasi-steady state (hence a steady MODFLOW run) while the interface will move. There are three stress periods with a length of 200, 12, and 18 years and 1,000, 120, and 180 steps.

```
[3]: ncol = 61
nrow = 61
nlay = 2

nper = 3
perlen = [365.25 * 200.0, 365.25 * 12.0, 365.25 * 18.0]
nstp = [1000, 120, 180]
```

(continues on next page)

(continued from previous page)

```
save_head = [200, 60, 60]
steady = True
```

Specify the cell size along the rows (delr) and along the columns (delc) and the top and bottom of the aquifer for the DIS package.

```
[4]: # dis data
delr, delc = 50.0, 50.0
botm = np.array([-10.0, -30.0, -50.0])
```

Define the IBOUND array and starting heads for the BAS package. The corners of the model are defined to be inactive.

```
[5]: # bas data
# ibound - active except for the corners
ibound = np.ones((nlay, nrow, ncol), dtype=int)
ibound[:, 0, 0] = 0
ibound[:, 0, -1] = 0
ibound[:, -1, 0] = 0
ibound[:, -1, -1] = 0
# initial head data
ihead = np.zeros((nlay, nrow, ncol), dtype=float)
```

Define the layers to be confined and define the horizontal and vertical hydraulic conductivity of the aquifer for the LPF package.

```
[6]: # lpf data
laytyp = 0
hk = 10.0
vka = 0.2
```

Define the boundary condition data for the model

```
[7]: # boundary condition data
# ghb data
colcell, rowcell = np.meshgrid(np.arange(0, ncol), np.arange(0, nrow))
index = np.zeros((nrow, ncol), dtype=int)
index[:, :10] = 1
index[:, -10:] = 1
index[:10, :] = 1
index[-10:, :] = 1
nghb = np.sum(index)
lrchc = np.zeros((nghb, 5))
lrchc[:, 0] = 0
lrchc[:, 1] = rowcell[index == 1]
lrchc[:, 2] = colcell[index == 1]
lrchc[:, 3] = 0.0
lrchc[:, 4] = 50.0 * 50.0 / 40.0
# create ghb dictionary
ghb_data = {0: lrchc}

# recharge data
rch = np.zeros((nrow, ncol), dtype=float)
rch[index == 0] = 0.0004
```

(continues on next page)

(continued from previous page)

```
# create recharge dictionary
rch_data = {0: rch}

# well data
nwwells = 2
lrcq = np.zeros((nwwells, 4))
lrcq[0, :] = np.array((0, 30, 35, 0))
lrcq[1, :] = np.array([1, 30, 35, 0])
lrcqw = lrcq.copy()
lrcqw[0, 3] = -250
lrcqsw = lrcq.copy()
lrcqsw[0, 3] = -250.0
lrcqsw[1, 3] = -25.0
# create well dictionary
base_well_data = {0: lrcq, 1: lrcqw}
swwells_well_data = {0: lrcq, 1: lrcqw, 2: lrcqsw}
```

```
[8]: # swi2 data
nadptmx = 10
nadptmn = 1
nu = [0, 0.025]
numult = 5.0
toeslope = nu[1] / numult # 0.005
tipslope = nu[1] / numult # 0.005
z1 = -10.0 * np.ones((nrow, ncol))
z1[index == 0] = -11.0
z = np.array([z1, z1])
iso = np.zeros((nlay, nrow, ncol), dtype=int)
iso[0, :, :][index == 0] = 1
iso[0, :, :][index == 1] = -2
iso[1, 30, 35] = 2
ssz = 0.2
# swi2 observations
obsnam = ["layer1_", "layer2_"]
obslrc = [[0, 30, 35], [1, 30, 35]]
nobs = len(obsnam)
iswiobs = 1051
```

Create output control (OC) data using words

```
[9]: # oc data
spd = {
    (0, 199): ["print budget", "save head"],
    (0, 200): [],
    (0, 399): ["print budget", "save head"],
    (0, 400): [],
    (0, 599): ["print budget", "save head"],
    (0, 600): [],
    (0, 799): ["print budget", "save head"],
    (0, 800): [],
    (0, 999): ["print budget", "save head"],
    (1, 0): [],
```

(continues on next page)

(continued from previous page)

```
(1, 59): ["print budget", "save head"],
(1, 60): [],
(1, 119): ["print budget", "save head"],
(1, 120): [],
(2, 0): [],
(2, 59): ["print budget", "save head"],
(2, 60): [],
(2, 119): ["print budget", "save head"],
(2, 120): [],
(2, 179): ["print budget", "save head"],
}
```

Create the model with the freshwater well (Simulation 1)

```
[10]: modelname = "swiex4_s1"
ml = flopy.modflow.Modflow(
    modelname, version="mf2005", exe_name=exe_name, model_ws=workspace
)

discret = flopy.modflow.ModflowDis(
    ml,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    laycbd=0,
    delr=delr,
    delc=delc,
    top=botm[0],
    botm=botm[1:],
    nper=nper,
    perlen=perlen,
    nstp=nstp,
)
bas = flopy.modflow.ModflowBas(ml, ibound=ibound, strt=ihead)
lpf = flopy.modflow.ModflowLpf(ml, laytyp=laytyp, hk=hk, vka=vka)
wel = flopy.modflow.ModflowWel(ml, stress_period_data=base_well_data)
ghb = flopy.modflow.ModflowGhb(ml, stress_period_data=ghb_data)
rch = flopy.modflow.ModflowRch(ml, rech=rch_data)
swi = flopy.modflow.ModflowSwi2(
    ml,
    nsrf=1,
    istrat=1,
    toeslope=toeslope,
    tipslope=tipslope,
    nu=nu,
    zeta=z,
    ssz=ssz,
    isource=iso,
    nsolver=1,
    nadptmx=nadptmx,
    nadptmn=nadptmn,
    nobs=nobs,
```

(continues on next page)

(continued from previous page)

```

        iswiobs=iswiobs,
        obsnam=obsnam,
        obslrc=obslrc,
        iswizt=55,
    )
    oc = flopy.modflow.ModflowOc(ml, stress_period_data=spd)
    pcg = flopy.modflow.ModflowPcg(
        ml, hclose=1.0e-6, rclose=3.0e-3, mxiter=100, iter1=50
    )

```

ModflowSwi2: specification of nobs is deprecated.

Write the simulation 1 MODFLOW input files and run the model

```

[11]: ml.write_input()
      ml.run_model(silent=True)

```

```

[11]: (True, [])

```

Create the model with the saltwater well (Simulation 2)

```

[12]: modelname2 = "swiex4_s2"
      ml2 = flopy.modflow.Modflow(
          modelname2, version="mf2005", exe_name=exe_name, model_ws=workspace
      )

      discret = flopy.modflow.ModflowDis(
          ml2,
          nlay=nlay,
          nrow=nrow,
          ncol=ncol,
          laycbd=0,
          delr=delr,
          delc=delc,
          top=botm[0],
          botm=botm[1:],
          nper=nper,
          perlen=perlen,
          nstp=nstp,
      )

      bas = flopy.modflow.ModflowBas(ml2, ibound=ibound, strt=ihead)
      lpf = flopy.modflow.ModflowLpf(ml2, laytyp=laytyp, hk=hk, vka=vka)
      wel = flopy.modflow.ModflowWel(ml2, stress_period_data=swwells_well_data)
      ghb = flopy.modflow.ModflowGhb(ml2, stress_period_data=ghb_data)
      rch = flopy.modflow.ModflowRch(ml2, rech=rch_data)
      swi = flopy.modflow.ModflowSwi2(
          ml2,
          nsrf=1,
          istrat=1,
          toeslope=toeslope,
          tipslope=tipslope,
          nu=nu,
          zeta=z,

```

(continues on next page)

(continued from previous page)

```

    ssz=ssz,
    isource=iso,
    nsolver=1,
    nadptmx=nadptmx,
    nadptmn=nadptmn,
    nobs=nobs,
    iswiobs=iswiobs,
    obsnam=obsnam,
    obslrc=obslrc,
    iswizt=55,
)
oc = flopy.modflow.ModflowOc(ml2, stress_period_data=spd)
pcg = flopy.modflow.ModflowPcg(
    ml2, hclose=1.0e-6, rclose=3.0e-3, mxiter=100, iter1=50
)

```

ModflowSwi2: specification of nobs is deprecated.

Write the simulation 2 MODFLOW input files and run the model

```

[13]: ml2.write_input()
      ml2.run_model(silent=True)

```

```

[13]: (True, [])

```

Load the simulation 1 ZETA data and ZETA observations.

```

[14]: # read base model zeta
      zfile = flopy.utils.CellBudgetFile(
          os.path.join(ml.model_ws, modelname + ".zta")
      )
      kstpker = zfile.get_kstpker()
      zeta = []
      for kk in kstpker:
          zeta.append(zfile.get_data(kstpker=kk, text="ZETASRF 1")[0])
      zeta = np.array(zeta)
      # read swi obs
      zobs = np.genfromtxt(
          os.path.join(ml.model_ws, modelname + ".zobs.out"), names=True
      )

```

Load the simulation 2 ZETA data and ZETA observations.

```

[15]: # read saltwater well model zeta
      zfile2 = flopy.utils.CellBudgetFile(
          os.path.join(ml2.model_ws, modelname2 + ".zta")
      )
      kstpker = zfile2.get_kstpker()
      zeta2 = []
      for kk in kstpker:
          zeta2.append(zfile2.get_data(kstpker=kk, text="ZETASRF 1")[0])
      zeta2 = np.array(zeta2)
      # read swi obs
      zobs2 = np.genfromtxt(

```

(continues on next page)

(continued from previous page)

```
os.path.join(ml2.model_ws, modelname2 + ".zobs.out"), names=True
)
```

Create arrays for the x-coordinates and the output years

```
[16]: x = np.linspace(-1500, 1500, 61)
xcell = np.linspace(-1500, 1500, 61) + delr / 2.0
xedge = np.linspace(-1525, 1525, 62)
years = [40, 80, 120, 160, 200, 6, 12, 18, 24, 30]
```

Define figure dimensions and colors used for plotting ZETA surfaces

```
[17]: # figure dimensions
fwid, fhgt = 8.00, 5.50
flft, frgt, fbot, ftop = 0.125, 0.95, 0.125, 0.925

# line color definition
icolor = 5
colormap = plt.cm.jet # winter
cc = []
cr = np.linspace(0.9, 0.0, icolor)
for idx in cr:
    cc.append(colormap(idx))
```

Recreate **Figure 9** from the SWI2 documentation (<https://pubs.usgs.gov/tm/6a46/>).

```
[18]: plt.rcParams.update({"legend.fontsize": 6, "legend.frameon": False})
fig = plt.figure(figsize=(fwid, fhgt), facecolor="w")
fig.subplots_adjust(
    wspace=0.25, hspace=0.25, left=flft, right=frgt, bottom=fbot, top=ftop
)
# first plot
ax = fig.add_subplot(2, 2, 1)
# axes limits
ax.set_xlim(-1500, 1500)
ax.set_ylim(-50, -10)
for idx in range(5):
    # layer 1
    ax.plot(
        xcell,
        zeta[idx, 0, 30, :],
        drawstyle="steps-mid",
        linewidth=0.5,
        color=cc[idx],
        label="{:2d} years".format(years[idx]),
    )
    # layer 2
    ax.plot(
        xcell,
        zeta[idx, 1, 30, :],
        drawstyle="steps-mid",
        linewidth=0.5,
        color=cc[idx],
```

(continues on next page)

(continued from previous page)

```

        label="_None",
    )
ax.plot([-1500, 1500], [-30, -30], color="k", linewidth=1.0)
# legend
plt.legend(loc="lower left")
# axes labels and text
ax.set_xlabel("Horizontal distance, in meters")
ax.set_ylabel("Elevation, in meters")
ax.text(
    0.025,
    0.55,
    "Layer 1",
    transform=ax.transAxes,
    va="center",
    ha="left",
    size="7",
)
ax.text(
    0.025,
    0.45,
    "Layer 2",
    transform=ax.transAxes,
    va="center",
    ha="left",
    size="7",
)
ax.text(
    0.975,
    0.1,
    "Recharge conditions",
    transform=ax.transAxes,
    va="center",
    ha="right",
    size="8",
)

# second plot
ax = fig.add_subplot(2, 2, 2)
# axes limits
ax.set_xlim(-1500, 1500)
ax.set_ylim(-50, -10)
for idx in range(5, len(years)):
    # layer 1
    ax.plot(
        xcell,
        zeta[idx, 0, 30, :],
        drawstyle="steps-mid",
        linewidth=0.5,
        color=cc[idx - 5],
        label="{:2d} years".format(years[idx]),
    )
    # layer 2

```

(continues on next page)

(continued from previous page)

```

    ax.plot(
        xcell,
        zeta[idx, 1, 30, :],
        drawstyle="steps-mid",
        linewidth=0.5,
        color=cc[idx - 5],
        label="_None",
    )
ax.plot([-1500, 1500], [-30, -30], color="k", linewidth=1.0)
# legend
plt.legend(loc="lower left")
# axes labels and text
ax.set_xlabel("Horizontal distance, in meters")
ax.set_ylabel("Elevation, in meters")
ax.text(
    0.025,
    0.55,
    "Layer 1",
    transform=ax.transAxes,
    va="center",
    ha="left",
    size="7",
)
ax.text(
    0.025,
    0.45,
    "Layer 2",
    transform=ax.transAxes,
    va="center",
    ha="left",
    size="7",
)
ax.text(
    0.975,
    0.1,
    "Freshwater well withdrawal",
    transform=ax.transAxes,
    va="center",
    ha="right",
    size="8",
)

# third plot
ax = fig.add_subplot(2, 2, 3)
# axes limits
ax.set_xlim(-1500, 1500)
ax.set_ylim(-50, -10)
for idx in range(5, len(years)):
    # layer 1
    ax.plot(
        xcell,
        zeta2[idx, 0, 30, :],

```

(continues on next page)

(continued from previous page)

```

        drawstyle="steps-mid",
        linewidth=0.5,
        color=cc[idx - 5],
        label="{:2d} years".format(years[idx]),
    )
    # layer 2
    ax.plot(
        xcell,
        zeta2[idx, 1, 30, :],
        drawstyle="steps-mid",
        linewidth=0.5,
        color=cc[idx - 5],
        label="_None",
    )
ax.plot([-1500, 1500], [-30, -30], color="k", linewidth=1.0)
# legend
plt.legend(loc="lower left")
# axes labels and text
ax.set_xlabel("Horizontal distance, in meters")
ax.set_ylabel("Elevation, in meters")
ax.text(
    0.025,
    0.55,
    "Layer 1",
    transform=ax.transAxes,
    va="center",
    ha="left",
    size="7",
)
ax.text(
    0.025,
    0.45,
    "Layer 2",
    transform=ax.transAxes,
    va="center",
    ha="left",
    size="7",
)
ax.text(
    0.975,
    0.1,
    "Freshwater and saltwater\nwell withdrawals",
    transform=ax.transAxes,
    va="center",
    ha="right",
    size="8",
)

# fourth plot
ax = fig.add_subplot(2, 2, 4)
# axes limits
ax.set_xlim(0, 30)

```

(continues on next page)

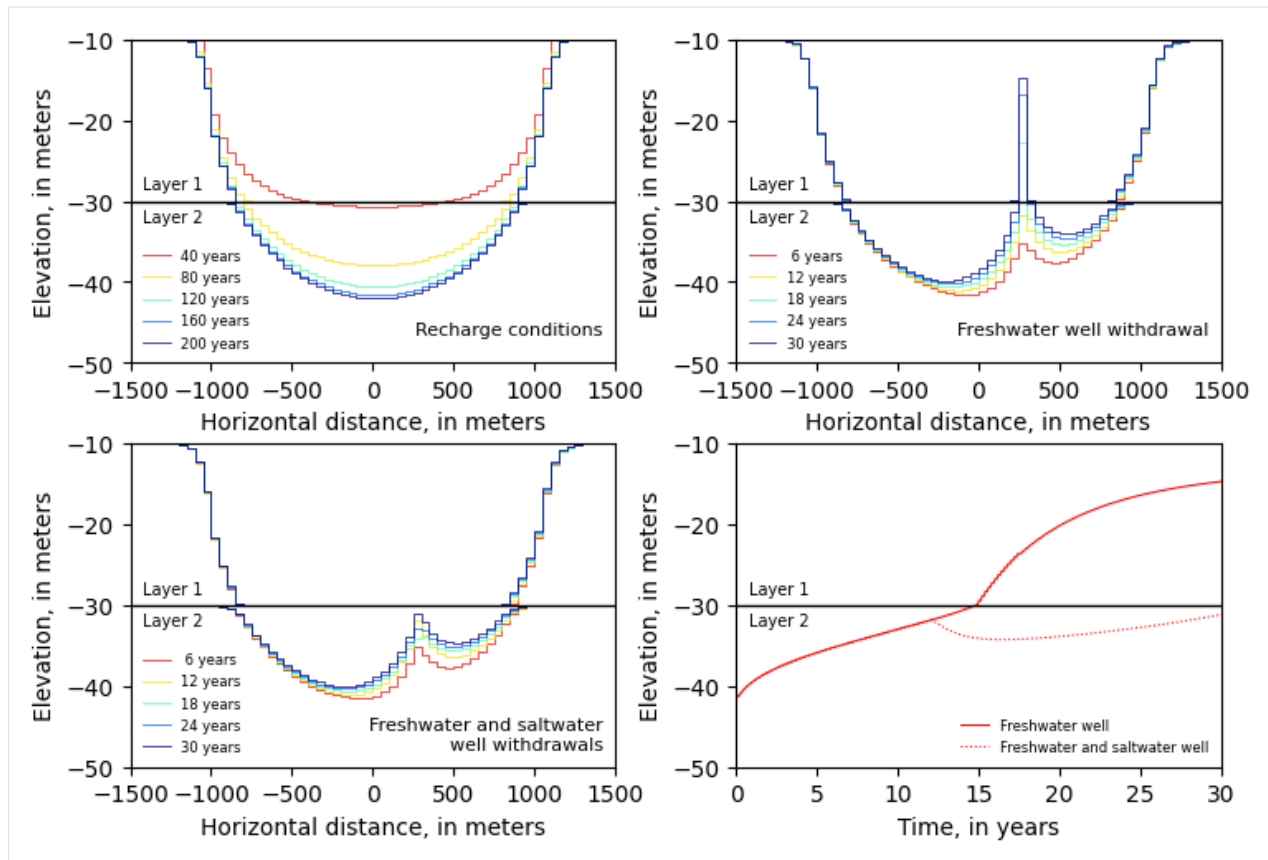
(continued from previous page)

```

ax.set_ylim(-50, -10)
t = zobs["TOTIM"][999:] / 365 - 200.0
tz2 = zobs["layer1_001"][999:]
tz3 = zobs2["layer1_001"][999:]
for i in range(len(t)):
    if zobs["layer2_001"][i + 999] < -30.0 - 0.1:
        tz2[i] = zobs["layer2_001"][i + 999]
    if zobs2["layer2_001"][i + 999] < 20.0 - 0.1:
        tz3[i] = zobs2["layer2_001"][i + 999]
ax.plot(
    t,
    tz2,
    linestyle="solid",
    color="r",
    linewidth=0.75,
    label="Freshwater well",
)
ax.plot(
    t,
    tz3,
    linestyle="dotted",
    color="r",
    linewidth=0.75,
    label="Freshwater and saltwater well",
)
ax.plot([0, 30], [-30, -30], "k", linewidth=1.0, label="_None")
# legend
leg = plt.legend(loc="lower right", numpoints=1)
# axes labels and text
ax.set_xlabel("Time, in years")
ax.set_ylabel("Elevation, in meters")
ax.text(
    0.025,
    0.55,
    "Layer 1",
    transform=ax.transAxes,
    va="center",
    ha="left",
    size="7",
)
ax.text(
    0.025,
    0.45,
    "Layer 2",
    transform=ax.transAxes,
    va="center",
    ha="left",
    size="7",
)

```

[18]: Text(0.025, 0.45, 'Layer 2')



Use ModelCrossSection plotting class and plot_fill_between() method to fill between zeta surfaces.

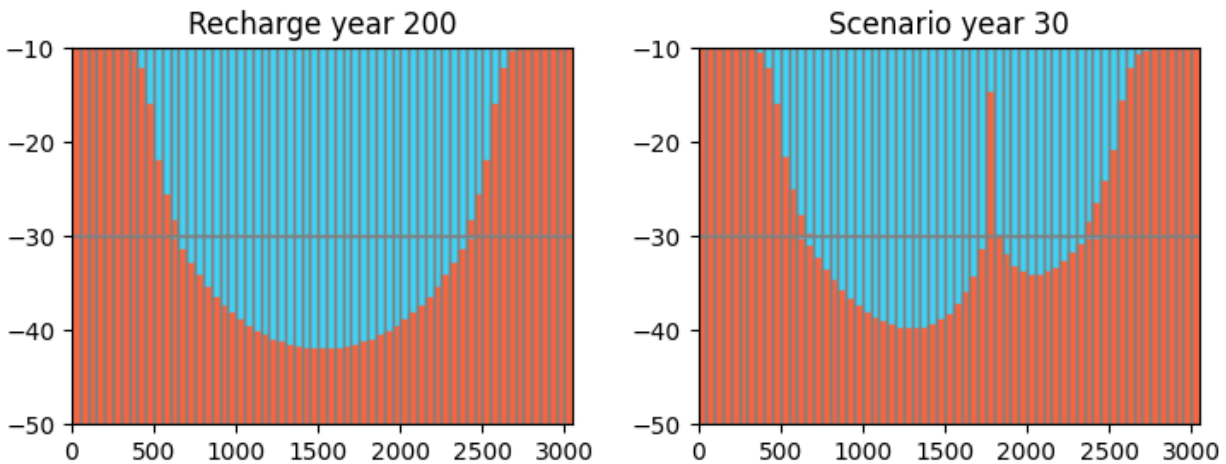
```
[19]: fig = plt.figure(figsize=(fwid, fhgt / 2))
fig.subplots_adjust(
    wspace=0.25, hspace=0.25, left=flft, right=frgt, bottom=fbot, top=ftop
)

colors = ["#40d3f7", "#F76541"]

ax = fig.add_subplot(1, 2, 1)
modelxsect = flopy.plot.PlotCrossSection(
    model=m1, line={"Row": 30}, extent=(0, 3050, -50, -10)
)
modelxsect.plot_fill_between(
    zeta[4, :, :, :], colors=colors, ax=ax, edgecolors="none"
)
linecollection = modelxsect.plot_grid(ax=ax)
ax.set_title("Recharge year {}".format(years[4]))

ax = fig.add_subplot(1, 2, 2)
ax.set_xlim(0, 3050)
ax.set_ylim(-50, -10)
modelxsect.plot_fill_between(zeta[-1, :, :, :], colors=colors, ax=ax)
linecollection = modelxsect.plot_grid(ax=ax)
ax.set_title("Scenario year {}".format(years[-1]))
```

```
[19]: Text(0.5, 1.0, 'Scenario year 30')
```



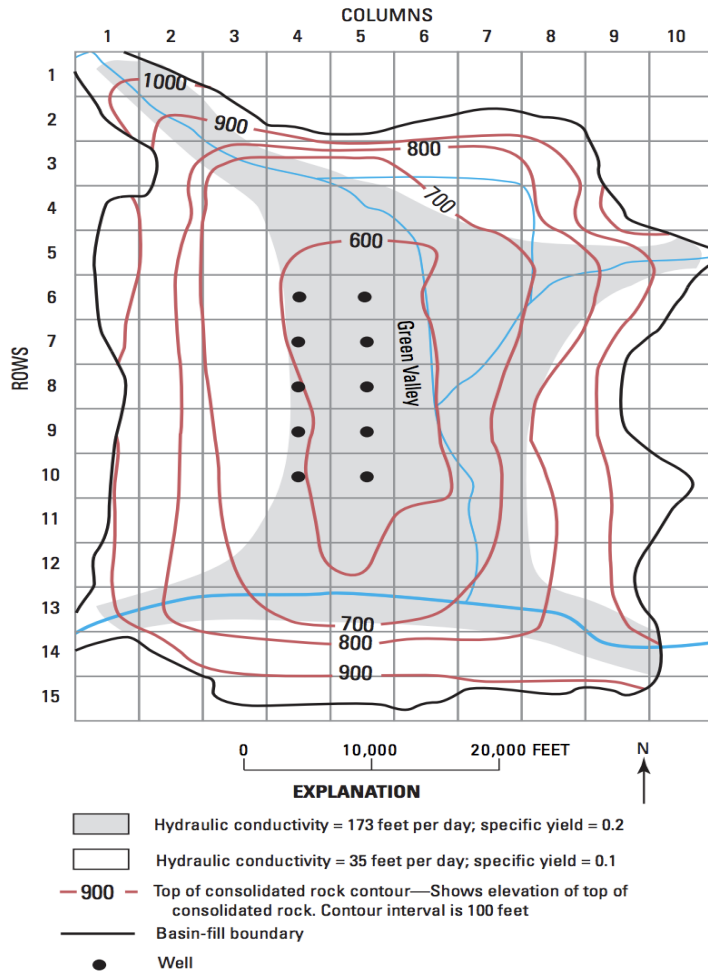
```
[20]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass
```

6.7.8 Unsaturated Zone Flow (UZF) Package demo

Demonstrates functionality of the flopy UZF module using the example from [Niswonger and others \(2006\)](#). This is the same as the SFR example problem from Prudic and others (2004; p. 13–19), except the UZF package replaces the ET and RCH packages.

Problem description:

- Grid dimensions: 1 Layer, 15 Rows, 10 Columns
- Stress periods: 12
- Units are in seconds and days
- Flow package: LPF
- Stress packages: SFR, GHB, UZF
- Solver: SIP



```
[1]: import glob
```

```
[2]: import os
import shutil
import sys
from pathlib import Path
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

proj_root = Path.cwd().parent.parent

import flopy
from flopy.utils import flopy_io

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
```

(continues on next page)

(continued from previous page)

```
print("pandas version: {}".format(pd.__version__))
print("flopy version: {}".format(flopy.__version__))
```

```
3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
pandas version: 2.0.3
flopy version: 3.4.3
```

```
[3]: # Set name of MODFLOW exe
# assumes executable is in users path statement
exe_name = "mf2005"
```

Set up a temporary workspace.

```
[4]: temp_dir = TemporaryDirectory()
path = Path(temp_dir.name)

gpth = proj_root / "examples" / "data" / "mf2005_test" / "UZFtest2.*"
for f in glob.glob(str(gpth)):
    shutil.copy(f, path)
```

Load example dataset, skipping the UZF package.

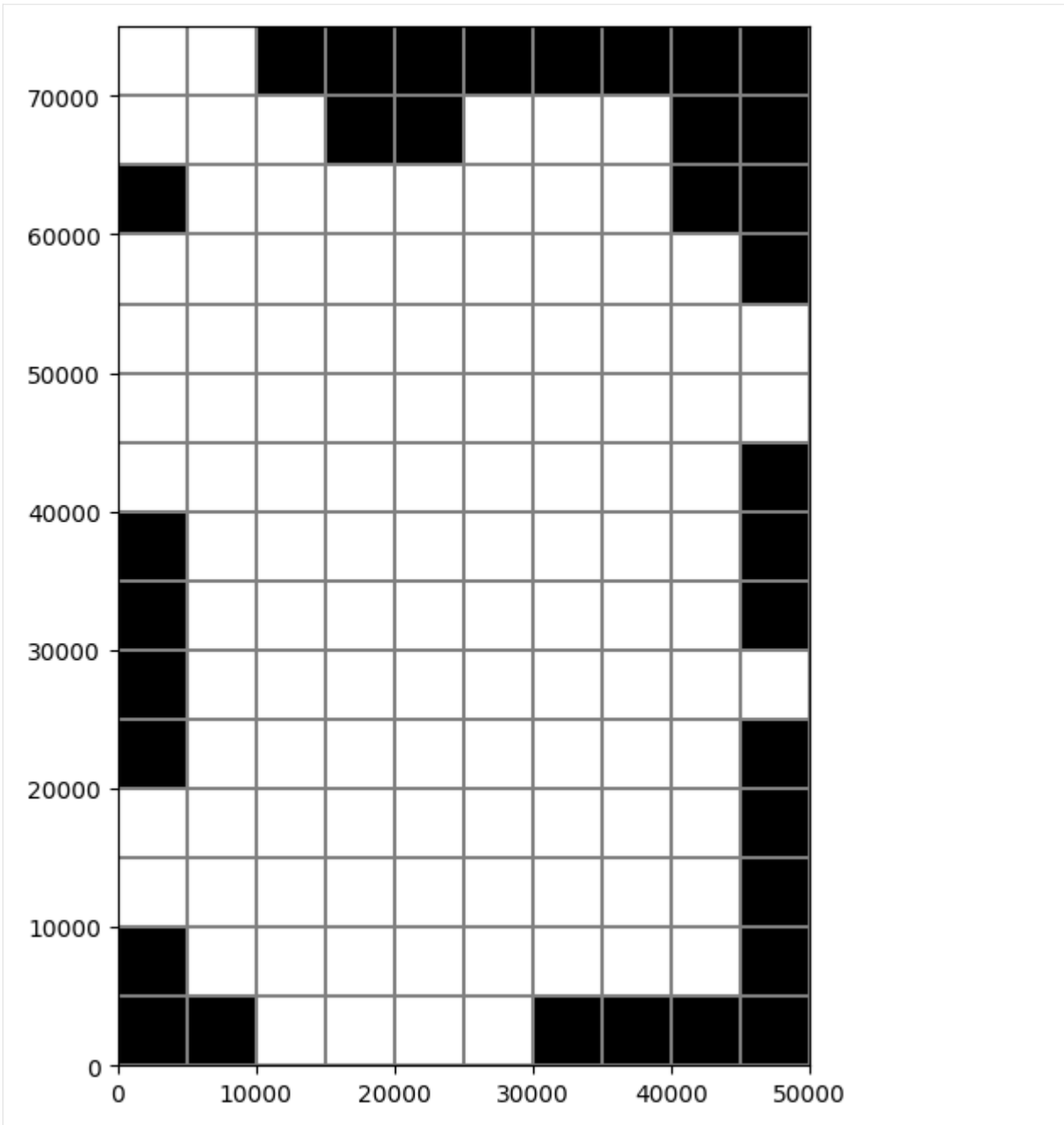
```
[5]: m = flopy.modflow.Modflow.load(
    "UZFtest2.nam",
    version="mf2005",
    exe_name=exe_name,
    model_ws=path,
    load_only=["ghb", "dis", "bas6", "oc", "sip", "lpf", "sfr"],
)
```

Remove previous UZF external file references so they don't conflict with the ones made by FloPy.

```
[6]: rm = [True if ".uz" in f else False for f in m.external_fnames]
m.external_fnames = [f for i, f in enumerate(m.external_fnames) if not rm[i]]
m.external_binflag = [f for i, f in enumerate(m.external_binflag) if not rm[i]]
m.external_output = [f for i, f in enumerate(m.external_output) if not rm[i]]
m.external_units = [f for i, f in enumerate(m.external_output) if not rm[i]]
```

Define the izufbnd array. In the example, the UZF package **izufbnd** array is the same as the ibound.

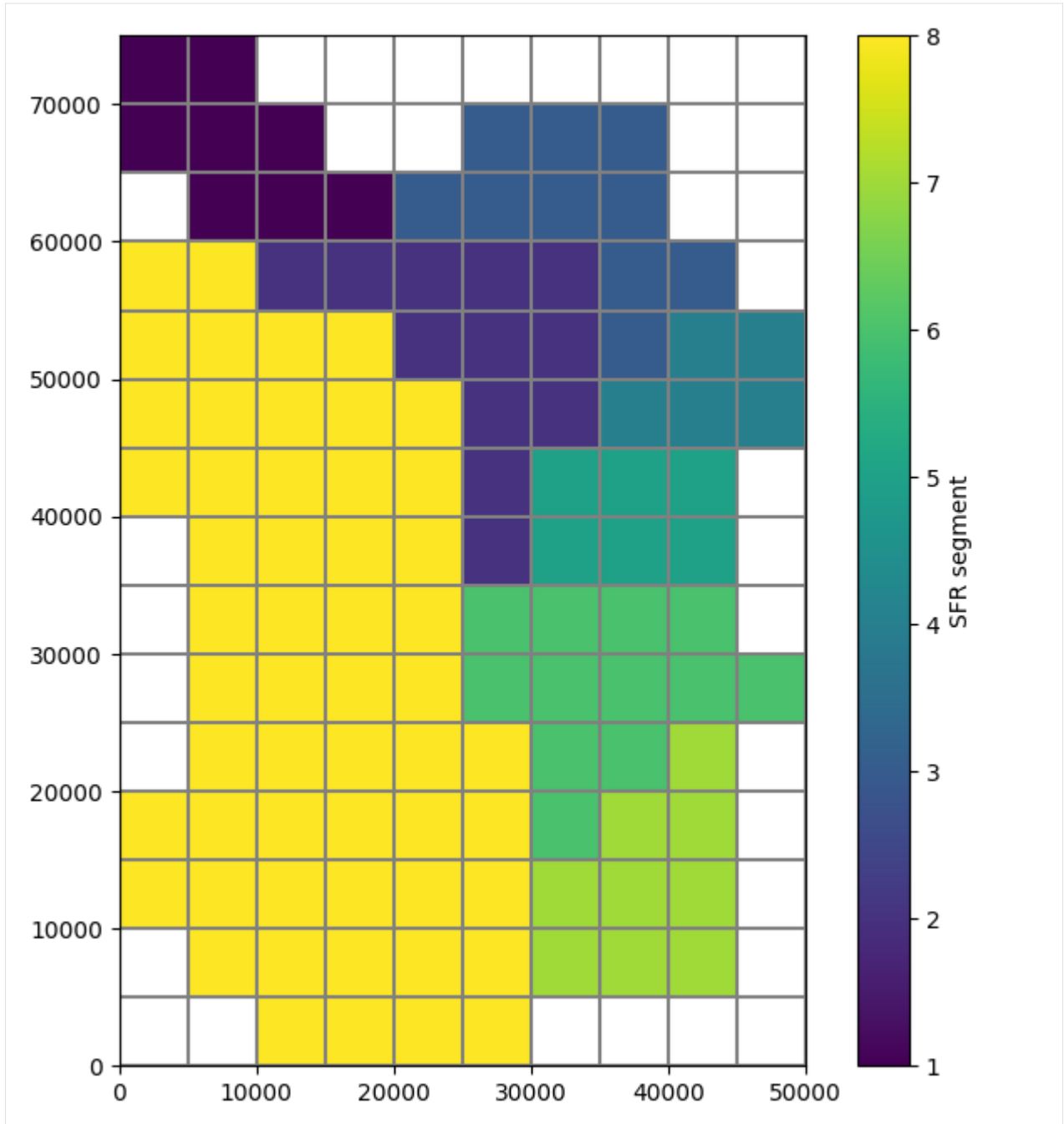
```
[7]: fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mapview = flopy.plot.PlotMapView(model=m)
quadmesh = mapview.plot_ibound()
linecollection = mapview.plot_grid()
```



Read the `irunbnd` array from an external file.

```
[8]: irunbndpth = proj_root / "examples" / "data" / "uzf_examples" / "irunbnd.dat"
      irunbnd = np.loadtxt(irunbndpth)

      fig = plt.figure(figsize=(8, 8))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      mapview = flopy.plot.PlotMapView(model=m)
      irunbndplt = mapview.plot_array(irunbnd)
      plt.colorbar(irunbndplt, ax=ax, label="SFR segment")
      linecollection = mapview.plot_grid()
```

Define the vks (unsaturated zone vertical hydraulic conductivity) array.

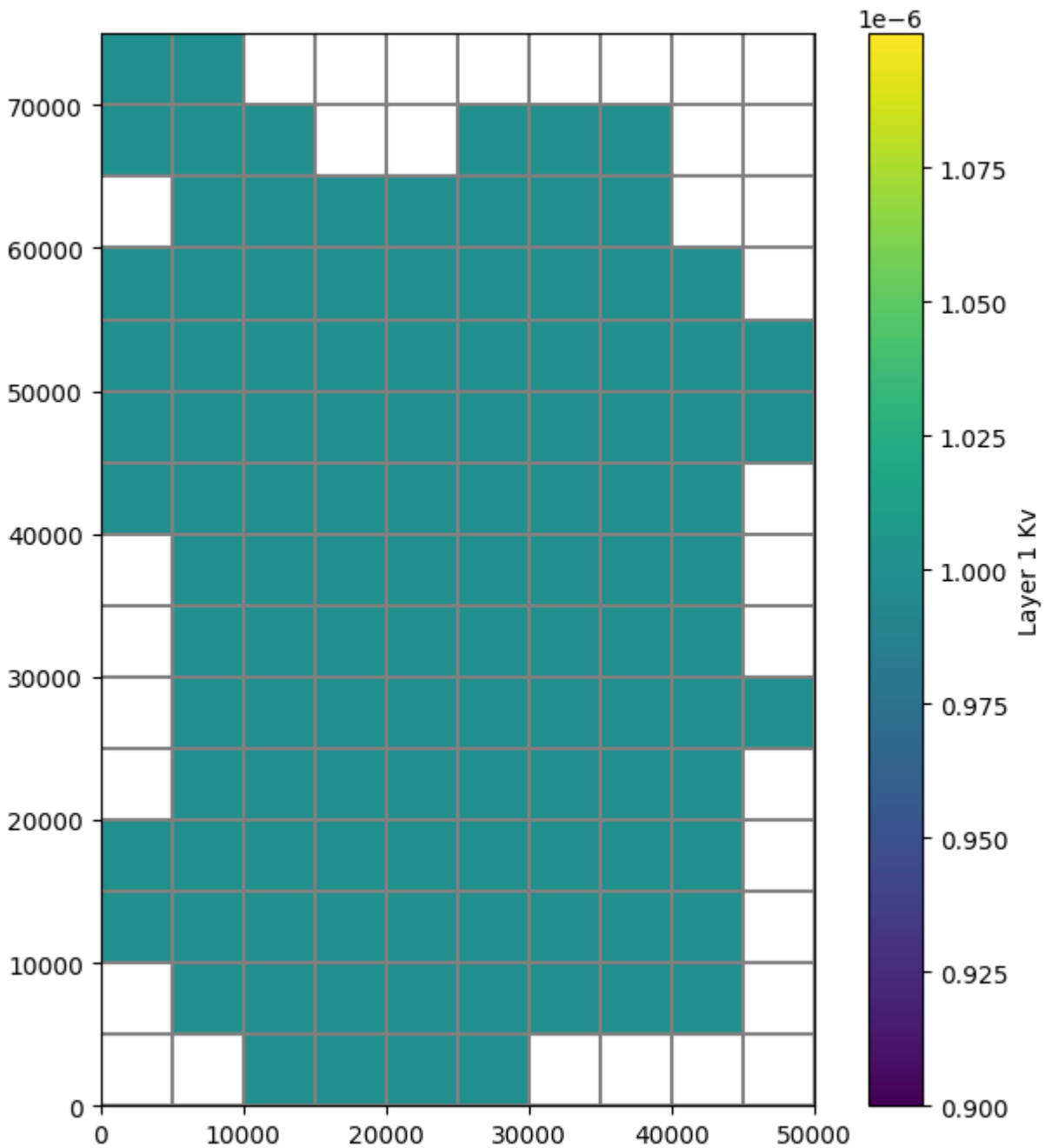
```
[9]: vksbndpth = proj_root / "examples" / "data" / "uzf_examples" / "vks.dat"
vks = np.loadtxt(vksbndpth)

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mapview = flopy.plot.PlotMapView(model=m)
vksplt = mapview.plot_array(vks)
plt.colorbar(vksplt, ax=ax, label="Layer 1 Kv")
```

(continues on next page)

(continued from previous page)

```
linecollection = mapview.plot_grid()
```



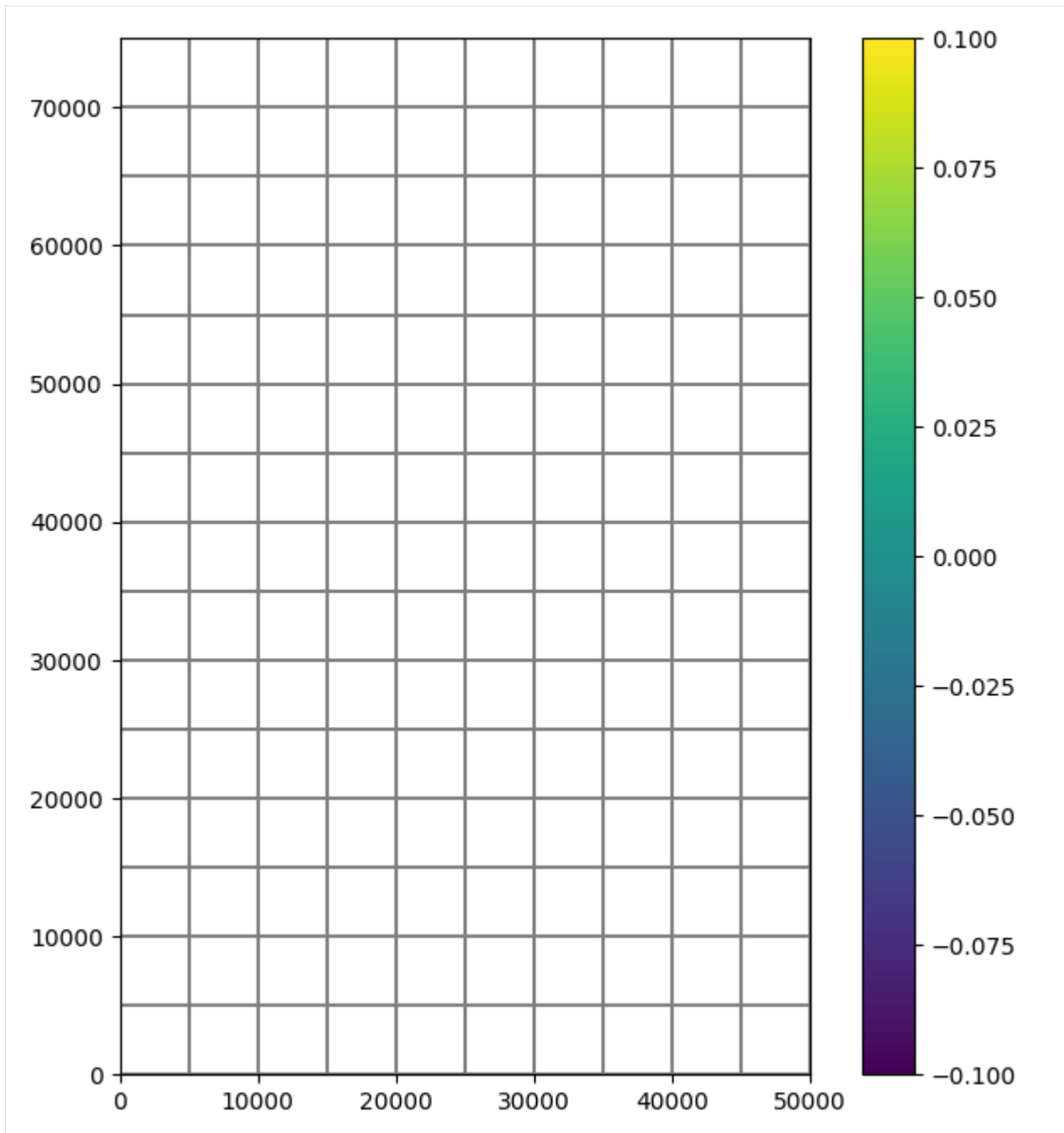
Define the `finf` array: * load infiltration rates from a file into a 3D array * `finf` can be submitted to FloPy as a 3D array, list of 2D arrays, list of numeric values, or single numeric value

```
[10]: m.nrow_ncol_nlay_nper
```

```
[10]: (15, 10, 1, 12)
```

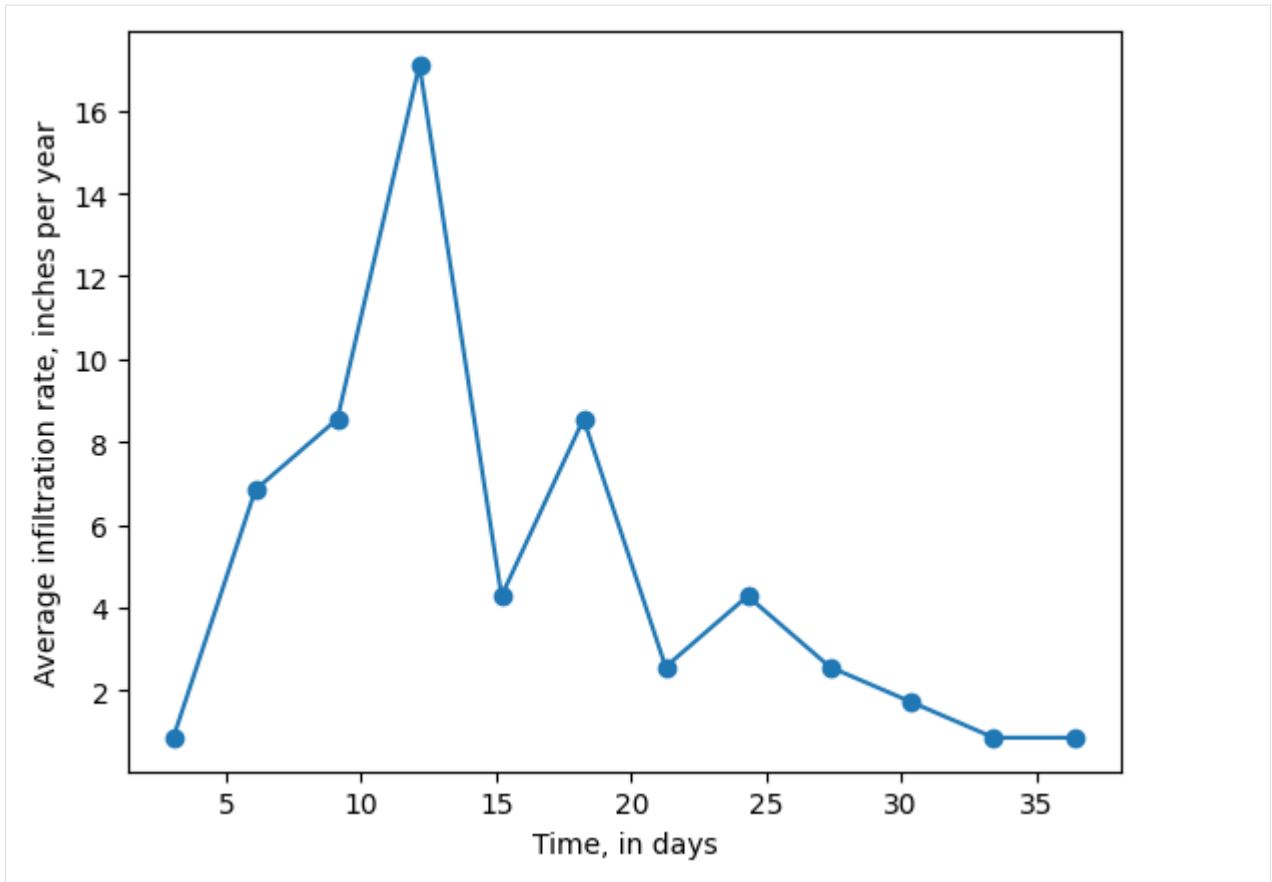
```
[11]: finf = np.loadtxt(
        proj_root / "examples" / "data" / "uzf_examples" / "finf.dat"
    )
    finf = np.reshape(finf, (m.nper, m.nrow, m.ncol))
    finf = {i: finf[i] for i in range(finf.shape[0])}

    # plot using PlotMapView
    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(1, 1, 1, aspect="equal")
    mapview = flopy.plot.PlotMapView(model=m)
    quadmesh = mapview.plot_array(finf[0])
    plt.colorbar(quadmesh)
    linecollection = mapview.plot_grid()
```



```
[12]: plt.plot(
    m.dis.perlen.array.cumsum() / 864600,
    [a.mean() * 86400 * 365 * 12 for a in finf.values()],
    marker="o",
)
plt.xlabel("Time, in days")
plt.ylabel("Average infiltration rate, inches per year")
```

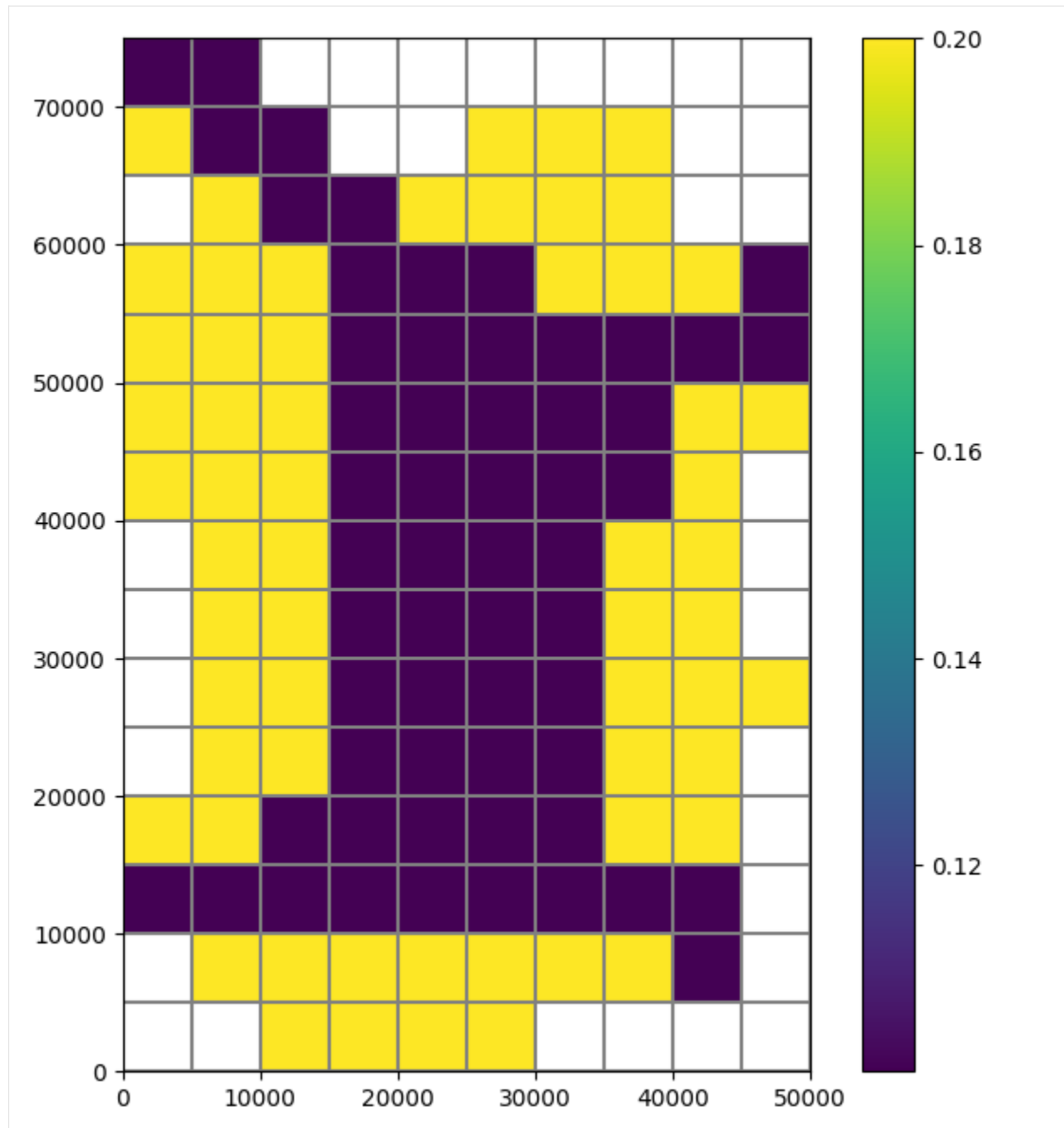
```
[12]: Text(0, 0.5, 'Average infiltration rate, inches per year')
```



Define extwc (extinction water content) array.

```
[13]: extwc = np.loadtxt(
    proj_root / "examples" / "data" / "uzf_examples" / "extwc.dat"
)

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mapview = flopy.plot.PlotMapView(model=m)
quadmesh = mapview.plot_array(extwc)
plt.colorbar(quadmesh)
linecollection = mapview.plot_grid()
```



Set up the gages (observation points)

- supplied as a dictionary keyed by IFTUNIT
- A positive value [of IFTUNIT] is for output of individual cells whereas a negative value is for output that is summed over all model cells.
- values are a list of [IUZROW, IUZCOL, IFTUNIT, IUZOPT]
- IUZROW and IUZCOL are zero based

```
[14]: uzgag = {
    -68: [-68],
    65: [
        2,
        5,
        65,
        1,
    ], # Print time, head, uz thickness and cum. vols of infiltration, recharge,
    ↪ storage, change in storage and ground-water discharge to land surface.
    66: [
        5,
        2,
        66,
        2,
    ], # Same as option 1 except rates of infiltration, recharge, change in storage,
    ↪ and ground-water discharge also are printed.
    67: [9, 4, 67, 3],
} # Prints time, ground-water head, thickness of unsaturated zone, followed by a series
    ↪ of depths and water contents in the unsaturated zone.
```

Make the UZF package.

```
[15]: uzf = flopy.modflow.ModflowUzf1(
    m,
    nuztop=1,
    iuzfopt=1,
    irunflg=1,
    ietflg=1,
    ipakcb=0,
    iuzfcb2=61, # binary output of recharge and groundwater discharge
    ntrail2=25,
    nsets=20,
    surfdep=1.0,
    uzgag=uzgag,
    iuzfbnd=m.bas6.ibound.array,
    irunbnd=irunbnd,
    vks=vks, # saturated vertical hydraulic conductivity of the uz
    finf=finf, # infiltration rates
    eps=3.5, # Brooks-Corey relation of water content to hydraulic conductivity
    ↪ (epsilon)
    thts=0.35, # saturated water content of the uz in units of volume of water to total
    ↪ volume
    pet=5.000000e-08, # potential ET
    extdp=15.0, # ET extinction depth(s)
```

(continues on next page)

(continued from previous page)

```
extwc=extwc, # extinction water content below which ET cannot be removed from the
↳unsaturated zone
    unitnumber=19,
)
```

Write model input files.

```
[16]: m.write_input()
```

Run the model.

```
[17]: success, buff = m.run_model()
assert success, f"{m.name} failed to run"
```

```
FloPy is using the following executable to run the model: ../../home/runner/.local/bin/
↳modflow/mf2005
```

```

                                MODFLOW-2005
      U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
                                Version 1.12.00 2/3/2017
```

```
Using NAME file: UZFtest2.nam
```

```
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:50:18
```

```

Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:      1      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:      2      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:      3      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:      4      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:      5      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:      6      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:      7      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:      8      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:      9      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:     10      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:     11      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:     12      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:     13      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:     14      Ground-Water Flow Eqn.
Solving: Stress period:      2      Time step:     15      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:      1      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:      2      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:      3      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:      4      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:      5      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:      6      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:      7      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:      8      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:      9      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:     10      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:     11      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:     12      Ground-Water Flow Eqn.
Solving: Stress period:      3      Time step:     13      Ground-Water Flow Eqn.
```

(continues on next page)

[illegible]

(continued from previous page)

[illegible]

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 10 Time step: 13 Ground-Water Flow Eqn.
Solving: Stress period: 10 Time step: 14 Ground-Water Flow Eqn.
Solving: Stress period: 10 Time step: 15 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 1 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 2 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 3 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 4 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 5 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 6 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 7 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 8 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 9 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 10 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 11 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 12 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 13 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 14 Ground-Water Flow Eqn.
Solving: Stress period: 11 Time step: 15 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 1 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 2 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 3 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 4 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 5 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 6 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 7 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 8 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 9 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 10 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 11 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 12 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 13 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 14 Ground-Water Flow Eqn.
Solving: Stress period: 12 Time step: 15 Ground-Water Flow Eqn.

```

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:50:21

Elapsed run time: 2.884 Seconds

Normal termination of simulation

Inspecting results

First, look at the budget output.

```

[18]: fpth = path / "UZTest2.uzfcb2.bin"
      avail = os.path.isfile(fpth)
      if avail:
          uzfbdojct = flopy.utils.CellBudgetFile(fpth)
          uzfbdojct.list_records()
      else:
          print("{} is not available".format(fpth))

```

```
(1, 1, b'          GW ET', 10, 15, -1, 4, 2628000., 2628000., 2628000., b'', b'', b'', b
→ '')
(1, 1, b'      UZF RECHARGE', 10, 15, -1, 4, 2628000., 2628000., 2628000., b'', b'', b'', b
→ '')
(1, 1, b' SURFACE LEAKAGE', 10, 15, -1, 4, 2628000., 2628000., 2628000., b'', b'', b'', b
→ '')
(1, 1, b'      HORT+DUNN', 10, 15, -1, 4, 2628000., 2628000., 2628000., b'', b'', b'', b
→ '')
(1, 1, b' STORAGE CHANGE', 10, 15, -1, 4, 2628000., 2628000., 2628000., b'', b'', b'', b
→ '')
(1, 2, b'          GW ET', 10, 15, -1, 4, 82713.07, 82713.07, 2710713., b'', b'', b'', b
→ '')
(1, 2, b'      UZF RECHARGE', 10, 15, -1, 4, 82713.07, 82713.07, 2710713., b'', b'', b'', b
→ '')
(1, 2, b' SURFACE LEAKAGE', 10, 15, -1, 4, 82713.07, 82713.07, 2710713., b'', b'', b'', b
→ '')
(1, 2, b'      HORT+DUNN', 10, 15, -1, 4, 82713.07, 82713.07, 2710713., b'', b'', b'', b
→ '')
(1, 2, b' STORAGE CHANGE', 10, 15, -1, 4, 82713.07, 82713.07, 2710713., b'', b'', b'', b
→ '')
(5, 2, b'          GW ET', 10, 15, -1, 4, 121100.21, 504971.6, 3132971.5, b'', b'', b'',
→ b'')
(5, 2, b'      UZF RECHARGE', 10, 15, -1, 4, 121100.21, 504971.6, 3132971.5, b'', b'', b'',
→ b'')
(5, 2, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 3132971.5, b'', b'', b'',
→ b'')
(5, 2, b'      HORT+DUNN', 10, 15, -1, 4, 121100.21, 504971.6, 3132971.5, b'', b'', b'',
→ b'')
(5, 2, b' STORAGE CHANGE', 10, 15, -1, 4, 121100.21, 504971.6, 3132971.5, b'', b'', b'',
→ b'')
(10, 2, b'          GW ET', 10, 15, -1, 4, 195033.11, 1318233.4, 3946233.2, b'', b'', b'
→ ', b'')
(10, 2, b'      UZF RECHARGE', 10, 15, -1, 4, 195033.11, 1318233.4, 3946233.2, b'', b'', b'
→ ', b'')
(10, 2, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 3946233.2, b'', b'', b'
→ ', b'')
(10, 2, b'      HORT+DUNN', 10, 15, -1, 4, 195033.11, 1318233.4, 3946233.2, b'', b'', b'
→ ', b'')
(10, 2, b' STORAGE CHANGE', 10, 15, -1, 4, 195033.11, 1318233.4, 3946233.2, b'', b'', b'
→ ', b'')
(15, 2, b'          GW ET', 10, 15, -1, 4, 314102.8, 2627999.8, 5256000., b'', b'', b'',
→ b'')
(15, 2, b'      UZF RECHARGE', 10, 15, -1, 4, 314102.8, 2627999.8, 5256000., b'', b'', b'',
→ b'')
(15, 2, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 5256000., b'', b'', b'',
→ b'')
(15, 2, b'      HORT+DUNN', 10, 15, -1, 4, 314102.8, 2627999.8, 5256000., b'', b'', b'',
→ b'')
(15, 2, b' STORAGE CHANGE', 10, 15, -1, 4, 314102.8, 2627999.8, 5256000., b'', b'', b'',
→ b'')
(1, 3, b'          GW ET', 10, 15, -1, 4, 82713.07, 82713.07, 5338713., b'', b'', b'', b
→ '')
(1, 3, b'      UZF RECHARGE', 10, 15, -1, 4, 82713.07, 82713.07, 5338713., b'', b'', b'', b
```

(continues on next page)

(continued from previous page)

```

→ ''')
(1, 3, b' SURFACE LEAKAGE', 10, 15, -1, 4, 82713.07, 82713.07, 5338713., b'', b'', b'', b
→ ''')
(1, 3, b' HORT+DUNN', 10, 15, -1, 4, 82713.07, 82713.07, 5338713., b'', b'', b'', b
→ ''')
(1, 3, b' STORAGE CHANGE', 10, 15, -1, 4, 82713.07, 82713.07, 5338713., b'', b'', b'', b
→ ''')
(5, 3, b' GW ET', 10, 15, -1, 4, 121100.21, 504971.6, 5760971.5, b'', b'', b'',
→ b'')
(5, 3, b' UZF RECHARGE', 10, 15, -1, 4, 121100.21, 504971.6, 5760971.5, b'', b'', b'',
→ b'')
(5, 3, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 5760971.5, b'', b'', b'',
→ b'')
(5, 3, b' HORT+DUNN', 10, 15, -1, 4, 121100.21, 504971.6, 5760971.5, b'', b'', b'',
→ b'')
(5, 3, b' STORAGE CHANGE', 10, 15, -1, 4, 121100.21, 504971.6, 5760971.5, b'', b'', b'',
→ b'')
(10, 3, b' GW ET', 10, 15, -1, 4, 195033.11, 1318233.4, 6574233.5, b'', b'', b'
→ ', b'')
(10, 3, b' UZF RECHARGE', 10, 15, -1, 4, 195033.11, 1318233.4, 6574233.5, b'', b'', b'
→ ', b'')
(10, 3, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 6574233.5, b'', b'', b'
→ ', b'')
(10, 3, b' HORT+DUNN', 10, 15, -1, 4, 195033.11, 1318233.4, 6574233.5, b'', b'', b'
→ ', b'')
(10, 3, b' STORAGE CHANGE', 10, 15, -1, 4, 195033.11, 1318233.4, 6574233.5, b'', b'', b'
→ ', b'')
(15, 3, b' GW ET', 10, 15, -1, 4, 314102.8, 2627999.8, 7884000., b'', b'', b'',
→ b'')
(15, 3, b' UZF RECHARGE', 10, 15, -1, 4, 314102.8, 2627999.8, 7884000., b'', b'', b'',
→ b'')
(15, 3, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 7884000., b'', b'', b'',
→ b'')
(15, 3, b' HORT+DUNN', 10, 15, -1, 4, 314102.8, 2627999.8, 7884000., b'', b'', b'',
→ b'')
(15, 3, b' STORAGE CHANGE', 10, 15, -1, 4, 314102.8, 2627999.8, 7884000., b'', b'', b'',
→ b'')
(5, 4, b' GW ET', 10, 15, -1, 4, 121100.21, 504971.6, 8388972., b'', b'', b'',
→ b'')
(5, 4, b' UZF RECHARGE', 10, 15, -1, 4, 121100.21, 504971.6, 8388972., b'', b'', b'',
→ b'')
(5, 4, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 8388972., b'', b'', b'',
→ b'')
(5, 4, b' HORT+DUNN', 10, 15, -1, 4, 121100.21, 504971.6, 8388972., b'', b'', b'',
→ b'')
(5, 4, b' STORAGE CHANGE', 10, 15, -1, 4, 121100.21, 504971.6, 8388972., b'', b'', b'',
→ b'')
(10, 4, b' GW ET', 10, 15, -1, 4, 195033.11, 1318233.4, 9202233., b'', b'', b'
→ ', b'')
(10, 4, b' UZF RECHARGE', 10, 15, -1, 4, 195033.11, 1318233.4, 9202233., b'', b'', b'
→ ', b'')
(10, 4, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 9202233., b'', b'', b'

```

(continues on next page)

(continued from previous page)

```

→ ', b'')
(10, 4, b'      HORT+DUNN', 10, 15, -1, 4, 195033.11, 1318233.4, 9202233., b'', b'', b'
→ ', b'')
(10, 4, b' STORAGE CHANGE', 10, 15, -1, 4, 195033.11, 1318233.4, 9202233., b'', b'', b'
→ ', b'')
(15, 4, b'      GW ET', 10, 15, -1, 4, 314102.8, 2627999.8, 10511999., b'', b'', b'
→ ', b'')
(15, 4, b'    UZF RECHARGE', 10, 15, -1, 4, 314102.8, 2627999.8, 10511999., b'', b'', b'
→ ', b'')
(15, 4, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 10511999., b'', b'', b'
→ ', b'')
(15, 4, b'      HORT+DUNN', 10, 15, -1, 4, 314102.8, 2627999.8, 10511999., b'', b'', b'
→ ', b'')
(15, 4, b' STORAGE CHANGE', 10, 15, -1, 4, 314102.8, 2627999.8, 10511999., b'', b'', b'
→ ', b'')
(5, 5, b'      GW ET', 10, 15, -1, 4, 121100.21, 504971.6, 11016970., b'', b'', b'',
→ b'')
(5, 5, b'    UZF RECHARGE', 10, 15, -1, 4, 121100.21, 504971.6, 11016970., b'', b'', b'',
→ b'')
(5, 5, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 11016970., b'', b'', b'',
→ b'')
(5, 5, b'      HORT+DUNN', 10, 15, -1, 4, 121100.21, 504971.6, 11016970., b'', b'', b'',
→ b'')
(5, 5, b' STORAGE CHANGE', 10, 15, -1, 4, 121100.21, 504971.6, 11016970., b'', b'', b'',
→ b'')
(10, 5, b'      GW ET', 10, 15, -1, 4, 195033.11, 1318233.4, 11830231., b'', b'', b'
→ ', b'')
(10, 5, b'    UZF RECHARGE', 10, 15, -1, 4, 195033.11, 1318233.4, 11830231., b'', b'', b'
→ ', b'')
(10, 5, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 11830231., b'', b'', b'
→ ', b'')
(10, 5, b'      HORT+DUNN', 10, 15, -1, 4, 195033.11, 1318233.4, 11830231., b'', b'', b'
→ ', b'')
(10, 5, b' STORAGE CHANGE', 10, 15, -1, 4, 195033.11, 1318233.4, 11830231., b'', b'', b'
→ ', b'')
(15, 5, b'      GW ET', 10, 15, -1, 4, 314102.8, 2627999.8, 13139997., b'', b'', b'
→ ', b'')
(15, 5, b'    UZF RECHARGE', 10, 15, -1, 4, 314102.8, 2627999.8, 13139997., b'', b'', b'
→ ', b'')
(15, 5, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 13139997., b'', b'', b'
→ ', b'')
(15, 5, b'      HORT+DUNN', 10, 15, -1, 4, 314102.8, 2627999.8, 13139997., b'', b'', b'
→ ', b'')
(15, 5, b' STORAGE CHANGE', 10, 15, -1, 4, 314102.8, 2627999.8, 13139997., b'', b'', b'
→ ', b'')
(5, 6, b'      GW ET', 10, 15, -1, 4, 121100.21, 504971.6, 13644968., b'', b'', b'',
→ b'')
(5, 6, b'    UZF RECHARGE', 10, 15, -1, 4, 121100.21, 504971.6, 13644968., b'', b'', b'',
→ b'')
(5, 6, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 13644968., b'', b'', b'',
→ b'')
(5, 6, b'      HORT+DUNN', 10, 15, -1, 4, 121100.21, 504971.6, 13644968., b'', b'', b'',

```

(continues on next page)

(continued from previous page)

```

→ b'')
(5, 6, b' STORAGE CHANGE', 10, 15, -1, 4, 121100.21, 504971.6, 13644968., b'', b'', b'',
→ b'')
(10, 6, b' GW ET', 10, 15, -1, 4, 195033.11, 1318233.4, 14458229., b'', b'', b'
→ ', b'')
(10, 6, b' UZF RECHARGE', 10, 15, -1, 4, 195033.11, 1318233.4, 14458229., b'', b'', b'
→ ', b'')
(10, 6, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 14458229., b'', b'', b'
→ ', b'')
(10, 6, b' HORT+DUNN', 10, 15, -1, 4, 195033.11, 1318233.4, 14458229., b'', b'', b'
→ ', b'')
(10, 6, b' STORAGE CHANGE', 10, 15, -1, 4, 195033.11, 1318233.4, 14458229., b'', b'', b'
→ ', b'')
(15, 6, b' GW ET', 10, 15, -1, 4, 314102.8, 2627999.8, 15767995., b'', b'', b'
→ ', b'')
(15, 6, b' UZF RECHARGE', 10, 15, -1, 4, 314102.8, 2627999.8, 15767995., b'', b'', b'
→ ', b'')
(15, 6, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 15767995., b'', b'', b'
→ ', b'')
(15, 6, b' HORT+DUNN', 10, 15, -1, 4, 314102.8, 2627999.8, 15767995., b'', b'', b'
→ ', b'')
(15, 6, b' STORAGE CHANGE', 10, 15, -1, 4, 314102.8, 2627999.8, 15767995., b'', b'', b'
→ ', b'')
(5, 7, b' GW ET', 10, 15, -1, 4, 121100.21, 504971.6, 16272966., b'', b'', b'',
→ b'')
(5, 7, b' UZF RECHARGE', 10, 15, -1, 4, 121100.21, 504971.6, 16272966., b'', b'', b'',
→ b'')
(5, 7, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 16272966., b'', b'', b'',
→ b'')
(5, 7, b' HORT+DUNN', 10, 15, -1, 4, 121100.21, 504971.6, 16272966., b'', b'', b'',
→ b'')
(5, 7, b' STORAGE CHANGE', 10, 15, -1, 4, 121100.21, 504971.6, 16272966., b'', b'', b'',
→ b'')
(10, 7, b' GW ET', 10, 15, -1, 4, 195033.11, 1318233.4, 17086228., b'', b'', b'
→ ', b'')
(10, 7, b' UZF RECHARGE', 10, 15, -1, 4, 195033.11, 1318233.4, 17086228., b'', b'', b'
→ ', b'')
(10, 7, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 17086228., b'', b'', b'
→ ', b'')
(10, 7, b' HORT+DUNN', 10, 15, -1, 4, 195033.11, 1318233.4, 17086228., b'', b'', b'
→ ', b'')
(10, 7, b' STORAGE CHANGE', 10, 15, -1, 4, 195033.11, 1318233.4, 17086228., b'', b'', b'
→ ', b'')
(15, 7, b' GW ET', 10, 15, -1, 4, 314102.8, 2627999.8, 18395994., b'', b'', b'
→ ', b'')
(15, 7, b' UZF RECHARGE', 10, 15, -1, 4, 314102.8, 2627999.8, 18395994., b'', b'', b'
→ ', b'')
(15, 7, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 18395994., b'', b'', b'
→ ', b'')
(15, 7, b' HORT+DUNN', 10, 15, -1, 4, 314102.8, 2627999.8, 18395994., b'', b'', b'
→ ', b'')
(15, 7, b' STORAGE CHANGE', 10, 15, -1, 4, 314102.8, 2627999.8, 18395994., b'', b'', b'

```

(continues on next page)

(continued from previous page)

```

→ ', b'')
(5, 8, b'          GW ET', 10, 15, -1, 4, 121100.21, 504971.6, 18900966., b'', b'', b'',
→ b'')
(5, 8, b'    UZF RECHARGE', 10, 15, -1, 4, 121100.21, 504971.6, 18900966., b'', b'', b'',
→ b'')
(5, 8, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 18900966., b'', b'', b'',
→ b'')
(5, 8, b'    HORT+DUNN', 10, 15, -1, 4, 121100.21, 504971.6, 18900966., b'', b'', b'',
→ b'')
(5, 8, b' STORAGE CHANGE', 10, 15, -1, 4, 121100.21, 504971.6, 18900966., b'', b'', b'',
→ b'')
(10, 8, b'          GW ET', 10, 15, -1, 4, 195033.11, 1318233.4, 19714228., b'', b'', b'
→ ', b'')
(10, 8, b'    UZF RECHARGE', 10, 15, -1, 4, 195033.11, 1318233.4, 19714228., b'', b'', b'
→ ', b'')
(10, 8, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 19714228., b'', b'', b'
→ ', b'')
(10, 8, b'    HORT+DUNN', 10, 15, -1, 4, 195033.11, 1318233.4, 19714228., b'', b'', b'
→ ', b'')
(10, 8, b' STORAGE CHANGE', 10, 15, -1, 4, 195033.11, 1318233.4, 19714228., b'', b'', b'
→ ', b'')
(15, 8, b'          GW ET', 10, 15, -1, 4, 314102.8, 2627999.8, 21023994., b'', b'', b'
→ ', b'')
(15, 8, b'    UZF RECHARGE', 10, 15, -1, 4, 314102.8, 2627999.8, 21023994., b'', b'', b'
→ ', b'')
(15, 8, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 21023994., b'', b'', b'
→ ', b'')
(15, 8, b'    HORT+DUNN', 10, 15, -1, 4, 314102.8, 2627999.8, 21023994., b'', b'', b'
→ ', b'')
(15, 8, b' STORAGE CHANGE', 10, 15, -1, 4, 314102.8, 2627999.8, 21023994., b'', b'', b'
→ ', b'')
(5, 9, b'          GW ET', 10, 15, -1, 4, 121100.21, 504971.6, 21528966., b'', b'', b'',
→ b'')
(5, 9, b'    UZF RECHARGE', 10, 15, -1, 4, 121100.21, 504971.6, 21528966., b'', b'', b'',
→ b'')
(5, 9, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 21528966., b'', b'', b'',
→ b'')
(5, 9, b'    HORT+DUNN', 10, 15, -1, 4, 121100.21, 504971.6, 21528966., b'', b'', b'',
→ b'')
(5, 9, b' STORAGE CHANGE', 10, 15, -1, 4, 121100.21, 504971.6, 21528966., b'', b'', b'',
→ b'')
(10, 9, b'          GW ET', 10, 15, -1, 4, 195033.11, 1318233.4, 22342228., b'', b'', b'
→ ', b'')
(10, 9, b'    UZF RECHARGE', 10, 15, -1, 4, 195033.11, 1318233.4, 22342228., b'', b'', b'
→ ', b'')
(10, 9, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 22342228., b'', b'', b'
→ ', b'')
(10, 9, b'    HORT+DUNN', 10, 15, -1, 4, 195033.11, 1318233.4, 22342228., b'', b'', b'
→ ', b'')
(10, 9, b' STORAGE CHANGE', 10, 15, -1, 4, 195033.11, 1318233.4, 22342228., b'', b'', b'
→ ', b'')
(15, 9, b'          GW ET', 10, 15, -1, 4, 314102.8, 2627999.8, 23651994., b'', b'', b'

```

(continues on next page)

(continued from previous page)

```

→', b'')
(15, 9, b'      UZF RECHARGE', 10, 15, -1, 4, 314102.8, 2627999.8, 23651994., b'', b'', b'
→', b'')
(15, 9, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 23651994., b'', b'', b'
→', b'')
(15, 9, b'      HORT+DUNN', 10, 15, -1, 4, 314102.8, 2627999.8, 23651994., b'', b'', b'
→', b'')
(15, 9, b' STORAGE CHANGE', 10, 15, -1, 4, 314102.8, 2627999.8, 23651994., b'', b'', b'
→', b'')
(5, 10, b'      GW ET', 10, 15, -1, 4, 121100.21, 504971.6, 24156966., b'', b'', b'
→', b'')
(5, 10, b'      UZF RECHARGE', 10, 15, -1, 4, 121100.21, 504971.6, 24156966., b'', b'', b'
→', b'')
(5, 10, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 24156966., b'', b'', b'
→', b'')
(5, 10, b'      HORT+DUNN', 10, 15, -1, 4, 121100.21, 504971.6, 24156966., b'', b'', b'
→', b'')
(5, 10, b' STORAGE CHANGE', 10, 15, -1, 4, 121100.21, 504971.6, 24156966., b'', b'', b'
→', b'')
(10, 10, b'      GW ET', 10, 15, -1, 4, 195033.11, 1318233.4, 24970228., b'', b'', b'
→', b'')
(10, 10, b'      UZF RECHARGE', 10, 15, -1, 4, 195033.11, 1318233.4, 24970228., b'', b'', b'
→', b'')
(10, 10, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 24970228., b'', b'', b'
→', b'')
(10, 10, b'      HORT+DUNN', 10, 15, -1, 4, 195033.11, 1318233.4, 24970228., b'', b'', b'
→', b'')
(10, 10, b' STORAGE CHANGE', 10, 15, -1, 4, 195033.11, 1318233.4, 24970228., b'', b'', b'
→', b'')
(15, 10, b'      GW ET', 10, 15, -1, 4, 314102.8, 2627999.8, 26279994., b'', b'', b'
→', b'')
(15, 10, b'      UZF RECHARGE', 10, 15, -1, 4, 314102.8, 2627999.8, 26279994., b'', b'', b'
→', b'')
(15, 10, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 26279994., b'', b'', b'
→', b'')
(15, 10, b'      HORT+DUNN', 10, 15, -1, 4, 314102.8, 2627999.8, 26279994., b'', b'', b'
→', b'')
(15, 10, b' STORAGE CHANGE', 10, 15, -1, 4, 314102.8, 2627999.8, 26279994., b'', b'', b'
→', b'')
(5, 11, b'      GW ET', 10, 15, -1, 4, 121100.21, 504971.6, 26784966., b'', b'', b'
→', b'')
(5, 11, b'      UZF RECHARGE', 10, 15, -1, 4, 121100.21, 504971.6, 26784966., b'', b'', b'
→', b'')
(5, 11, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 26784966., b'', b'', b'
→', b'')
(5, 11, b'      HORT+DUNN', 10, 15, -1, 4, 121100.21, 504971.6, 26784966., b'', b'', b'
→', b'')
(5, 11, b' STORAGE CHANGE', 10, 15, -1, 4, 121100.21, 504971.6, 26784966., b'', b'', b'
→', b'')
(10, 11, b'      GW ET', 10, 15, -1, 4, 195033.11, 1318233.4, 27598228., b'', b'', b'
→', b'')
(10, 11, b'      UZF RECHARGE', 10, 15, -1, 4, 195033.11, 1318233.4, 27598228., b'', b'', b'

```

(continues on next page)

(continued from previous page)

```

→'', b'')
(10, 11, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 27598228., b'', b'', b
→'', b'')
(10, 11, b'          HORT+DUNN', 10, 15, -1, 4, 195033.11, 1318233.4, 27598228., b'', b'', b
→'', b'')
(10, 11, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 27598228., b'', b'', b
→'', b'')
(15, 11, b'          GW ET', 10, 15, -1, 4, 314102.8, 2627999.8, 28907994., b'', b'', b'
→'', b'')
(15, 11, b'      UZF RECHARGE', 10, 15, -1, 4, 314102.8, 2627999.8, 28907994., b'', b'', b'
→'', b'')
(15, 11, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 28907994., b'', b'', b'
→'', b'')
(15, 11, b'          HORT+DUNN', 10, 15, -1, 4, 314102.8, 2627999.8, 28907994., b'', b'', b'
→'', b'')
(15, 11, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 28907994., b'', b'', b'
→'', b'')
(5, 12, b'          GW ET', 10, 15, -1, 4, 121100.21, 504971.6, 29412966., b'', b'', b'
→'', b'')
(5, 12, b'      UZF RECHARGE', 10, 15, -1, 4, 121100.21, 504971.6, 29412966., b'', b'', b'
→'', b'')
(5, 12, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 29412966., b'', b'', b'
→'', b'')
(5, 12, b'          HORT+DUNN', 10, 15, -1, 4, 121100.21, 504971.6, 29412966., b'', b'', b'
→'', b'')
(5, 12, b' SURFACE LEAKAGE', 10, 15, -1, 4, 121100.21, 504971.6, 29412966., b'', b'', b'
→'', b'')
(10, 12, b'          GW ET', 10, 15, -1, 4, 195033.11, 1318233.4, 30226228., b'', b'', b
→'', b'')
(10, 12, b'      UZF RECHARGE', 10, 15, -1, 4, 195033.11, 1318233.4, 30226228., b'', b'', b
→'', b'')
(10, 12, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 30226228., b'', b'', b
→'', b'')
(10, 12, b'          HORT+DUNN', 10, 15, -1, 4, 195033.11, 1318233.4, 30226228., b'', b'', b
→'', b'')
(10, 12, b' SURFACE LEAKAGE', 10, 15, -1, 4, 195033.11, 1318233.4, 30226228., b'', b'', b
→'', b'')
(15, 12, b'          GW ET', 10, 15, -1, 4, 314102.8, 2627999.8, 31535994., b'', b'', b'
→'', b'')
(15, 12, b'      UZF RECHARGE', 10, 15, -1, 4, 314102.8, 2627999.8, 31535994., b'', b'', b'
→'', b'')
(15, 12, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 31535994., b'', b'', b'
→'', b'')
(15, 12, b'          HORT+DUNN', 10, 15, -1, 4, 314102.8, 2627999.8, 31535994., b'', b'', b'
→'', b'')
(15, 12, b' SURFACE LEAKAGE', 10, 15, -1, 4, 314102.8, 2627999.8, 31535994., b'', b'', b'
→'', b'')

```

```

[19]: if success and avail:
      r = uzfbdojct.get_data(text="UZF RECHARGE")
      et = uzfbdojct.get_data(text="GW ET")

```

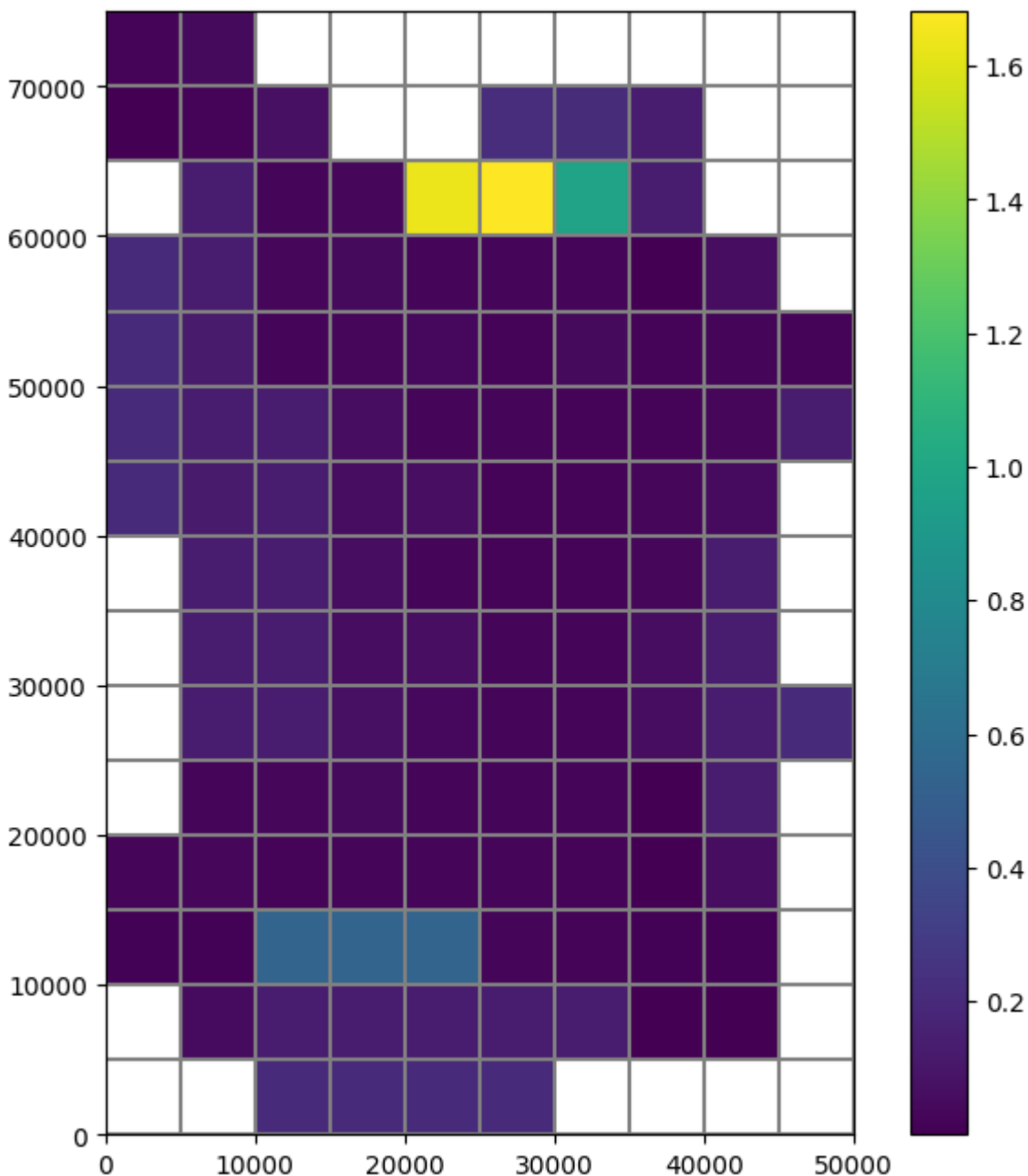
(continues on next page)

(continued from previous page)

```

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mapview = flopy.plot.PlotMapView(model=m)
quadmesh = mapview.plot_array(r[6])
plt.colorbar(quadmesh)
linecollection = mapview.plot_grid()

```



```

[20]: if avail:
    rtot = [rp.sum() for rp in r]
    ettot = [etp.sum() for etp in et]

```

(continues on next page)

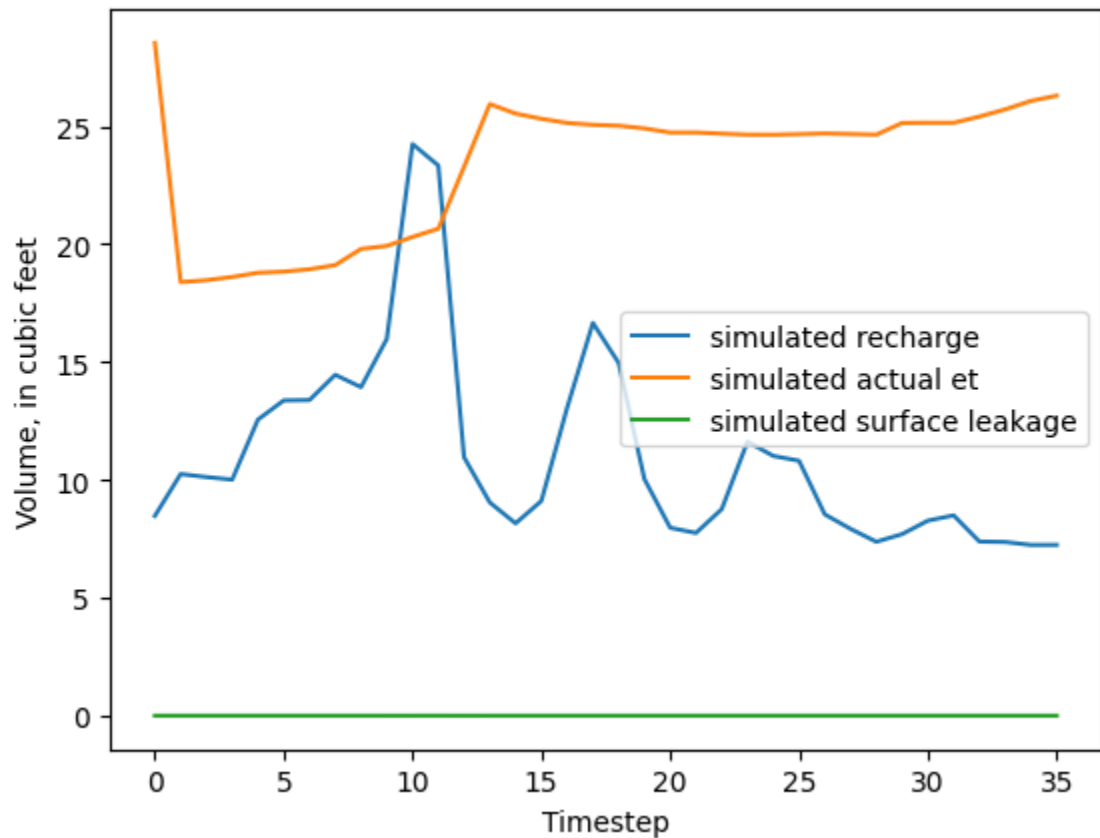
(continued from previous page)

```

sltot = [sl.sum() for sl in uzfbdojct.get_data(text="SURFACE LEAKAGE")]

plt.plot(rtot, label="simulated recharge")
plt.plot(np.abs(ettot), label="simulated actual et")
plt.plot(np.abs(sltot), label="simulated surface leakage")
plt.xlabel("Timestep")
plt.ylabel("Volume, in cubic feet")
plt.legend()

```



Look at the gages.

```

[21]: fpth = path / "UZFtest2.uzf68.out"
      avail = os.path.isfile(fpth)
      if avail:
          dtype = [
              ("TIME", float),
              ("APPLIED-INFIL", float),
              ("RUNOFF", float),
              ("ACTUAL-INFIL", float),
              ("SURFACE-LEAK", float),
              ("UZ-ET", float),
              ("GW-ET", float),
              ("UZSTOR-CHANGE", float),
              ("RECHARGE", float),

```

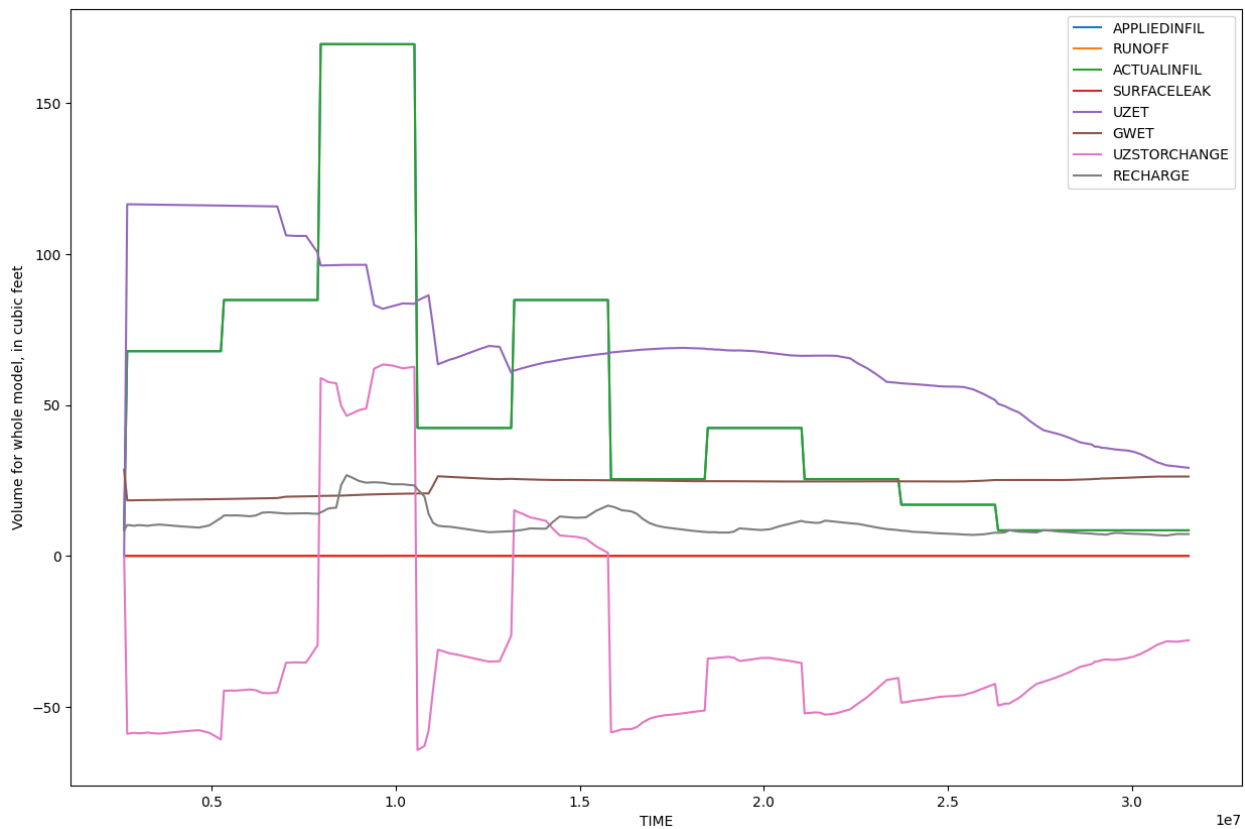
(continues on next page)

(continued from previous page)

```

]
# read data from file
df = np.genfromtxt(fpth, skip_header=3, dtype=dtype)
# convert numpy recarray to pandas dataframe
df = pd.DataFrame(data=df)
# set index to the time column
df.set_index(["TIME"], inplace=True)
# plot the data
ax = df.plot(legend=False, figsize=(15, 10))
patches, labels = ax.get_legend_handles_labels()
ax.legend(patches, labels, loc=1)
ax.set_ylabel("Volume for whole model, in cubic feet")

```



Plot water content profile through time at row 10, column 5.

```

[22]: fpth = path / "UZFtest2.uzf67.out"
avail = os.path.isfile(fpth)
if avail:
    data = []
    with open(fpth) as input:
        for i in range(3):
            next(input)
        for line in input:
            line = line.strip().split()
            if len(line) == 6:

```

(continues on next page)

(continued from previous page)

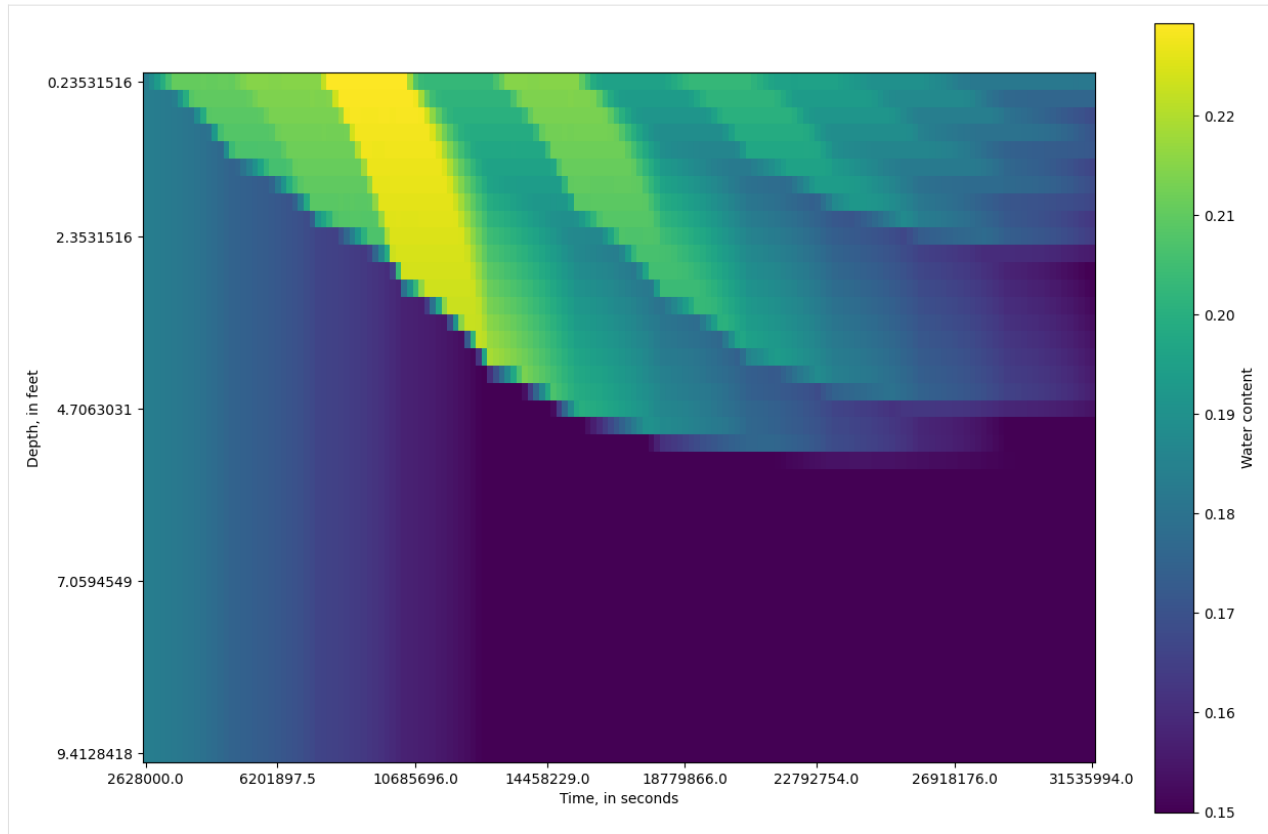
```
        layer = int(line.pop(0))
        time = float(line.pop(0))
        head = float(line.pop(0))
        uzthick = float(line.pop(0))
        depth = float(line.pop(0))
        watercontent = float(line.pop(0))
        data.append([layer, time, head, uzthick, depth, watercontent])
```

```
[23]: if avail:
        df3 = pd.DataFrame(
            data,
            columns=["layer", "time", "head", "uzthick", "depth", "watercontent"],
        )
        df3.head(41)
```

```
[24]: if avail:
        wc = df3.watercontent.values.reshape(len(df3.time.unique()), 40).T
        wc = pd.DataFrame(wc, columns=df3.time.unique(), index=df3.depth[0:40])
        wc.head()
```

```
[25]: if avail:
        fig, ax = plt.subplots(figsize=(15, 10))
        plt.imshow(wc, interpolation="None")
        ax.set_aspect(3)
        r, c = wc.shape
        xcol_locs = np.linspace(0, c - 1, 8, dtype=int)
        ycol_locs = np.linspace(0, r - 1, 5, dtype=int)
        ax.set_xticks(xcol_locs)

        xlabels = wc.columns
        ax.set_xticklabels(xlabels[xcol_locs])
        ax.set_ylabel("Depth, in feet")
        ax.set_yticks(ycol_locs)
        ax.set_yticklabels(wc.index[ycol_locs])
        ax.set_xlabel("Time, in seconds")
        plt.colorbar(label="Water content")
```



Clean up the temporary directory.

```
[26]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.8 MODPATH examples

6.8.1 Working with MODPATH 6

This notebook demonstrates forward and backward tracking with MODPATH. The notebook also shows how to create subsets of pathline and endpoint information, plot MODPATH results on ModelMap objects, and export endpoints and pathlines as shapefiles.

```
[1]: import glob
      import os
      import shutil
```

```
[2]: import sys
      from tempfile import TemporaryDirectory

      import matplotlib as mpl
```

(continues on next page)

(continued from previous page)

```

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join(".", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("pandas version: {}".format(pd.__version__))
print("flopy version: {}".format(flopy.__version__))

```

```

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
pandas version: 2.0.3
flopy version: 3.4.3

```

Load the MODFLOW model, then switch to a temporary working directory.

```

[3]: from pathlib import Path

# temporary directory
temp_dir = TemporaryDirectory()
model_ws = temp_dir.name

model_path = Path.cwd().parent.parent / "examples" / "data" / "mp6"
mffiles = list(model_path.glob("EXAMPLE.*"))

m = flopy.modflow.Modflow.load("EXAMPLE.nam", model_ws=model_path)

hdsfile = flopy.utils.HeadFile(os.path.join(model_path, "EXAMPLE.HED"))
hdsfile.get_kstpker()

hds = hdsfile.get_data(kstpker=(0, 2))

```

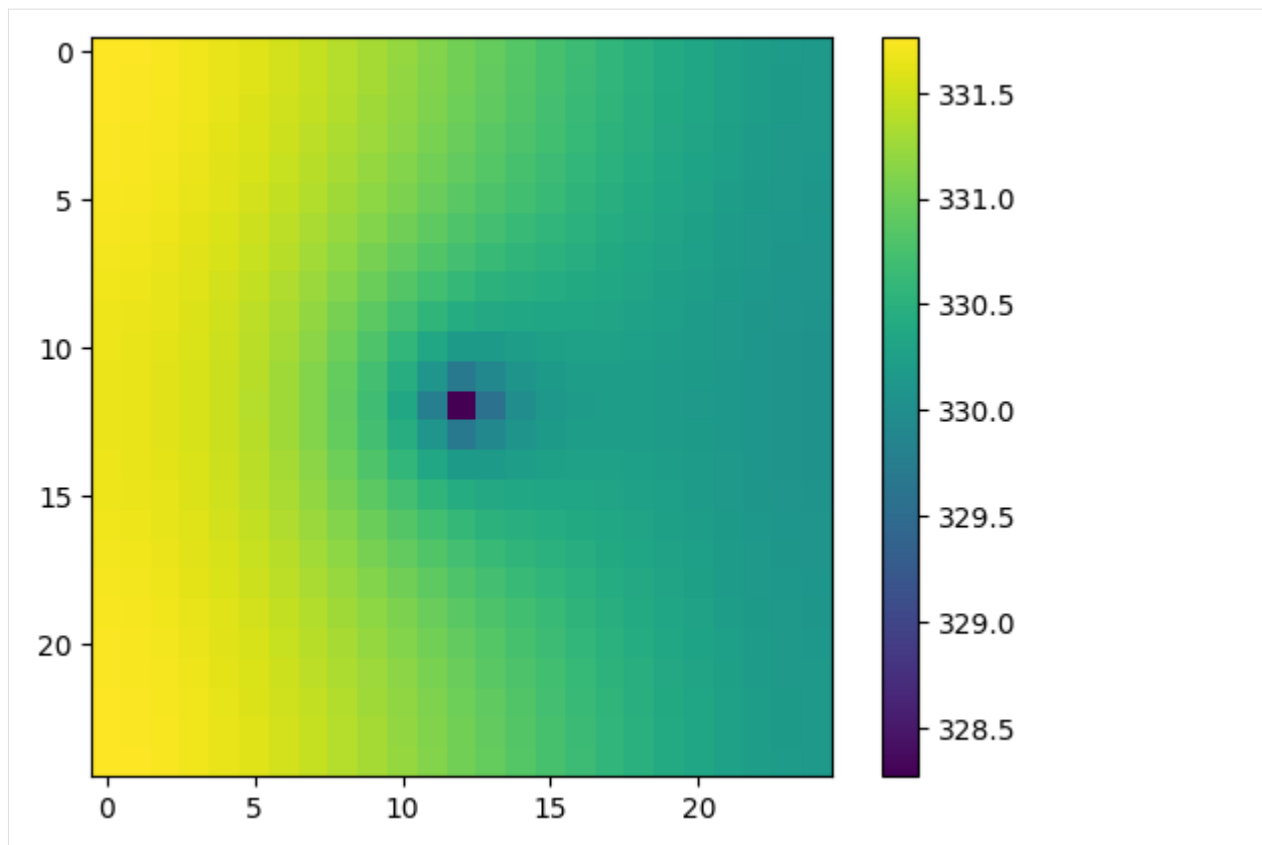
Plot RIV bc and head results.

```

[4]: plt.imshow(hds[4, :, :])
plt.colorbar()

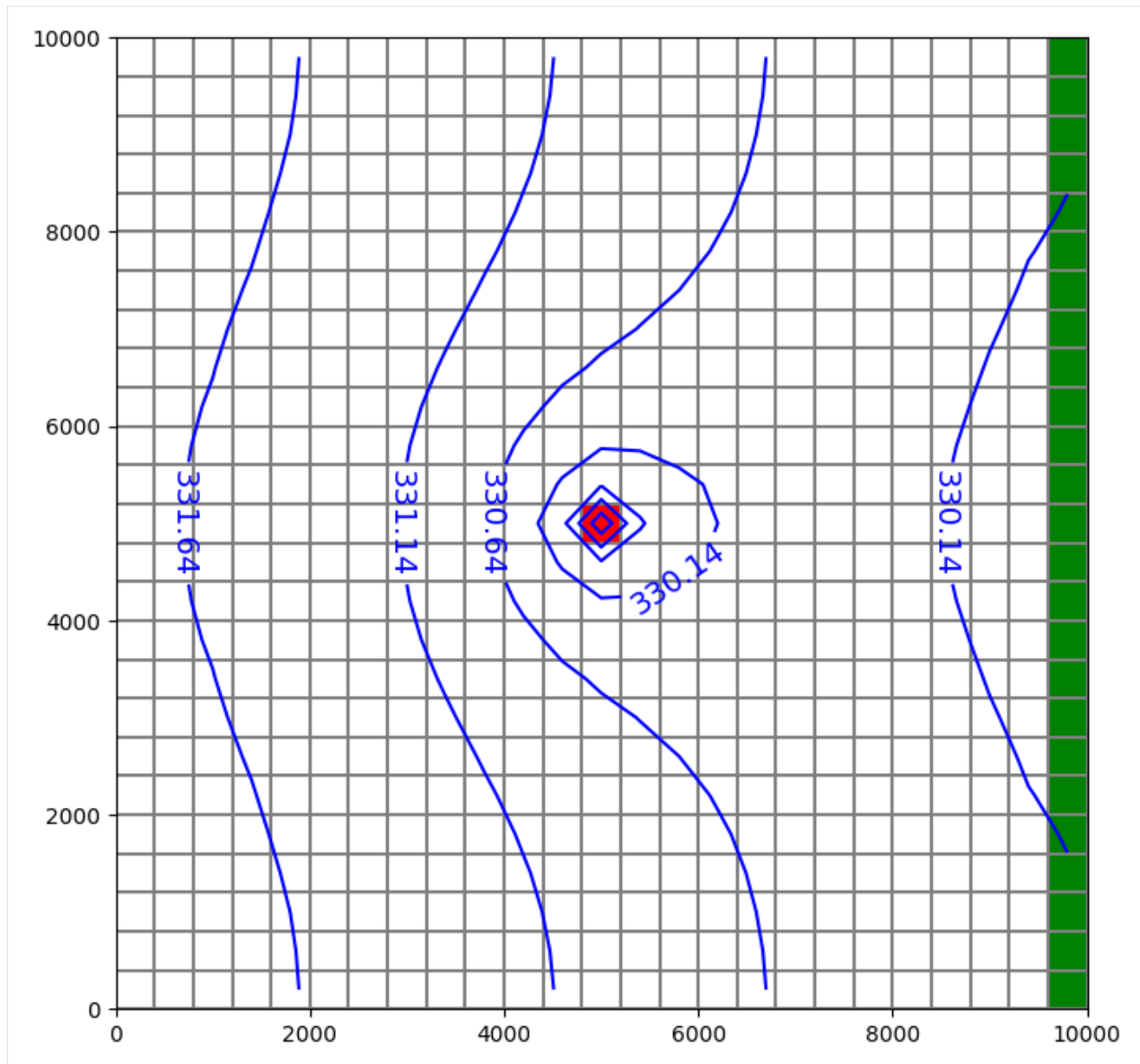
[4]: <matplotlib.colorbar.Colorbar at 0x7ffb7afa33d0>

```

```
[5]: fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mapview = flopy.plot.PlotMapView(model=m, layer=4)
quadmesh = mapview.plot_ibound()
linecollection = mapview.plot_grid()
riv = mapview.plot_bc("RIV", color="g", plotAll=True)
quadmesh = mapview.plot_bc("WEL", kper=1, plotAll=True)
contour_set = mapview.contour_array(
    hds, levels=np.arange(np.min(hds), np.max(hds), 0.5), colors="b"
)
plt.clabel(contour_set, inline=1, fontsize=14)
```

```
[5]: <a list of 5 text.Text objects>
```



Now create forward particle tracking simulation where particles are released at the top of each cell in layer 1: * specifying the recharge package in `create_mpsim` releases a single particle on `iface=6` of each top cell * start the particles at beginning of per 3, step 1, as in example 3 in MODPATH6 manual

Note: in FloPy version 3.3.5 and previous, the `Modpath6` constructor `dis_file`, `head_file` and `budget_file` arguments expected filenames relative to the model workspace. In 3.3.6 and later, full paths must be provided — if they are not, discretization, head and budget data are read directly from the model, as before.

```
[6]: from os.path import join

mp = flopy.modpath.Modpath6(
    modelname="ex6",
    exe_name="mp6",
    modflowmodel=m,
    model_ws=str(model_path),
)
```

(continues on next page)

(continued from previous page)

```

mpb = flopy.modpath.Modpath6Bas(
    mp, hdry=m.lpf.hdry, laytyp=m.lpf.laytyp, ibound=1, prsity=0.1
)

# start the particles at begining of per 3, step 1, as in example 3 in MODPATH6 manual
# (otherwise particles will all go to river)
sim = mp.create_mpsim(
    trackdir="forward",
    simtype="pathline",
    packages="RCH",
    start_time=(2, 0, 1.0),
)

shutil.copy(model_path / "EXAMPLE.DIS", join(model_ws, "EXAMPLE.DIS"))
shutil.copy(model_path / "EXAMPLE.HED", join(model_ws, "EXAMPLE.HED"))
shutil.copy(model_path / "EXAMPLE.BUD", join(model_ws, "EXAMPLE.BUD"))

mp.change_model_ws(model_ws)
mp.write_name_file()
mp.write_input()
success, buff = mp.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

Processing basic data ...
Checking head file ...
Checking budget file and building index ...

Run particle tracking simulation ...
Processing Time Step      1 Period      3. Time = 4.10000E+05
Particle tracking complete. Writing endpoint file ...
End of MODPATH simulation. Normal termination.

```

Read in the endpoint file and plot particles that terminated in the well.

```

[7]: fpth = os.path.join(model_ws, "ex6.mpend")
    epobj = flopy.utils.EndpointFile(fpth)
    well_epd = epobj.get_destination_endpoint_data(dest_cells=[(4, 12, 12)])
    # returns record array of same form as epobj.get_all_data()

```

```

[8]: well_epd[0:2]

```

```

[8]: rec.array([(50, 0, 2, 0., 29565.62, 1, 0, 2, 0, 6, 0, 0.5, 0.5, 1., 200., 9000., 339.
    ↪ 1231, 1, 4, 12, 12, 2, 0, 1.,          , 0.9178849, 0.09755219, 5200.,    , 5167.154, 9.
    ↪ 755219, b'rch'),
    (75, 0, 2, 0., 26106.59, 1, 0, 3, 0, 6, 0, 0.5, 0.5, 1., 200., 8600., 339.
    ↪ 1203, 1, 4, 12, 12, 4, 0, 0.7848778, 1.,          , 0.1387314 , 5113.951, 5200.,    , 13.
    ↪ 87314 , b'rch')],
    dtype=[('particleid', '<i4'), ('particlegroup', '<i4'), ('status', '<i4'), (

```

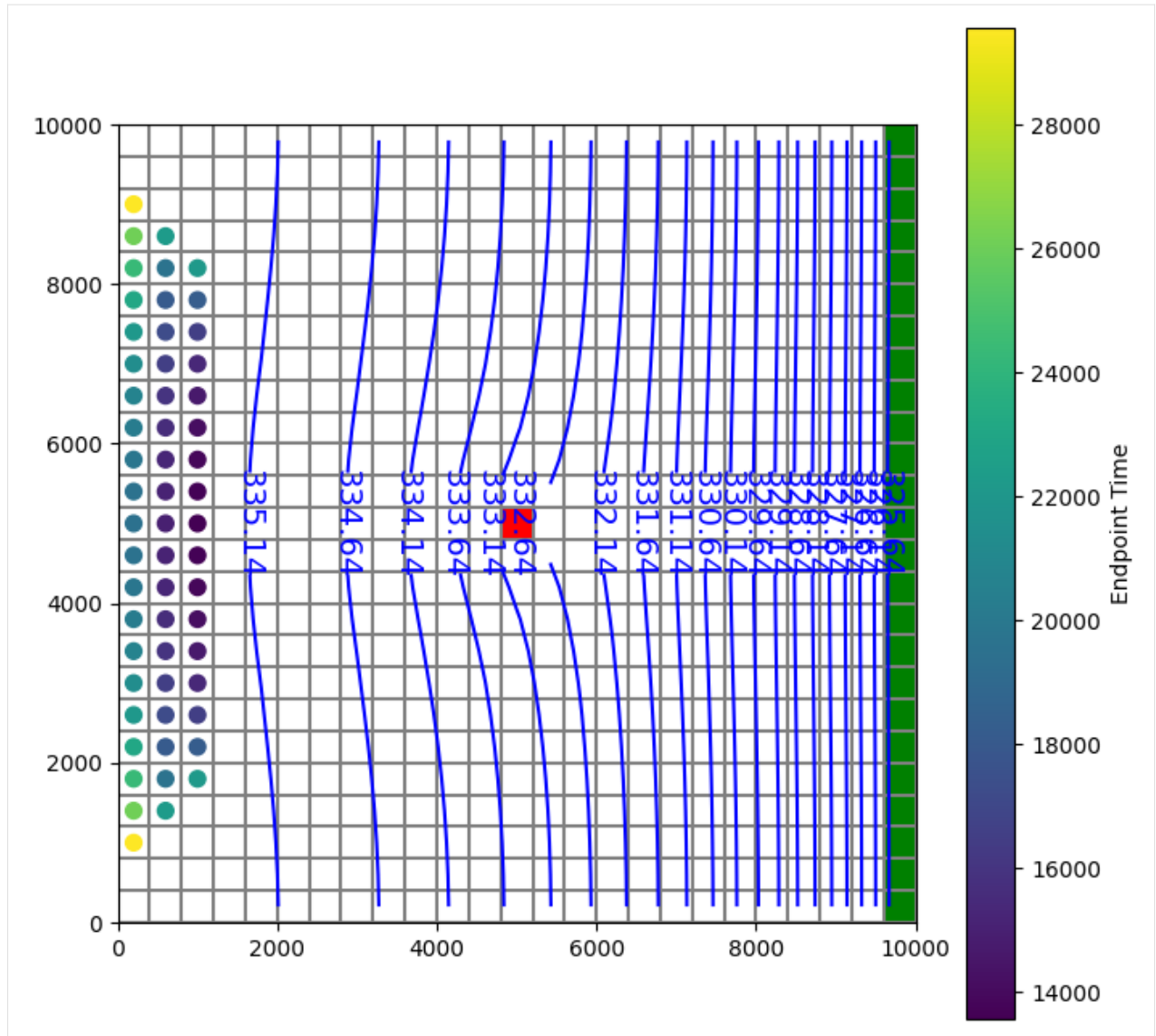
(continues on next page)

(continued from previous page)

```
→ 'time0', '<f4'), ('time', '<f4'), ('initialgrid', '<i4'), ('k0', '<i4'), ('i0', '<i4'),  
→ ('j0', '<i4'), ('cellface0', '<i4'), ('zone0', '<i4'), ('xloc0', '<f4'), ('yloc0', '  
→ <f4'), ('zloc0', '<f4'), ('x0', '<f4'), ('y0', '<f4'), ('z0', '<f4'), ('finalgrid', '  
→ <i4'), ('k', '<i4'), ('i', '<i4'), ('j', '<i4'), ('cellface', '<i4'), ('zone', '<i4'),  
→ ('xloc', '<f4'), ('yloc', '<f4'), ('zloc', '<f4'), ('x', '<f4'), ('y', '<f4'), ('z', '  
→ <f4'), ('label', '0'))]
```

```
[9]: fig = plt.figure(figsize=(8, 8))  
    ax = fig.add_subplot(1, 1, 1, aspect="equal")  
    mapview = flopy.plot.PlotMapView(model=m, layer=2)  
    quadmesh = mapview.plot_ibound()  
    linecollection = mapview.plot_grid()  
    riv = mapview.plot_bc("RIV", color="g", plotAll=True)  
    quadmesh = mapview.plot_bc("WEL", kper=1, plotAll=True)  
    contour_set = mapview.contour_array(  
        hds, levels=np.arange(np.min(hds), np.max(hds), 0.5), colors="b"  
    )  
    plt.clabel(contour_set, inline=1, fontsize=14)  
    mapview.plot_endpoint(well_epd, direction="starting", colorbar=True)
```

```
[9]: <matplotlib.collections.PathCollection at 0x7ffb78c22670>
```



Write starting locations to a shapefile.

```
[10]: fpth = os.path.join(model_ws, "starting_locs.shp")
print(type(fpth))
epobj.write_shapefile(
    well_epd, direction="starting", shpname=fpth, mg=m.modelgrid
)

<class 'str'>
No CRS information for writing a .prj file.
Supply an valid coordinate system reference to the attached modelgrid object or .
  export() method.
wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks
```

Read in the pathline file and subset to pathlines that terminated in the well .

```
[11]: # make a selection of cells that terminate in the well cell = (4, 12, 12)
```

(continues on next page)

(continued from previous page)

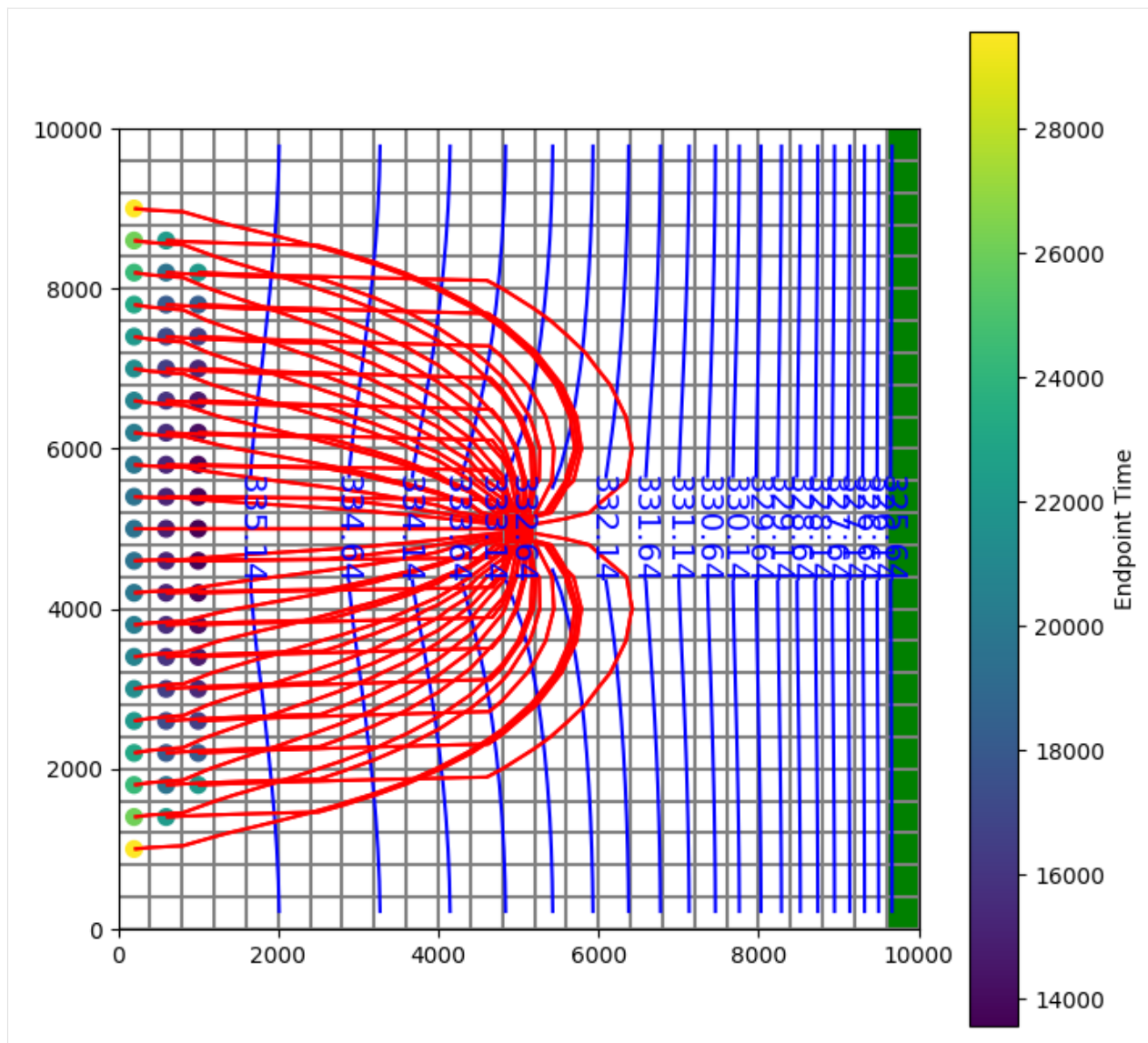
```
pthobj = flopy.utils.PathlineFile(os.path.join(model_ws, "ex6.mppth"))
well_pathlines = pthobj.get_destination_pathline_data(dest_cells=[(4, 12, 12)])
```

Plot the pathlines that terminate in the well and the starting locations of the particles.

```
[12]: fig = plt.figure(figsize=(8, 8))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      mapview = flopy.plot.PlotMapView(model=m, layer=2)
      quadmesh = mapview.plot_ibound()
      linecollection = mapview.plot_grid()
      riv = mapview.plot_bc("RIV", color="g", plotAll=True)
      quadmesh = mapview.plot_bc("WEL", kper=1, plotAll=True)
      contour_set = mapview.contour_array(
          hds, levels=np.arange(np.min(hds), np.max(hds), 0.5), colors="b"
      )
      plt.clabel(contour_set, inline=1, fontsize=14)

      mapview.plot_endpoint(well_epd, direction="starting", colorbar=True)
      # for now, each particle must be plotted individually
      # (plot_pathline() will plot a single line for recarray with multiple particles)
      # for pid in np.unique(well_pathlines.particleid):
      #     modelmap.plot_pathline(pthobj.get_data(pid), layer='all', colors='red');
      mapview.plot_pathline(well_pathlines, layer="all", colors="red")

[12]: <matplotlib.collections.LineCollection at 0x7ffb75840e80>
```



Write pathlines to a shapefile.

```
[13]: # one line feature per particle
pthobj.write_shapefile(
    well_pathlines,
    direction="starting",
    shpname=os.path.join(model_ws, "pathlines.shp"),
    mg=m.modelgrid,
    verbose=False,
)

# one line feature for each row in pathline file
# (can be used to color lines by time or layer in a GIS)
pthobj.write_shapefile(
    well_pathlines,
    one_per_particle=False,
```

(continues on next page)

(continued from previous page)

```

shpname=os.path.join(model_ws, "pathlines_1per.shp"),
mg=m.modelgrid,
verbose=False,
)

(numpy.record, {'names': ['x', 'y', 'z', 'time', 'k', 'particleid'], 'formats': ['<f4', '
↪<f4', '<f4', '<f4', '<i4', '<i4'], 'offsets': [20, 24, 28, 16, 32, 0], 'itemsizes': 64})
No CRS information for writing a .prj file.
Supply an valid coordinate system reference to the attached modelgrid object or .
↪export() method.
wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks
(numpy.record, {'names': ['x', 'y', 'z', 'time', 'k', 'particleid'], 'formats': ['<f4', '
↪<f4', '<f4', '<f4', '<i4', '<i4'], 'offsets': [20, 24, 28, 16, 32, 0], 'itemsizes': 64})
No CRS information for writing a .prj file.
Supply an valid coordinate system reference to the attached modelgrid object or .
↪export() method.
wrote ../../../../home/runner/work/flopy/flopy/.docs/Notebooks

```

Replace WEL package with MNW2, and create backward tracking simulation using particles released at MNW well.

```

[14]: m2 = flopy.modflow.Modflow.load(
        "EXAMPLE.nam", model_ws=str(model_path), exe_name="mf2005"
    )
m2.get_package_list()

```

```

[14]: ['DIS', 'BAS6', 'WEL', 'RIV', 'RCH', 'OC', 'PCG', 'LPF']

```

```

[15]: m2.nrow_ncol_nlay_nper

```

```

[15]: (25, 25, 5, 3)

```

```

[16]: m2.wel.stress_period_data.data

```

```

[16]: {0: 0,
      1: rec.array([(4, 12, 12, -150000., 0.)],
                    dtype=[('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('flux', '<f4'), ('iface', '
↪<f4')]),
      2: rec.array([(4, 12, 12, -150000., 0.)],
                    dtype=[('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('flux', '<f4'), ('iface', '
↪<f4')])})

```

```

[17]: node_data = np.array(
    [
        (3, 12, 12, "well1", "skin", -1, 0, 0, 0, 1.0, 2.0, 5.0, 6.2),
        (4, 12, 12, "well1", "skin", -1, 0, 0, 0, 0.5, 2.0, 5.0, 6.2),
    ],
    dtype=[
        ("k", int),
        ("i", int),
        ("j", int),
        ("wellid", object),
        ("losstype", object),
        ("pumploc", int),
    ],
)

```

(continues on next page)

(continued from previous page)

```

        ("qlimit", int),
        ("ppflag", int),
        ("pumpcap", int),
        ("rw", float),
        ("rskin", float),
        ("kskin", float),
        ("zpump", float),
    ],
).view(np.recarray)

stress_period_data = {
    0: np.array(
        [(0, "well1", -150000.0)],
        dtype=[("per", int), ("wellid", object), ("qdes", float)],
    )
}

```

```

[18]: m2.name = "Example_mnw"
m2.remove_package("WEL")
mnw2 = flopy.modflow.ModflowMnw2(
    model=m2,
    mnwmax=1,
    node_data=node_data,
    stress_period_data=stress_period_data,
    itmp=[1, -1, -1],
)
m2.get_package_list()

```

```

[18]: ['DIS', 'BAS6', 'RIV', 'RCH', 'OC', 'PCG', 'LPF', 'MNW2']

```

Write and run MODFLOW.

```

[19]: m2.change_model_ws(model_ws)
m2.write_name_file()
m2.write_input()
success, buff = m2.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

```

```

                                MODFLOW-2005
      U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
                                Version 1.12.00 2/3/2017

Using NAME file: Example_mnw.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:35

Solving:  Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Solving:  Stress period:      2      Time step:      1      Ground-Water Flow Eqn.
Solving:  Stress period:      2      Time step:      2      Ground-Water Flow Eqn.

```

(continues on next page)

(continued from previous page)

```

Solving: Stress period: 2 Time step: 3 Ground-Water Flow Eqn.
Solving: Stress period: 2 Time step: 4 Ground-Water Flow Eqn.
Solving: Stress period: 2 Time step: 5 Ground-Water Flow Eqn.
Solving: Stress period: 2 Time step: 6 Ground-Water Flow Eqn.
Solving: Stress period: 2 Time step: 7 Ground-Water Flow Eqn.
Solving: Stress period: 2 Time step: 8 Ground-Water Flow Eqn.
Solving: Stress period: 2 Time step: 9 Ground-Water Flow Eqn.
Solving: Stress period: 2 Time step: 10 Ground-Water Flow Eqn.
Solving: Stress period: 3 Time step: 1 Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:35
Elapsed run time: 0.051 Seconds

```

Normal termination of simulation

Create a new Modpath6 object.

```

[20]: mp = flopy.modpath.Modpath6(
        modelname="ex6mnw",
        exe_name="mp6",
        modflowmodel=m2,
        model_ws=model_ws,
    )

    mpb = flopy.modpath.Modpath6Bas(
        mp, hdry=m2.lpf.hdry, laytyp=m2.lpf.laytyp, ibound=1, prsity=0.1
    )
    sim = mp.create_mpsim(trackdir="backward", simtype="pathline", packages="MNW2")

    mp.change_model_ws(model_ws)
    mp.write_name_file()
    mp.write_input()
    success, buff = mp.run_model(silent=True, report=True)
    if success:
        for line in buff:
            print(line)
    else:
        raise ValueError("Failed to run.")

```

```

Processing basic data ...
Checking head file ...
Checking budget file and building index ...

```

```

Run particle tracking simulation ...
Processing Time Step 1 Period 1. Time = 2.000000E+05
Particle tracking complete. Writing endpoint file ...
End of MODPATH simulation. Normal termination.

```

Read in results and plot.

```

[21]: pthobj = flopy.utils.PathlineFile(os.path.join(model_ws, "ex6mnw.mppth"))
    epdobj = flopy.utils.EndpointFile(os.path.join(model_ws, "ex6mnw.mpend"))
    well_epd = epdobj.get_alldata()
    well_pathlines = (

```

(continues on next page)

(continued from previous page)

```

    pthobj.get_alldata()
) # returns a list of recarrays; one per pathline

```

```

[22]: fig = plt.figure(figsize=(8, 8))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      mapview = flopy.plot.PlotMapView(model=m2, layer=2)
      quadmesh = mapview.plot_ibound()
      linecollection = mapview.plot_grid()
      riv = mapview.plot_bc("RIV", color="g", plotAll=True)
      quadmesh = mapview.plot_bc("MNW2", kper=1, plotAll=True)
      contour_set = mapview.contour_array(
          hds, levels=np.arange(np.min(hds), np.max(hds), 0.5), colors="b"
      )
      plt.clabel(contour_set, inline=1, fontsize=14)

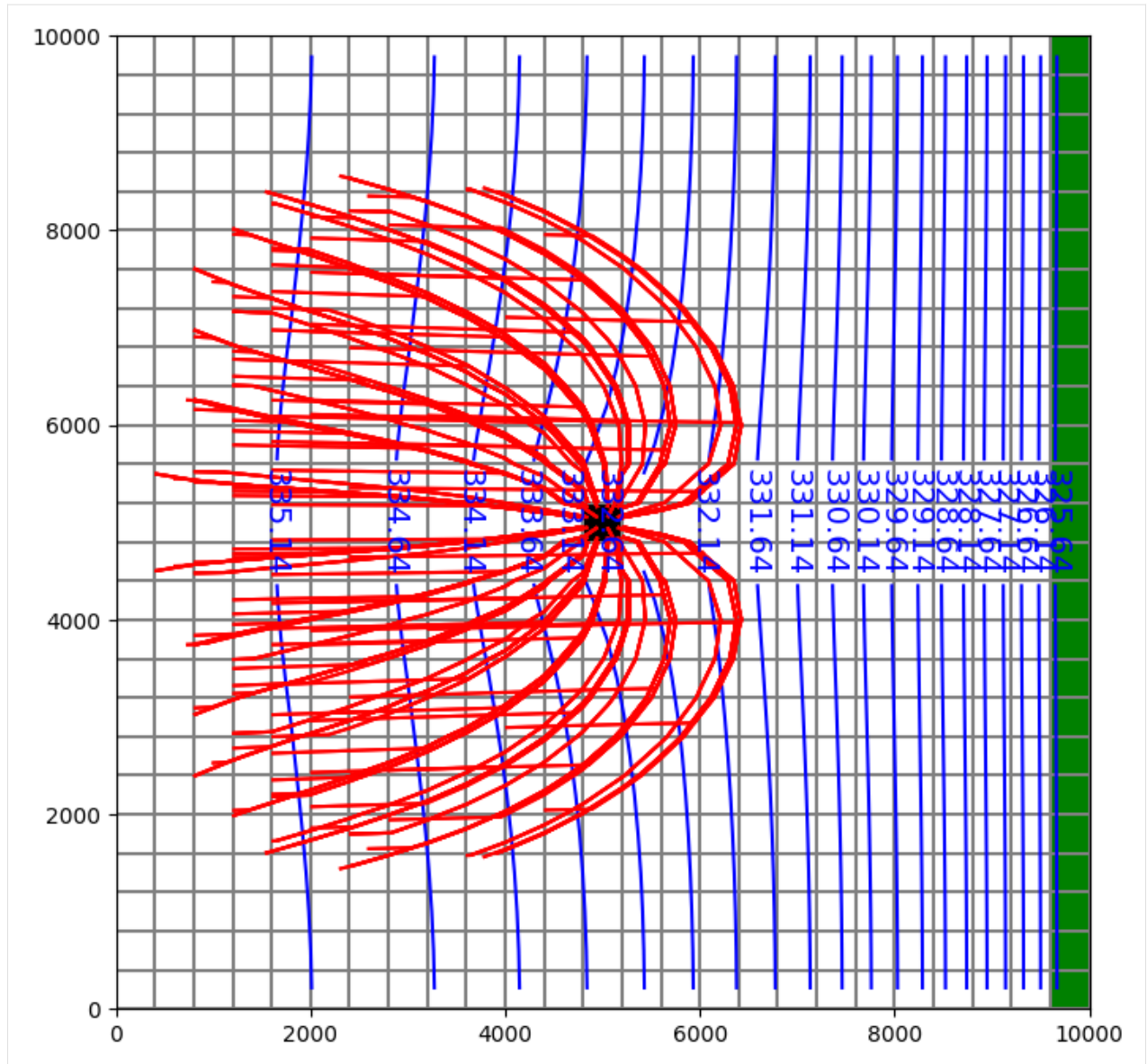
      mapview.plot_pathline(
          well_pathlines, travel_time="<10000", layer="all", colors="red"
      )

```

```

[22]: <matplotlib.collections.LineCollection at 0x7ffb755ab100>

```



```
[23]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.8.2 Creating a MODPATH 7 simulation

This notebook demonstrates how to create a simple forward and backward MODPATH 7 simulation using the `.create_mp7()` method. The notebooks also shows how to create subsets of endpoint output and plot MODPATH results on ModelMap objects.

```
[1]: import os
```

```
[2]: import sys
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

# temporary directory
temp_dir = TemporaryDirectory()
model_ws = temp_dir.name

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[3]: # define executable names
mpexe = "mp7"
mfexe = "mf6"
```

Flow model data

```
[4]: nper, nstp, perlen, tsmult = 1, 1, 1.0, 1.0
nlay, nrow, ncol = 3, 21, 20
delr = delc = 500.0
top = 400.0
botm = [220.0, 200.0, 0.0]
laytyp = [1, 0, 0]
kh = [50.0, 0.01, 200.0]
kv = [10.0, 0.01, 20.0]
```

(continues on next page)

(continued from previous page)

```
wel_loc = (2, 10, 9)
wel_q = -150000.0
rch = 0.005
riv_h = 320.0
riv_z = 317.0
riv_c = 1.0e5
```

```
[5]: def get_nodes(locs):
      nodes = []
      for k, i, j in locs:
          nodes.append(k * nrow * ncol + i * ncol + j)
      return nodes
```

MODPATH 7 using MODFLOW 6

Create and run MODFLOW 6

```
[6]: ws = os.path.join(model_ws, "mp7_ex1_cs")
      nm = "ex01_mf6"

      # Create the Flopy simulation object
      sim = flopy.mf6.MFSimulation(
          sim_name=nm, exe_name=mfexe, version="mf6", sim_ws=ws
      )

      # Create the Flopy temporal discretization object
      pd = (perlen, nstp, tsmult)
      tdis = flopy.mf6.modflow.mftdis.ModflowTdis(
          sim, pname="tdis", time_units="DAYS", nper=nper, perioddata=[pd]
      )

      # Create the Flopy groundwater flow (gwf) model object
      model_nam_file = "{}.nam".format(nm)
      gwf = flopy.mf6.ModflowGwf(
          sim, modelname=nm, model_nam_file=model_nam_file, save_flows=True
      )

      # Create the Flopy iterative model solver (ims) Package object
      ims = flopy.mf6.modflow.mfims.ModflowIms(
          sim,
          pname="ims",
          complexity="SIMPLE",
          outer_hclose=1e-6,
          inner_hclose=1e-6,
          rcloserecord=1e-6,
      )

      # create gwf file
      dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
          gwf,
```

(continues on next page)

(continued from previous page)

```

    pname="dis",
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    length_units="FEET",
    delr=delr,
    delc=delc,
    top=top,
    botm=botm,
)
# Create the initial conditions package
ic = flopy.mf6.modflow.mfgwfic.ModflowGwfic(gwf, pname="ic", strt=top)

# Create the node property flow package
npf = flopy.mf6.modflow.mfgwfnpf.ModflowGwfnpf(
    gwf, pname="npf", icelltype=laytyp, k=kh, k33=kv
)

# recharge
flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha(gwf, recharge=rch)
# wel
wd = [(wel_loc, wel_q)]
flopy.mf6.modflow.mfgwfwel.ModflowGwfwel(
    gwf, maxbound=1, stress_period_data={0: wd}
)
# river
rd = []
for i in range(nrow):
    rd.append([(0, i, ncol - 1), riv_h, riv_c, riv_z])
flopy.mf6.modflow.mfgwfriv.ModflowGwfriv(gwf, stress_period_data={0: rd})
# Create the output control package
headfile = "{}.hds".format(nm)
head_record = [headfile]
budgetfile = "{}.cbb".format(nm)
budget_record = [budgetfile]
saverecord = [("HEAD", "ALL"), ("BUDGET", "ALL")]
oc = flopy.mf6.modflow.mfgwfoc.ModflowGwfoc(
    gwf,
    pname="oc",
    saverecord=saverecord,
    head_filerecord=head_record,
    budget_filerecord=budget_record,
)

# Write the datasets
sim.write_simulation()
# Run the simulation
success, buff = sim.run_simulation(silent=True, report=True)
assert success, "mf6 model did not run"
for line in buff:
    print(line)

```

```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims...
  writing model ex01_mf6...
    writing model name file...
    writing package dis...
    writing package ic...
    writing package npf...
    writing package rcha_0...
    writing package wel_0...
    writing package riv_0...
INFORMATION: maxbound in ('gwf6', 'riv', 'dimensions') changed to 21 based on size of
↳ stress_period_data
  writing package oc...
```

```

MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023
```

```
MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000
```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

```
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:39
```

```
Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam
```

```
Solving: Stress period:      1      Time step:      1
```

```
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:44:39
Elapsed run time: 0.042 Seconds
```

WARNING REPORT:

1. NONLINEAR BLOCK VARIABLE 'OUTER_HCLOSE' IN FILE 'ex01_mf6.ims' WAS DEPRECATED IN VERSION 6.1.1. SETTING OUTER_DVCLOSE TO OUTER_HCLOSE VALUE.
2. LINEAR BLOCK VARIABLE 'INNER_HCLOSE' IN FILE 'ex01_mf6.ims' WAS

(continues on next page)

(continued from previous page)

DEPRECATED IN VERSION 6.1.1. SETTING INNER_DVCLOSE TO INNER_HCLOSE VALUE.
Normal termination of simulation.

Get locations to extract data

```
[7]: nodew = get_nodes([wel_loc])
      cellids = gwf.riv.stress_period_data.get_data()[0]["cellid"]
      nodesr = get_nodes(cellids)
```

Create and run MODPATH 7

Forward tracking

```
[8]: # create modpath files
      mpnamf = nm + "_mp_forward"

      # create basic forward tracking modpath simulation
      mp = flopy.modpath.Modpath7.create_mp7(
          modelname=mpnamf,
          trackdir="forward",
          flowmodel=gwf,
          model_ws=ws,
          rowcelldivisions=1,
          columncelldivisions=1,
          layercelldivisions=1,
          exe_name=mpexe,
      )

      # write modpath datasets
      mp.write_input()

      # run modpath
      succes, buff = mp.run_model(silent=True, report=True)
      assert succes, "mp7 forward tracking failed to run"
      for line in buff:
          print(line)
```

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.000000E+00 Steady-state flow

Particle Summary:

```
0 particles are pending release.
0 particles remain active.
0 particles terminated at boundary faces.
0 particles terminated at weak sink cells.
0 particles terminated at weak source cells.
```

(continues on next page)

(continued from previous page)

```
1260 particles terminated at strong source/sink cells.
0 particles terminated in cells with a specified zone number.
0 particles were stranded in inactive or dry cells.
0 particles were unreleased.
0 particles have an unknown status.
```

Normal termination.

Backward tracking from well and river locations

```
[9]: # create modpath files
mpnamb = nm + "_mp_backward"

# create basic forward tracking modpath simulation
mp = flopy.modpath.Modpath7.create_mp7(
    modelname=mpnamb,
    trackdir="backward",
    flowmodel=gwf,
    model_ws=ws,
    rowcelldivisions=5,
    columncelldivisions=5,
    layercelldivisions=5,
    nodes=nodew + nodesr,
    exe_name=mpexe,
)

# write modpath datasets
mp.write_input()

# run modpath
success, buff = mp.run_model(silent=True, report=True)
assert success, "mp7 backward tracking failed to run"
for line in buff:
    print(line)
```

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.000000E+00 Steady-state flow

Particle Summary:

```
0 particles are pending release.
0 particles remain active.
0 particles terminated at boundary faces.
0 particles terminated at weak sink cells.
0 particles terminated at weak source cells.
2750 particles terminated at strong source/sink cells.
0 particles terminated in cells with a specified zone number.
0 particles were stranded in inactive or dry cells.
0 particles were unreleased.
```

(continues on next page)

(continued from previous page)

```
0 particles have an unknown status.
```

```
Normal termination.
```

Load and Plot MODPATH 7 output

Forward Tracking

Load forward tracking pathline data

```
[10]: fpth = os.path.join(ws, mpnamf + ".mppth")
      p = flopy.utils.PathlineFile(fpth)
      pw = p.get_destination_pathline_data(dest_cells=nodew)
      pr = p.get_destination_pathline_data(dest_cells=nodesr)
```

Load forward tracking endpoint data

```
[11]: fpth = os.path.join(ws, mpnamf + ".mpend")
      e = flopy.utils.EndpointFile(fpth)
```

Get forward particles that terminate in the well

```
[12]: well_epd = e.get_destination_endpoint_data(dest_cells=nodew)
```

Get particles that terminate in the river boundaries

```
[13]: riv_epd = e.get_destination_endpoint_data(dest_cells=nodesr)
```

Well and river forward tracking pathlines

```
[14]: colors = ["green", "orange", "red"]
```

```
[15]: f, axes = plt.subplots(
      ncols=3, nrows=2, sharey=True, sharex=True, figsize=(15, 10)
      )
      axes = axes.flatten()

      idax = 0
      for k in range(nlay):
          ax = axes[idax]
          ax.set_aspect("equal")
          ax.set_title("Well pathlines - Layer {}".format(k + 1))
          mm = flopy.plot.PlotMapView(model=gwf, ax=ax)
          mm.plot_grid(lw=0.5)
          mm.plot_pathline(pw, layer=k, colors=colors[k], lw=0.75)
          idax += 1

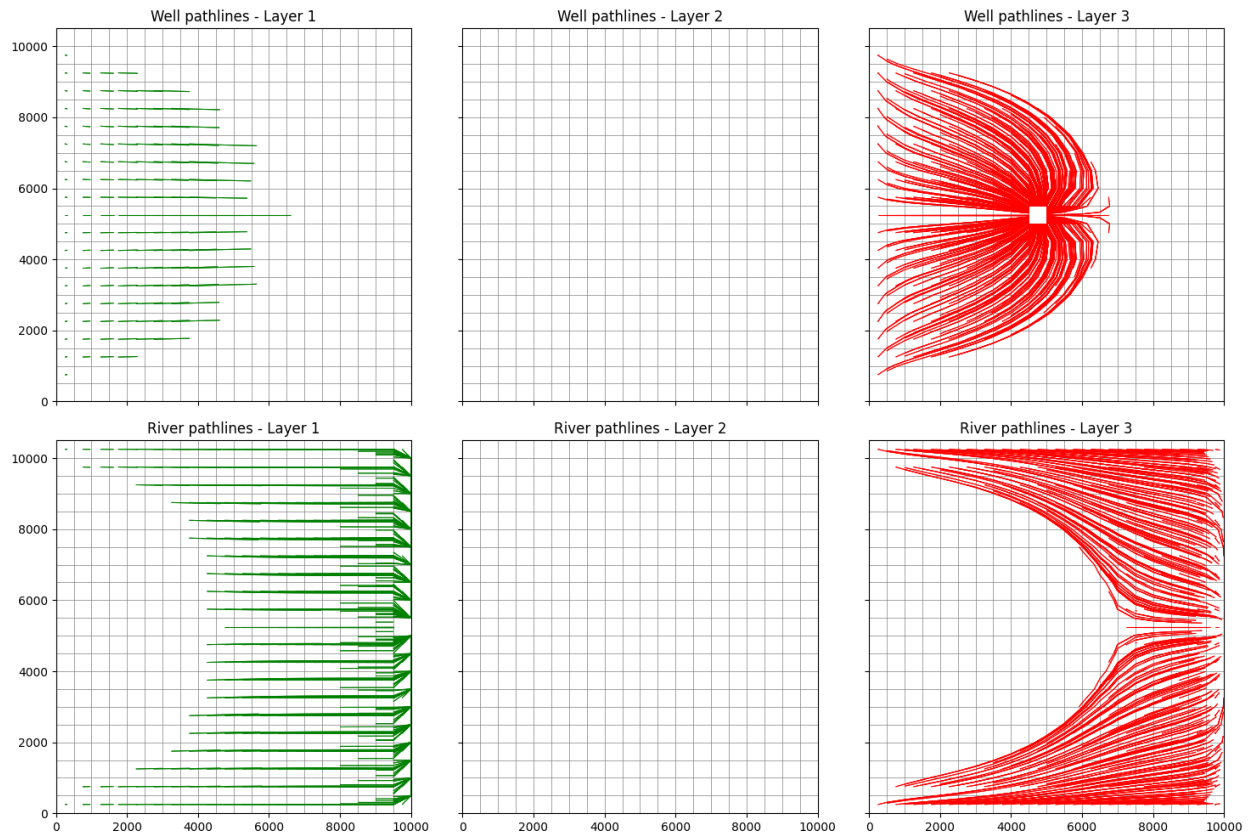
      for k in range(nlay):
          ax = axes[idax]
          ax.set_aspect("equal")
          ax.set_title("River pathlines - Layer {}".format(k + 1))
```

(continues on next page)

(continued from previous page)

```
mm = flopy.plot.PlotMapView(model=gwf, ax=ax)
mm.plot_grid(lw=0.5)
mm.plot_pathline(pr, layer=k, colors=colors[k], lw=0.75)
idax += 1
```

```
plt.tight_layout()
```



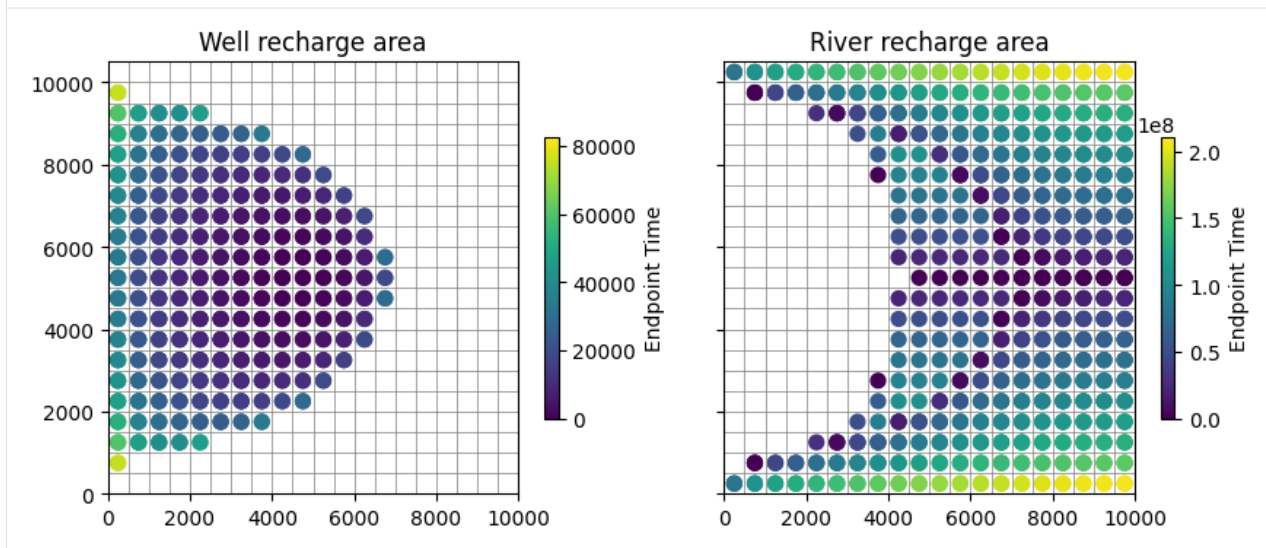
Forward tracking endpoints captured by the well and river

```
[16]: f, axes = plt.subplots(ncols=2, nrows=1, sharey=True, figsize=(10, 5))
axes = axes.flatten()

ax = axes[0]
ax.set_aspect("equal")
ax.set_title("Well recharge area")
mm = flopy.plot.PlotMapView(model=gwf, ax=ax)
mm.plot_grid(lw=0.5)
mm.plot_endpoint(well_epd, direction="starting", colorbar=True, shrink=0.5)

ax = axes[1]
ax.set_aspect("equal")
ax.set_title("River recharge area")
mm = flopy.plot.PlotMapView(model=gwf, ax=ax)
mm.plot_grid(lw=0.5)
mm.plot_endpoint(riv_epd, direction="starting", colorbar=True, shrink=0.5)
```

```
[16]: <matplotlib.collections.PathCollection at 0x7f75fc37a430>
```



Backward tracking

Load backward tracking pathlines

```
[17]: fpth = os.path.join(ws, mpnamb + ".mppth")
p = flopy.utils.PathlineFile(fpth)
pwb = p.get_destination_pathline_data(dest_cells=nodew)
prb = p.get_destination_pathline_data(dest_cells=nodesr)
```

Load backward tracking endpoints

```
[18]: fpth = os.path.join(ws, mpnamb + ".mpend")
e = flopy.utils.EndpointFile(fpth)
ewb = e.get_destination_endpoint_data(dest_cells=nodew, source=True)
erb = e.get_destination_endpoint_data(dest_cells=nodesr, source=True)
```

Well backward tracking pathlines

```
[19]: f, axes = plt.subplots(ncols=2, nrows=1, figsize=(10, 5))

ax = axes[0]
ax.set_aspect("equal")
ax.set_title("Well recharge area")
mm = flopy.plot.PlotMapView(model=gwf, ax=ax)
mm.plot_grid(lw=0.5)
mm.plot_pathline(
    pwb,
    layer="all",
    colors="blue",
    lw=0.5,
    linestyle=":",
    label="captured by wells",
)
```

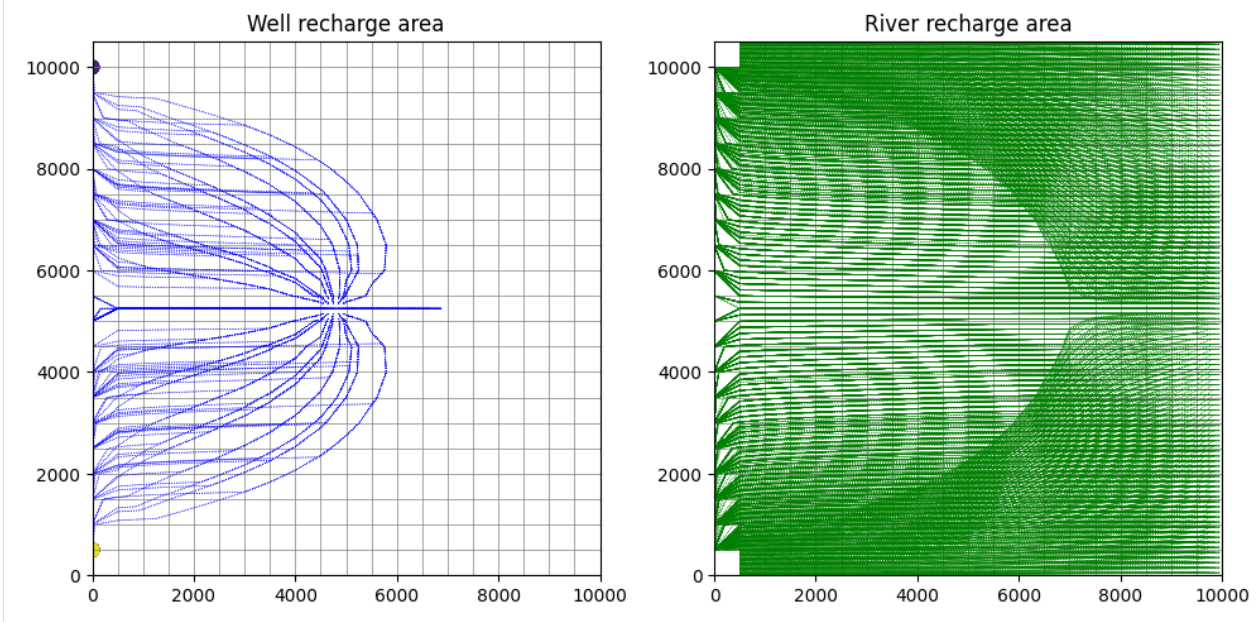
(continues on next page)

(continued from previous page)

```
mm.plot_endpoint(ewb, direction="ending") # , colorbar=True, shrink=0.5);

ax = axes[1]
ax.set_aspect("equal")
ax.set_title("River recharge area")
mm = flopy.plot.PlotMapView(model=gwf, ax=ax)
mm.plot_grid(lw=0.5)
mm.plot_pathline(
    prb,
    layer="all",
    colors="green",
    lw=0.5,
    linestyle=":",
    label="captured by rivers",
)

plt.tight_layout()
```



6.8.3 Using MODPATH 7 with structured grids

This notebook demonstrates how to create and run example 1a from the MODPATH 7 documentation for MODFLOW-2005 and MODFLOW 6. The notebooks also shows how to create subsets of endpoint output and plot MODPATH results on PlotMapView objects.

```
[1]: import os

[2]: import sys
    from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
```

(continues on next page)

(continued from previous page)

```

import numpy as np

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

    print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

# temporary directory
temp_dir = TemporaryDirectory()
workspace = temp_dir.name

numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3

```

Flow model data

```

[3]: nper, nstp, perlen, tsmult = 1, 1, 1.0, 1.0
     nlay, nrow, ncol = 3, 21, 20
     delr = delc = 500.0
     top = 400.0
     botm = [220.0, 200.0, 0.0]
     laytyp = [1, 0, 0]
     kh = [50.0, 0.01, 200.0]
     kv = [10.0, 0.01, 20.0]
     wel_loc = (2, 10, 9)
     wel_q = -150000.0
     rch = 0.005
     riv_h = 320.0
     riv_z = 317.0
     riv_c = 1.0e5

```

MODPATH 7 data

```

[4]: # MODPATH zones
     zone3 = np.ones((nrow, ncol), dtype=np.int32)
     zone3[wel_loc[1:]] = 2
     zones = [1, 1, zone3]

     # create particles
     # particle group 1

```

(continues on next page)

(continued from previous page)

```

plocs = []
pids = []
for idx in range(nrow):
    plocs.append((0, idx, 2))
    pids.append(idx)
part0 = flopy.modpath.ParticleData(
    plocs, drape=0, structured=True, particleids=pids
)
pg0 = flopy.modpath.ParticleGroup(
    particlegroupname="PG1", particledata=part0, filename="ex01a.pg1.sloc"
)

# particle group 2
v = [(2, 0, 0), (0, 20, 0)]
part1 = flopy.modpath.ParticleData(
    v, drape=1, structured=True, particleids=[1000, 1001]
)
pg1 = flopy.modpath.ParticleGroup(
    particlegroupname="PG2", particledata=part1, filename="ex01a.pg2.sloc"
)

locsa = [[0, 0, 0, 0, nrow - 1, ncol - 1], [1, 0, 0, 1, nrow - 1, ncol - 1]]
locsb = [[2, 0, 0, 2, nrow - 1, ncol - 1]]
sd = flopy.modpath.CellDataType(
    drape=0, columncelldivisions=1, rowcelldivisions=1, layercelldivisions=1
)
p = flopy.modpath.LRCParticleData(
    subdivisiondata=[sd, sd], lrcregions=[locsa, locsb]
)
pg2 = flopy.modpath.ParticleGroupLRCTemplate(
    particlegroupname="PG3", particledata=p, filename="ex01a.pg3.sloc"
)

particlegroups = [pg2]

# default iface for MODFLOW-2005 and MODFLOW 6
defaultiface = {"RECHARGE": 6, "ET": 6}
defaultiface6 = {"RCH": 6, "EVT": 6}

```

MODPATH 7 using MODFLOW-2005

Create and run MODFLOW-2005

```

[5]: ws = os.path.join(workspace, "mp7_ex1_mf2005_dis")
nm = "ex01_mf2005"
exe_name = "mf2005"
iu_cbc = 130
m = flopy.modflow.Modflow(nm, model_ws=ws, exe_name=exe_name)
flopy.modflow.ModflowDis(
    m,

```

(continues on next page)

(continued from previous page)

```

nlay=nlay,
nrow=nrow,
ncol=ncol,
nper=nper,
itmuni=4,
lenuni=2,
perlen=perlen,
nstp=nstp,
tsmult=tsmult,
steady=True,
delr=delr,
delc=delc,
top=top,
botm=botm,
)
flopy.modflow.ModflowLpf(
    m, ipakcb=iu_cbc, laytyp=laytyp, hk=kh, vka=kv, constantcv=True
)
flopy.modflow.ModflowBas(m, ibound=1, strt=top)
# recharge
flopy.modflow.ModflowRch(m, ipakcb=iu_cbc, rech=rch)
# wel
wd = [i for i in wel_loc] + [wel_q]
flopy.modflow.ModflowWel(m, ipakcb=iu_cbc, stress_period_data={0: wd})
# river
rd = []
for i in range(nrow):
    rd.append([0, i, ncol - 1, riv_h, riv_c, riv_z])
flopy.modflow.ModflowRiv(m, ipakcb=iu_cbc, stress_period_data={0: rd})
# output control
flopy.modflow.ModflowOc(
    m, stress_period_data={(0, 0): ["save head", "save budget", "print head"]}
)
flopy.modflow.ModflowPcg(m, hclose=1e-6, rclose=1e-6)

m.write_input()
success, buff = m.run_model(silent=True, report=True)
assert success, "mf2005 model did not run"
for line in buff:
    print(line)

```

MODFLOW-2005
 U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
 Version 1.12.00 2/3/2017

Using NAME file: ex01_mf2005.nam
 Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:07

Solving: Stress period: 1 Time step: 1 Ground-Water Flow Eqn.
 Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:07
 Elapsed run time: 0.014 Seconds

(continues on next page)

(continued from previous page)

Normal termination of simulation

Create and run MODPATH 7

```
[6]: # create modpath files
exe_name = "mp7"
mp = flopy.modpath.Modpath7(
    modelname=nm + "_mp", flowmodel=m, exe_name=exe_name, model_ws=ws
)
mpbas = flopy.modpath.Modpath7Bas(mp, porosity=0.1, defaultiface=defaultiface)
mpsim = flopy.modpath.Modpath7Sim(
    mp,
    simulationtype="combined",
    trackingdirection="forward",
    weaksinkoption="pass_through",
    weaksourceoption="pass_through",
    budgetoutputoption="summary",
    budgetcellnumbers=[1049, 1259],
    traceparticledata=[1, 1000],
    referencetime=[0, 0, 0.0],
    stoptimeoption="extend",
    timepointdata=[500, 1000.0],
    zonedataoption="on",
    zones=zones,
    particlegroups=particlegroups,
)

# write modpath datasets
mp.write_input()

# run modpath
success, buff = mp.run_model(silent=True, report=True)
assert success, "mp7 failed to run"
for line in buff:
    print(line)
```

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.000000E+00 Steady-state flow

Particle Summary:

- 0 particles are pending release.
- 0 particles remain active.
- 0 particles terminated at boundary faces.
- 0 particles terminated at weak sink cells.
- 0 particles terminated at weak source cells.

(continues on next page)

(continued from previous page)

```

1260 particles terminated at strong source/sink cells.
    0 particles terminated in cells with a specified zone number.
    0 particles were stranded in inactive or dry cells.
    0 particles were unreleased.
    0 particles have an unknown status.

```

Normal termination.

Load MODPATH 7 output

Get locations to extract pathline data

```

[7]: nodew = m.dis.get_node([wel_loc])
    riv_locs = flopy.utils.ra_slice(m.riv.stress_period_data[0], ["k", "i", "j"])
    nodesr = m.dis.get_node(riv_locs.tolist())

```

Pathline data

```

[8]: fpth = os.path.join(ws, nm + "_mp.mppth")
    p = flopy.utils.PathlineFile(fpth)
    pw0 = p.get_destination_pathline_data(nodew, to_reccarray=True)
    pr0 = p.get_destination_pathline_data(nodesr, to_reccarray=True)

```

Endpoint data

Get particles that terminate in the well

```

[9]: fpth = os.path.join(ws, nm + "_mp.mpend")
    e = flopy.utils.EndpointFile(fpth)
    well_epd = e.get_destination_endpoint_data(dest_cells=nodew)
    well_epd.shape

```

```

[9]: (564,)

```

Get particles that terminate in the river boundaries

```

[10]: riv_epd = e.get_destination_endpoint_data(dest_cells=nodesr)
    riv_epd.shape

```

```

[10]: (696,)

```

Merge the particles that end in the well and the river boundaries.

```

[11]: epd0 = np.concatenate((well_epd, riv_epd))
    epd0.shape

```

```

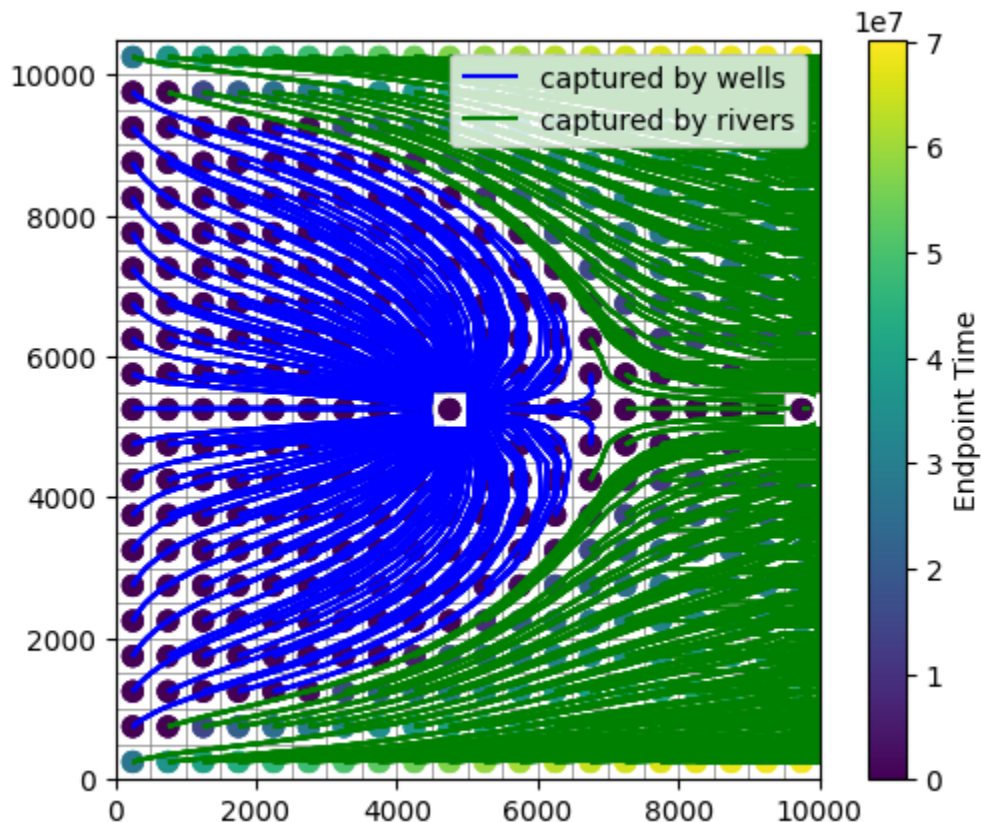
[11]: (1260,)

```

Plot MODPATH 7 output

```
[12]: mm = flopy.plot.PlotMapView(model=m)
mm.plot_grid(lw=0.5)
mm.plot_pathline(pw0, layer="all", colors="blue", label="captured by wells")
mm.plot_pathline(pr0, layer="all", colors="green", label="captured by rivers")
mm.plot_endpoint(epd0, direction="starting", colorbar=True)
mm.ax.legend()

[12]: <matplotlib.legend.Legend at 0x7f59dbac93a0>
```



MODPATH 7 using MODFLOW 6

Create and run MODFLOW 6

```
[13]: ws = os.path.join(workspace, "mp7_ex1_mf6_dis")
nm = "ex01_mf6"
exe_name = "mf6"

# Create the Flopy simulation object
sim = flopy.mf6.MFSimulation(
    sim_name=nm, exe_name="mf6", version="mf6", sim_ws=ws
)
```

(continues on next page)

(continued from previous page)

```

# Create the Flopy temporal discretization object
pd = (perlen, nstp, tsmult)
tdis = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim, pname="tdis", time_units="DAYS", nper=nper, perioddata=[pd]
)

# Create the Flopy groundwater flow (gwf) model object
model_nam_file = "{}.nam".format(nm)
gwf = flopy.mf6.ModflowGwf(
    sim, modelname=nm, model_nam_file=model_nam_file, save_flows=True
)

# Create the Flopy iterative model solver (ims) Package object
ims = flopy.mf6.modflow.mfims.ModflowIms(
    sim,
    pname="ims",
    complexity="SIMPLE",
    outer_dvclose=1e-6,
    inner_dvclose=1e-6,
    rcloserecord=1e-6,
)

# create gwf file
dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf,
    pname="dis",
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    length_units="FEET",
    delr=delr,
    delc=delc,
    top=top,
    botm=botm,
)

# Create the initial conditions package
ic = flopy.mf6.modflow.mfgwfic.ModflowGwfic(gwf, pname="ic", strt=top)

# Create the node property flow package
npf = flopy.mf6.modflow.mfgwfnpf.ModflowGwfnpf(
    gwf, pname="npf", icelltype=laytyp, k=kh, k33=kv
)

# recharge
flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha(gwf, recharge=rch)
# wel
wd = [(wel_loc, wel_q)]
flopy.mf6.modflow.mfgfwel.ModflowGfwel(
    gwf, maxbound=1, stress_period_data={0: wd}
)
# river

```

(continues on next page)

(continued from previous page)

```

rd = []
for i in range(nrow):
    rd.append([(0, i, ncol - 1), riv_h, riv_c, riv_z])
flopy.mf6.modflow.mfgwfriv.ModflowGwfriv(gwf, stress_period_data={0: rd})
# Create the output control package
headfile = "{}.hds".format(nm)
head_record = [headfile]
budgetfile = "{}.cbb".format(nm)
budget_record = [budgetfile]
saverrecord = [("HEAD", "ALL"), ("BUDGET", "ALL")]
oc = flopy.mf6.modflow.mfgwfoc.ModflowGwfoc(
    gwf,
    pname="oc",
    saverrecord=saverrecord,
    head_filerecord=head_record,
    budget_filerecord=budget_record,
)

# Write the datasets
sim.write_simulation()
# Run the simulation
success, buff = sim.run_simulation(silent=True, report=True)
assert success, "mf6 model did not run"
for line in buff:
    print(line)

```

writing simulation...

writing simulation name file...

writing simulation tdis package...

writing solution package ims...

writing model ex01_mf6...

writing model name file...

writing package dis...

writing package ic...

writing package npf...

writing package rcha_0...

writing package wel_0...

writing package riv_0...

INFORMATION: maxbound in ('gwf6', 'riv', 'dimensions') changed to 21 based on size of ↵
↵stress_period_data

writing package oc...

MODFLOW 6

U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL

VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

This software has been approved for release by the U.S. Geological
Survey (USGS). Although the software has been subjected to rigorous
review, the USGS reserves the right to update the software as needed

(continues on next page)

(continued from previous page)

pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:21

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

Solving: Stress period: 1 Time step: 1

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:21
Elapsed run time: 0.042 Seconds

Normal termination of simulation.

Create and run MODPATH 7

```
[14]: # create modpath files
exe_name = "mp7"
mp = flopy.modpath.Modpath7(
    modelname=nm + "_mp", flowmodel=gwf, exe_name=exe_name, model_ws=ws
)
mpbas = flopy.modpath.Modpath7Bas(mp, porosity=0.1, defaultiface=defaultiface6)
mpsim = flopy.modpath.Modpath7Sim(
    mp,
    simulationtype="combined",
    trackingdirection="forward",
    weaksinkoption="pass_through",
    weaksourceoption="pass_through",
    budgetoutputoption="summary",
    budgetcellnumbers=[1049, 1259],
    traceparticledata=[1, 1000],
    referencetime=[0, 0, 0.0],
    stoptimeoption="extend",
    timepointdata=[500, 1000.0],
    zonedataoption="on",
    zones=zones,
    particlegroups=particlegroups,
)

# write modpath datasets
mp.write_input()
```

(continues on next page)

(continued from previous page)

```
# run modpath
success, buff = mp.run_model(silent=True, report=True)
assert success, "mp7 failed to run"
for line in buff:
    print(line)
```

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.00000E+00 Steady-state flow

Particle Summary:

```
0 particles are pending release.
0 particles remain active.
0 particles terminated at boundary faces.
0 particles terminated at weak sink cells.
0 particles terminated at weak source cells.
1260 particles terminated at strong source/sink cells.
0 particles terminated in cells with a specified zone number.
0 particles were stranded in inactive or dry cells.
0 particles were unreleased.
0 particles have an unknown status.
```

Normal termination.

Load MODPATH 7 output

Pathline data

```
[15]: fpth = os.path.join(ws, nm + "_mp.mppth")
p = flopy.utils.PathlineFile(fpth)
pw1 = p.get_destination_pathline_data(nodew, to_reccarray=True)
pr1 = p.get_destination_pathline_data(nodesr, to_reccarray=True)
```

Endpoint data

Get particles that terminate in the well

```
[16]: fpth = os.path.join(ws, nm + "_mp.mpend")
e = flopy.utils.EndpointFile(fpth)
well_epd = e.get_destination_endpoint_data(dest_cells=nodew)
```

Get particles that terminate in the river boundaries

```
[17]: riv_epd = e.get_destination_endpoint_data(dest_cells=nodesr)
```

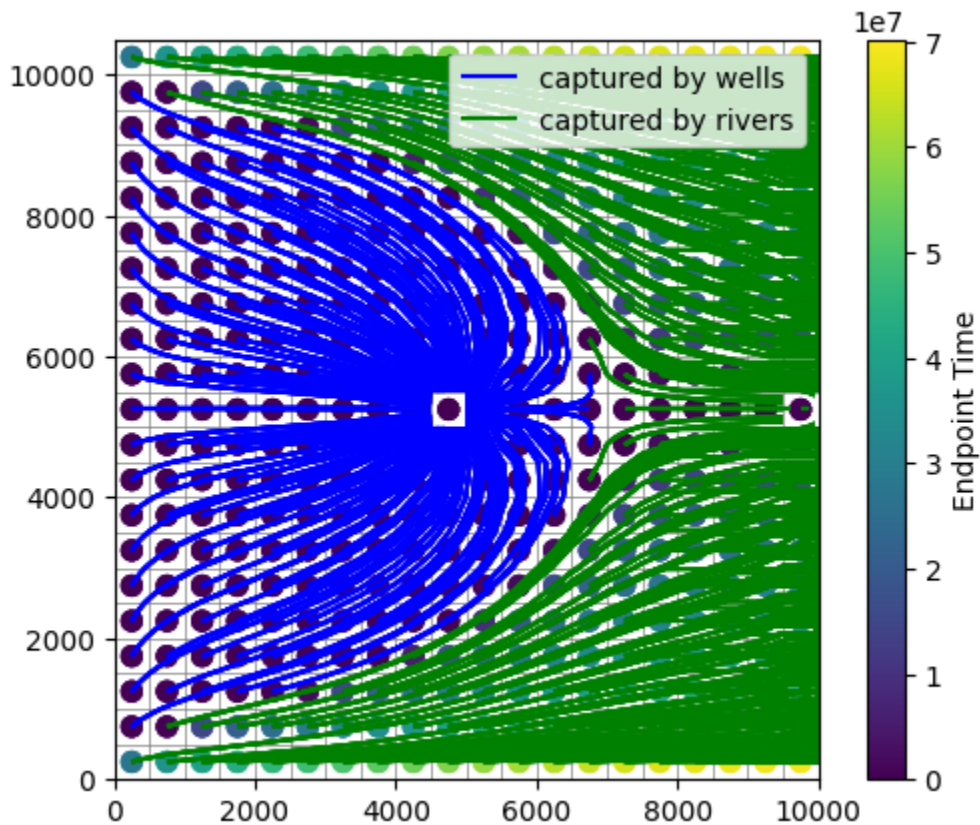
Merge the particles that end in the well and the river boundaries.


```
[18]: epd1 = np.concatenate((well_epd, riv_epd))
```

Plot MODPATH 7 output

```
[19]: mm = flopy.plot.PlotMapView(model=gwf)
mm.plot_grid(lw=0.5)
mm.plot_pathline(pw1, layer="all", colors="blue", label="captured by wells")
mm.plot_pathline(pr1, layer="all", colors="green", label="captured by rivers")
mm.plot_endpoint(epd1, direction="starting", colorbar=True)
mm.ax.legend()
```

```
[19]: <matplotlib.legend.Legend at 0x7f59dee8a640>
```



Compare MODPATH results

Compare MODPATH results for MODFLOW-2005 and MODFLOW 6. Also show pathline points every 5th point.

```
[20]: f, axes = plt.subplots(ncols=3, nrows=1, sharey=True, figsize=(15, 10))
axes = axes.flatten()
ax = axes[0]
ax.set_aspect("equal")
mm = flopy.plot.PlotMapView(model=m, ax=ax)
mm.plot_grid(lw=0.5)
```

(continues on next page)

(continued from previous page)

```

mm.plot_pathline(
    pw0,
    layer="all",
    colors="blue",
    lw=1,
    marker="o",
    markercolor="black",
    markersize=3,
    markerevery=5,
)
mm.plot_pathline(
    pr0,
    layer="all",
    colors="green",
    lw=1,
    marker="o",
    markercolor="black",
    markersize=3,
    markerevery=5,
)
ax.set_title("MODFLOW-2005")

ax = axes[1]
ax.set_aspect("equal")
mm = flopy.plot.PlotMapView(model=gwf, ax=ax)
mm.plot_grid(lw=0.5)
mm.plot_pathline(
    pw1,
    layer="all",
    colors="blue",
    lw=1,
    marker="o",
    markercolor="black",
    markersize=3,
    markerevery=5,
)
mm.plot_pathline(
    pr1,
    layer="all",
    colors="green",
    lw=1,
    marker="o",
    markercolor="black",
    markersize=3,
    markerevery=5,
)
ax.set_title("MODFLOW 6")

ax = axes[2]
ax.set_aspect("equal")
mm = flopy.plot.PlotMapView(model=m, ax=ax)

```

(continues on next page)

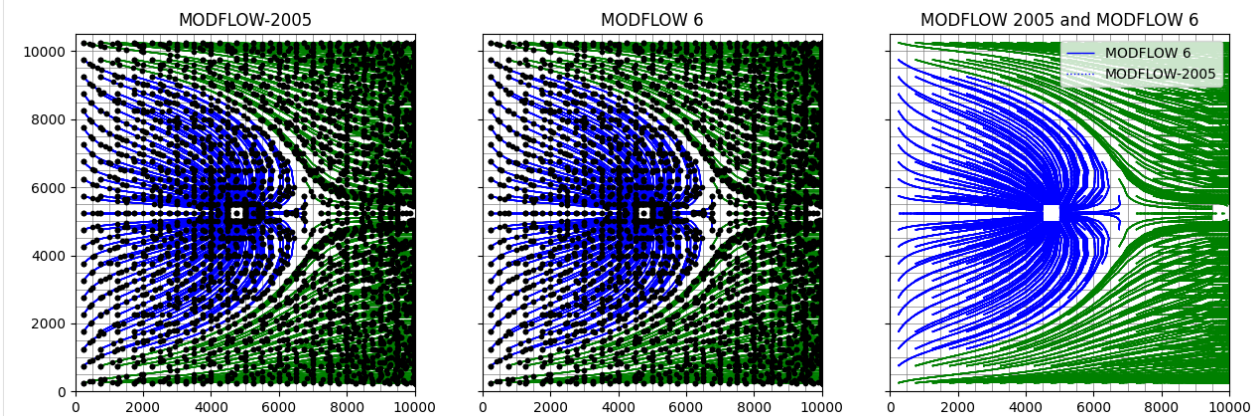
(continued from previous page)

```

mm.plot_grid(lw=0.5)
mm.plot_pathline(pw1, layer="all", colors="blue", lw=1, label="MODFLOW 6")
mm.plot_pathline(
    pw0, layer="all", colors="blue", lw=1, linestyle=":", label="MODFLOW-2005"
)
mm.plot_pathline(pr1, layer="all", colors="green", lw=1, label="_none")
mm.plot_pathline(
    pr0, layer="all", colors="green", lw=1, linestyle=":", label="_none"
)
ax.legend()
ax.set_title("MODFLOW 2005 and MODFLOW 6")

```

[20]: `Text(0.5, 1.0, 'MODFLOW 2005 and MODFLOW 6')`



6.8.4 Using MODPATH 7 with structured grids (transient example)

This notebook reproduces example 3a from the MODPATH 7 documentation, demonstrating a transient MODFLOW 6 simulation based on the same flow system as the basic structured and unstructured examples. Particles are released at 10 20-day intervals for the first 200 days of the simulation. 2 discharge wells are added 100,000 days into the simulation and pump at a constant rate for the remainder. There are three stress periods:

Stress period	Type	Time steps	Length (days)
1	steady-state	1	100000
2	transient	10	36500
3	steady-state	1	100000

Setting up the simulation

First import FloPy and set up a temporary workspace.

[1]: `import os`

[2]: `import sys
from os.path import join
from pathlib import Path`

(continues on next page)

(continued from previous page)

```

from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

proj_root = Path.cwd().parent.parent

# run installed version of flopy or add local path
try:
    import flopy
except:
    sys.path.append(proj_root)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

temp_dir = TemporaryDirectory()
sim_name = "mp7_ex03a_mf6"
workspace = Path(temp_dir.name) / sim_name

```

3.8.18 (default, Aug 28 2023, 08:27:22)
 [GCC 11.4.0]
 numpy version: 1.24.4
 matplotlib version: 3.7.3
 flopy version: 3.4.3

Define flow model data.

```

[3]: nlay, nrow, ncol = 3, 21, 20
    delr = delc = 500.0
    top = 400.0
    botm = [220.0, 200.0, 0.0]
    laytyp = [1, 0, 0]
    kh = [50.0, 0.01, 200.0]
    kv = [10.0, 0.01, 20.0]
    rch = 0.005
    riv_h = 320.0
    riv_z = 317.0
    riv_c = 1.0e5

```

Define well data. Although this notebook will refer to layer/row/column indices starting at 1, indices in FloPy (and more generally in Python) are zero-based. A negative discharge indicates pumping, while a positive value indicates injection.

```

[4]: wells = [
    # layer, row, col, discharge
    (0, 10, 9, -750000),
    (2, 12, 4, -1000000),
]

```

Define the drain location.

```
[5]: drain = (0, 14, (9, 20))
```

Configure locations for particle tracking to terminate. We have three explicitly defined termination zones:

- 2: the well in layer 1, at row 11, column 10
- 3: the well in layer 3, at row 13, column 5
- 4: the drain in layer 1, running through row 15 from column 10-20

MODFLOW 6 reserves zone number 1 to indicate that particles may move freely within the zone.

The river running through column 20 is also a termination zone, but it doesn't need to be defined separately since we are using the RIV package.

```
[6]: zone_maps = []

# zone 1 is the default (non-terminating regions)
def fill_zone_1():
    return np.ones((nrow, ncol), dtype=np.int32)

# zone map for layer 1
za = fill_zone_1()
za[wells[0][1:3]] = 2
za[drain[1], drain[2][0] : drain[2][1]] = 4
zone_maps.append(za)

# constant layer 2 (zone 1)
zone_maps.append(1)

# zone map for layer 3
za = fill_zone_1()
za[wells[1][1:3]] = 3
zone_maps.append(za)
```

Define particles to track. We release particles from the top of a 2x2 square of cells in the upper left of the model grid's top layer.

```
[7]: rel_minl = rel_maxl = 1
rel_minr = 2
rel_maxr = 3
rel_minc = 2
rel_maxc = 3
sd = flopy.modpath.CellDataType(
    drape=0
) # particles added at top of cell (no drape)
pd = flopy.modpath.LRCParticleData(
    subdivisiondata=[sd],
    lrcregions=[
        [rel_minl, rel_minr, rel_minc, rel_maxl, rel_maxr, rel_maxc]
    ],
)
```

(continues on next page)

(continued from previous page)

```
pg = flopy.modpath.ParticleGroupLRCTemplate(
    particlegroupname="PG1", particledata=pd, filename=f"{sim_name}.pg1.sloc"
)
pgs = [pg]
defaultiface = {"RECHARGE": 6, "ET": 6}
```

Create the MODFLOW 6 simulation.

```
[8]: # simulation
sim = flopy.mf6.MFSimulation(
    sim_name=sim_name, exe_name="mf6", version="mf6", sim_ws=workspace
)

# temporal discretization
nper = 3
pd = [
    # perlen, nstp, tsmult
    (100000, 1, 1),
    (36500, 10, 1),
    (100000, 1, 1),
]
tdis = flopy.mf6.modflow.mftdis.ModflowTdis(
    sim, pname="tdis", time_units="DAYS", nper=nper, perioddata=pd
)

# groundwater flow (gwf) model
model_nam_file = "{}.nam".format(sim_name)
gwf = flopy.mf6.ModflowGwf(
    sim, modelname=sim_name, model_nam_file=model_nam_file, save_flows=True
)

# iterative model solver (ims) package
ims = flopy.mf6.modflow.mfims.ModflowIms(
    sim,
    pname="ims",
    complexity="SIMPLE",
    outer_dvclose=1e-6,
    inner_dvclose=1e-6,
    rcloserecord=1e-6,
)

# grid discretization
dis = flopy.mf6.modflow.mfgwfdis.ModflowGwfdis(
    gwf,
    pname="dis",
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    length_units="FEET",
    delr=delr,
    delc=delc,
    top=top,
```

(continues on next page)

(continued from previous page)

```

    botm=botm,
)

# initial conditions
ic = flopy.mf6.modflow.mfgwfic.ModflowGwfic(gwf, pname="ic", strt=top)

# node property flow
npf = flopy.mf6.modflow.mfgwnpf.ModflowGwnpf(
    gwf, pname="npf", icelltype=laytyp, k=kh, k33=kv
)

# recharge
rch = flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha(gwf, recharge=rch)

# wells
def no_flow(w):
    return w[0], w[1], w[2], 0

wel = flopy.mf6.modflow.mfgfwel.ModflowGfwel(
    gwf,
    maxbound=1,
    stress_period_data={0: [no_flow(w) for w in wells], 1: wells, 2: wells},
)

# river
rd = [[(0, i, ncol - 1), riv_h, riv_c, riv_z] for i in range(nrow)]
flopy.mf6.modflow.mfgwfriv.ModflowGwfriv(
    gwf, stress_period_data={0: rd, 1: rd, 2: rd}
)

# drain (set auxiliary IFACE var to 6 for top of cell)
dd = [
    [drain[0], drain[1], i + drain[2][0], 322.5, 100000.0, 6]
    for i in range(drain[2][1] - drain[2][0])
]
drn = flopy.mf6.modflow.mfgwdrn.ModflowGwdrn(gwf, stress_period_data={0: dd})

# output control
headfile = "{}.hds".format(sim_name)
head_record = [headfile]
budgetfile = "{}.cbb".format(sim_name)
budget_record = [budgetfile]
saverecord = [("HEAD", "ALL"), ("BUDGET", "ALL")]
oc = flopy.mf6.modflow.mfgwfoc.ModflowGwfoc(
    gwf,
    pname="oc",
    saverecord=saverecord,
    head_filerecord=head_record,
    budget_filerecord=budget_record,
)

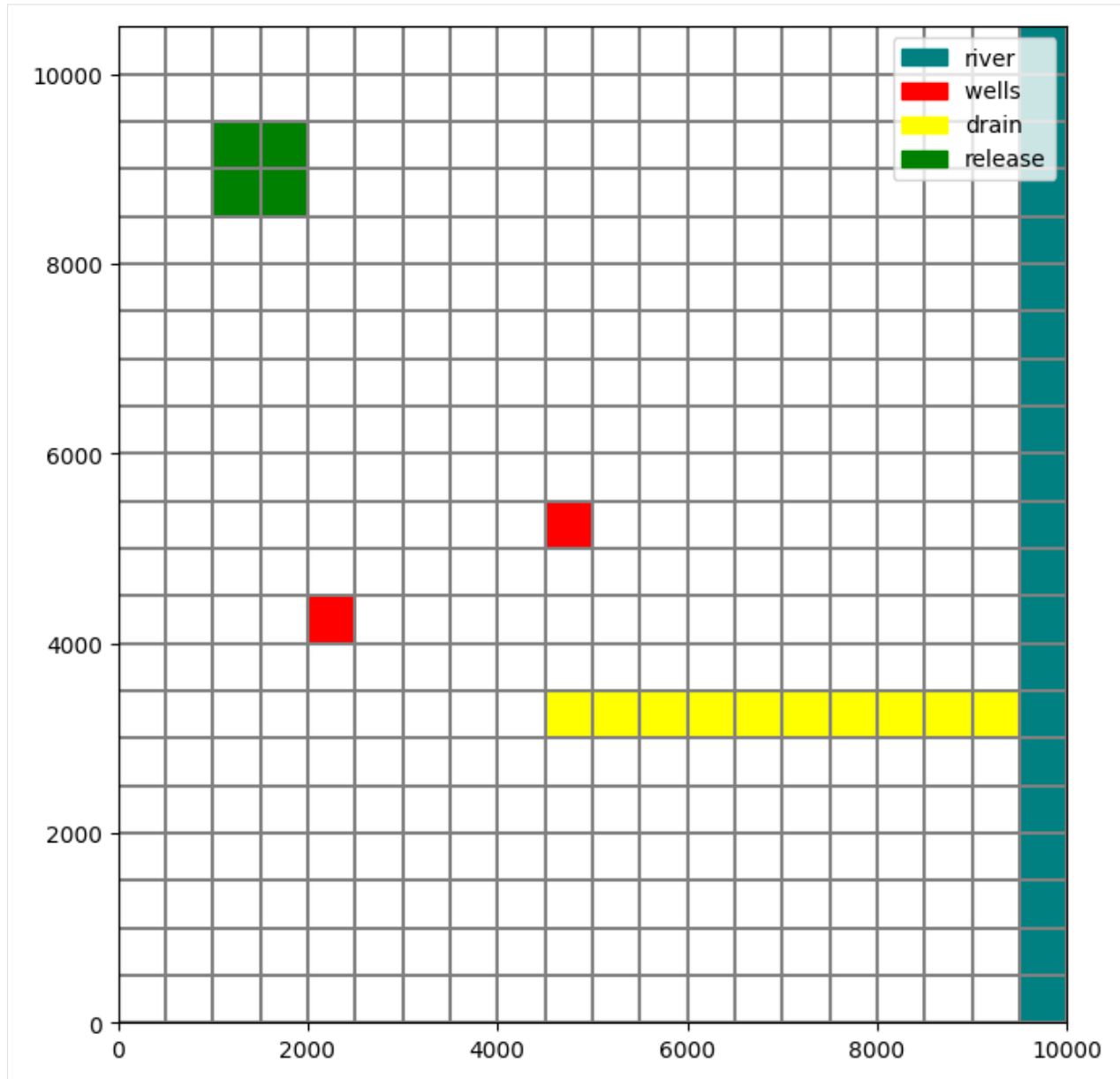
```

Take a look at the model grid before running the simulation.

```
[9]: def add_release(ax):
    ax.add_patch(
        mpl.patches.Rectangle(
            (2 * delc, (nrow - 2) * delr),
            1000,
            -1000,
            facecolor="green",
        )
    )

def add_legend(ax):
    ax.legend(
        handles=[
            mpl.patches.Patch(color="teal", label="river"),
            mpl.patches.Patch(color="red", label="wells "),
            mpl.patches.Patch(color="yellow", label="drain"),
            mpl.patches.Patch(color="green", label="release"),
        ]
    )

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mv = flopy.plot.PlotMapView(model=gwf)
mv.plot_grid()
mv.plot_bc("DRN")
mv.plot_bc("RIV")
mv.plot_bc("WEL", plotAll=True) # include both wells (1st and 3rd layer)
add_release(ax)
add_legend(ax)
plt.show()
```

Running the simulation

Run the MODFLOW 6 flow simulation.

```
[10]: sim.write_simulation()
      success, buff = sim.run_simulation(silent=True, report=True)
      assert success, "Failed to run simulation."
      for line in buff:
          print(line)
```

```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
```

(continues on next page)

(continued from previous page)

```
writing solution package ims...
writing model mp7_ex03a_mf6...
  writing model name file...
  writing package dis...
  writing package ic...
  writing package npf...
  writing package rcha_0...
  writing package wel_0...
INFORMATION: maxbound in ('gwf6', 'wel', 'dimensions') changed to 2 based on size of ↵
↵stress_period_data
  writing package riv_0...
INFORMATION: maxbound in ('gwf6', 'riv', 'dimensions') changed to 21 based on size of ↵
↵stress_period_data
  writing package drn_0...
INFORMATION: maxbound in ('gwf6', 'drn', 'dimensions') changed to 11 based on size of ↵
↵stress_period_data
  writing package oc...
```

MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:52

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

Solving:	Stress period:	1	Time step:	1
Solving:	Stress period:	2	Time step:	1
Solving:	Stress period:	2	Time step:	2
Solving:	Stress period:	2	Time step:	3
Solving:	Stress period:	2	Time step:	4
Solving:	Stress period:	2	Time step:	5
Solving:	Stress period:	2	Time step:	6
Solving:	Stress period:	2	Time step:	7

(continues on next page)

(continued from previous page)

```

Solving: Stress period:      2      Time step:      8
Solving: Stress period:      2      Time step:      9
Solving: Stress period:      2      Time step:     10
Solving: Stress period:      3      Time step:      1

```

```

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:52
Elapsed run time: 0.083 Seconds

```

```

Normal termination of simulation.

```

Create and run MODPATH 7 particle tracking model in combined mode, which includes both pathline and timeseries.

```

[11]: # create modpath files
mp = flopy.modpath.Modpath7(
    modelname=f"{sim_name}_mp",
    flowmodel=gwf,
    exe_name="mp7",
    model_ws=workspace,
)
mpbas = flopy.modpath.Modpath7Bas(mp, porosity=0.1, defaultiface=defaultiface)
mpsim = flopy.modpath.Modpath7Sim(
    mp,
    simulationtype="combined",
    trackingdirection="forward",
    weaksinkoption="pass_through",
    weaksourceoption="pass_through",
    budgetoutputoption="summary",
    referencetime=[0, 0, 0.9],
    timepointdata=[10, 20.0], # release every 20 days, for 200 days
    zonedataoption="on",
    zones=zone_maps,
    particlegroups=pgs,
)

mp.write_input()
success, buff = mp.run_model(silent=True, report=True)
assert success
for line in buff:
    print(line)

```

```

MODPATH Version 7.2.001

```

```

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

```

```

Run particle tracking simulation ...

```

```

Processing Time Step      1 Period      1. Time = 1.00000E+05 Steady-state flow
Processing Time Step      1 Period      2. Time = 1.03650E+05 Steady-state flow
Processing Time Step      2 Period      2. Time = 1.07300E+05 Steady-state flow
Processing Time Step      3 Period      2. Time = 1.10950E+05 Steady-state flow
Processing Time Step      4 Period      2. Time = 1.14600E+05 Steady-state flow
Processing Time Step      5 Period      2. Time = 1.18250E+05 Steady-state flow
Processing Time Step      6 Period      2. Time = 1.21900E+05 Steady-state flow

```

(continues on next page)

(continued from previous page)

```

Processing Time Step    7 Period    2. Time = 1.25550E+05 Steady-state flow
Processing Time Step    8 Period    2. Time = 1.29200E+05 Steady-state flow
Processing Time Step    9 Period    2. Time = 1.32850E+05 Steady-state flow
Processing Time Step   10 Period    2. Time = 1.36500E+05 Steady-state flow
Processing Time Step    1 Period    3. Time = 2.36500E+05 Steady-state flow

```

Particle Summary:

```

    0 particles are pending release.
    0 particles remain active.
    0 particles terminated at boundary faces.
    0 particles terminated at weak sink cells.
    0 particles terminated at weak source cells.
  108 particles terminated at strong source/sink cells.
    0 particles terminated in cells with a specified zone number.
    0 particles were stranded in inactive or dry cells.
    0 particles were unreleased.
    0 particles have an unknown status.

```

Normal termination.

Inspecting results

First we need the particle termination locations.

```

[12]: wel_locs = [w[0:3] for w in wells]
      riv_locs = [(0, i, 19) for i in range(20)]
      drn_locs = [(drain[0], drain[1], d) for d in range(drain[2][0], drain[2][1])]
      wel_nids = gwf.modelgrid.get_node(wel_locs)
      riv_nids = gwf.modelgrid.get_node(riv_locs)
      drn_nids = gwf.modelgrid.get_node(drn_locs)

```

Next, load pathline data from the MODPATH 7 pathline output file, filtering by termination location.

```

[13]: fpth = workspace / f"{sim_name}_mp.mppth"
      p = flopy.utils.PathlineFile(fpth)

      pl1 = p.get_destination_pathline_data(wel_nids, to_reccarray=True)
      pl2 = p.get_destination_pathline_data(riv_nids + drn_nids, to_reccarray=True)

```

Load endpoint data from the MODPATH 7 endpoint output file.

```

[14]: fpth = workspace / f"{sim_name}_mp.mpend"
      e = flopy.utils.EndpointFile(fpth)

      ep1 = e.get_destination_endpoint_data(dest_cells=wel_nids)
      ep2 = e.get_destination_endpoint_data(dest_cells=riv_nids + drn_nids)

```

Extract head data from the GWF model's output files.

```

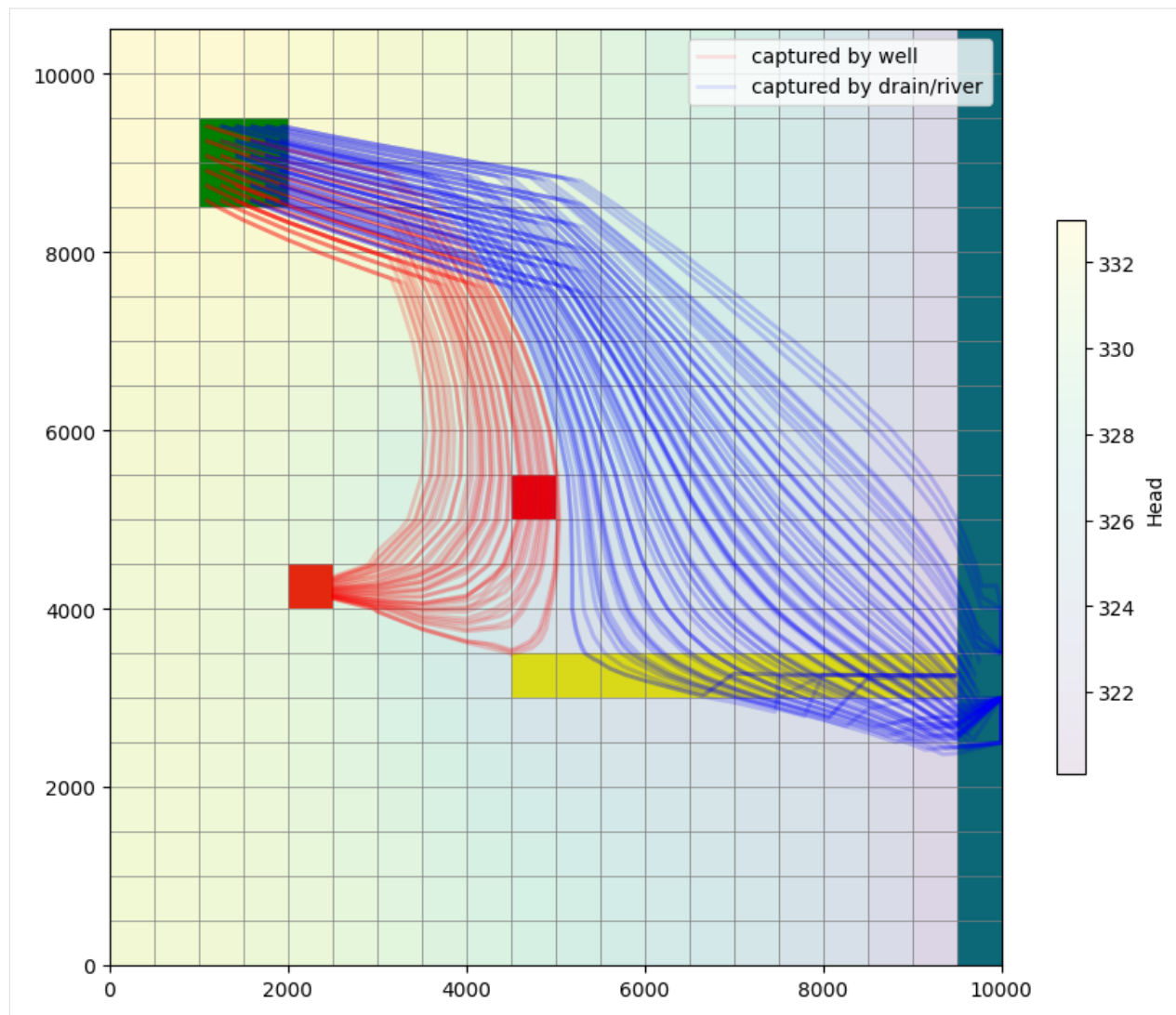
[15]: hf = flopy.utils.HeadFile(workspace / f"{sim_name}.hds")
      head = hf.get_data()

```

Plot heads over a map view of the model, then add particle starting points and pathlines. The apparent number of particle starting locations is less than the total number of particles because a separate particle begins at each location every 20 days during the release period at the beginning of the simulation.

```
[16]: fig = plt.figure(figsize=(10, 10))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")

      mv = flopy.plot.PlotMapView(model=gwf)
      mv.plot_grid(lw=0.5)
      mv.plot_bc("DRN")
      mv.plot_bc("RIV")
      mv.plot_bc("WEL", plotAll=True)
      hd = mv.plot_array(head, alpha=0.1)
      cb = plt.colorbar(hd, shrink=0.5)
      cb.set_label("Head")
      mv.plot_pathline(
          pl1, layer="all", alpha=0.1, colors=["red"], lw=2, label="captured by well"
      )
      mv.plot_pathline(
          pl2,
          layer="all",
          alpha=0.1,
          colors=["blue"],
          lw=2,
          label="captured by drain/river",
      )
      add_release(ax)
      mv.ax.legend()
      plt.show()
```



Clean up the temporary directory.

```
[17]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.8.5 Using MODPATH 7 with a DISV unstructured model

This is a replication of the MODPATH Problem 2 example that is described on page 12 of the `modpath_7_examples.pdf` file. The results shown here should be the same as the results in the MODPATH example, however, the vertex and node numbering used here may be different from the numbering used in MODPATH, so head values may not be compared directly without some additional mapping.

Part I. Setup Notebook

```
[1]: import os

[2]: import sys
    from pathlib import Path
    from tempfile import TemporaryDirectory

    import matplotlib as mpl
    import matplotlib.pyplot as plt
    import numpy as np

    proj_root = Path.cwd().parent.parent

    # run installed version of flopy or add local path
    try:
        import flopy
    except:
        sys.path.append(proj_root)
        import flopy

    print(sys.version)
    print("numpy version: {}".format(np.__version__))
    print("matplotlib version: {}".format(mpl.__version__))
    print("flopy version: {}".format(flopy.__version__))

    # temporary directory
    temp_dir = TemporaryDirectory()
    workspace = Path(temp_dir.name)

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

Part II. Gridgen Creation of Model Grid

Create the base model grid.

```
[3]: Lx = 10000.0
      Ly = 10500.0
      nlay = 3
      nrow = 21
      ncol = 20
      delr = Lx / ncol
      delc = Ly / nrow
      top = 400
      botm = [220, 200, 0]
```

```
[4]: ms = flopy.modflow.Modflow()
      dis5 = flopy.modflow.ModflowDis(
          ms,
          nlay=nlay,
          nrow=nrow,
          ncol=ncol,
          delr=delr,
          delc=delc,
          top=top,
          botm=botm,
      )
```

Create the Gridgen object.

```
[5]: from flopy.utils.gridgen import Gridgen

      model_name = "mp7p2_u"
      model_ws = workspace / "mp7_ex2" / "mf6"
      gridgen_ws = model_ws / "gridgen"
      g = Gridgen(ms.modelgrid, model_ws=gridgen_ws)
```

Refine the grid.

```
[6]: rf0shp = gridgen_ws / "rf0"
      xmin = 7 * delr
      xmax = 12 * delr
      ymin = 8 * delc
      ymax = 13 * delc
      rfpoly = [
          [
              list(
                  reversed(
                      [
                          (xmin, ymin),
                          (xmax, ymin),
                          (xmax, ymax),
                          (xmin, ymax),
                          (xmin, ymin),
                      ]
                  )
              )
          ]
```

(continues on next page)

(continued from previous page)

```

        )
    )
]
]
g.add_refinement_features(rfpoly, "polygon", 1, range(nlay))

rf1shp = gridgen_ws / "rf1"
xmin = 8 * delr
xmax = 11 * delr
ymin = 9 * delc
ymax = 12 * delc
rfpoly = [
    [
        list(
            reversed(
                [
                    (xmin, ymin),
                    (xmax, ymin),
                    (xmax, ymax),
                    (xmin, ymax),
                    (xmin, ymin),
                ]
            )
        )
    ]
]
g.add_refinement_features(rfpoly, "polygon", 2, range(nlay))

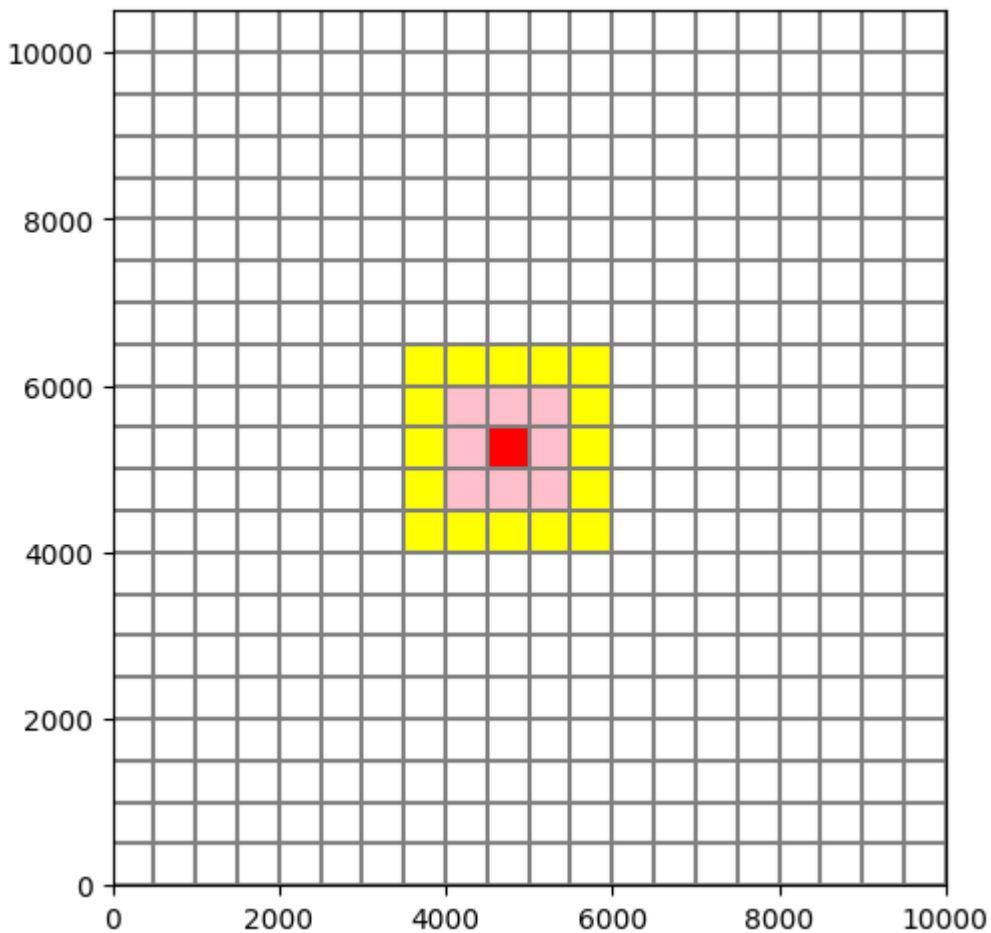
rf2shp = gridgen_ws / "rf2"
xmin = 9 * delr
xmax = 10 * delr
ymin = 10 * delc
ymax = 11 * delc
rfpoly = [
    [
        list(
            reversed(
                [
                    (xmin, ymin),
                    (xmax, ymin),
                    (xmax, ymax),
                    (xmin, ymax),
                    (xmin, ymin),
                ]
            )
        )
    ]
]
g.add_refinement_features(rfpoly, "polygon", 3, range(nlay))

```

Show the model grid with refinement levels superimposed.

```
[7]: fig = plt.figure(figsize=(5, 5), constrained_layout=True)
ax = fig.add_subplot(1, 1, 1)
mm = flopy.plot.PlotMapView(model=ms)
mm.plot_grid()
flopy.plot.plot_shapefile(rf0shp, ax=ax, facecolor="yellow", edgecolor="none")
flopy.plot.plot_shapefile(rf1shp, ax=ax, facecolor="pink", edgecolor="none")
flopy.plot.plot_shapefile(rf2shp, ax=ax, facecolor="red", edgecolor="none")
```

```
[7]: <matplotlib.collections.PatchCollection at 0x7fad30be6ac0>
```



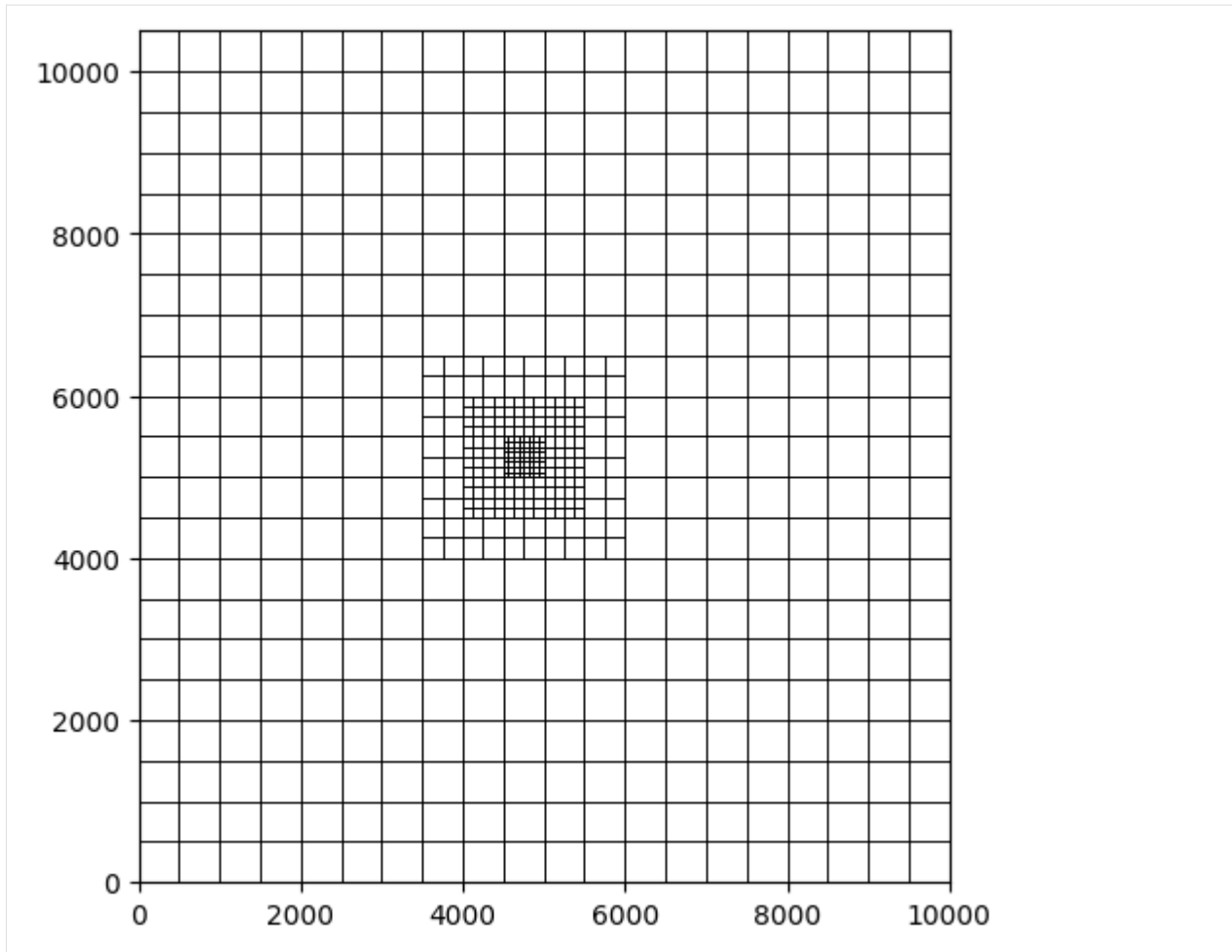
Build the refined grid.

```
[8]: g.build(verbose=False)
```

Show the refined grid.

```
[9]: fig = plt.figure(figsize=(5, 5), constrained_layout=True)
ax = fig.add_subplot(1, 1, 1, aspect="equal")
g.plot(ax, linewidth=0.5)
```

```
[9]: <matplotlib.collections.PatchCollection at 0x7fad30c65ac0>
```



Extract the refined grid's properties.

```
[10]: gridprops = g.get_gridprops_disv()
      ncpl = gridprops["ncpl"]
      top = gridprops["top"]
      botm = gridprops["botm"]
      nvert = gridprops["nvert"]
      vertices = gridprops["vertices"]
      cell2d = gridprops["cell2d"]
```

Part III. Create the FloPy Model

```
[11]: # create simulation
      sim = flopy.mf6.MFSimulation(
          sim_name=model_name, version="mf6", exe_name="mf6", sim_ws=model_ws
      )

      # create tdis package
      tdis_rc = [(1000.0, 1, 1.0)]
      tdis = flopy.mf6.ModflowTdis(
```

(continues on next page)

(continued from previous page)

```

    sim, pname="tdis", time_units="DAYS", perioddata=tdis_rc
)

# create gwf model
gwf = flopy.mf6.ModflowGwf(
    sim, modelname=model_name, model_nam_file("{} .nam".format(model_name))
)
gwf.name_file.save_flows = True

# create iterative model solution and register the gwf model with it
ims = flopy.mf6.ModflowIms(
    sim,
    pname="ims",
    print_option="SUMMARY",
    complexity="SIMPLE",
    outer_dvclose=1.0e-5,
    outer_maximum=100,
    under_relaxation="NONE",
    inner_maximum=100,
    inner_dvclose=1.0e-6,
    rcloserecord=0.1,
    linear_acceleration="BICGSTAB",
    scaling_method="NONE",
    reordering_method="NONE",
    relaxation_factor=0.99,
)
sim.register_ims_package(ims, [gwf.name])

# disv
disv = flopy.mf6.ModflowGwfdisv(
    gwf,
    nlay=nlay,
    ncpl=ncpl,
    top=top,
    botm=botm,
    nvert=nvert,
    vertices=vertices,
    cell2d=cell2d,
)

# initial conditions
ic = flopy.mf6.ModflowGwfic(gwf, pname="ic", strt=320.0)

# node property flow
npf = flopy.mf6.ModflowGwfnpf(
    gwf,
    xt3doptions=["xt3d"],
    icelltype=[1, 0, 0],
    k=[50.0, 0.01, 200.0],
    k33=[10.0, 0.01, 20.0],
)

```

(continues on next page)

(continued from previous page)

```

# wel
wellpoints = [(4750.0, 5250.0)]
welcells = g.intersect(wellpoints, "point", 0)
# welspd = flopy.mf6.ModflowGwfwel.stress_period_data.empty(gwf, maxbound=1, aux_vars=[
↪ 'iface'])
welspd = [[(2, icpl), -150000, 0] for icpl in welcells["nodenumber"]]
wel = flopy.mf6.ModflowGwfwel(
    gwf, print_input=True, auxiliary=[("iface",)], stress_period_data=welspd
)

# rch
aux = [np.ones(ncpl, dtype=int) * 6]
rch = flopy.mf6.ModflowGwfrcha(
    gwf, recharge=0.005, auxiliary=[("iface",)], aux={0: [6]}
)

# riv
riverline = [(Lx - 1.0, Ly), (Lx - 1.0, 0.0)]
rivcells = g.intersect(riverline, "line", 0)
rivspd = [(0, icpl), 320.0, 100000.0, 318] for icpl in rivcells["nodenumber"]]
riv = flopy.mf6.ModflowGwfriv(gwf, stress_period_data=rivspd)

# output control
oc = flopy.mf6.ModflowGwfoc(
    gwf,
    pname="oc",
    budget_filerecord="{}.cbb".format(model_name),
    head_filerecord="{}.hds".format(model_name),
    headprintrecord=[("COLUMNS", 10, "WIDTH", 15, "DIGITS", 6, "GENERAL")],
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
    printrecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
)

```

WARNING: Unable to resolve dimension of ('gwf6', 'disv', 'cell2d', 'cell2d', 'icvert')
↪ based on shape "ncvert".

Now write the simulation input files.

```
[12]: sim.write_simulation()
```

```

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims...
  writing model mp7p2_u...
    writing model name file...
    writing package disv...
    writing package ic...
    writing package npf...
    writing package wel_0...
INFORMATION: maxbound in ('gwf6', 'wel', 'dimensions') changed to 1 based on size of
↪ stress_period_data
  writing package rcha_0...
  writing package riv_0...

```

(continues on next page)

(continued from previous page)

```

INFORMATION: maxbound in ('gwf6', 'riv', 'dimensions') changed to 21 based on size of
↳ stress_period_data
   writing package oc...

```

Part IV. Run the MODFLOW 6 Model

```

[13]: success, buff = sim.run_simulation(silent=True, report=True)
      assert success, "mf6 failed to run"
      for line in buff:
          print(line)

```

```

MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

```

```

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

```

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:33

```

```

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

```

```

Solving: Stress period:      1      Time step:      1

```

```

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:33
Elapsed run time: 0.134 Seconds

```

```

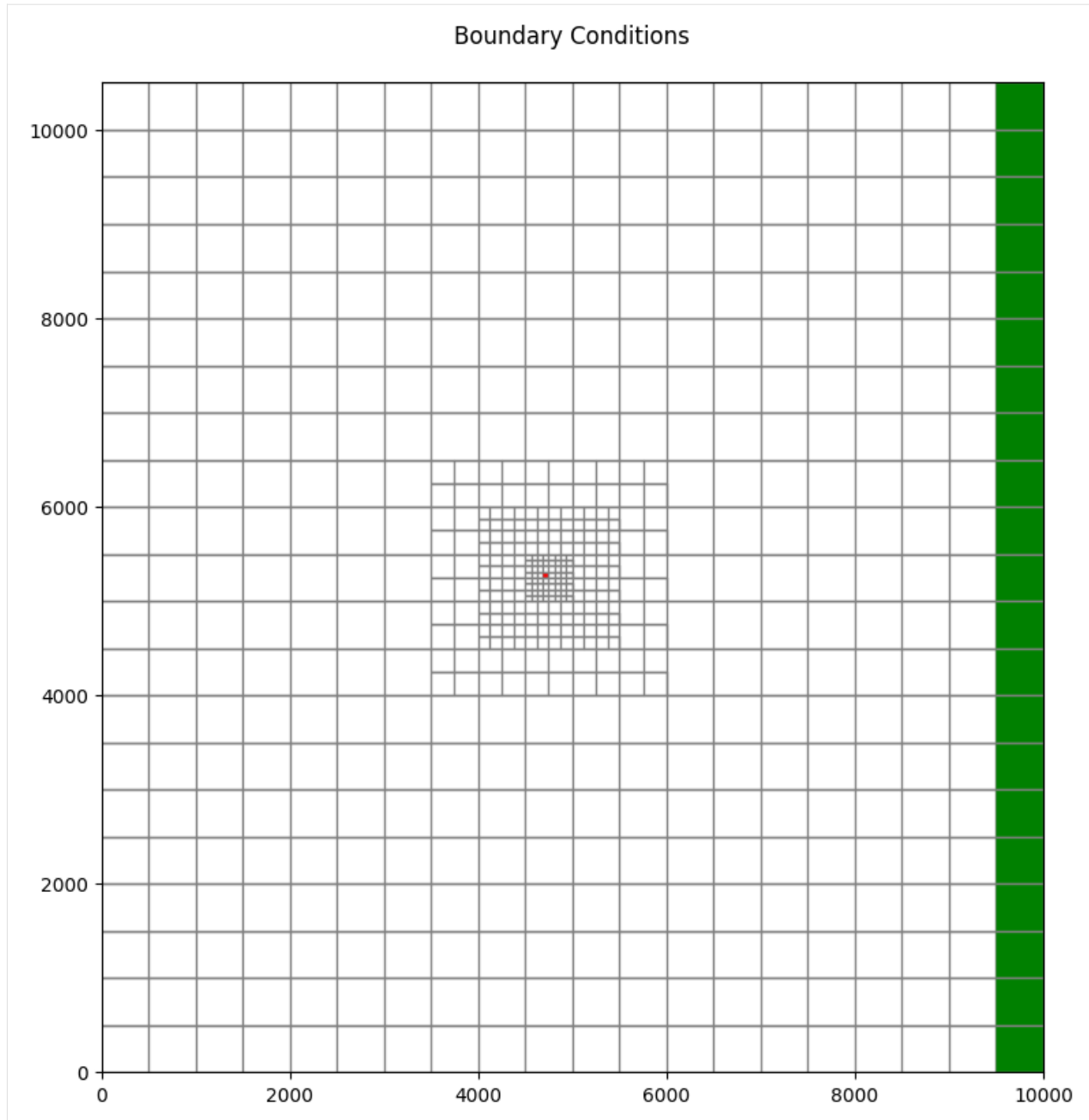
Normal termination of simulation.

```

Part V. Import and Plot the Results

Plot the boundary conditions on the grid.

```
[14]: fname = os.path.join(model_ws, model_name + ".disv.grb")
      grd = flopy.mf6.utils.MfGrdFile(fname, verbose=False)
      mg = grd.modelgrid
      ibd = np.zeros((ncpl), dtype=int)
      ibd[welcells["nodenumber"]] = 1
      ibd[rivcells["nodenumber"]] = 2
      ibd = np.ma.masked_equal(ibd, 0)
      fig = plt.figure(figsize=(8, 8), constrained_layout=True)
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      pmv = flopy.plot.PlotMapView(modelgrid=mg, ax=ax)
      ax.set_xlim(0, Lx)
      ax.set_ylim(0, Ly)
      cmap = mpl.colors.ListedColormap(
          [
              "r",
              "g",
          ]
      )
      pc = pmv.plot_array(ibd, cmap=cmap, edgecolor="gray")
      t = ax.set_title("Boundary Conditions\n")
```



```
[15]: fname = os.path.join(model_ws, model_name + ".hds")
      hdoobj = flopy.utils.HeadFile(fname)
      head = hdoobj.get_data()
      head.shape
```

```
[15]: (3, 1, 651)
```

```
[16]: ilay = 2
      cint = 0.25
      fig = plt.figure(figsize=(8, 8), constrained_layout=True)
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
```

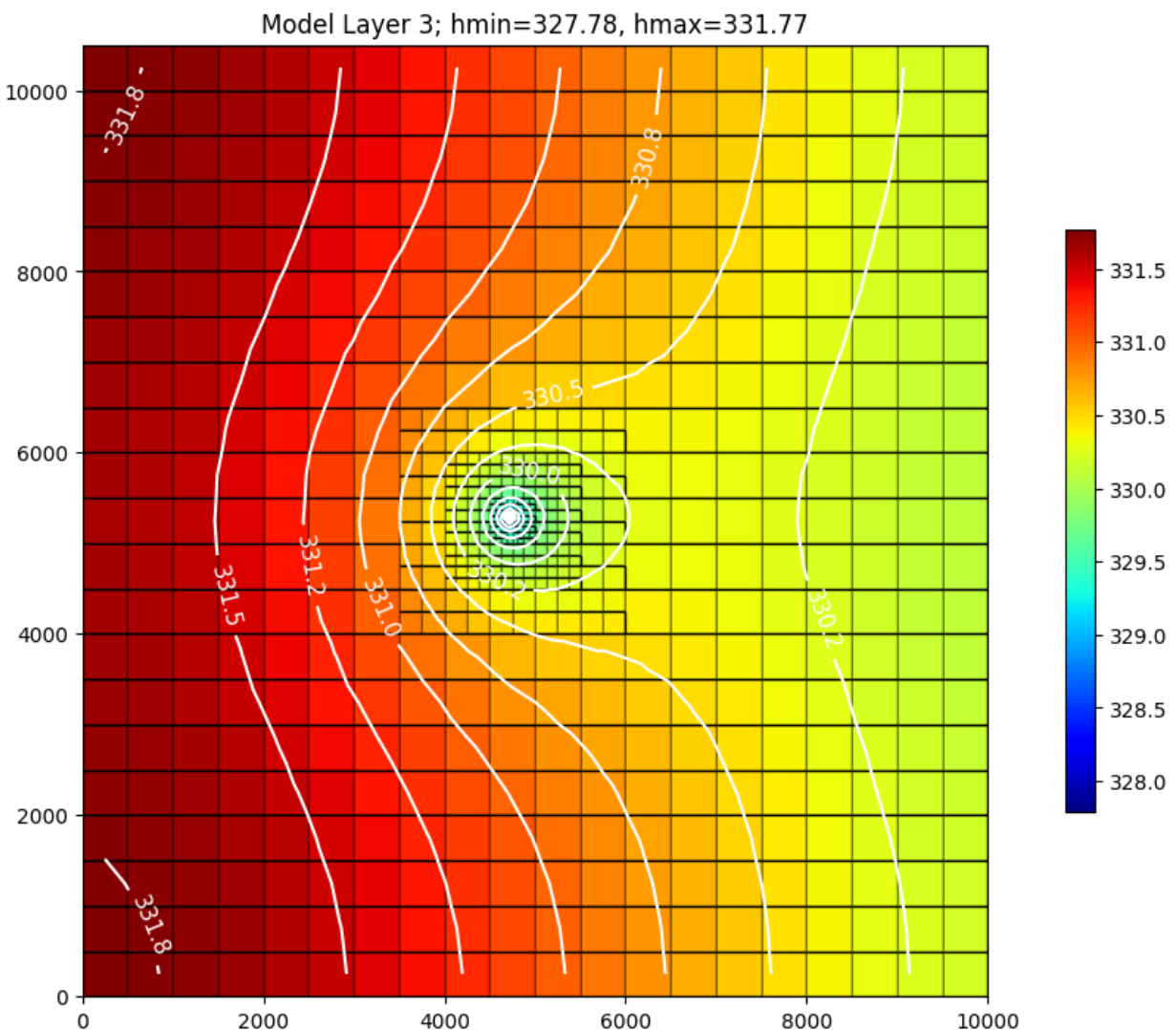
(continues on next page)

(continued from previous page)

```

mm = flopy.plot.PlotMapView(modelgrid=mg, ax=ax, layer=ilay)
ax.set_xlim(0, Lx)
ax.set_ylim(0, Ly)
pc = mm.plot_array(head[:, 0, :], cmap="jet", edgecolor="black")
hmin = head[ilay, 0, :].min()
hmax = head[ilay, 0, :].max()
levels = np.arange(np.floor(hmin), np.ceil(hmax) + cint, cint)
cs = mm.contour_array(head[:, 0, :], colors="white", levels=levels)
plt.clabel(cs, fmt="%.1f", colors="white", fontsize=11)
cb = plt.colorbar(pc, shrink=0.5)
t = ax.set_title(
    "Model Layer {}; hmin={:6.2f}, hmax={:6.2f}".format(ilay + 1, hmin, hmax)
)

```



Inspect model cells and vertices.

```
[17]: # zoom area
```

(continues on next page)

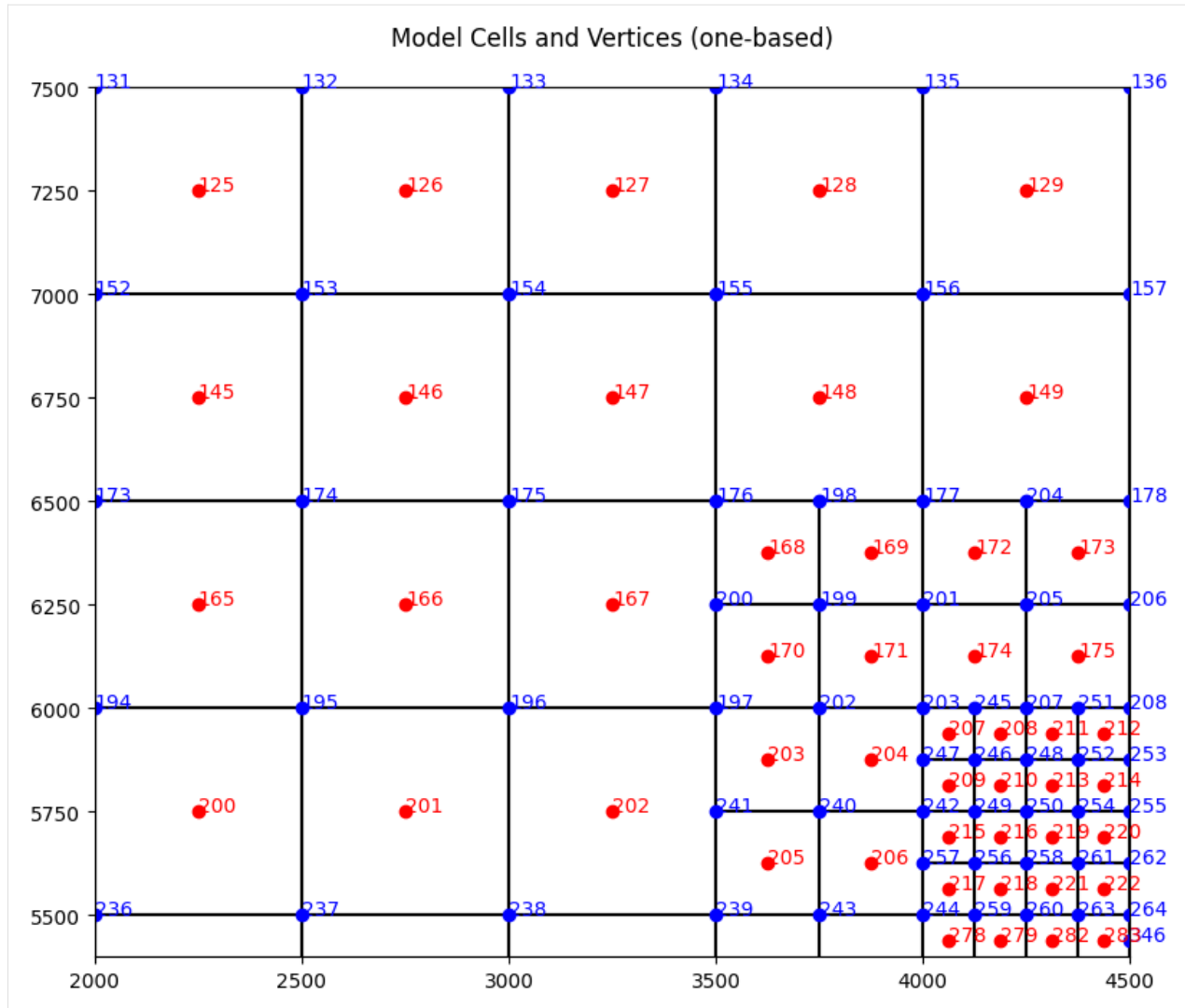
(continued from previous page)

```
xmin, xmax = 2000, 4500
ymin, ymax = 5400, 7500

mg.get_cell_vertices
fig = plt.figure(figsize=(8, 8), constrained_layout=True)
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mm = flopy.plot.PlotMapView(modelgrid=mg, ax=ax)
v = mm.plot_grid(edgecolor="black")
t = ax.set_title("Model Cells and Vertices (one-based)\n")
ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)

verts = mg.verts
ax.plot(verts[:, 0], verts[:, 1], "bo")
for i in range(ncpl):
    x, y = verts[i, 0], verts[i, 1]
    if xmin <= x <= xmax and ymin <= y <= ymax:
        ax.annotate(str(i + 1), verts[i, :], color="b")

xc, yc = mg.get_xcellcenters_for_layer(0), mg.get_ycellcenters_for_layer(0)
for i in range(ncpl):
    x, y = xc[i], yc[i]
    ax.plot(x, y, "ro")
    if xmin <= x <= xmax and ymin <= y <= ymax:
        ax.annotate(str(i + 1), (x, y), color="r")
```



Part VI. Create the FloPy MODPATH7 Models

Define names for the MODPATH 7 simulations.

```
[18]: mp_namea = model_name + "a_mp"
      mp_nameb = model_name + "b_mp"
```

Create particles for the pathline and timeseries analysis.

```
[19]: pcoord = np.array(
    [
        [0.000, 0.125, 0.500],
        [0.000, 0.375, 0.500],
        [0.000, 0.625, 0.500],
        [0.000, 0.875, 0.500],
        [1.000, 0.125, 0.500],
        [1.000, 0.375, 0.500],
```

(continues on next page)

(continued from previous page)

```

        [1.000, 0.625, 0.500],
        [1.000, 0.875, 0.500],
        [0.125, 0.000, 0.500],
        [0.375, 0.000, 0.500],
        [0.625, 0.000, 0.500],
        [0.875, 0.000, 0.500],
        [0.125, 1.000, 0.500],
        [0.375, 1.000, 0.500],
        [0.625, 1.000, 0.500],
        [0.875, 1.000, 0.500],
    ]
)
nodew = gwfdisc.ncpl.array * 2 + welcells["nodenumber"][0]
plocs = [nodew for i in range(pcoord.shape[0])]

# create particle data
pa = flopy.modpath.ParticleData(
    plocs,
    structured=False,
    localx=pcoord[:, 0],
    localy=pcoord[:, 1],
    localz=pcoord[:, 2],
    drape=0,
)

# create backward particle group
fpth = mp_namea + ".sloc"
pga = flopy.modpath.ParticleGroup(
    particlegroupname="BACKWARD1", particledata=pa, filename=fpth
)

```

Create particles for endpoint analysis.

```

[20]: facedata = flopy.modpath.FaceDataType(
    drape=0,
    verticaldivisions1=10,
    horizontaldivisions1=10,
    verticaldivisions2=10,
    horizontaldivisions2=10,
    verticaldivisions3=10,
    horizontaldivisions3=10,
    verticaldivisions4=10,
    horizontaldivisions4=10,
    rowdivisions5=0,
    columndivisions5=0,
    rowdivisions6=4,
    columndivisions6=4,
)
pb = flopy.modpath.NodeParticleData(subdivisiondata=facedata, nodes=nodew)
# create forward particle group
fpth = mp_nameb + ".sloc"
pgb = flopy.modpath.ParticleGroupNodeTemplate(

```

(continues on next page)

(continued from previous page)

```
particlegroupname="BACKWARD2", particledata=pb, filename=fpth
)
```

Create and run the pathline and timeseries analysis model.

```
[21]: # create modpath files
mp = flopy.modpath.Modpath7(
    modelname=mp_namea, flowmodel=gwf, exe_name="mp7", model_ws=model_ws
)
flopy.modpath.Modpath7Bas(mp, porosity=0.1)
flopy.modpath.Modpath7Sim(
    mp,
    simulationtype="combined",
    trackingdirection="backward",
    weaksinkoption="pass_through",
    weaksourceoption="pass_through",
    referencetime=0.0,
    stoptimeoption="extend",
    timepointdata=[500, 1000.0],
    particlegroups=pga,
)

# write modpath datasets
mp.write_input()

# run modpath
success, buff = mp.run_model(silent=True, report=True)
assert success, "mp7 failed to run"
for line in buff:
    print(line)
```

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.000000E+03 Steady-state flow

Particle Summary:

```
0 particles are pending release.
0 particles remain active.
16 particles terminated at boundary faces.
0 particles terminated at weak sink cells.
0 particles terminated at weak source cells.
0 particles terminated at strong source/sink cells.
0 particles terminated in cells with a specified zone number.
0 particles were stranded in inactive or dry cells.
0 particles were unreleased.
0 particles have an unknown status.
```

Normal termination.

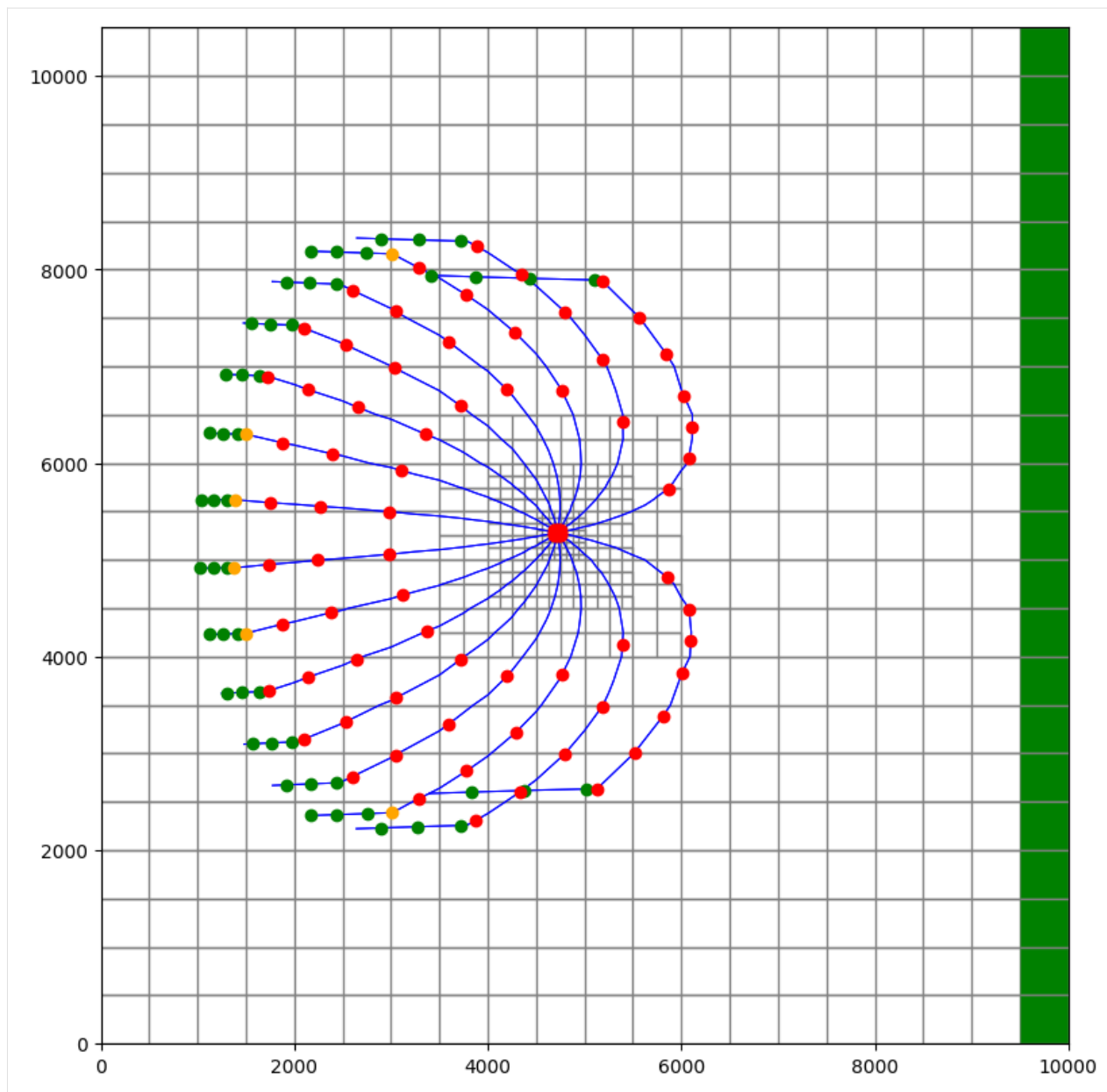
Load the pathline and timeseries data.

```
[22]: fpth = model_ws / f"{mp_namea}.mppth"
      p = flopy.utils.PathlineFile(fpth)
      p0 = p.get_alldata()
```

```
[23]: fpth = model_ws / f"{mp_namea}.timeseries"
      ts = flopy.utils.TimeseriesFile(fpth)
      ts0 = ts.get_alldata()
```

Plot the pathline and timeseries data.

```
[24]: fig = plt.figure(figsize=(8, 8), constrained_layout=True)
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      mm = flopy.plot.PlotMapView(modelgrid=mg, ax=ax)
      ax.set_xlim(0, Lx)
      ax.set_ylim(0, Ly)
      cmap = mpl.colors.ListedColormap(
          [
              "r",
              "g",
          ]
      )
      v = mm.plot_array(ibd, cmap=cmap, edgecolor="gray")
      mm.plot_pathline(p0, layer="all", colors="blue", lw=0.75)
      colors = ["green", "orange", "red"]
      for k in range(nlay):
          mm.plot_timeseries(ts0, layer=k, marker="o", lw=0, color=colors[k])
```



Create and run the endpoint analysis model.

```
[25]: # create modpath files
mp = flopy.modpath.Modpath7(
    modelname=mp_nameb, flowmodel=gwf, exe_name="mp7", model_ws=model_ws
)
flopy.modpath.Modpath7Bas(mp, porosity=0.1)
flopy.modpath.Modpath7Sim(
    mp,
    simulationtype="endpoint",
    trackingdirection="backward",
    weaksinkoption="pass_through",
    weaksourceoption="pass_through",
```

(continues on next page)

(continued from previous page)

```

    referencetime=0.0,
    stoptimeoption="extend",
    particlegroups=pgb,
)

# write modpath datasets
mp.write_input()

# run modpath
success, buff = mp.run_model(silent=True, report=True)
assert success, "mp7 failed to run"
for line in buff:
    print(line)

```

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.00000E+03 Steady-state flow

Particle Summary:

```

    0 particles are pending release.
    0 particles remain active.
416 particles terminated at boundary faces.
    0 particles terminated at weak sink cells.
    0 particles terminated at weak source cells.
    0 particles terminated at strong source/sink cells.
    0 particles terminated in cells with a specified zone number.
    0 particles were stranded in inactive or dry cells.
    0 particles were unreleased.
    0 particles have an unknown status.

```

Normal termination.

Load the endpoint data.

```

[26]: fpth = model_ws / f"{mp_nameb}.mpend"
      e = flopy.utils.EndpointFile(fpth)
      e0 = e.get_alldata()

```

Plot the endpoint data.

```

[27]: fig = plt.figure(figsize=(8, 8), constrained_layout=True)
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      mm = flopy.plot.PlotMapView(modelgrid=mg, ax=ax)
      ax.set_xlim(0, Lx)
      ax.set_ylim(0, Ly)
      cmap = mpl.colors.ListedColormap(
          [
              "r",
              "g",

```

(continues on next page)

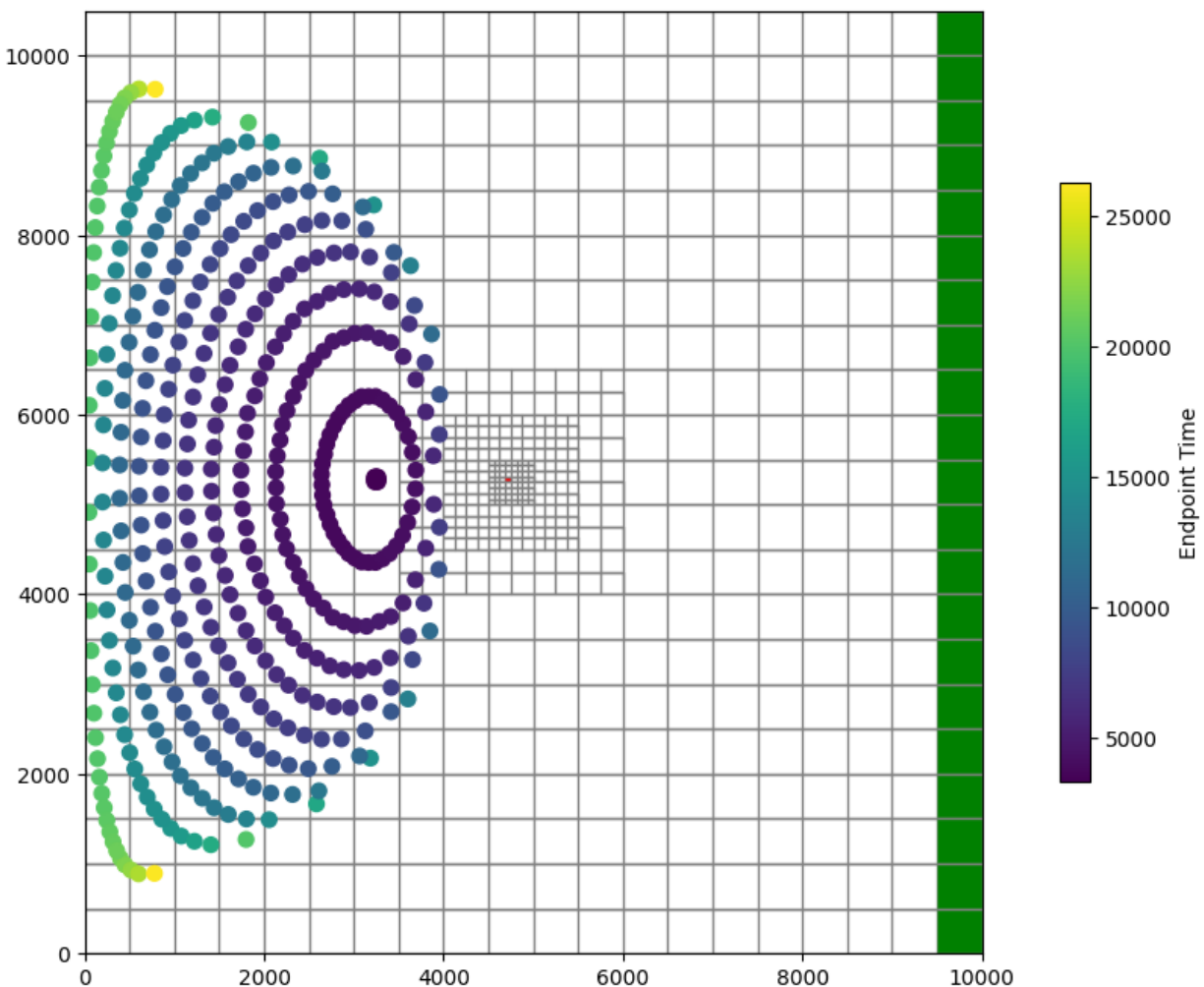
(continued from previous page)

```

]
)
v = mm.plot_array(ibd, cmap=cmap, edgecolor="gray")
mm.plot_endpoint(e0, direction="ending", colorbar=True, shrink=0.5)

```

[27]: <matplotlib.collections.PathCollection at 0x7fad23a8c2e0>



Clean up the temporary workspace.

```

[28]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass

```

6.8.6 Using MODPATH 7: DISV quadpatch example

This notebook demonstrates example 4 from the MODPATH 7 documentation, a steady-state MODFLOW 6 simulation using a quadpatch DISV grid with an irregular domain and a large number of inactive cells. Particles are tracked backwards from terminating locations, including a pair of wells in a locally-refined region of the grid and constant-head cells along the grid's right side, to release locations along the left border of the grid's active region. Injection wells along the left-hand border are used to generate boundary flows.

First import FloPy and set up a temporary workspace.

```
[1]: import sys
    from pathlib import Path
    from tempfile import TemporaryDirectory

    import matplotlib as mpl
    import matplotlib.pyplot as plt
    import numpy as np

    proj_root = Path.cwd().parent.parent

    # run installed version of flopy or add local path
    try:
        import flopy
    except:
        sys.path.append(proj_root)
        import flopy

    temp_dir = TemporaryDirectory()
    workspace = Path(temp_dir.name)
    sim_name = "ex04_mf6"

    print("Python version:", sys.version)
    print("NumPy version:", np.__version__)
    print("Matplotlib version:", mpl.__version__)
    print("FloPy version:", flopy.__version__)

Python version: 3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
NumPy version: 1.24.4
Matplotlib version: 3.7.3
FloPy version: 3.4.3
```

Grid creation/refinement

In this example we use GRIDGEN to create a quadpatch grid with a refined region in the upper left quadrant.

The grid has 3 nested refinement levels, all nearly but not perfectly rectangular (a 500x500 area is carved out of each corner of each). Outer levels of refinement have a width of 500. To produce this pattern we use 5 rectangular polygons for each level.

First, create the coarse-grained grid discretization.

```
[2]: nlay, nrow, ncol = 1, 21, 26 # coarsest-grained grid is 21x26
delr = delc = 500.0
top = 100.0
botm = np.zeros((nlay, nrow, ncol), dtype=np.float32)
ms = flopy.modflow.Modflow()
dis = flopy.modflow.ModflowDis(
    ms,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=delr,
    delc=delc,
    top=top,
    botm=botm,
)
```

Next, refine the grid. Create a Gridgen object from the base grid, then add refinement features (3 groups of polygons).

```
[3]: from flopy.utils.gridgen import Gridgen

# create Gridgen workspace
gridgen_ws = workspace / "gridgen"
gridgen_ws.mkdir()

# create Gridgen object
g = Gridgen(ms.modelgrid, model_ws=gridgen_ws)

# add polygon for each refinement level
outer_polygon = [
    [
        (2500, 6000),
        (2500, 9500),
        (3000, 9500),
        (3000, 10000),
        (6000, 10000),
        (6000, 9500),
        (6500, 9500),
        (6500, 6000),
        (6000, 6000),
        (6000, 5500),
        (3000, 5500),
        (3000, 6000),
        (2500, 6000),
    ]
]
g.add_refinement_features([outer_polygon], "polygon", 1, range(nlay))
refshp0 = gridgen_ws / "rf0"

middle_polygon = [
    [
        (3000, 6500),
        (3000, 9000),
        (3500, 9000),
```

(continues on next page)

(continued from previous page)

```

        (3500, 9500),
        (5500, 9500),
        (5500, 9000),
        (6000, 9000),
        (6000, 6500),
        (5500, 6500),
        (5500, 6000),
        (3500, 6000),
        (3500, 6500),
        (3000, 6500),
    ]
]
g.add_refinement_features([middle_polygon], "polygon", 2, range(nlay))
refshp1 = gridgen_ws / "rf1"

inner_polygon = [
    [
        (3500, 7000),
        (3500, 8500),
        (4000, 8500),
        (4000, 9000),
        (5000, 9000),
        (5000, 8500),
        (5500, 8500),
        (5500, 7000),
        (5000, 7000),
        (5000, 6500),
        (4000, 6500),
        (4000, 7000),
        (3500, 7000),
    ]
]
g.add_refinement_features([inner_polygon], "polygon", 3, range(nlay))
refshp2 = gridgen_ws / "rf2"

```

Create and plot the refined grid with refinement levels superimposed.

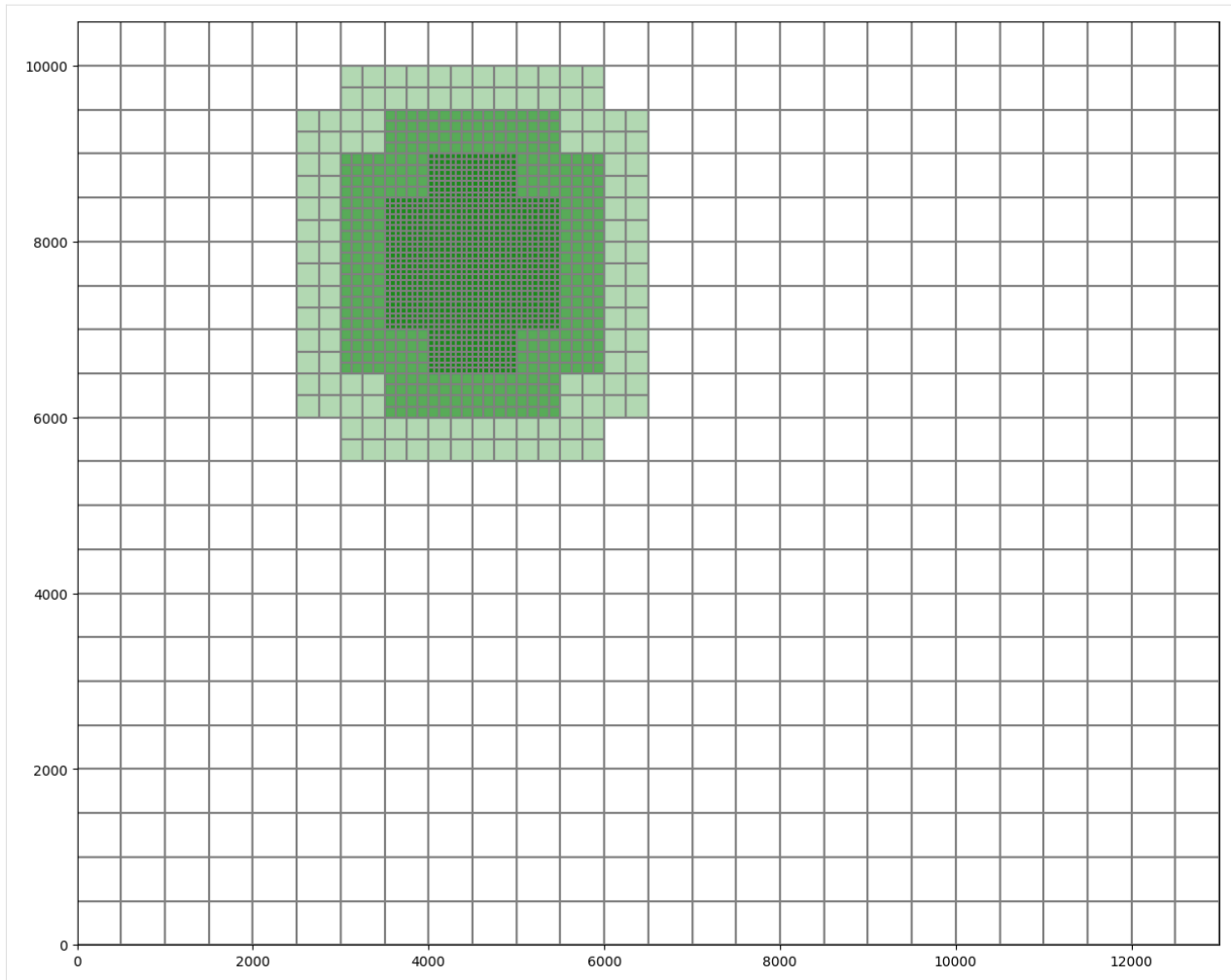
```

[4]: g.build(verbose=False)
grid = flopy.discretization.VertexGrid(**g.get_gridprops_vertexgrid())

fig = plt.figure(figsize=(15, 15))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mm = flopy.plot.PlotMapView(model=ms)
grid.plot(ax=ax)
flopy.plot.plot_shapefile(refshp0, ax=ax, facecolor="green", alpha=0.3)
flopy.plot.plot_shapefile(refshp1, ax=ax, facecolor="green", alpha=0.5)
flopy.plot.plot_shapefile(str(refshp2), ax=ax, facecolor="green", alpha=0.7)

[4]: <matplotlib.collections.PatchCollection at 0x7f6de0fda2b0>

```



Groundwater flow model

Next, create a GWF model. The particle-tracking model will consume its output.

```
[5]: # simulation
sim = flopy.mf6.MFSimulation(
    sim_name=sim_name, sim_ws=workspace, exe_name="mf6", version="mf6"
)

# temporal discretization
tdis = flopy.mf6.ModflowTdis(
    sim, time_units="days", nper=1, perioddata=[(10000, 1, 1.0)]
)

# iterative model solver
ims = flopy.mf6.ModflowIms(
    sim,
    pname="ims",
    complexity="SIMPLE",
    outer_dvclose=1e-4,
```

(continues on next page)

(continues on next page)

(continues on next page)

(continued from previous page)

```

# initial conditions
ic = flopy.mf6.ModflowGwfic(gwf, strt=150.0)

# wells are tuples (layer, node number, q, iface)
wells = [
    # negative q: discharge
    (0, 861, -30000.0, 0),
    (0, 891, -30000.0, 0),
    # positive q: injection
    (0, 1959, 10000.0, 1),
    (0, 1932, 10000.0, 3),
    (0, 1931, 10000.0, 3),
    (0, 1930, 5000.0, 1),
    (0, 1930, 5000.0, 3),
    (0, 1903, 5000.0, 1),
    (0, 1903, 5000.0, 3),
    (0, 1876, 10000.0, 3),
    (0, 1875, 10000.0, 3),
    (0, 1874, 5000.0, 1),
    (0, 1874, 5000.0, 3),
    (0, 1847, 10000.0, 3),
    (0, 1846, 5000.0, 3),
    (0, 1845, 5000.0, 1),
    (0, 1845, 5000.0, 3),
    (0, 1818, 5000.0, 1),
    (0, 1818, 5000.0, 3),
    (0, 1792, 10000.0, 1),
    (0, 1766, 10000.0, 1),
    (0, 1740, 5000.0, 1),
    (0, 1740, 5000.0, 4),
    (0, 1715, 5000.0, 1),
    (0, 1715, 5000.0, 4),
    (0, 1690, 10000.0, 1),
    (0, 1646, 5000.0, 1),
    (0, 1646, 5000.0, 4),
    (0, 1549, 5000.0, 1),
    (0, 1549, 5000.0, 4),
    (0, 1332, 5000.0, 4),
    (0, 1332, 5000.0, 1),
    (0, 1021, 2500.0, 1),
    (0, 1021, 2500.0, 4),
    (0, 1020, 5000.0, 1),
    (0, 708, 2500.0, 1),
    (0, 708, 2500.0, 4),
    (0, 711, 625.0, 1),
    (0, 711, 625.0, 4),
    (0, 710, 625.0, 1),
    (0, 710, 625.0, 4),
    (0, 409, 1250.0, 1),
    (0, 407, 625.0, 1),
    (0, 407, 625.0, 4),

```

(continues on next page)

(continued from previous page)

```

(0, 402, 625.0, 1),
(0, 402, 625.0, 4),
(0, 413, 1250.0, 1),
(0, 411, 1250.0, 1),
(0, 203, 1250.0, 1),
(0, 202, 1250.0, 1),
(0, 202, 1250.0, 4),
(0, 199, 2500.0, 1),
(0, 197, 1250.0, 1),
(0, 197, 1250.0, 4),
(0, 96, 2500.0, 1),
(0, 97, 1250.0, 1),
(0, 97, 1250.0, 4),
(0, 103, 1250.0, 1),
(0, 103, 1250.0, 4),
(0, 102, 1250.0, 1),
(0, 102, 1250.0, 4),
(0, 43, 2500.0, 1),
(0, 43, 2500.0, 4),
(0, 44, 2500.0, 1),
(0, 44, 2500.0, 4),
(0, 45, 5000.0, 4),
(0, 10, 10000.0, 1),
]
flopy.mf6.modflow.mfgwfwel.ModflowGwfwel(
    gwf,
    maxbound=68,
    auxiliary="IFACE",
    save_flows=True,
    stress_period_data={0: wells},
)

# node property flow
npf = flopy.mf6.ModflowGwnpf(
    gwf,
    xt3doptions=True,
    save_flows=True,
    save_specific_discharge=True,
    icelltype=[0],
    k=[50],
)

# constant head boundary (period, node number, head)
chd_bound = [
    (0, 1327, 150.0),
    (0, 1545, 150.0),
    (0, 1643, 150.0),
    (0, 1687, 150.0),
    (0, 1713, 150.0),
]
chd = flopy.mf6.ModflowGwfchd(
    gwf, pname="chd", save_flows=True, stress_period_data=chd_bound

```

(continues on next page)

(continued from previous page)

```

)

# output control
budget_file = f"{sim_name}.bud"
head_file = f"{sim_name}.hds"
oc = flopy.mf6.ModflowGwfoc(
    gwf,
    pname="oc",
    budget_filerecord=[budget_file],
    head_filerecord=[head_file],
    saverecord=[("HEAD", "ALL"), ("BUDGET", "ALL")],
)

WARNING: Unable to resolve dimension of ('gwf6', 'disv', 'cell2d', 'cell2d', 'icvert')_
↳ based on shape "ncvert".

```

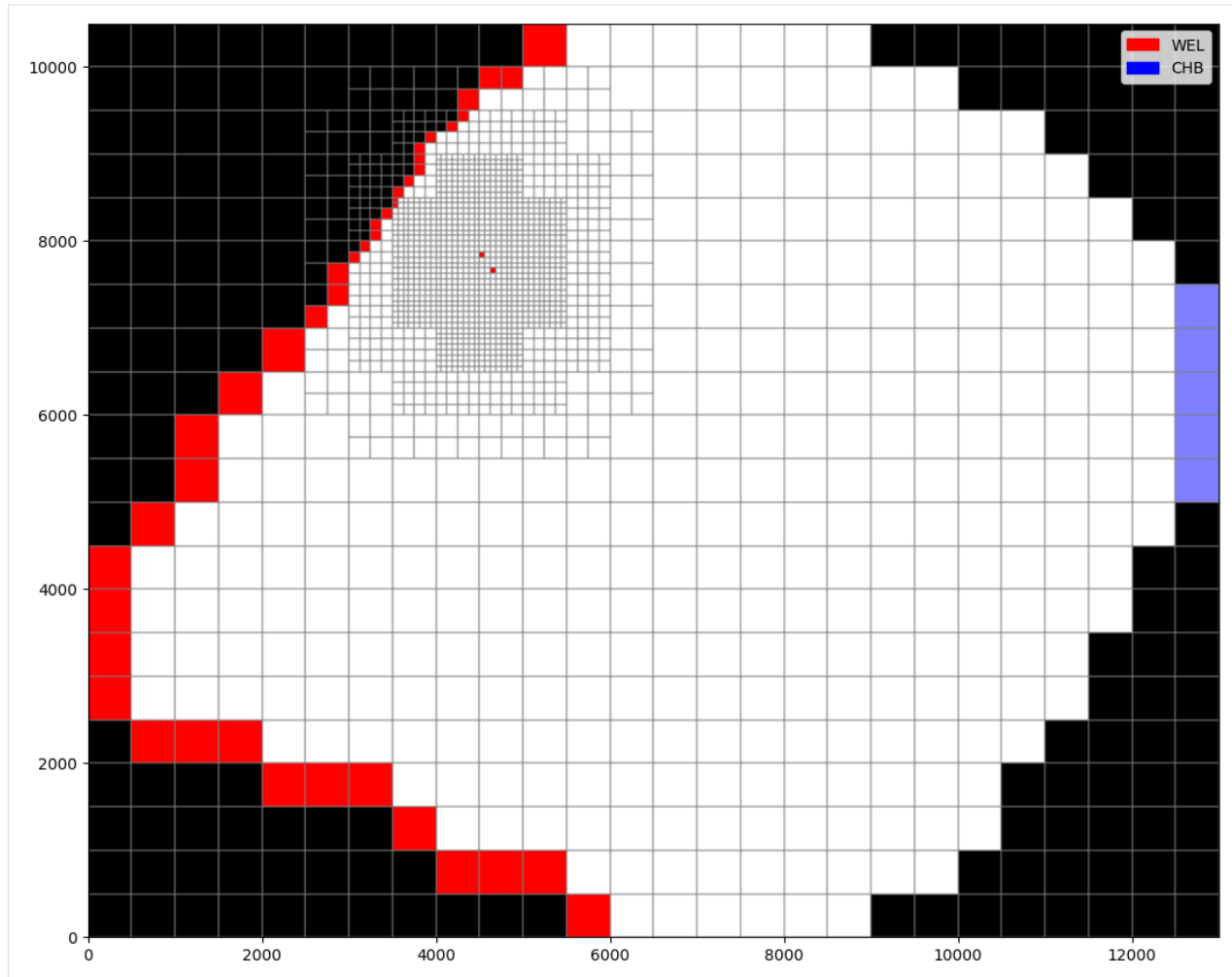
Before running the simulation, view the model's boundary conditions.

```

[6]: fig = plt.figure(figsize=(13, 13))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mv = flopy.plot.PlotMapView(model=gwf, ax=ax)
mv.plot_grid(alpha=0.3)
mv.plot_ibound()
mv.plot_bc("WEL")
ax.add_patch(
    mpl.patches.Rectangle(
        ((ncol - 1) * delc, (nrow - 6) * delr),
        1000,
        -2500,
        linewidth=5,
        facecolor="blue",
        alpha=0.5,
    )
)
ax.legend(
    handles=[
        mpl.patches.Patch(color="red", label="WEL"),
        mpl.patches.Patch(color="blue", label="CHB"),
    ]
)

plt.show()

```



Run the simulation.

```
[7]: sim.set_sim_path(workspace)
sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
assert success, f"Failed to run MF6 simulation."
for line in buff:
    print(line)
```

```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package ims...
  writing model ex04_mf6...
    writing model name file...
    writing package disv...
    writing package ic...
    writing package wel_0...
    writing package npf...
    writing package chd...
INFORMATION: maxbound in ('gwf6', 'chd', 'dimensions') changed to 5 based on size of_
```

(continues on next page)

(continued from previous page)

```

↪stress_period_data
    writing package oc...

                                MODFLOW 6
                                U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
                                VERSION 6.4.2 06/28/2023

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:47

Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam

Solving: Stress period: 1 Time step: 1

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:47
Elapsed run time: 0.073 Seconds

Normal termination of simulation.

Particle tracking

This example is a reverse-tracking model, with termination and release zones inverted: we “release” particles from the constant head boundary on the grid’s right edge and from the two pumping wells, and track the particles backwards to release locations at the wells along the left boundary of the active domain.

```

[8]: particles = [
    # node number, localx, localy, localz
    (1327, 0.000, 0.125, 0.500),
    (1327, 0.000, 0.375, 0.500),
    (1327, 0.000, 0.625, 0.500),
    (1327, 0.000, 0.875, 0.500),
    (1545, 0.000, 0.125, 0.500),
    (1545, 0.000, 0.375, 0.500),
    (1545, 0.000, 0.625, 0.500),

```

(continues on next page)

(continued from previous page)

```

(1545, 0.000, 0.875, 0.500),
(1643, 0.000, 0.125, 0.500),
(1643, 0.000, 0.375, 0.500),
(1643, 0.000, 0.625, 0.500),
(1643, 0.000, 0.875, 0.500),
(1687, 0.000, 0.125, 0.500),
(1687, 0.000, 0.375, 0.500),
(1687, 0.000, 0.625, 0.500),
(1687, 0.000, 0.875, 0.500),
(1713, 0.000, 0.125, 0.500),
(1713, 0.000, 0.375, 0.500),
(1713, 0.000, 0.625, 0.500),
(1713, 0.000, 0.875, 0.500),
(861, 0.000, 0.125, 0.500),
(861, 0.000, 0.375, 0.500),
(861, 0.000, 0.625, 0.500),
(861, 0.000, 0.875, 0.500),
(861, 1.000, 0.125, 0.500),
(861, 1.000, 0.375, 0.500),
(861, 1.000, 0.625, 0.500),
(861, 1.000, 0.875, 0.500),
(861, 0.125, 0.000, 0.500),
(861, 0.375, 0.000, 0.500),
(861, 0.625, 0.000, 0.500),
(861, 0.875, 0.000, 0.500),
(861, 0.125, 1.000, 0.500),
(861, 0.375, 1.000, 0.500),
(861, 0.625, 1.000, 0.500),
(861, 0.875, 1.000, 0.500),
(891, 0.000, 0.125, 0.500),
(891, 0.000, 0.375, 0.500),
(891, 0.000, 0.625, 0.500),
(891, 0.000, 0.875, 0.500),
(891, 1.000, 0.125, 0.500),
(891, 1.000, 0.375, 0.500),
(891, 1.000, 0.625, 0.500),
(891, 1.000, 0.875, 0.500),
(891, 0.125, 0.000, 0.500),
(891, 0.375, 0.000, 0.500),
(891, 0.625, 0.000, 0.500),
(891, 0.875, 0.000, 0.500),
(891, 0.125, 1.000, 0.500),
(891, 0.375, 1.000, 0.500),
(891, 0.625, 1.000, 0.500),
(891, 0.875, 1.000, 0.500),
]

pd = flopy.modpath.ParticleData(
    partlocs=[p[0] for p in particles],
    localx=[p[1] for p in particles],
    localy=[p[2] for p in particles],
    localz=[p[3] for p in particles],

```

(continues on next page)

(continued from previous page)

```

        timeoffset=0,
        drape=0,
    )
    pg = flopy.modpath.ParticleGroup(
        particlegroupname="G1", particledata=pd, filename=f"{sim_name}.sloc"
    )

```

Create and run the backwards particle tracking model in pathline mode.

```

[9]: mp = flopy.modpath.Modpath7(
    modelname=sim_name + "_mp",
    flowmodel=gwf,
    exe_name="mp7",
    model_ws=workspace,
)
mpbas = flopy.modpath.Modpath7Bas(
    mp,
    porosity=0.1,
)
mpsim = flopy.modpath.Modpath7Sim(
    mp,
    simulationtype="pathline",
    trackingdirection="backward",
    budgetoutputoption="summary",
    particlegroups=[pg],
)

mp.write_input()
success, buff = mp.run_model(silent=True, report=True)
assert success, "Failed to run particle-tracking model."
for line in buff:
    print(line)

```

MODPATH Version 7.2.001

Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...

Processing Time Step 1 Period 1. Time = 1.00000E+04 Steady-state flow

Particle Summary:

```

    0 particles are pending release.
    0 particles remain active.
  51 particles terminated at boundary faces.
    0 particles terminated at weak sink cells.
    1 particles terminated at weak source cells.
    0 particles terminated at strong source/sink cells.
    0 particles terminated in cells with a specified zone number.
    0 particles were stranded in inactive or dry cells.
    0 particles were unreleased.
    0 particles have an unknown status.

```

(continues on next page)

(continued from previous page)

Normal termination.

Load pathline data from the model's pathline output file.

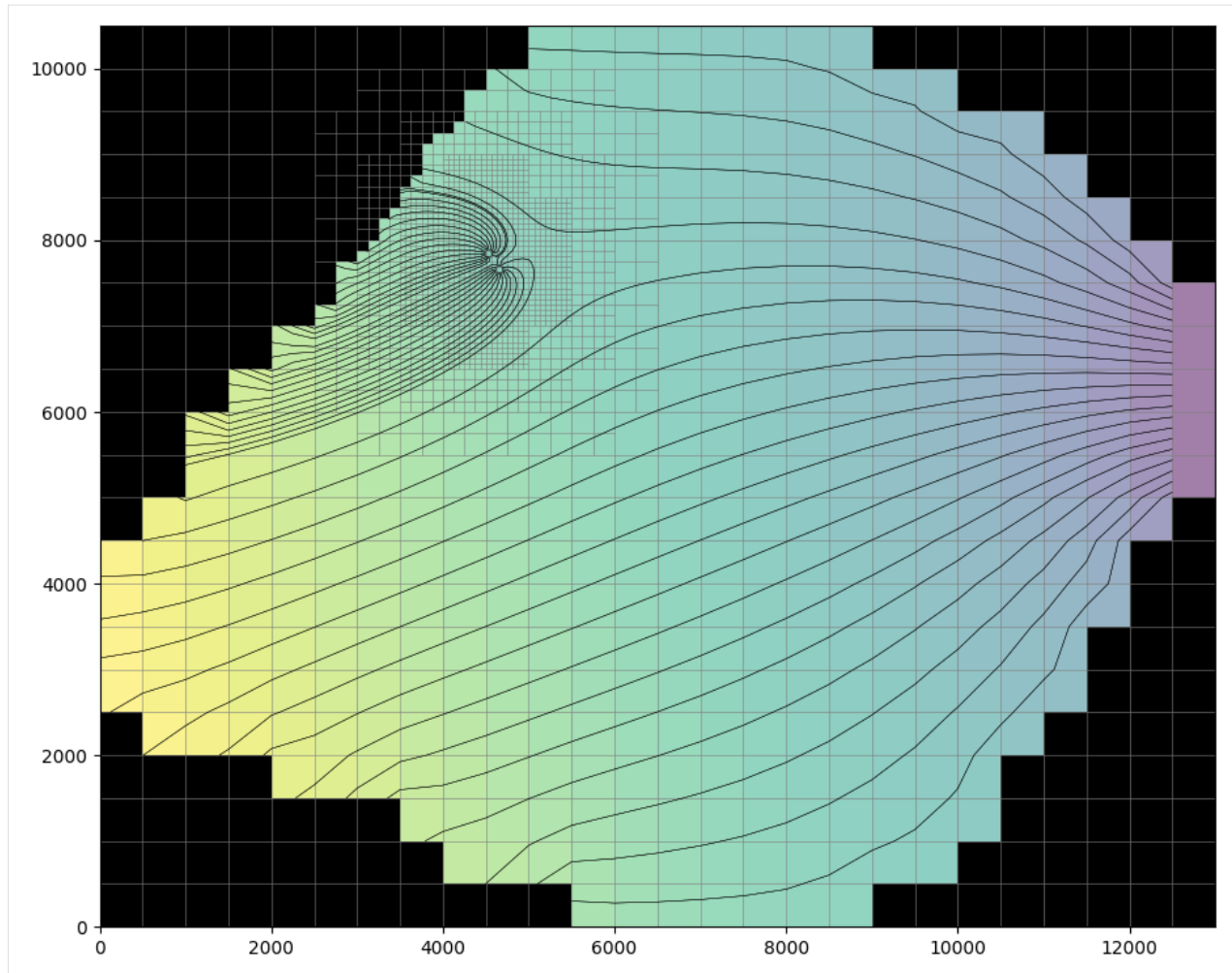
```
[10]: fpth = workspace / f"{sim_name}_mp.mppth"
p = flopy.utils.PathlineFile(fpth)
pl = p.get_destination_pathline_data(
    range(gwf.modelgrid.nnodes), to_rearray=True
)
```

Load head data.

```
[11]: hf = flopy.utils.HeadFile(workspace / f"{sim_name}.hds")
hd = hf.get_data()
```

Plot heads and particle paths over the grid.

```
[12]: fig = plt.figure(figsize=(11, 11))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mm = flopy.plot.PlotMapView(model=gwf)
mm.plot_grid(lw=0.5, alpha=0.5)
mm.plot_ibound()
mm.plot_array(hd, alpha=0.5)
mm.plot_pathline(pl, layer="all", lw=0.3, colors=["black"])
plt.show()
```



Clean up the temporary workspace.

```
[13]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
    except:
      pass
```

6.8.7 FloPy VTK/PyVista particle tracking pathline visualization demo

This notebook arranges and runs a steady state MODFLOW 6 groundwater flow model, then a MODPATH 7 particle tracking model, using the Freyberg grid. Particle tracks are exported to VTK files, then loaded and visualized with [PyVista](#), first in a static plot and then an animated GIF.

First, import FloPy and show the version.

```
[1]: import sys

import flopy
```

(continues on next page)

(continued from previous page)

```
print(sys.version)
print("flopy version: {}".format(flopy.__version__))
```

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
flopy version: 3.4.3

Load the Freyberg MF6 groundwater flow model.

```
[2]: from pathlib import Path

from flopy.mf6 import MFSimulation

mdl_name = "freyberg"
sim_name = f"mf6-{mdl_name}-vtk-pathlines"
sim_path = Path.cwd().parent / "../examples/data" / f"mf6-{mdl_name}"

sim = MFSimulation.load(sim_name=sim_name, sim_ws=sim_path)

loading simulation...
  loading simulation name file...
  loading tdis package...
  loading model gwf6...
    loading package dis...
    loading package ic...
WARNING: Block "options" is not a valid block name for file type ic.
    loading package oc...
    loading package npf...
    loading package sto...
    loading package chd...
    loading package riv...
    loading package wel...
    loading package rch...
  loading solution package freyberg...
```

Create a temporary directory and change the simulation's workspace.

```
[3]: from tempfile import TemporaryDirectory

temp_path = TemporaryDirectory()
workspace = Path(temp_path.name)
sim.set_sim_path(workspace)
```

Write the input files to the temporary workspace.

```
[4]: sim.write_simulation()

writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing solution package freyberg...
  writing model freyberg...
    writing model name file...
    writing package dis...
```

(continues on next page)

(continued from previous page)

```
writing package ic...
writing package oc...
writing package npf...
writing package sto...
writing package chd-1...
writing package riv-1...
writing package wel-1...
writing package rch-1...
```

Run the groundwater flow simulation.

```
[5]: success, buff = sim.run_simulation(silent=True, report=True)
assert success, "MODFLOW 6 simulation failed"
for line in buff:
    print(line)
```

```
MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023
```

```
MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000
```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

```
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:50:27
```

```
Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam
```

```
Solving: Stress period:      1      Time step:      1
```

```
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:50:28
Elapsed run time: 0.082 Seconds
```

```
Normal termination of simulation.
```

Define particle release locations. In this example we will release 16 particles, with each particle released at the center of a unique grid cell. Cells containing particle release points are clustered into four 2x2 square regions.

```
[6]: pgs = []
      gwf = sim.get_model mdl_name)

      for i in range(1, 5):
          nrow = gwf.modelgrid.nrow
          ncol = gwf.modelgrid.ncol
          m = i * 2 if i < 3 else (nrow - i * 4)
          n = i * 2 if i < 3 else (ncol - i * 4)
          celldata = flopy.modpath.CellDataType(
              drape=0,
              columncelldivisions=1,
              rowcelldivisions=1,
              layercelldivisions=1,
          )
          lrcpd = flopy.modpath.LRCPParticleData(
              subdivisiondata=[celldata],
              lrcregions=[[[0, m, n, 0, m + 1, n + 1]]],
          )
          pg = flopy.modpath.ParticleGroupLRCTemplate(
              particlegroupname=f"PG{i}",
              particledata=lrcpd,
              filename=f"{sim_name}.pg{i}.sloc",
          )
          pgs.append(pg)
```

Retrieve well locations (to define termination zones).

```
[7]: import numpy as np

      wel_locs = [
          (rec[0][1], rec[0][2]) for rec in (gwf.wel.stress_period_data.data[0])
      ]
      print(wel_locs)

      [(8, 15), (10, 12), (19, 13), (25, 9), (28, 5), (33, 11)]
```

Define particle termination zones.

```
[8]: zone_maps = []

      # zone 1 is the default (non-terminating regions)
      def fill_zone_1():
          return np.ones((nrow, ncol), dtype=np.int32)

      za = fill_zone_1()
      for wl in wel_locs:
          za[wl] = 2 # wells
      za[:, 14] = 3 # river is in column 14
      zone_maps.append(za)
```

Create a MODPATH 7 simulation forward tracking model in combined (pathline & timeseries) mode.

```
[9]: mp = flopy.modpath.Modpath7(
    modelname=f"{sim_name}_mp",
    flowmodel=gwf,
    exe_name="mp7",
    model_ws=workspace,
)
mpbas = flopy.modpath.Modpath7Bas(mp, porosity=0.1)
mpsim = flopy.modpath.Modpath7Sim(
    mp,
    simulationtype="combined",
    trackingdirection="forward",
    budgetoutputoption="summary",
    referencetime=[0, 0, 0.0],
    timepointdata=[1, [0]],
    zonedataoption="on",
    zones=zone_maps,
    particlegroups=pgs,
)
```

Write and run the particle tracking model.

```
[10]: mp.write_input()
success, buff = mp.run_model(silent=True, report=True)
assert success, "MODPATH 7 simulation failed"
for line in buff:
    print(line)
```

```
MODPATH Version 7.2.001
Program compiled Jul 05 2023 20:31:49 with IFORT compiler (ver. 20.21.7)

Run particle tracking simulation ...
Processing Time Step      1 Period      1. Time =  1.000000E+01  Steady-state flow

Particle Summary:
    0 particles are pending release.
    0 particles remain active.
    0 particles terminated at boundary faces.
   16 particles terminated at weak sink cells.
    0 particles terminated at weak source cells.
    0 particles terminated at strong source/sink cells.
    0 particles terminated in cells with a specified zone number.
    0 particles were stranded in inactive or dry cells.
    0 particles were unreleased.
    0 particles have an unknown status.

Normal termination.
```

Open the pathline output file and read pathline data.

```
[11]: from flopy.utils import PathlineFile

pf = PathlineFile(workspace / mpsim.pathlinefilename)
```

(continues on next page)

(continued from previous page)

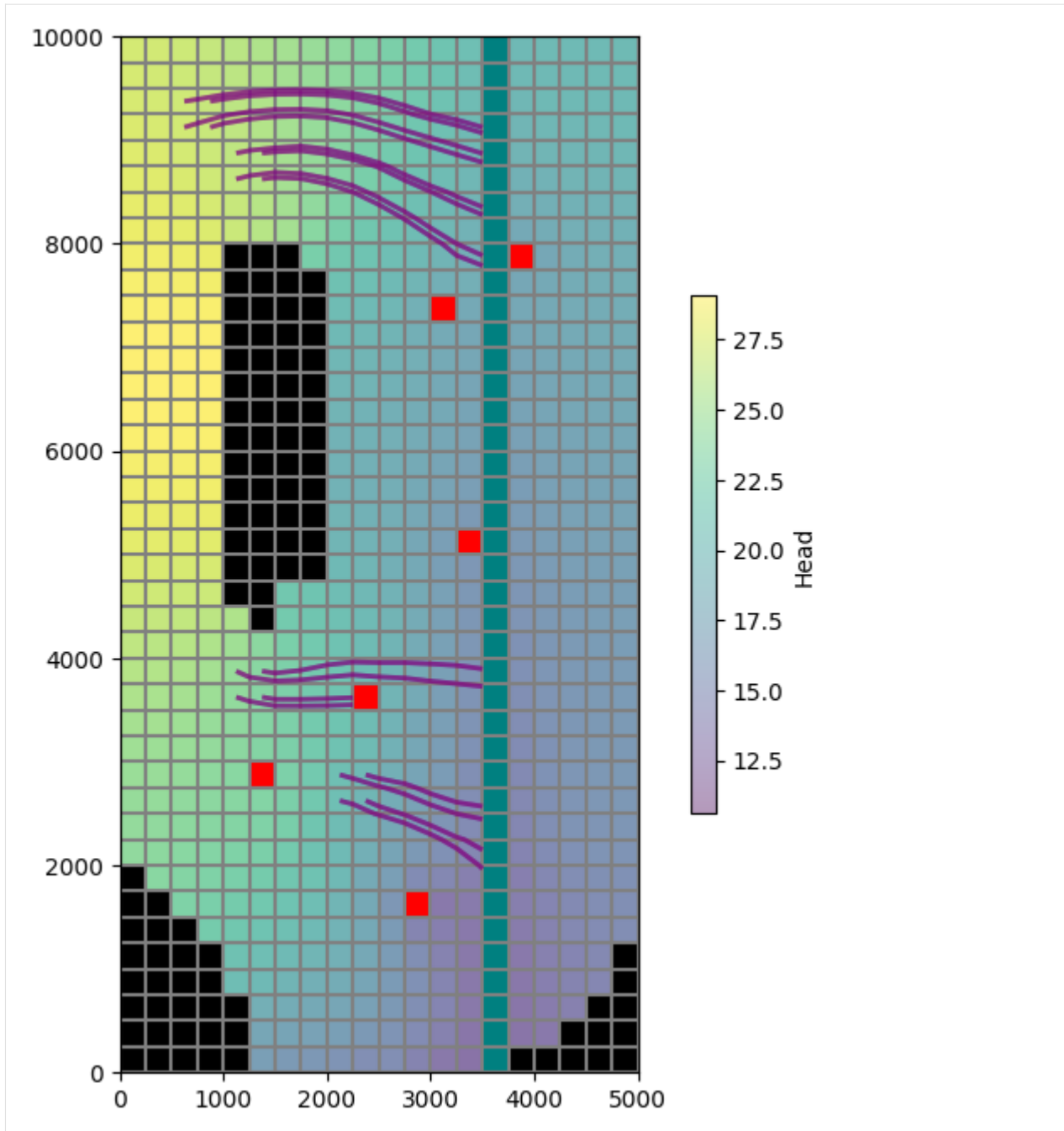
```
pl = pf.get_alldata()
```

Plot the grid, heads, boundary conditions, and pathlines.

```
[12]: import matplotlib.pyplot as plt

hf = flopy.utils.HeadFile(workspace / f"{mdl_name}.hds")
hds = hf.get_data()

fig = plt.figure(figsize=(8, 8))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
mv = flopy.plot.PlotMapView(model=gwf)
mv.plot_grid()
mv.plot_ibound()
hd = mv.plot_array(hds, alpha=0.4)
cb = plt.colorbar(hd, shrink=0.5)
cb.set_label("Head")
mv.plot_bc("RIV")
mv.plot_bc("WEL", plotAll=True)
mv.plot_pathline(pl, layer="all", alpha=0.5, colors=["purple"], lw=2)
plt.show()
```



Create a Vtk object and add the flow model outputs and pathlines.

```
[13]: from flopy.export.vtk import Vtk

vtk = Vtk(model=gwf, binary=False, vertical_exaggeration=50, smooth=False)
vtk.add_model(gwf)
vtk.add_pathline_points(pl)
```

Convert the VTK object to PyVista meshes.

```
[14]: grid, pathlines = vtk.to_pyvista()
```

Rotate the meshes to match the orientation of the map view plot above.

```
[15]: import pyvista as pv

axes = pv.Axes(show_actor=True, actor_scale=2.0, line_width=5)

grid.rotate_z(160, point=axes.origin, inplace=True)
pathlines.rotate_z(160, point=axes.origin, inplace=True)
```

```
[15]: UnstructuredGrid (0x7fd948d0bc40)
      N Cells:      223
      N Points:     207
      X Bounds:    -6.410e+03, -2.297e+03
      Y Bounds:    -8.596e+03, -6.525e+02
      Z Bounds:     3.392e+02, 1.160e+03
      N Arrays:      2
```

Check some grid properties to make sure the export succeeded.

```
[16]: assert grid.n_cells == gwf.modelgrid.nnodes
print("Model grid has", grid.n_cells, "cells")
print("Model grid has", grid.n_arrays, "arrays")
```

```
Model grid has 800 cells
Model grid has 9 arrays
```

Select particle release locations and build a dictionary of particle tracks (pathlines). This will be used below for particle labelling, as well as for animation.

Note: while below we construct pathlines manually from data read from the exported VTK files, pathlines may also be read directly from the MODPATH 7 pathline output file (provided the simulation was run in pathline or combined mode, as this one was).

```
[17]: tracks = {}
particle_ids = set()
release_locs = list()

for i, t in enumerate(pathlines["time"]):
    pid = str(round(float(pathlines["particleid"][i])))
    loc = pathlines.points[i]

    if pid not in tracks:
        tracks[pid] = []
        particle_ids.add(pid)
        release_locs.append(loc)

    # store the particle location in the corresponding track
    tracks[pid].append((loc, t))

release_locs = np.array(release_locs)
tracks = {k: np.array(v, dtype=object) for k, v in tracks.items()}
max_track_len = max([len(v) for v in tracks.values()])
print("The maximum number of locations per particle track is", max_track_len)
```

The maximum number of locations per particle track is 18

View the grid and pathlines with PyVista, with particle tracks/locations colored by time. Also add particle ID labels to a few particles' release locations.

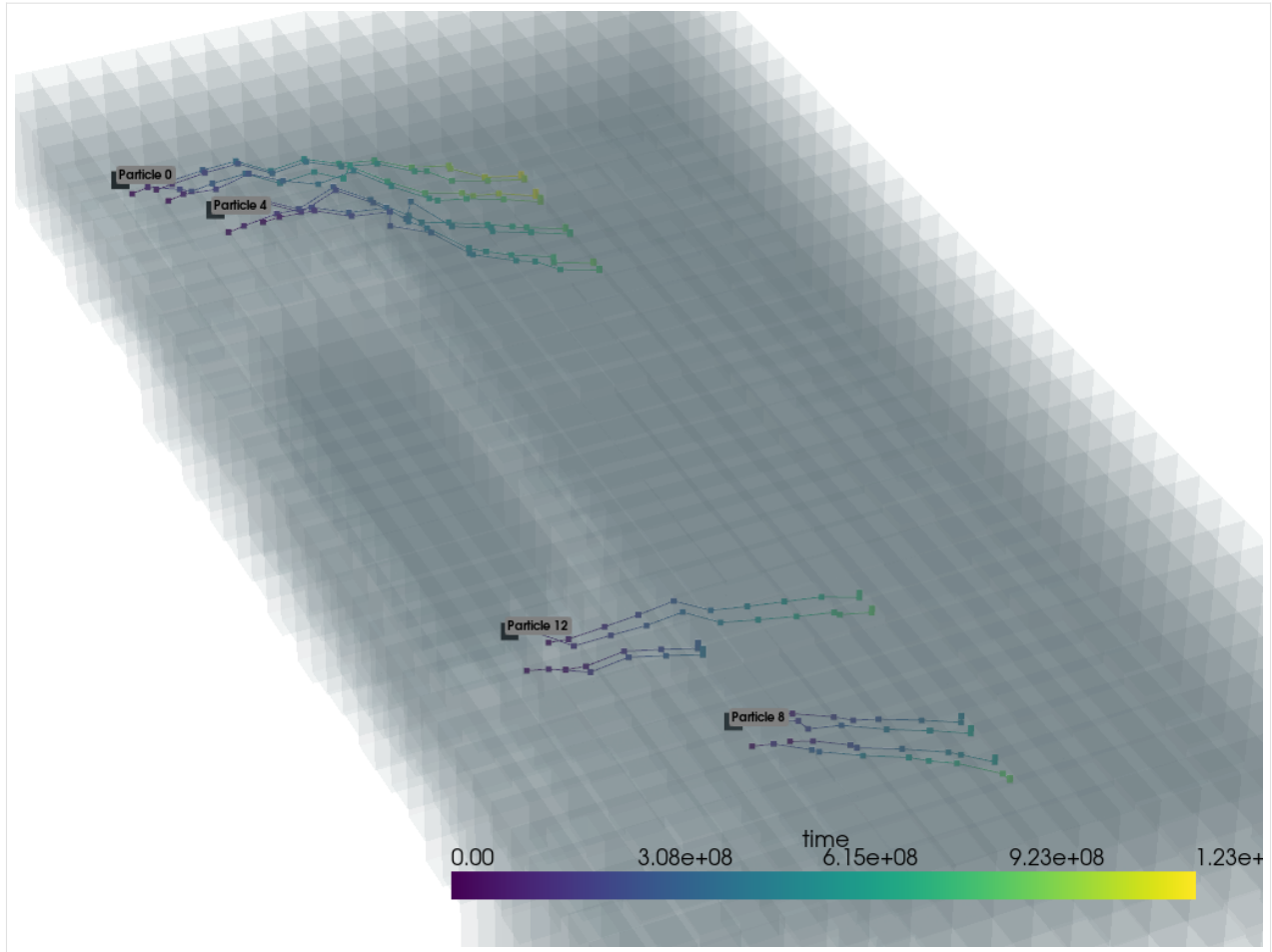
```
[18]: pv.set_plot_theme("document")
pv.set_jupyter_backend("static")

# create the plot and add the grid and pathline meshes
p = pv.Plotter()
p.add_mesh(grid, opacity=0.05)
p.add_mesh(pathlines, scalars="time")

# add a particle ID label to each 4th particle's starting point
label_coords = []
start_labels = []
for pid, track in tracks.items():
    if int(pid) % 4 == 0:
        label_coords.append(track[0][0])
        start_labels.append(f"Particle {pid}")

p.add_point_labels(
    label_coords,
    start_labels,
    font_size=10,
    point_size=15,
    point_color="black",
)

# zoom in and show the plot
p.camera.zoom(2.4)
p.show()
```

Create an animated GIF of the particles traveling along their pathlines, with particles colored by time.

```
[19]: # create plotter
p = pv.Plotter(notebook=False, off_screen=True)

# open GIF file
gif_path = workspace / f"{sim_name}_tracks.gif"
p.open_gif(str(gif_path))

# create mesh from release locations
spls = pv.PolyData(release_locs)
spls.point_data["time"] = np.zeros(len(spls.points))

# add the underlying grid mesh and particle data, then zoom in
p.add_mesh(grid, opacity=0.05)
p.add_mesh(spls, clim=[0, 1.23e09])
p.camera.zoom(2.4)

# cycle through time steps and update particle location
for i in range(1, max_track_len):
    pts = []
    times = []
```

(continues on next page)

(continued from previous page)

```

segments = []

for pid in particle_ids:
    track = tracks[pid]
    npts = len(track)
    # use last locn if particle has already terminated
    loc, t = track[i] if i < npts else track[npts - 1]
    pts.append(loc)
    times.append(t)
    if i < npts:
        segments.append(track[i - 1][0])
        segments.append(loc)

p.update_coordinates(np.vstack(pts), render=False)
p.update_scalars(np.array(times), mesh=spls, render=False)
p.add_lines(np.array(segments), width=1, color="black")
p.write_frame() # write frame to file

# close the plotter and the GIF file
p.close()

```

Show the GIF.

```

[20]: from IPython.core.display import Image

display(Image(data=open(gif_path, "rb").read(), format="gif"))

<IPython.core.display.Image object>

```

Clean up the temporary workspace.

```

[21]: try:
    # ignore PermissionError on Windows
    temp_path.cleanup()
except:
    pass

```

6.9 MT3D and SEAWAT examples

6.9.1 MT3D-USGS Example

Demonstrates functionality of the flopy MT3D-USGS module using the ‘Crank-Nicolson’ example distributed with MT3D-USGS.

Problem description:

- Grid dimensions: 1 Layer, 3 Rows, 650 Columns
- Stress periods: 3
- Units are in seconds and meters
- Flow package: UPW
- Stress packages: SFR, GHB
- Solvers: NWT, GCG

```
[1]: import math
import os
import string
```

```
[2]: import sys
from io import BytesIO, StringIO
from tempfile import TemporaryDirectory

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[3]: temp_dir = TemporaryDirectory()
modelpth = temp_dir.name
modelname = "CrnkNic"
mfexe = "mfnnwt"
mtexe = "mt3dusgs"

# Make sure modelpth directory exists
if not os.path.isdir(modelpth):
    os.makedirs(modelpth, exist_ok=True)

# Instantiate MODFLOW object in flopy
```

(continues on next page)

(continued from previous page)

```
mf = flopy.modflow.Modflow(  
    modelname=modelname, exe_name=mfexe, model_ws=modelpth, version="mfnwt"  
)
```

Set up model discretization

```
[4]: Lx = 650.0  
     Ly = 15  
     nrow = 3  
     ncol = 650  
     nlay = 1  
  
     delr = Lx / ncol  
     delc = Ly / nrow  
  
     xmax = ncol * delr  
     ymax = nrow * delc  
  
     X, Y = np.meshgrid(  
         np.linspace(delr / 2, xmax - delr / 2, ncol),  
         np.linspace(ymax - delc / 2, 0 + delc / 2, nrow),  
     )
```

Instantiate output control (oc) package for MODFLOW-NWT

```
[5]: # Output Control: Create a flopy output control object  
     oc = flopy.modflow.ModflowOc(mf)
```

Instantiate solver package for MODFLOW-NWT

```
[6]: # Newton-Raphson Solver: Create a flopy nwt package object  
  
     headtol = 1.0e-4  
     fluxtol = 5  
     maxiterout = 5000  
     thickfact = 1e-06  
     linmeth = 2  
     iprnwt = 1  
     ibotav = 1  
  
     nwt = flopy.modflow.ModflowNwt(  
         mf,  
         headtol=headtol,  
         fluxtol=fluxtol,  
         maxiterout=maxiterout,  
         thickfact=thickfact,  
         linmeth=linmeth,  
         iprnwt=iprnwt,  
         ibotav=ibotav,  
         options="SIMPLE",  
     )
```

Instantiate discretization (DIS) package for MODFLOW-NWT

```
[7]: # The equations for calculating the ground elevation in the 1 Layer CrnkNic model.
# Although Y isn't used, keeping it here for symetry

def topElev(X, Y):
    return 100.0 - (np.ceil(X) - 1) * 0.03

grndElev = topElev(X, Y)
bedRockElev = grndElev - 3.0

Steady = [False, False, False]
nstp = [1, 1, 1]
tsmult = [1.0, 1.0, 1.0]

# Stress periods extend from (12AM-8:29:59AM); (8:30AM-11:30:59AM); (11:31AM-23:59:59PM)
perlen = [30600, 10800, 45000]

# Create the discretization object
# itmuni = 1 (seconds); lenuni = 2 (meters)
dis = flopy.modflow.ModflowDis(
    mf,
    nlay,
    nrow,
    ncol,
    nper=3,
    delr=delr,
    delc=delc,
    top=grndElev,
    botm=bedRockElev,
    laycbd=0,
    itmuni=1,
    lenuni=2,
    steady=Steady,
    nstp=nstp,
    tsmult=tsmult,
    perlen=perlen,
)
```

Instantiate upstream weighting (UPW) flow package for MODFLOW-NWT

```
[8]: # UPW parameters
# UPW must be instantiated after DIS. Otherwise, during the mf.write_input() procedures,
# flopy will crash.

laytyp = 1
layavg = 2
chani = 1.0
layvka = 1
iphdry = 0
hk = 0.1
hani = 1
```

(continues on next page)

(continued from previous page)

```

vka = 1.0
ss = 0.000001
sy = 0.20
hdry = -888

upw = flopy.modflow.ModflowUpw(
    mf,
    laytyp=laytyp,
    layavg=layavg,
    chani=chani,
    layvka=layvka,
    ipakcb=53,
    hdry=hdry,
    iphdry=iphdry,
    hk=hk,
    hani=hani,
    vka=vka,
    ss=ss,
    sy=sy,
)

```

Instantiate basic (BAS or BA6) package for MODFLOW-NWT

```

[9]: # Create a flopy basic package object
def calc_strtElev(X, Y):
    return 99.5 - (np.ceil(X) - 1) * 0.0001

ibound = np.ones((nlay, nrow, ncol))
ibound[:, 0, :] *= -1
ibound[:, 2, :] *= -1

strtElev = calc_strtElev(X, Y)

bas = flopy.modflow.ModflowBas(mf, ibound=ibound, hnoflo=hdry, strt=strtElev)

```

Instantiate streamflow routing (SFR2) package for MODFLOW-NWT

```

[10]: # Streamflow Routing Package: Try and set up with minimal options in use
# 9 11 IFACE # Data Set 1: ISTCB1 ISTCB2

nstrm = ncol
nss = 6
const = 1.0
dleak = 0.0001
istcb1 = -10
istcb2 = 11
isfropt = 1

segment_data = None
channel_geometry_data = None
channel_flow_data = None

```

(continues on next page)

(continued from previous page)

```

dataset_5 = None
reachinginput = True

# The next couple of lines set up the reach_data for the 30x100 hypothetical model.
# Will need to adjust the row based on which grid discretization we're doing.
# Ensure that the stream goes down one of the middle rows of the model.

strmBed_Elev = 98.75 - (np.ceil(X[1, :]) - 1) * 0.0001

s1 = "k,i,j,iseq,ireach,rchlen,strttop,slope,strthick,strhc1\n"
iseq = 0
irch = 0

for y in range(ncol):
    if y <= 37:
        if iseq == 0:
            irch = 1
        else:
            irch += 1
        iseq = 1
        strhc1 = 1.0e-10
    elif y <= 104:
        if iseq == 1:
            irch = 1
        else:
            irch += 1
        iseq = 2
        strhc1 = 1.0e-10
    elif y <= 280:
        if iseq == 2:
            irch = 1
        else:
            irch += 1
        iseq = 3
        strhc1 = 2.946219199e-6
    elif y <= 432:
        if iseq == 3:
            irch = 1
        else:
            irch += 1
        iseq = 4
        strhc1 = 1.375079882e-6
    elif y <= 618:
        if iseq == 4:
            irch = 1
        else:
            irch += 1
        iseq = 5
        strhc1 = 1.764700062e-6
    else:
        if iseq == 5:

```

(continues on next page)

(continued from previous page)

```

        irch = 1
    else:
        irch += 1
    iseg = 6
    strhc1 = 1e-10

    # remember that lay, row, col need to be zero-based and are adjusted accordingly by
    ↪ flopy
    #      layer + row      +      col      +      iseg      +      irch      +      ↪
    ↪ rchlen      +      strtot      +      slope      +      strthick      +      ↪
    ↪ strmbed K
    s1 += "0,{}".format(1)
    s1 += ",{}".format(y)
    s1 += ",{}".format(iseg)
    s1 += ",{}".format(irch)
    s1 += ",{}".format(delr)
    s1 += ",{}".format(strmBed_Elev[y])
    s1 += ",{}".format(0.0001)
    s1 += ",{}".format(0.50)
    s1 += ",{}\n".format(strhc1)

    fpth = os.path.join(modelpth, "s1.csv")
    f = open(fpth, "w")
    f.write(s1)
    f.close()

    dtype = [
        ("k", "<i4"),
        ("i", "<i4"),
        ("j", "<i4"),
        ("iseg", "<i4"),
        ("ireach", "<f8"),
        ("rchlen", "<f8"),
        ("strtot", "<f8"),
        ("slope", "<f8"),
        ("strthick", "<f8"),
        ("strhc1", "<f8"),
    ]

    if sys.version_info > (3, 0):
        f = open(fpth, "rb")
    else:
        f = open(fpth, "r")

    reach_data = np.genfromtxt(f, delimiter=",", names=True, dtype=dtype)
    f.close()

    s2 = "nseg,icalc,outseg,iupseg,nstrpts,    flow,runoff,etsw,pptsw,          roughch,
    ↪ roughbk,cdpth,fdpth,awdth,bwdth,width1,width2\n
    1,    1,    2,    0,    0, 0.0125,    0.0, 0.0,    0.0, 0.082078856000, 0.
    ↪ 082078856000,    0.0,    0.0,    0.0,    0.0,    1.5,    1.5\n

```

(continues on next page)

(continued from previous page)

```

    2,    1,    3,    0,    0,    0.0,    0.0, 0.0,    0.0, 0.1438063000000, 0.
↪ 1438063000000, 0.0, 0.0, 0.0, 0.0,    1.5,    1.5\n \
    3,    1,    4,    0,    0,    0.0,    0.0, 0.0,    0.0, 0.104569661821, 0.
↪ 104569661821, 0.0, 0.0, 0.0, 0.0,    1.5,    1.5\n \
    4,    1,    5,    0,    0,    0.0,    0.0, 0.0,    0.0, 0.126990045841, 0.
↪ 126990045841, 0.0, 0.0, 0.0, 0.0,    1.5,    1.5\n \
    5,    1,    6,    0,    0,    0.0,    0.0, 0.0,    0.0, 0.183322283828, 0.
↪ 183322283828, 0.0, 0.0, 0.0, 0.0,    1.5,    1.5\n \
    6,    1,    0,    0,    0,    0.0,    0.0, 0.0,    0.0, 0.183322283828, 0.
↪ 183322283828, 0.0, 0.0, 0.0, 0.0,    1.5,    1.5"

fpth = os.path.join(modelpth, "s2.csv")
f = open(fpth, "w")
f.write(s2)
f.close()

if sys.version_info > (3, 0):
    f = open(fpth, "rb")
else:
    f = open(fpth, "r")

segment_data = np.genfromtxt(f, delimiter=",", names=True)
f.close()

# Be sure to convert segment_data to a dictionary keyed on stress period.
segment_data = np.atleast_1d(segment_data)
segment_data = {0: segment_data, 1: segment_data, 2: segment_data}

# There are 3 stress periods
dataset_5 = {0: [nss, 0, 0], 1: [nss, 0, 0], 2: [nss, 0, 0]}

sfr = flopy.modflow.ModflowSfr2(
    mf,
    nstrm=nstrm,
    nss=nss,
    const=const,
    dleak=dleak,
    isfropt=isfropt,
    istcb2=0,
    reachinput=True,
    reach_data=reach_data,
    dataset_5=dataset_5,
    segment_data=segment_data,
    channel_geometry_data=channel_geometry_data,
)

```

Instantiate gage package for use with MODFLOW-NWT package

```

[11]: gages = [
    [1, 38, 61, 1],
    [2, 67, 62, 1],
    [3, 176, 63, 1],

```

(continues on next page)

(continued from previous page)

```

    [4, 152, 64, 1],
    [5, 186, 65, 1],
    [6, 31, 66, 1],
]
files = [
    "CrnkNic.gage",
    "CrnkNic.gag1",
    "CrnkNic.gag2",
    "CrnkNic.gag3",
    "CrnkNic.gag4",
    "CrnkNic.gag5",
    "CrnkNic.gag6",
]
gage = flopy.modflow.ModflowGage(
    mf, numgage=6, gage_data=gages, filenames=files
)

```

Instantiate linkage with mass transport routing (LMT) package for MODFLOW-NWT (generates linker file)

```

[12]: lmt = flopy.modflow.ModflowLmt(
    mf,
    output_file_name="CrnkNic.ftl",
    output_file_header="extended",
    output_file_format="formatted",
    package_flows=["sfr"],
)

```

Write the MODFLOW input files

```

[13]: mf.write_input()

# run the model
success, buff = mf.run_model(silent=True, report=True)
if success:
    for line in buff:
        print(line)
else:
    raise ValueError("Failed to run.")

```

```

                                MODFLOW-NWT-SWR1
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUNDWATER-FLOW MODEL
                                WITH NEWTON FORMULATION
                                Version 1.3.0 07/01/2022
                                BASED ON MODFLOW-2005 Version 1.12.0 02/03/2017

                                SWR1 Version 1.05.0 03/10/2022

```

Using NAME file: CrnkNic.nam

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:51

```

Solving: Stress period:      1      Time step:      1      Groundwater-Flow Eqn.
Solving: Stress period:      2      Time step:      1      Groundwater-Flow Eqn.

```

(continues on next page)

(continued from previous page)

```
Solving: Stress period:    3    Time step:    1    Groundwater-Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:51
Elapsed run time: 0.060 Seconds

Normal termination of simulation
```

Now draft up MT3D-USGS input files.

```
[14]: # Instantiate MT3D-USGS object in flopy
mt = flopy.mt3d.Mt3dms(
    modflowmodel=mf,
    modelname=modelname,
    model_ws=modelpth,
    version="mt3d-usgs",
    namefile_ext="mtnam",
    exe_name=mtexe,
    ftlfilename="CrnkNic.ftl",
    ftlfree=True,
)
```

Instantiate basic transport (BTN) package for MT3D-USGS

```
[15]: btn = flopy.mt3d.Mt3dBtn(
    mt,
    sconc=3.7,
    ncomp=1,
    prsity=0.2,
    cinact=-1.0,
    thkmin=0.001,
    nprs=-1,
    nprobs=10,
    chkmas=True,
    nprmas=10,
    dt0=180,
    mxstrn=2500,
)
```

Instantiate advection (ADV) package for MT3D-USGS

```
[16]: adv = flopy.mt3d.Mt3dAdv(mt, mixelm=0, percel=1.00, mxpart=5000, nadvfd=1)
```

Instantiate generalized conjugate gradient solver (GCG) package for MT3D-USGS

```
[17]: rct = flopy.mt3d.Mt3dRct(mt, isothm=0, ireact=100, igetsc=0, rc1=0.01)
```

```
[18]: gcg = flopy.mt3d.Mt3dGcg(
    mt, mxiter=10, iter1=50, isolve=3, ncrs=0, accl=1, cclose=1e-6, iprgcg=1
)
```

Instantiate source-sink mixing (SSM) package for MT3D-USGS

```
[19]: # For SSM, need to set the constant head boundary conditions to the ambient concentration
# for all 1,300 constant head boundary cells.
```

(continues on next page)

(continued from previous page)

```

itype = flopy.mt3d.Mt3dSsm.itype_dict()
ssm_data = {}
ssm_data[0] = [(0, 0, 0, 3.7, itype["CHD"])]
ssm_data[0].append((0, 2, 0, 3.7, itype["CHD"]))
for i in [0, 2]:
    for j in range(1, ncol):
        ssm_data[0].append((0, i, j, 3.7, itype["CHD"]))

ssm = flopy.mt3d.Mt3dSsm(mt, stress_period_data=ssm_data)

```

Instantiate streamflow transport (SFT) package for MT3D-USGS

```

[20]: dispsf = []
for y in range(ncol):
    if y <= 37:
        dispsf.append(0.12)
    elif y <= 104:
        dispsf.append(0.15)
    elif y <= 280:
        dispsf.append(0.24)
    elif y <= 432:
        dispsf.append(0.31)
    elif y <= 618:
        dispsf.append(0.40)
    else:
        dispsf.append(0.40)

# Enter a list of the observation points
# Each observation is taken as the last reach within the first 5 segments

seg_len = np.unique(reach_data["iseg"], return_counts=True)
obs_sf = np.cumsum(seg_len[1])
obs_sf = obs_sf.tolist()

# The last reach is not an observation point, therefore drop
obs_sf.pop(-1)

# In the first and last stress periods, concentration at the headwater is 3.7
sf_stress_period_data = {0: [0, 0, 3.7], 1: [0, 0, 11.4], 2: [0, 0, 3.7]}

gage_output = [None, None, "CrnkNic.sftobs"]

sft = flopy.mt3d.Mt3dSft(
    mt,
    nsfinit=650,
    mxsfbc=650,
    icbcsf=81,
    ioutobs=82,
    isfsolv=1,
    cclosesf=1.0e-6,
    mxitersf=10,

```

(continues on next page)

(continued from previous page)

```

crntsf=1.0,
iprtxmd=0,
coldsf=3.7,
dispsf=dispsf,
nobssf=5,
obs_sf=obs_sf,
sf_stress_period_data=sf_stress_period_data,
filenames=gage_output,
)

sft.dispsf[0].format.fortran = "(10E15.6)"

```

Write the MT3D-USGS input files and run the model.

```

[21]: mt.write_input()
      mt.run_model(silent=True, report=True)

[21]: (False,
      [
        '
        ' MT3D-USGS - Modular 3D Multi-Species Transport Model [Ver 1.1.0]   ',
        ' and based on MT3DMS. MT3D-USGS developed in cooperation by ',
        ' S.S. Papadopoulos & Associates and the U.S. Geological Survey',
        '
        ' Using NAME File: CrnkNic.mtnam',
        '
        ' STRESS PERIOD NO.      1',
        '
        ' TIME STEP NO.      1',
        ' FROM TIME =  0.0000      TO      30600.      ',
        '
        ' Transport Step:      1  Step Size:  180.0      Total Elapsed Time:  180.00      ',
        ' Outer Iter.  1  Inner Iter.  1:  Max. DC =  0.4865      [K,I,J]      1  2  289',
        ' Outer Iter.  1  Inner Iter.  2:  Max. DC =  0.5960E-07 [K,I,J]      1  1  36',
        ' Outer Iter.  2  Inner Iter.  1:  Max. DC =  0.000      [K,I,J]      1  1  1',
        ' Transport Step:      2  Step Size:  180.0      Total Elapsed Time:  360.00      ',
        ' Outer Iter.  1  Inner Iter.  1:  Max. DC =  0.9385      [K,I,J]      1  2  394',
        ' Outer Iter.  1  Inner Iter.  2:  Max. DC =  0.1788E-06 [K,I,J]      1  2  294',
        ' Outer Iter.  2  Inner Iter.  1:  Max. DC =  0.3934E-05 [K,I,J]      1  2  294',
        ' Outer Iter.  2  Inner Iter.  2:  Max. DC =  0.000      [K,I,J]      1  1  1',
        ' Outer Iter.  3  Inner Iter.  1:  Max. DC =  0.000      [K,I,J]      1  1  1',
        ' Transport Step:      3  Step Size:  180.0      Total Elapsed Time:  540.00      ',
        ' Outer Iter.  1  Inner Iter.  1:  Max. DC =  0.7602      [K,I,J]      1  1  1',
        ' Outer Iter.  1  Inner Iter.  2:  Max. DC =  0.1788E-06 [K,I,J]      1  2  2',
        ' Outer Iter.  2  Inner Iter.  1:  Max. DC =  0.8884E-06 [K,I,J]      1  1  642',
        ' Transport Step:      4  Step Size:  180.0      Total Elapsed Time:  720.00      ',
        ' Outer Iter.  1  Inner Iter.  1:  Max. DC =  0.3366E-03 [K,I,J]      1  2  1',
        ' Outer Iter.  1  Inner Iter.  2:  Max. DC =  0.4657E-09 [K,I,J]      1  2  1',
        ' Outer Iter.  2  Inner Iter.  1:  Max. DC =  0.000      [K,I,J]      1  1  1',
        ' Transport Step:      5  Step Size:  180.0      Total Elapsed Time:  900.00      ',
        ' Outer Iter.  1  Inner Iter.  1:  Max. DC =  0.000      [K,I,J]      1  1  1',
        ' Outer Iter.  2  Inner Iter.  1:  Max. DC =  0.000      [K,I,J]      1  1  1',
        ' Transport Step:      6  Step Size:  180.0      Total Elapsed Time:  1080.0      ',
        ' Outer Iter.  1  Inner Iter.  1:  Max. DC =  0.000      [K,I,J]      1  1  1',
      ]

```

(continues on next page)

(continued from previous page)

```
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 7 Step Size: 180.0 Total Elapsed Time: 1260.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 8 Step Size: 180.0 Total Elapsed Time: 1440.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 9 Step Size: 180.0 Total Elapsed Time: 1620.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 10 Step Size: 180.0 Total Elapsed Time: 1800.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 11 Step Size: 180.0 Total Elapsed Time: 1980.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 12 Step Size: 180.0 Total Elapsed Time: 2160.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 13 Step Size: 180.0 Total Elapsed Time: 2340.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 14 Step Size: 180.0 Total Elapsed Time: 2520.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 15 Step Size: 180.0 Total Elapsed Time: 2700.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 16 Step Size: 180.0 Total Elapsed Time: 2880.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 17 Step Size: 180.0 Total Elapsed Time: 3060.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 18 Step Size: 180.0 Total Elapsed Time: 3240.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 19 Step Size: 180.0 Total Elapsed Time: 3420.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 20 Step Size: 180.0 Total Elapsed Time: 3600.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 21 Step Size: 180.0 Total Elapsed Time: 3780.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 22 Step Size: 180.0 Total Elapsed Time: 3960.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 23 Step Size: 180.0 Total Elapsed Time: 4140.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
```

(continues on next page)

(continued from previous page)

```

' Transport Step: 24 Step Size: 180.0 Total Elapsed Time: 4320.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 25 Step Size: 180.0 Total Elapsed Time: 4500.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 26 Step Size: 180.0 Total Elapsed Time: 4680.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 27 Step Size: 180.0 Total Elapsed Time: 4860.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 28 Step Size: 180.0 Total Elapsed Time: 5040.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 29 Step Size: 180.0 Total Elapsed Time: 5220.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 30 Step Size: 180.0 Total Elapsed Time: 5400.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 31 Step Size: 180.0 Total Elapsed Time: 5580.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 32 Step Size: 180.0 Total Elapsed Time: 5760.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 33 Step Size: 180.0 Total Elapsed Time: 5940.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 34 Step Size: 180.0 Total Elapsed Time: 6120.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 35 Step Size: 180.0 Total Elapsed Time: 6300.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 36 Step Size: 180.0 Total Elapsed Time: 6480.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 37 Step Size: 180.0 Total Elapsed Time: 6660.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 38 Step Size: 180.0 Total Elapsed Time: 6840.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 39 Step Size: 180.0 Total Elapsed Time: 7020.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 40 Step Size: 180.0 Total Elapsed Time: 7200.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 41 Step Size: 180.0 Total Elapsed Time: 7380.0 ',

```

(continues on next page)

(continued from previous page)

```
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 42 Step Size: 180.0 Total Elapsed Time: 7560.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 43 Step Size: 180.0 Total Elapsed Time: 7740.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 44 Step Size: 180.0 Total Elapsed Time: 7920.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 45 Step Size: 180.0 Total Elapsed Time: 8100.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 46 Step Size: 180.0 Total Elapsed Time: 8280.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 47 Step Size: 180.0 Total Elapsed Time: 8460.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 48 Step Size: 180.0 Total Elapsed Time: 8640.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 49 Step Size: 180.0 Total Elapsed Time: 8820.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 50 Step Size: 180.0 Total Elapsed Time: 9000.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 51 Step Size: 180.0 Total Elapsed Time: 9180.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 52 Step Size: 180.0 Total Elapsed Time: 9360.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 53 Step Size: 180.0 Total Elapsed Time: 9540.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 54 Step Size: 180.0 Total Elapsed Time: 9720.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 55 Step Size: 180.0 Total Elapsed Time: 9900.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 56 Step Size: 180.0 Total Elapsed Time: 10080.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 57 Step Size: 180.0 Total Elapsed Time: 10260.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 58 Step Size: 180.0 Total Elapsed Time: 10440.0 ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
```

(continues on next page)

(continued from previous page)

```

' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 59 Step Size: 180.0 Total Elapsed Time: 10620. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 60 Step Size: 180.0 Total Elapsed Time: 10800. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 61 Step Size: 180.0 Total Elapsed Time: 10980. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 62 Step Size: 180.0 Total Elapsed Time: 11160. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 63 Step Size: 180.0 Total Elapsed Time: 11340. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 64 Step Size: 180.0 Total Elapsed Time: 11520. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 65 Step Size: 180.0 Total Elapsed Time: 11700. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 66 Step Size: 180.0 Total Elapsed Time: 11880. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 67 Step Size: 180.0 Total Elapsed Time: 12060. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 68 Step Size: 180.0 Total Elapsed Time: 12240. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 69 Step Size: 180.0 Total Elapsed Time: 12420. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 70 Step Size: 180.0 Total Elapsed Time: 12600. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 71 Step Size: 180.0 Total Elapsed Time: 12780. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 72 Step Size: 180.0 Total Elapsed Time: 12960. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 73 Step Size: 180.0 Total Elapsed Time: 13140. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 74 Step Size: 180.0 Total Elapsed Time: 13320. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 75 Step Size: 180.0 Total Elapsed Time: 13500. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',

```

(continues on next page)

(continued from previous page)

```
' Transport Step: 76 Step Size: 180.0 Total Elapsed Time: 13680. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 77 Step Size: 180.0 Total Elapsed Time: 13860. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 78 Step Size: 180.0 Total Elapsed Time: 14040. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 79 Step Size: 180.0 Total Elapsed Time: 14220. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 80 Step Size: 180.0 Total Elapsed Time: 14400. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 81 Step Size: 180.0 Total Elapsed Time: 14580. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 82 Step Size: 180.0 Total Elapsed Time: 14760. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 83 Step Size: 180.0 Total Elapsed Time: 14940. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 84 Step Size: 180.0 Total Elapsed Time: 15120. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 85 Step Size: 180.0 Total Elapsed Time: 15300. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 86 Step Size: 180.0 Total Elapsed Time: 15480. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 87 Step Size: 180.0 Total Elapsed Time: 15660. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 88 Step Size: 180.0 Total Elapsed Time: 15840. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 89 Step Size: 180.0 Total Elapsed Time: 16020. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 90 Step Size: 180.0 Total Elapsed Time: 16200. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 91 Step Size: 180.0 Total Elapsed Time: 16380. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 92 Step Size: 180.0 Total Elapsed Time: 16560. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 93 Step Size: 180.0 Total Elapsed Time: 16740. ',
```

(continues on next page)

(continued from previous page)

```

' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 94 Step Size: 180.0 Total Elapsed Time: 16920. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 95 Step Size: 180.0 Total Elapsed Time: 17100. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 96 Step Size: 180.0 Total Elapsed Time: 17280. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 97 Step Size: 180.0 Total Elapsed Time: 17460. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 98 Step Size: 180.0 Total Elapsed Time: 17640. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 99 Step Size: 180.0 Total Elapsed Time: 17820. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 100 Step Size: 180.0 Total Elapsed Time: 18000. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 101 Step Size: 180.0 Total Elapsed Time: 18180. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 102 Step Size: 180.0 Total Elapsed Time: 18360. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 103 Step Size: 180.0 Total Elapsed Time: 18540. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 104 Step Size: 180.0 Total Elapsed Time: 18720. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 105 Step Size: 180.0 Total Elapsed Time: 18900. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 106 Step Size: 180.0 Total Elapsed Time: 19080. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 107 Step Size: 180.0 Total Elapsed Time: 19260. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 108 Step Size: 180.0 Total Elapsed Time: 19440. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 109 Step Size: 180.0 Total Elapsed Time: 19620. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 110 Step Size: 180.0 Total Elapsed Time: 19800. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',

```

(continues on next page)

(continued from previous page)

```
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 111 Step Size: 180.0 Total Elapsed Time: 19980. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 112 Step Size: 180.0 Total Elapsed Time: 20160. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 113 Step Size: 180.0 Total Elapsed Time: 20340. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 114 Step Size: 180.0 Total Elapsed Time: 20520. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 115 Step Size: 180.0 Total Elapsed Time: 20700. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 116 Step Size: 180.0 Total Elapsed Time: 20880. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 117 Step Size: 180.0 Total Elapsed Time: 21060. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 118 Step Size: 180.0 Total Elapsed Time: 21240. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 119 Step Size: 180.0 Total Elapsed Time: 21420. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 120 Step Size: 180.0 Total Elapsed Time: 21600. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 121 Step Size: 180.0 Total Elapsed Time: 21780. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 122 Step Size: 180.0 Total Elapsed Time: 21960. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 123 Step Size: 180.0 Total Elapsed Time: 22140. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 124 Step Size: 180.0 Total Elapsed Time: 22320. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 125 Step Size: 180.0 Total Elapsed Time: 22500. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 126 Step Size: 180.0 Total Elapsed Time: 22680. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 127 Step Size: 180.0 Total Elapsed Time: 22860. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
```

(continues on next page)

(continued from previous page)

```
' Transport Step: 128 Step Size: 180.0 Total Elapsed Time: 23040. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 129 Step Size: 180.0 Total Elapsed Time: 23220. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 130 Step Size: 180.0 Total Elapsed Time: 23400. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 131 Step Size: 180.0 Total Elapsed Time: 23580. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 132 Step Size: 180.0 Total Elapsed Time: 23760. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 133 Step Size: 180.0 Total Elapsed Time: 23940. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 134 Step Size: 180.0 Total Elapsed Time: 24120. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 135 Step Size: 180.0 Total Elapsed Time: 24300. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 136 Step Size: 180.0 Total Elapsed Time: 24480. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 137 Step Size: 180.0 Total Elapsed Time: 24660. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 138 Step Size: 180.0 Total Elapsed Time: 24840. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 139 Step Size: 180.0 Total Elapsed Time: 25020. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 140 Step Size: 180.0 Total Elapsed Time: 25200. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 141 Step Size: 180.0 Total Elapsed Time: 25380. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 142 Step Size: 180.0 Total Elapsed Time: 25560. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 143 Step Size: 180.0 Total Elapsed Time: 25740. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 144 Step Size: 180.0 Total Elapsed Time: 25920. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 145 Step Size: 180.0 Total Elapsed Time: 26100. ',
```

(continues on next page)

(continued from previous page)

```
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 146 Step Size: 180.0 Total Elapsed Time: 26280. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 147 Step Size: 180.0 Total Elapsed Time: 26460. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 148 Step Size: 180.0 Total Elapsed Time: 26640. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 149 Step Size: 180.0 Total Elapsed Time: 26820. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 150 Step Size: 180.0 Total Elapsed Time: 27000. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 151 Step Size: 180.0 Total Elapsed Time: 27180. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 152 Step Size: 180.0 Total Elapsed Time: 27360. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 153 Step Size: 180.0 Total Elapsed Time: 27540. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 154 Step Size: 180.0 Total Elapsed Time: 27720. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 155 Step Size: 180.0 Total Elapsed Time: 27900. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 156 Step Size: 180.0 Total Elapsed Time: 28080. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 157 Step Size: 180.0 Total Elapsed Time: 28260. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 158 Step Size: 180.0 Total Elapsed Time: 28440. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 159 Step Size: 180.0 Total Elapsed Time: 28620. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 160 Step Size: 180.0 Total Elapsed Time: 28800. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 161 Step Size: 180.0 Total Elapsed Time: 28980. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 162 Step Size: 180.0 Total Elapsed Time: 29160. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
```

(continues on next page)

(continued from previous page)

```

' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 163 Step Size: 180.0 Total Elapsed Time: 29340. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 164 Step Size: 180.0 Total Elapsed Time: 29520. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 165 Step Size: 180.0 Total Elapsed Time: 29700. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 166 Step Size: 180.0 Total Elapsed Time: 29880. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 167 Step Size: 180.0 Total Elapsed Time: 30060. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 168 Step Size: 180.0 Total Elapsed Time: 30240. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 169 Step Size: 180.0 Total Elapsed Time: 30420. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 170 Step Size: 180.0 Total Elapsed Time: 30600. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
',
' STRESS PERIOD NO. 2',
',
' TIME STEP NO. 1',
' FROM TIME = 30600. TO 41400. ',
',
' Transport Step: 1 Step Size: 180.0 Total Elapsed Time: 30780. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.2062E-04 [K,I,J] 1 1 1',
' Outer Iter. 1 Inner Iter. 2: Max. DC = 0.6548E-10 [K,I,J] 1 2 3',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 2 Step Size: 180.0 Total Elapsed Time: 30960. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 3 Step Size: 180.0 Total Elapsed Time: 31140. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 4 Step Size: 180.0 Total Elapsed Time: 31320. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 5 Step Size: 180.0 Total Elapsed Time: 31500. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 6 Step Size: 180.0 Total Elapsed Time: 31680. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 7 Step Size: 180.0 Total Elapsed Time: 31860. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',

```

(continues on next page)

(continued from previous page)

```
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 8 Step Size: 180.0 Total Elapsed Time: 32040. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 9 Step Size: 180.0 Total Elapsed Time: 32220. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 10 Step Size: 180.0 Total Elapsed Time: 32400. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 11 Step Size: 180.0 Total Elapsed Time: 32580. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 12 Step Size: 180.0 Total Elapsed Time: 32760. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 13 Step Size: 180.0 Total Elapsed Time: 32940. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 14 Step Size: 180.0 Total Elapsed Time: 33120. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 15 Step Size: 180.0 Total Elapsed Time: 33300. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 16 Step Size: 180.0 Total Elapsed Time: 33480. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 17 Step Size: 180.0 Total Elapsed Time: 33660. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 18 Step Size: 180.0 Total Elapsed Time: 33840. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 19 Step Size: 180.0 Total Elapsed Time: 34020. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 20 Step Size: 180.0 Total Elapsed Time: 34200. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 21 Step Size: 180.0 Total Elapsed Time: 34380. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 22 Step Size: 180.0 Total Elapsed Time: 34560. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 23 Step Size: 180.0 Total Elapsed Time: 34740. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 24 Step Size: 180.0 Total Elapsed Time: 34920. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
```

(continues on next page)

(continued from previous page)

```

' Transport Step: 25 Step Size: 180.0 Total Elapsed Time: 35100. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 26 Step Size: 180.0 Total Elapsed Time: 35280. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 27 Step Size: 180.0 Total Elapsed Time: 35460. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 28 Step Size: 180.0 Total Elapsed Time: 35640. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 29 Step Size: 180.0 Total Elapsed Time: 35820. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 30 Step Size: 180.0 Total Elapsed Time: 36000. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 31 Step Size: 180.0 Total Elapsed Time: 36180. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 32 Step Size: 180.0 Total Elapsed Time: 36360. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 33 Step Size: 180.0 Total Elapsed Time: 36540. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 34 Step Size: 180.0 Total Elapsed Time: 36720. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 35 Step Size: 180.0 Total Elapsed Time: 36900. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 36 Step Size: 180.0 Total Elapsed Time: 37080. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 37 Step Size: 180.0 Total Elapsed Time: 37260. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 38 Step Size: 180.0 Total Elapsed Time: 37440. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 39 Step Size: 180.0 Total Elapsed Time: 37620. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 40 Step Size: 180.0 Total Elapsed Time: 37800. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 41 Step Size: 180.0 Total Elapsed Time: 37980. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 42 Step Size: 180.0 Total Elapsed Time: 38160. ',

```

(continues on next page)

(continued from previous page)

```
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 43 Step Size: 180.0 Total Elapsed Time: 38340. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 44 Step Size: 180.0 Total Elapsed Time: 38520. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 45 Step Size: 180.0 Total Elapsed Time: 38700. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 46 Step Size: 180.0 Total Elapsed Time: 38880. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 47 Step Size: 180.0 Total Elapsed Time: 39060. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 48 Step Size: 180.0 Total Elapsed Time: 39240. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 49 Step Size: 180.0 Total Elapsed Time: 39420. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 50 Step Size: 180.0 Total Elapsed Time: 39600. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 51 Step Size: 180.0 Total Elapsed Time: 39780. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 52 Step Size: 180.0 Total Elapsed Time: 39960. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 53 Step Size: 180.0 Total Elapsed Time: 40140. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 54 Step Size: 180.0 Total Elapsed Time: 40320. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 55 Step Size: 180.0 Total Elapsed Time: 40500. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 56 Step Size: 180.0 Total Elapsed Time: 40680. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 57 Step Size: 180.0 Total Elapsed Time: 40860. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 58 Step Size: 180.0 Total Elapsed Time: 41040. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 59 Step Size: 180.0 Total Elapsed Time: 41220. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
```

(continues on next page)

(continued from previous page)

```

' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 60 Step Size: 180.0 Total Elapsed Time: 41400. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
',
' STRESS PERIOD NO. 3',
',
' TIME STEP NO. 1',
' FROM TIME = 41400. TO 86400. ',
',
' Transport Step: 1 Step Size: 180.0 Total Elapsed Time: 41580. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 2 Step Size: 180.0 Total Elapsed Time: 41760. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 3 Step Size: 180.0 Total Elapsed Time: 41940. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 4 Step Size: 180.0 Total Elapsed Time: 42120. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 5 Step Size: 180.0 Total Elapsed Time: 42300. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 6 Step Size: 180.0 Total Elapsed Time: 42480. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 7 Step Size: 180.0 Total Elapsed Time: 42660. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 8 Step Size: 180.0 Total Elapsed Time: 42840. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 9 Step Size: 180.0 Total Elapsed Time: 43020. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 10 Step Size: 180.0 Total Elapsed Time: 43200. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 11 Step Size: 180.0 Total Elapsed Time: 43380. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 12 Step Size: 180.0 Total Elapsed Time: 43560. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 13 Step Size: 180.0 Total Elapsed Time: 43740. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 14 Step Size: 180.0 Total Elapsed Time: 43920. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',

```

(continues on next page)

(continued from previous page)

```
' Transport Step: 15 Step Size: 180.0 Total Elapsed Time: 44100. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 16 Step Size: 180.0 Total Elapsed Time: 44280. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 17 Step Size: 180.0 Total Elapsed Time: 44460. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 18 Step Size: 180.0 Total Elapsed Time: 44640. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 19 Step Size: 180.0 Total Elapsed Time: 44820. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 20 Step Size: 180.0 Total Elapsed Time: 45000. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 21 Step Size: 180.0 Total Elapsed Time: 45180. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 22 Step Size: 180.0 Total Elapsed Time: 45360. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 23 Step Size: 180.0 Total Elapsed Time: 45540. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 24 Step Size: 180.0 Total Elapsed Time: 45720. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 25 Step Size: 180.0 Total Elapsed Time: 45900. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 26 Step Size: 180.0 Total Elapsed Time: 46080. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 27 Step Size: 180.0 Total Elapsed Time: 46260. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 28 Step Size: 180.0 Total Elapsed Time: 46440. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 29 Step Size: 180.0 Total Elapsed Time: 46620. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 30 Step Size: 180.0 Total Elapsed Time: 46800. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 31 Step Size: 180.0 Total Elapsed Time: 46980. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 32 Step Size: 180.0 Total Elapsed Time: 47160. ',
```

(continues on next page)

(continued from previous page)

```

' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 33 Step Size: 180.0 Total Elapsed Time: 47340. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 34 Step Size: 180.0 Total Elapsed Time: 47520. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 35 Step Size: 180.0 Total Elapsed Time: 47700. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 36 Step Size: 180.0 Total Elapsed Time: 47880. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 37 Step Size: 180.0 Total Elapsed Time: 48060. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 38 Step Size: 180.0 Total Elapsed Time: 48240. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 39 Step Size: 180.0 Total Elapsed Time: 48420. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 40 Step Size: 180.0 Total Elapsed Time: 48600. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 41 Step Size: 180.0 Total Elapsed Time: 48780. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 42 Step Size: 180.0 Total Elapsed Time: 48960. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 43 Step Size: 180.0 Total Elapsed Time: 49140. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 44 Step Size: 180.0 Total Elapsed Time: 49320. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 45 Step Size: 180.0 Total Elapsed Time: 49500. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 46 Step Size: 180.0 Total Elapsed Time: 49680. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 47 Step Size: 180.0 Total Elapsed Time: 49860. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 48 Step Size: 180.0 Total Elapsed Time: 50040. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 49 Step Size: 180.0 Total Elapsed Time: 50220. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',

```

(continues on next page)

(continued from previous page)

```
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 50 Step Size: 180.0 Total Elapsed Time: 50400. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 51 Step Size: 180.0 Total Elapsed Time: 50580. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 52 Step Size: 180.0 Total Elapsed Time: 50760. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 53 Step Size: 180.0 Total Elapsed Time: 50940. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 54 Step Size: 180.0 Total Elapsed Time: 51120. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 55 Step Size: 180.0 Total Elapsed Time: 51300. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 56 Step Size: 180.0 Total Elapsed Time: 51480. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 57 Step Size: 180.0 Total Elapsed Time: 51660. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 58 Step Size: 180.0 Total Elapsed Time: 51840. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 59 Step Size: 180.0 Total Elapsed Time: 52020. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 60 Step Size: 180.0 Total Elapsed Time: 52200. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 61 Step Size: 180.0 Total Elapsed Time: 52380. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 62 Step Size: 180.0 Total Elapsed Time: 52560. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 63 Step Size: 180.0 Total Elapsed Time: 52740. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 64 Step Size: 180.0 Total Elapsed Time: 52920. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 65 Step Size: 180.0 Total Elapsed Time: 53100. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 66 Step Size: 180.0 Total Elapsed Time: 53280. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
```

(continues on next page)

(continued from previous page)

```

' Transport Step: 67 Step Size: 180.0 Total Elapsed Time: 53460. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 68 Step Size: 180.0 Total Elapsed Time: 53640. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 69 Step Size: 180.0 Total Elapsed Time: 53820. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 70 Step Size: 180.0 Total Elapsed Time: 54000. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 71 Step Size: 180.0 Total Elapsed Time: 54180. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 72 Step Size: 180.0 Total Elapsed Time: 54360. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 73 Step Size: 180.0 Total Elapsed Time: 54540. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 74 Step Size: 180.0 Total Elapsed Time: 54720. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 75 Step Size: 180.0 Total Elapsed Time: 54900. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 76 Step Size: 180.0 Total Elapsed Time: 55080. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 77 Step Size: 180.0 Total Elapsed Time: 55260. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 78 Step Size: 180.0 Total Elapsed Time: 55440. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 79 Step Size: 180.0 Total Elapsed Time: 55620. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 80 Step Size: 180.0 Total Elapsed Time: 55800. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 81 Step Size: 180.0 Total Elapsed Time: 55980. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 82 Step Size: 180.0 Total Elapsed Time: 56160. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 83 Step Size: 180.0 Total Elapsed Time: 56340. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 84 Step Size: 180.0 Total Elapsed Time: 56520. ',

```

(continues on next page)

(continued from previous page)

```
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 85 Step Size: 180.0 Total Elapsed Time: 56700. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 86 Step Size: 180.0 Total Elapsed Time: 56880. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 87 Step Size: 180.0 Total Elapsed Time: 57060. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 88 Step Size: 180.0 Total Elapsed Time: 57240. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 89 Step Size: 180.0 Total Elapsed Time: 57420. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 90 Step Size: 180.0 Total Elapsed Time: 57600. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 91 Step Size: 180.0 Total Elapsed Time: 57780. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 92 Step Size: 180.0 Total Elapsed Time: 57960. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Transport Step: 93 Step Size: 180.0 Total Elapsed Time: 58140. ',
' Outer Iter. 1 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
' Outer Iter. 2 Inner Iter. 1: Max. DC = 0.000 [K,I,J] 1 1 1',
...])
```

Compare mt3d-usgs results to an analytical solution

```
[22]: # Define a function to read SFT output file
def load_ts_from_SFT_output(fname, nd=1):
    f = open(fname, "r")
    iline = 0
    lst = []
    for line in f:
        if line.strip().split()[0].replace(".", "", 1).isdigit():
            l = line.strip().split()
            t = float(l[0])
            loc = int(l[1])
            conc = float(l[2])
            if loc == nd:
                lst.append([t, conc])

    ts = np.array(lst)
    f.close()
    return ts
```

(continues on next page)

(continued from previous page)

```

# Also define a function to read OTIS output file
def load_ts_from_otis(fname, iobs=1):
    f = open(fname, "r")
    iline = 0
    lst = []
    for line in f:
        l = line.strip().split()
        t = float(l[0])
        val = float(l[iobs])
        lst.append([t, val])

    ts = np.array(lst)
    f.close()
    return ts

```

Load output from SFT as well as from the OTIS solution

```

[23]: # Model output
fname_SFTout = os.path.join(modelpth, "CrnkNic.sftcobs.out")

# Loading MT3D-USGS output
ts1_mt3d = load_ts_from_SFT_output(fname_SFTout, nd=38)
ts2_mt3d = load_ts_from_SFT_output(fname_SFTout, nd=105)
ts3_mt3d = load_ts_from_SFT_output(fname_SFTout, nd=281)
ts4_mt3d = load_ts_from_SFT_output(fname_SFTout, nd=433)
ts5_mt3d = load_ts_from_SFT_output(fname_SFTout, nd=619)

# OTIS results located here
fname_OTIS = "../examples/data/mt3d_test/mfnwt_mt3dusgs/sft_crnkNic/OTIS_solution.out"

# Loading OTIS output
ts1_Otis = load_ts_from_otis(fname_OTIS, 1)
ts2_Otis = load_ts_from_otis(fname_OTIS, 2)
ts3_Otis = load_ts_from_otis(fname_OTIS, 3)
ts4_Otis = load_ts_from_otis(fname_OTIS, 4)
ts5_Otis = load_ts_from_otis(fname_OTIS, 5)

```

Set up some plotting functions

```

[24]: def set_plot_params():
    import matplotlib as mpl
    from matplotlib.font_manager import FontProperties

    mpl.rcParams["font.sans-serif"] = "Arial"
    mpl.rcParams["font.serif"] = "Times"
    mpl.rcParams["font.cursive"] = "Zapf Chancery"
    mpl.rcParams["font.fantasy"] = "Comic Sans MS"
    mpl.rcParams["font.monospace"] = "Courier New"
    mpl.rcParams["pdf.compression"] = 0
    mpl.rcParams["pdf.fonttype"] = 42

```

(continues on next page)

(continued from previous page)

```

    ticksize = 10
    mpl.rcParams["legend.fontsize"] = 7
    mpl.rcParams["axes.labelsize"] = 12
    mpl.rcParams["xtick.labelsize"] = ticksize
    mpl.rcParams["ytick.labelsize"] = ticksize
    return

def set_sizeaxis(a, fmt, sz):
    success = 0
    x = a.get_xticks()
    # print x
    xc = np.chararray(len(x), itemsize=16)
    for i in range(0, len(x)):
        text = fmt % (x[i])
        xc[i] = string.strip(string.ljust(text, 16))
    # print xc
    a.set_xticklabels(xc, size=sz)
    success = 1
    return success

def set_sizeyaxis(a, fmt, sz):
    success = 0
    y = a.get_yticks()
    # print y
    yc = np.chararray(len(y), itemsize=16)
    for i in range(0, len(y)):
        text = fmt % (y[i])
        yc[i] = string.strip(string.ljust(text, 16))
    # print yc
    a.set_yticklabels(yc, size=sz)
    success = 1
    return success

```

Compare output:

```

[25]: # set up figure
try:
    plt.close("all")
except:
    pass

set_plot_params()

fig = plt.figure(figsize=(6, 4), facecolor="w")
ax = fig.add_subplot(1, 1, 1)

ax.plot(ts1_Otis[:, 0], ts1_Otis[:, 1], "k-", linewidth=1.0)
ax.plot(ts2_Otis[:, 0], ts2_Otis[:, 1], "b-", linewidth=1.0)
ax.plot(ts3_Otis[:, 0], ts3_Otis[:, 1], "r-", linewidth=1.0)

```

(continues on next page)

(continued from previous page)

```

ax.plot(ts4_Otis[:, 0], ts4_Otis[:, 1], "g-", linewidth=1.0)
ax.plot(ts5_Otis[:, 0], ts5_Otis[:, 1], "c-", linewidth=1.0)

ax.plot(
    (ts1_mt3d[:, 0]) / 3600,
    ts1_mt3d[:, 1],
    "kD",
    markersize=2.0,
    mfc="none",
    mec="k",
)
ax.plot(
    (ts2_mt3d[:, 0]) / 3600,
    ts2_mt3d[:, 1],
    "b*",
    markersize=3.0,
    mfc="none",
    mec="b",
)
ax.plot((ts3_mt3d[:, 0]) / 3600, ts3_mt3d[:, 1], "r+", markersize=3.0)
ax.plot(
    (ts4_mt3d[:, 0]) / 3600,
    ts4_mt3d[:, 1],
    "g^",
    markersize=2.0,
    mfc="none",
    mec="g",
)
ax.plot(
    (ts5_mt3d[:, 0]) / 3600,
    ts5_mt3d[:, 1],
    "co",
    markersize=2.0,
    mfc="none",
    mec="c",
)

# customize plot
ax.set_xlabel("Time, hours")
ax.set_ylabel("Concentration, mg L-1")
ax.set_ylim([3.5, 13])
ticksize = 10

# legend
leg = ax.legend(
    (
        "Otis, Site 1",
        "Otis, Site 2",
        "Otis, Site 3",
        "Otis, Site 4",
        "Otis, Site 5",
        "MT3D-USGS, Site 1",
    )

```

(continues on next page)

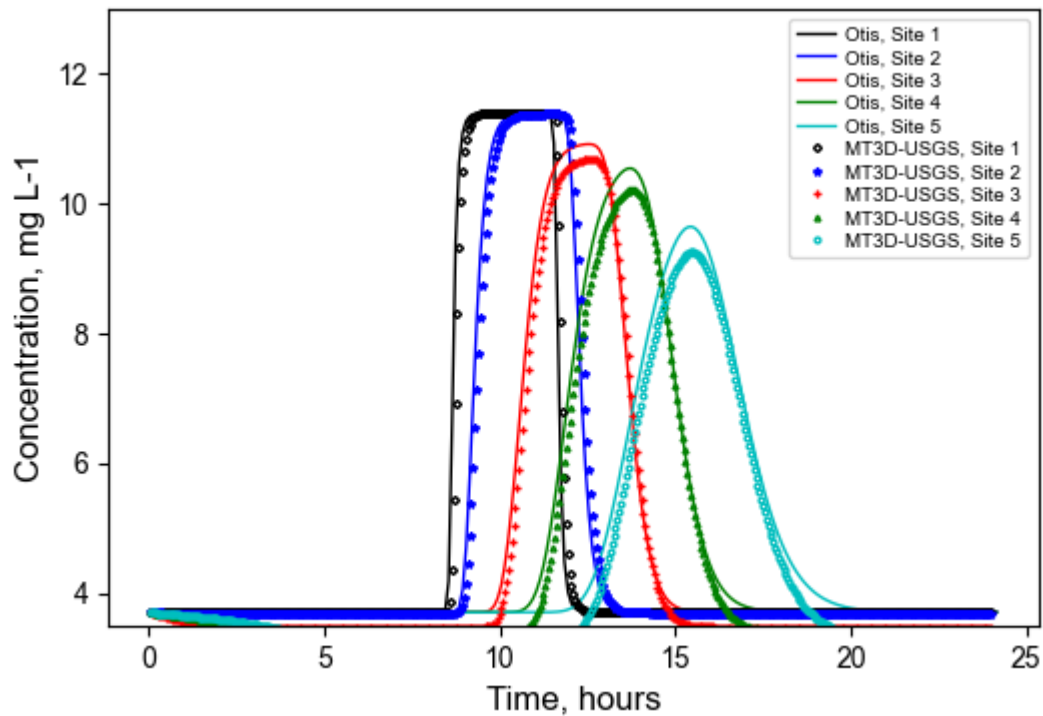
(continued from previous page)

```

    "MT3D-USGS, Site 2",
    "MT3D-USGS, Site 3",
    "MT3D-USGS, Site 4",
    "MT3D-USGS, Site 5",
),
loc="upper right",
labelspacing=0.25,
columnspacing=1,
handletextpad=0.5,
handlelength=2.0,
numpoints=1,
)
leg._drawFrame = False

plt.show()

```



```

[26]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass

```

6.9.2 MT3DMS Example Problems

The purpose of this notebook is to recreate the example problems that are described in the 1999 MT3DMS report.

There are 10 example problems: 1. One-Dimensional Transport in a Uniform Flow Field 2. One-Dimensional Transport with Nonlinear or Nonequilibrium Sorption 3. Two-Dimensional Transport in a Uniform Flow Field 4. Two-Dimensional Transport in a Diagonal Flow Field 5. Two-Dimensional Transport in a Radial Flow Field 6. Concentration at an Injection/Extraction Well 7. Three-Dimensional Transport in a Uniform Flow Field 8. Two-Dimensional, Vertical Transport in a Heterogeneous Aquifer 9. Two-Dimensional Application Example 10. Three-Dimensional Field Case Study

```
[1]: import os

[2]: import sys
    from tempfile import TemporaryDirectory

    import matplotlib as mpl
    import matplotlib.pyplot as plt
    import numpy as np

    # run installed version of flopy or add local path
    try:
        import flopy
    except:
        fpth = os.path.abspath(os.path.join("..", ".."))
        sys.path.append(fpth)
        import flopy

    from flopy.utils.util_array import read1d

    mpl.rcParams["figure.figsize"] = (8, 8)

    exe_name_mf = "mf2005"
    exe_name_mt = "mt3dms"
    datadir = os.path.join("..", "..", "examples", "data", "mt3d_test", "mt3dms")

    # temporary directory
    temp_dir = TemporaryDirectory()
    workdir = temp_dir.name

    print(sys.version)
    print("numpy version: {}".format(np.__version__))
    print("matplotlib version: {}".format(mpl.__version__))
    print("flopy version: {}".format(flopy.__version__))

    3.8.18 (default, Aug 28 2023, 08:27:22)
    [GCC 11.4.0]
    numpy version: 1.24.4
    matplotlib version: 3.7.3
    flopy version: 3.4.3
```

Example 1. One-Dimensional Transport in a Uniform Flow Field

This example has 4 cases: * Case 1a: Advection only * Case 1b: Advection and dispersion * Case 1c: Advection, dispersion, and sorption * Case 1d: Advection, dispersion, sorption, and decay

```
[3]: def p01(dirname, al, retardation, lambda1, mixelm):
    model_ws = os.path.join(workdir, dirname)
    nlay = 1
    nrow = 1
    ncol = 101
    delr = 10
    delc = 1
    delv = 1
    Lx = (ncol - 1) * delr
    v = 0.24
    prsity = 0.25
    q = v * prsity

    perlen = 2000.0
    hk = 1.0
    laytyp = 0
    rhob = 0.25
    kd = (retardation - 1.0) * prsity / rhob

    modelname_mf = dirname + "_mf"
    mf = flopy.modflow.Modflow(
        modelname=modelname_mf, model_ws=model_ws, exe_name=exe_name_mf
    )
    dis = flopy.modflow.ModflowDis(
        mf,
        nlay=nlay,
        nrow=nrow,
        ncol=ncol,
        delr=delr,
        delc=delc,
        top=0.0,
        botm=[0 - delv],
        perlen=perlen,
    )
    ibound = np.ones((nlay, nrow, ncol), dtype=int)
    ibound[0, 0, 0] = -1
    ibound[0, 0, -1] = -1
    strt = np.zeros((nlay, nrow, ncol), dtype=float)
    h1 = q * Lx
    strt[0, 0, 0] = h1
    bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
    lpf = flopy.modflow.ModflowLpf(mf, hk=hk, laytyp=laytyp)
    pcg = flopy.modflow.ModflowPcg(mf)
    lmt = flopy.modflow.ModflowLmt(mf)
    mf.write_input()
    success, buff = mf.run_model(silent=True, report=True)
    if success:
        for line in buff:
```

(continues on next page)

(continued from previous page)

```

        print(line)
    else:
        raise ValueError("Failed to run.")

    modelname_mt = dirname + "_mt"
    mt = flopy.mt3d.Mt3dms(
        modelname=modelname_mt,
        model_ws=model_ws,
        exe_name=exe_name_mt,
        modflowmodel=mf,
    )
    c0 = 1.0
    icbund = np.ones((nlay, nrow, ncol), dtype=int)
    icbund[0, 0, 0] = -1
    sconc = np.zeros((nlay, nrow, ncol), dtype=float)
    sconc[0, 0, 0] = c0
    btn = flopy.mt3d.Mt3dBtn(mt, icbund=icbund, prsity=prsity, sconc=sconc)
    dceps = 1.0e-5
    nplane = 1
    npl = 0
    nph = 4
    npmin = 0
    npmax = 8
    nlsink = nplane
    npsink = nph
    adv = flopy.mt3d.Mt3dAdv(
        mt,
        mixelm=mixelm,
        dceps=dceps,
        nplane=nplane,
        npl=npl,
        nph=nph,
        npmin=npmin,
        npmax=npmax,
        nlsink=nlsink,
        npsink=npsink,
        percel=0.5,
    )
    dsp = flopy.mt3d.Mt3dDsp(mt, al=al)
    rct = flopy.mt3d.Mt3dRct(
        mt,
        isothm=1,
        ireact=1,
        igetsc=0,
        rhob=rhob,
        sp1=kd,
        rc1=lambda1,
        rc2=lambda1,
    )
    ssm = flopy.mt3d.Mt3dSsm(mt)
    gcg = flopy.mt3d.Mt3dGcg(mt)
    mt.write_input()

```

(continues on next page)

(continued from previous page)

```

fname = os.path.join(model_ws, "MT3D001.UCN")
if os.path.isfile(fname):
    os.remove(fname)
mt.run_model(silent=True)

fname = os.path.join(model_ws, "MT3D001.UCN")
ucnobj = flopy.utils.UcnFile(fname)
times = ucnobj.get_times()
conc = ucnobj.get_alldata()

fname = os.path.join(model_ws, "MT3D001.OBS")
if os.path.isfile(fname):
    cvt = mt.load_obs(fname)
else:
    cvt = None

fname = os.path.join(model_ws, "MT3D001.MAS")
mvt = mt.load_mas(fname)

return mf, mt, conc, cvt, mvt

```

```

[4]: mf, mt, conc, cvt, mvt = p01("casela", 0, 1, 0, 1)
x = mf.modelgrid.xcellcenters.ravel()
y = conc[0, 0, 0, :]
plt.plot(x, y, label="Case 1a")

mf, mt, conc, cvt, mvt = p01("case1b", 10, 1, 0, 1)
y = conc[0, 0, 0, :]
plt.plot(x, y, label="Case 1b")

mf, mt, conc, cvt, mvt = p01("case1c", 10, 5, 0, 2)
y = conc[0, 0, 0, :]
plt.plot(x, y, label="Case 1c")

mf, mt, conc, cvt, mvt = p01("case1d", 10, 5, 0.002, 2)
y = conc[0, 0, 0, :]
plt.plot(x, y, label="Case 1d")

plt.xlim(0, 1000)
plt.ylim(0, 1.2)
plt.xlabel("DISTANCE, IN METERS")
plt.ylabel("NORMALIZED CONCENTRATION, UNITLESS")
plt.legend()

```

MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017

Using NAME file: casela_mf.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:58

(continues on next page)

(continued from previous page)

```
Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:58
Elapsed run time: 0.001 Seconds
```

```
Normal termination of simulation
```

```
MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017
```

```
Using NAME file: case1b_mf.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
```

```
Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
Elapsed run time: 0.001 Seconds
```

```
Normal termination of simulation
```

```
MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017
```

```
Using NAME file: case1c_mf.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
```

```
Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
Elapsed run time: 0.001 Seconds
```

```
Normal termination of simulation
```

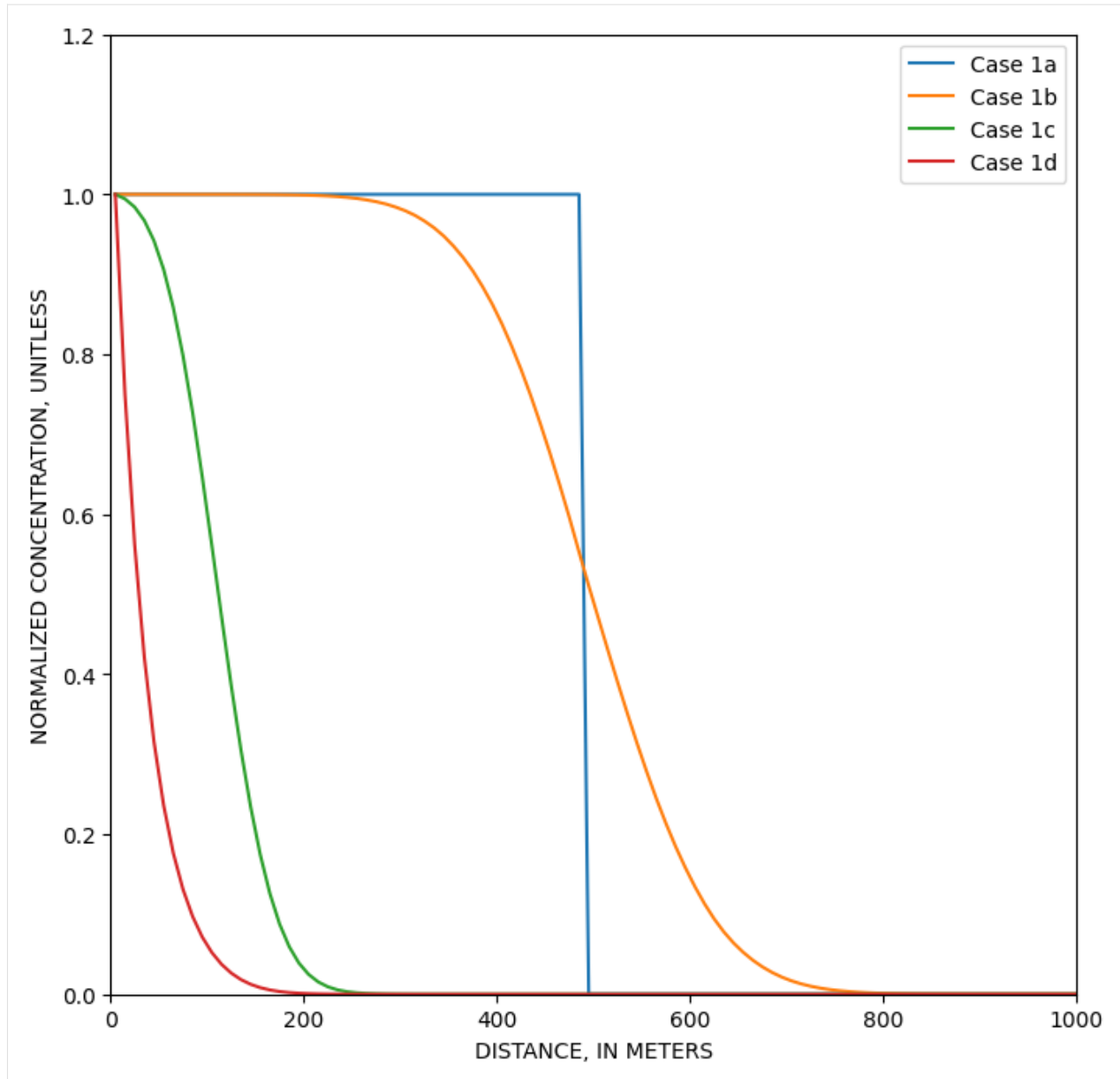
```
MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017
```

```
Using NAME file: case1d_mf.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
```

```
Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
Elapsed run time: 0.001 Seconds
```

```
Normal termination of simulation
```

```
[4]: <matplotlib.legend.Legend at 0x7f05412db8b0>
```



```
[5]: mf, mt, conc, cvt, mvt = p01("case1e", 0, 1, 0, 1)
     y = conc[0, 0, 0, :]
     plt.plot(x, y, label="MOC")

     mf, mt, conc, cvt, mvt = p01("case1f", 0, 1, 0, -1)
     y = conc[0, 0, 0, :]
     plt.plot(x, y, label="ULTIMATE")

     mf, mt, conc, cvt, mvt = p01("case1g", 0, 1, 0, 0)
     y = conc[0, 0, 0, :]
     plt.plot(x, y, label="Upstream FD")

     plt.xlim(0, 1000)
```

(continues on next page)

(continued from previous page)

```
plt.ylim(0, 1.2)
plt.xlabel("DISTANCE, IN METERS")
plt.ylabel("NORMALIZED CONCENTRATION, UNITLESS")
plt.legend()
```

```
MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017
```

```
Using NAME file: case1e_mf.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
```

```
Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
Elapsed run time: 0.002 Seconds
```

```
Normal termination of simulation
```

```
MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017
```

```
Using NAME file: case1f_mf.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
```

```
Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
Elapsed run time: 0.002 Seconds
```

```
Normal termination of simulation
```

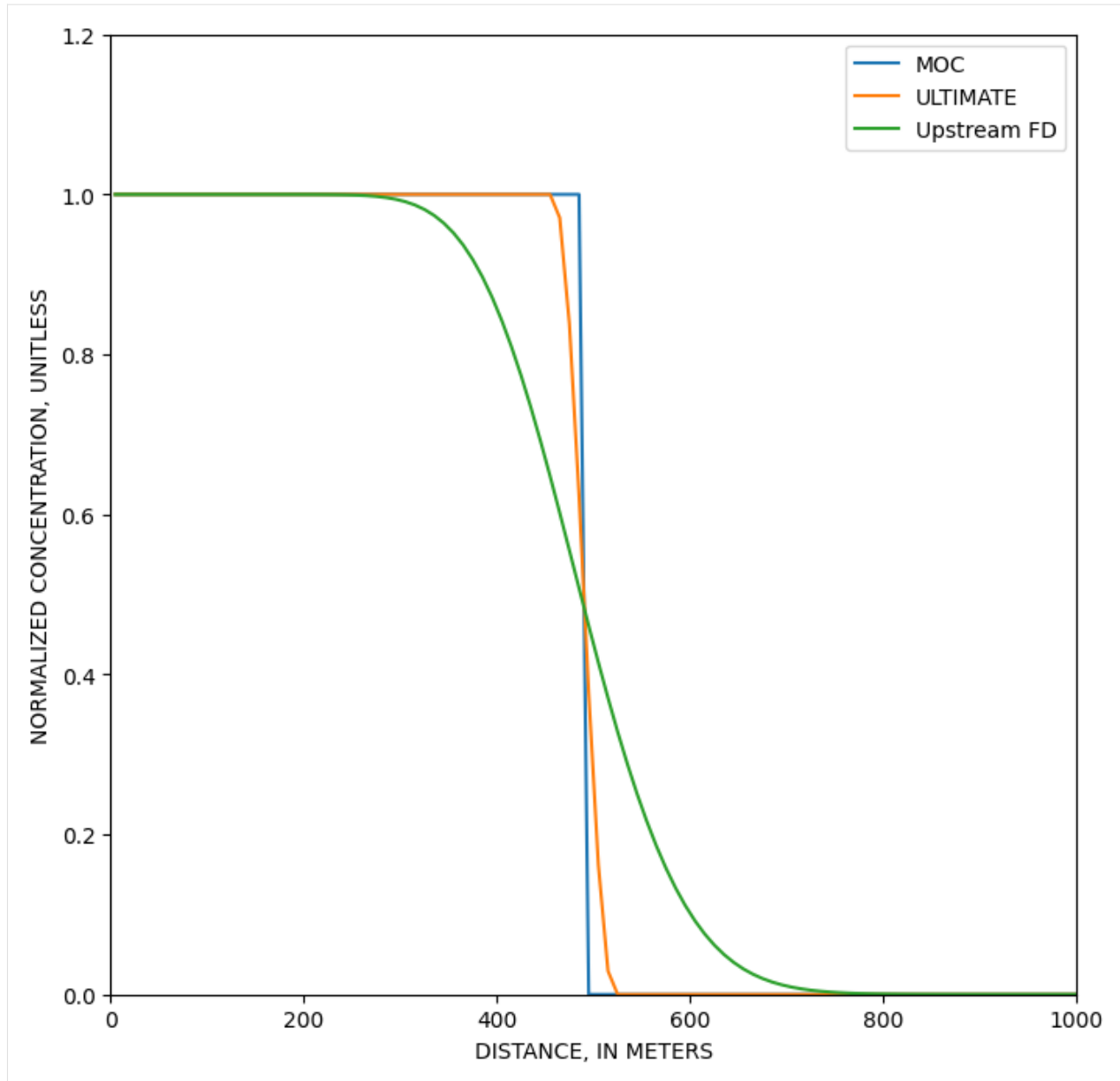
```
MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017
```

```
Using NAME file: case1g_mf.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
```

```
Solving: Stress period:      1      Time step:      1      Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59
Elapsed run time: 0.002 Seconds
```

```
Normal termination of simulation
```

```
[5]: <matplotlib.legend.Legend at 0x7f0541282c10>
```



```
[6]: mf, mt, conc, cvt, mvt = p01("case1e", 10, 1, 0, 1)
     y = conc[0, 0, 0, :]
     plt.plot(x, y, label="MOC")

     mf, mt, conc, cvt, mvt = p01("case1f", 10, 1, 0, -1)
     y = conc[0, 0, 0, :]
     plt.plot(x, y, label="ULTIMATE")

     mf, mt, conc, cvt, mvt = p01("case1g", 10, 1, 0, 0)
     y = conc[0, 0, 0, :]
     plt.plot(x, y, label="Upstream FD")

     plt.xlim(0, 1000)
```

(continues on next page)

(continued from previous page)

```
plt.ylim(0, 1.2)
plt.xlabel("DISTANCE, IN METERS")
plt.ylabel("NORMALIZED CONCENTRATION, UNITLESS")
plt.legend()
```

```

MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017

```

Using NAME file: case1e_mf.nam

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59

Solving: Stress period: 1 Time step: 1 Ground-Water Flow Eqn.

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:45:59

Elapsed run time: 0.002 Seconds

Normal termination of simulation

```

MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017

```

Using NAME file: case1f_mf.nam

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:46:00

Solving: Stress period: 1 Time step: 1 Ground-Water Flow Eqn.

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:46:00

Elapsed run time: 0.002 Seconds

Normal termination of simulation

```

MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017

```

Using NAME file: case1g_mf.nam

Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:46:00

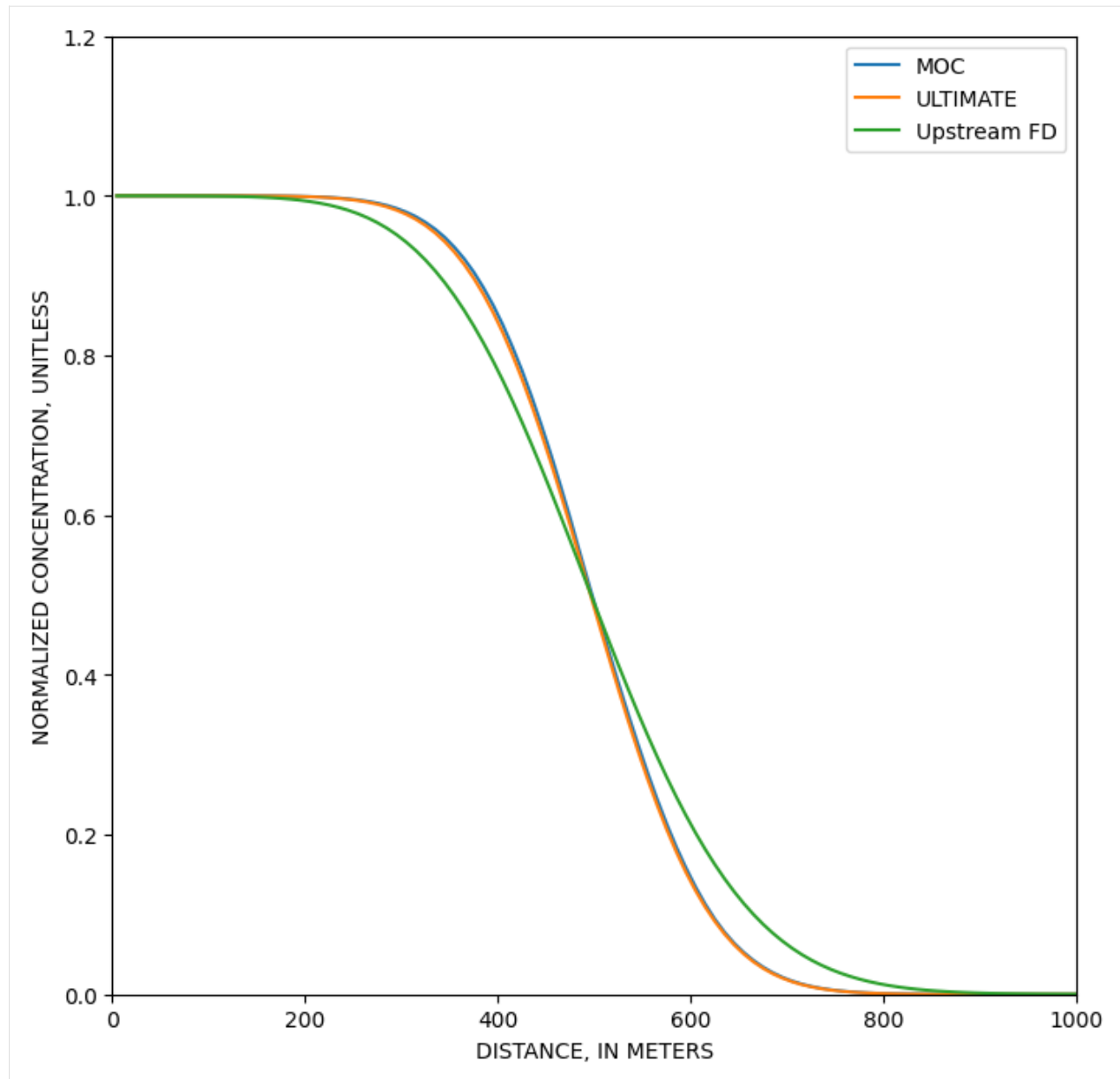
Solving: Stress period: 1 Time step: 1 Ground-Water Flow Eqn.

Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:46:00

Elapsed run time: 0.001 Seconds

Normal termination of simulation

[6]: <matplotlib.legend.Legend at 0x7f0540e04190>



Example 2. One-Dimensional Transport with Nonlinear or Nonequilibrium Sorption

```
[7]: def p02(dirname, isothm, sp1, sp2, mixelm):
    model_ws = os.path.join(workdir, dirname)
    nlay = 1
    nrow = 1
    ncol = 101
    delr = 0.16
    delc = 1
    delv = 1
    Lx = (ncol - 1) * delr
    v = 0.1
```

(continues on next page)

(continued from previous page)

```

prcity = 0.37
q = v * prcity

perlen_mf = 1.0
perlen_mt = [160, 1320]
hk = 1.0
laytyp = 0
rhob = 1.587

modelname_mf = dirname + "_mf"
mf = flopy.modflow.Modflow(
    modelname=modelname_mf, model_ws=model_ws, exe_name=exe_name_mf
)
dis = flopy.modflow.ModflowDis(
    mf,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=delr,
    delc=delc,
    top=0.0,
    botm=[0 - delv],
    perlen=perlen_mf,
)
ibound = np.ones((nlay, nrow, ncol), dtype=int)
ibound[0, 0, 0] = -1
ibound[0, 0, -1] = -1
strt = np.zeros((nlay, nrow, ncol), dtype=float)
h1 = q * Lx
strt[0, 0, 0] = h1
bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
lpf = flopy.modflow.ModflowLpf(mf, hk=hk, laytyp=laytyp)
pcg = flopy.modflow.ModflowPcg(mf)
lmt = flopy.modflow.ModflowLmt(mf)
mf.write_input()
mf.run_model(silent=True)

modelname_mt = dirname + "_mt"
mt = flopy.mt3d.Mt3dms(
    modelname=modelname_mt,
    model_ws=model_ws,
    exe_name=exe_name_mt,
    modflowmodel=mf,
)
btn = flopy.mt3d.Mt3dBtn(
    mt,
    icbund=1,
    prcity=prcity,
    sconc=0,
    nper=2,
    perlen=perlen_mt,
    obs=[(0, 0, 50)],

```

(continues on next page)

(continued from previous page)

```

)
dceps = 1.0e-5
nplane = 1
npl = 0
nph = 4
npmin = 0
npmax = 8
nlsink = nplane
npsink = nph
adv = flopy.mt3d.Mt3dAdv(
    mt,
    mixelm=mixelm,
    dceps=dceps,
    nplane=nplane,
    npl=npl,
    nph=nph,
    npmin=npmin,
    npmax=npmax,
    nlsink=nlsink,
    npsink=npsink,
    percel=0.5,
)
al = 1.0
dsp = flopy.mt3d.Mt3dDsp(mt, al=al)
rct = flopy.mt3d.Mt3dRct(
    mt, isothm=isothm, ireact=0, igetsc=0, rhob=rhob, sp1=sp1, sp2=sp2
)
c0 = 0.05
spd = {0: [0, 0, 0, c0, 1], 1: [0, 0, 0, 0.0, 1]}
ssm = flopy.mt3d.Mt3dSsm(mt, stress_period_data=spd)
gcg = flopy.mt3d.Mt3dGcg(mt)
mt.write_input()
fname = os.path.join(model_ws, "MT3D001.UCN")
if os.path.isfile(fname):
    os.remove(fname)
mt.run_model(silent=True)

fname = os.path.join(model_ws, "MT3D001.UCN")
ucnobj = flopy.utils.UcnFile(fname)
times = ucnobj.get_times()
conc = ucnobj.get_alldata()

fname = os.path.join(model_ws, "MT3D001.OBS")
if os.path.isfile(fname):
    cvt = mt.load_obs(fname)
else:
    cvt = None

fname = os.path.join(model_ws, "MT3D001.MAS")
mvt = mt.load_mas(fname)

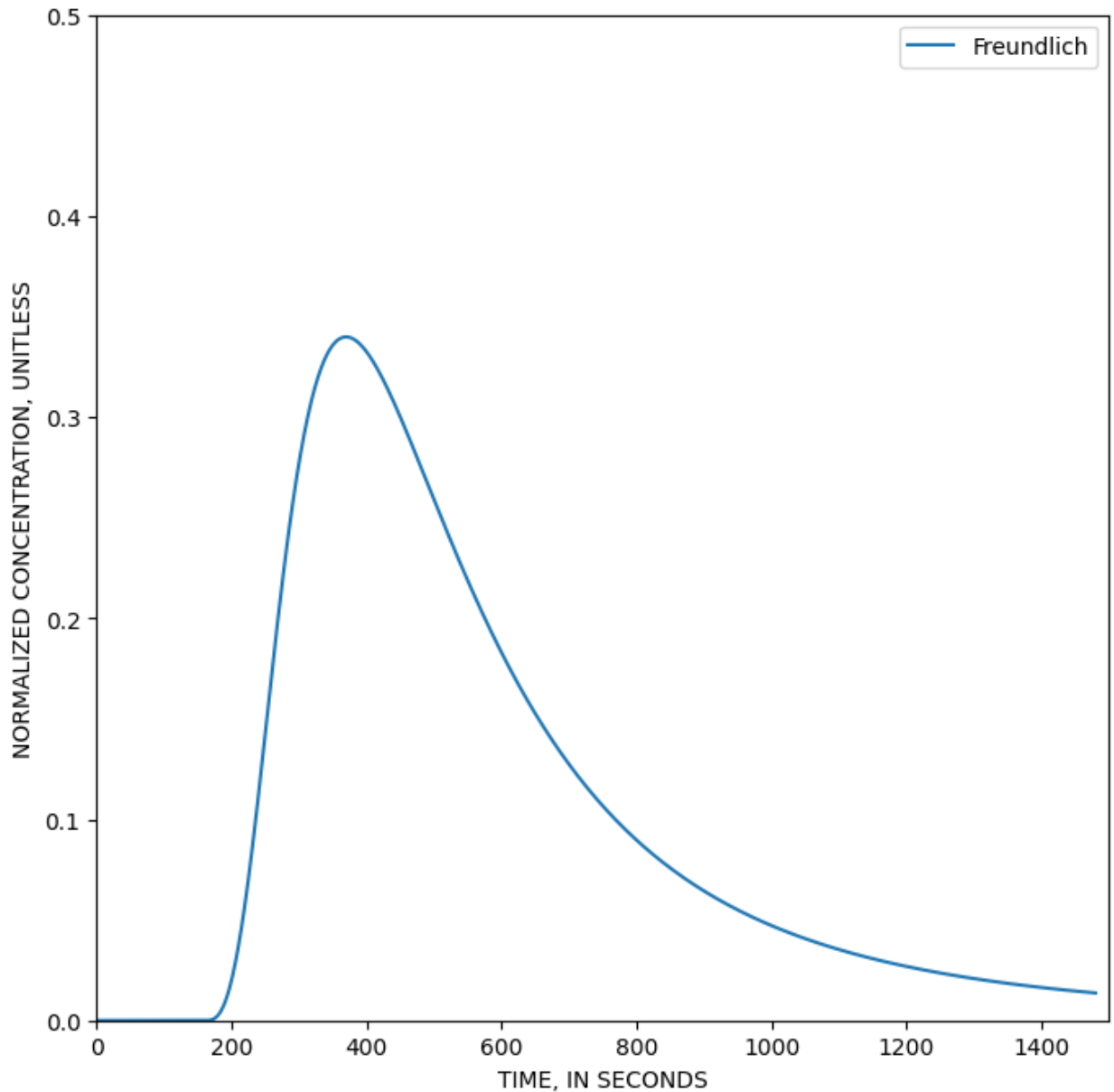
return mf, mt, conc, cvt, mvt

```



```
[8]: mf, mt, conc, cvt, mvt = p02("freundlich", 2, 0.3, 0.7, -1)
x = cvt["time"]
y = cvt["(1, 1, 51)"] / 0.05
plt.plot(x, y, label="Freundlich")
plt.xlim(0, 1500)
plt.ylim(0, 0.5)
plt.xlabel("TIME, IN SECONDS")
plt.ylabel("NORMALIZED CONCENTRATION, UNITLESS")
plt.legend()
```

```
[8]: <matplotlib.legend.Legend at 0x7f0540de6a90>
```



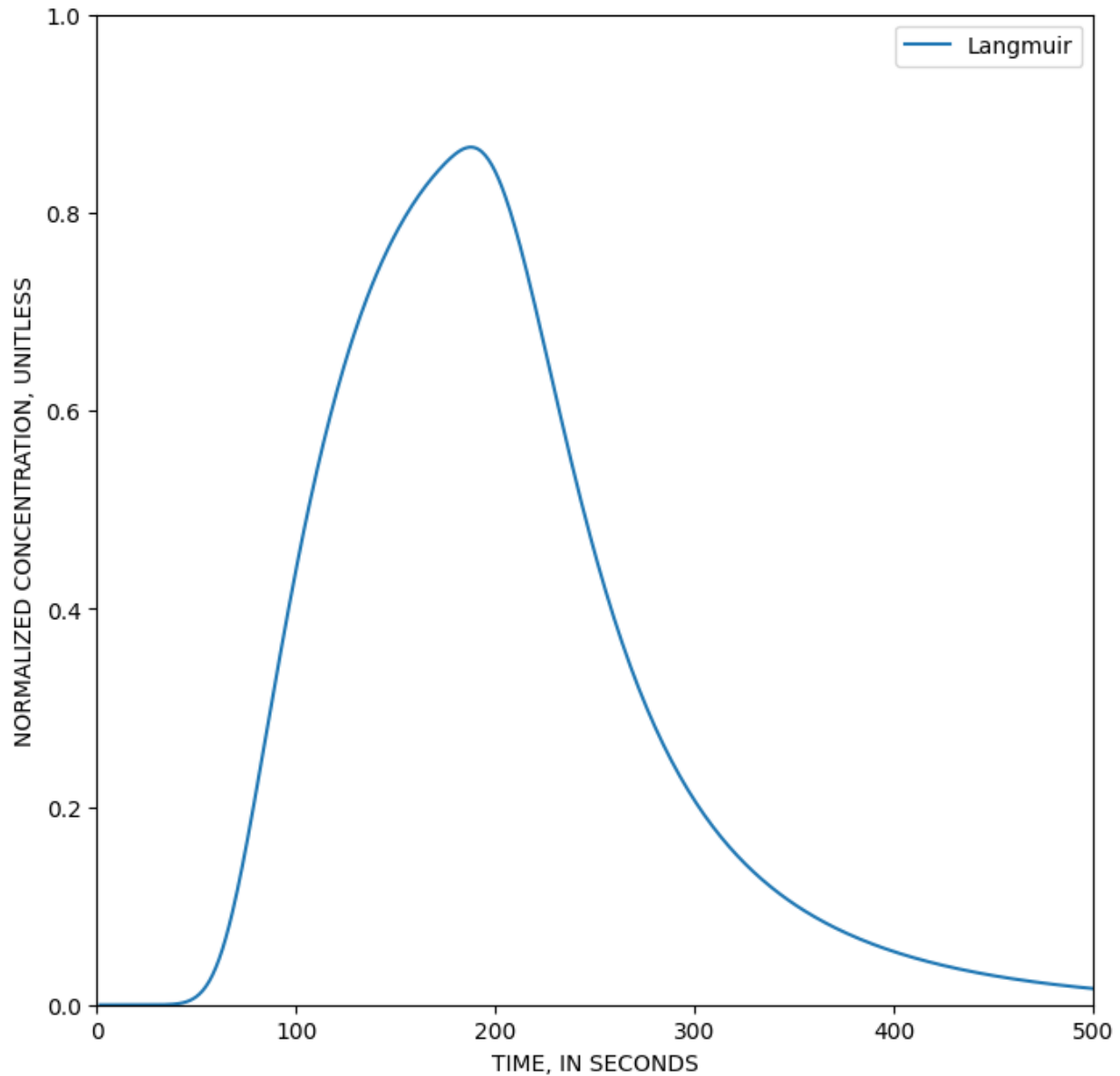
```
[9]: mf, mt, conc, cvt, mvt = p02("langmuir", 3, 100.0, 0.003, -1)
x = cvt["time"]
```

(continues on next page)

(continued from previous page)

```
y = cvt["(1, 1, 51)"] / 0.05
plt.plot(x, y, label="Langmuir")
plt.xlim(0, 500)
plt.ylim(0, 1.0)
plt.xlabel("TIME, IN SECONDS")
plt.ylabel("NORMALIZED CONCENTRATION, UNITLESS")
plt.legend()
```

[9]: <matplotlib.legend.Legend at 0x7f0540dc38b0>



```
[10]: for beta in [0, 2.0e-3, 1.0e-2, 20.0]:
    lbl = "beta={}".format(beta)
    mf, mt, conc, cvt, mvt = p02("nonequilibrium", 4, 0.933, beta, -1)
    x = cvt["time"]
```

(continues on next page)

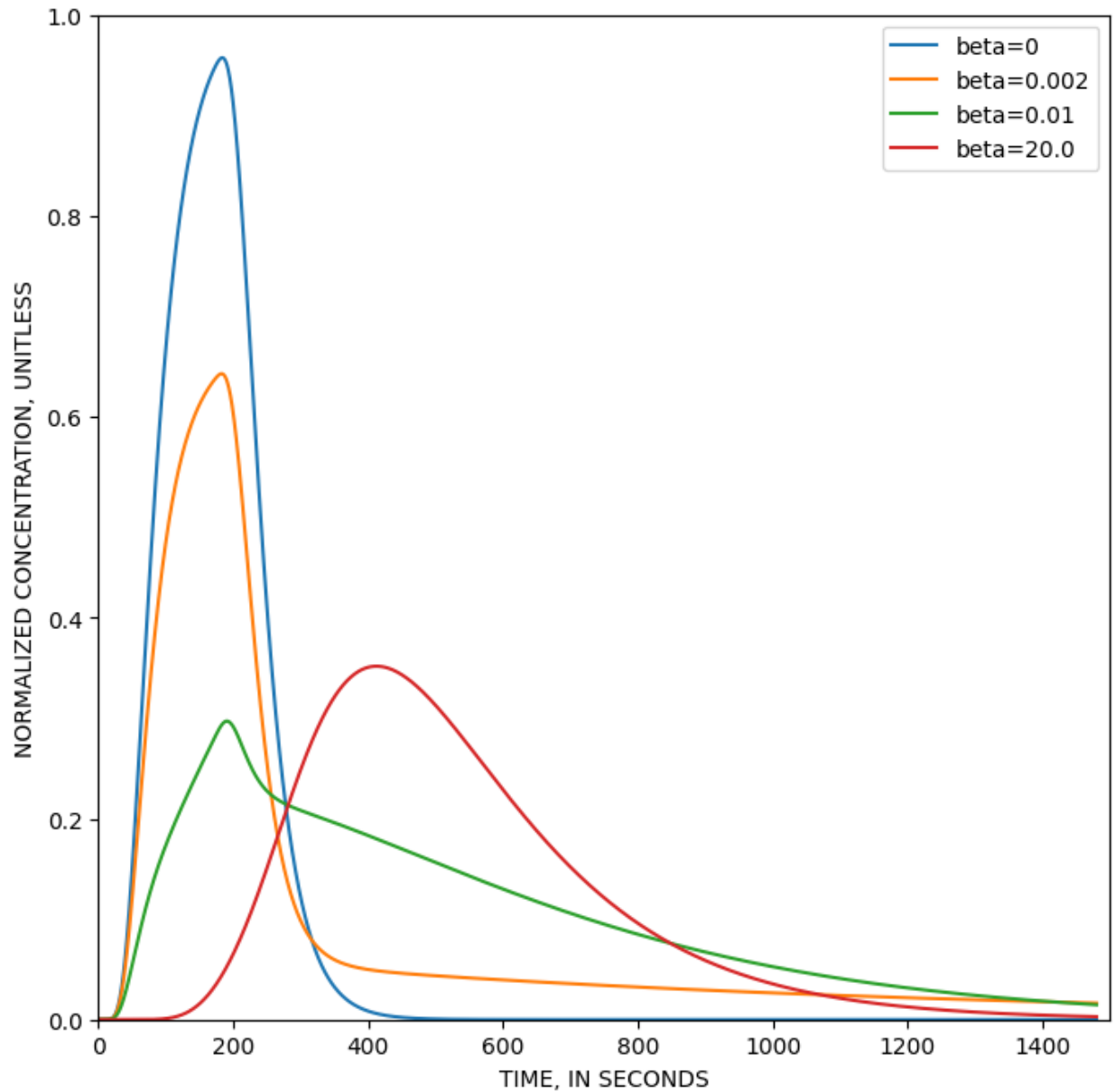
(continued from previous page)

```

y = cvt["(1, 1, 51)"] / 0.05
plt.plot(x, y, label=lbl)
plt.xlim(0, 1500)
plt.ylim(0, 1.0)
plt.xlabel("TIME, IN SECONDS")
plt.ylabel("NORMALIZED CONCENTRATION, UNITLESS")
plt.legend()

```

[10]: <matplotlib.legend.Legend at 0x7f0540db7e80>



Example 3. Two-Dimensional Transport in a Uniform Flow Field

```
[11]: def p03(dirname, mixelm):
    model_ws = os.path.join(workdir, dirname)
    nlay = 1
    nrow = 31
    ncol = 46
    delr = 10
    delc = 10
    delv = 10
    Lx = (ncol - 1) * delr
    v = 1.0 / 3.0
    prsity = 0.3
    q = v * prsity
    al = 10.0
    trpt = 0.3
    q0 = 1.0
    c0 = 1000.0

    perlen_mf = 365.0
    perlen_mt = 365.0
    hk = 1.0
    laytyp = 0

    modelname_mf = dirname + "_mf"
    mf = flopy.modflow.Modflow(
        modelname=modelname_mf, model_ws=model_ws, exe_name=exe_name_mf
    )
    dis = flopy.modflow.ModflowDis(
        mf,
        nlay=nlay,
        nrow=nrow,
        ncol=ncol,
        delr=delr,
        delc=delc,
        top=0.0,
        botm=[0 - delv],
        perlen=perlen_mf,
    )
    ibound = np.ones((nlay, nrow, ncol), dtype=int)
    ibound[0, :, 0] = -1
    ibound[0, :, -1] = -1
    strt = np.zeros((nlay, nrow, ncol), dtype=float)
    h1 = q * Lx
    strt[0, :, 0] = h1
    bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
    lpf = flopy.modflow.ModflowLpf(mf, hk=hk, laytyp=laytyp)
    wel = flopy.modflow.ModflowWel(mf, stress_period_data=[[0, 15, 15, q0]])
    pcg = flopy.modflow.ModflowPcg(mf)
    lmt = flopy.modflow.ModflowLmt(mf)
    mf.write_input()
    mf.run_model(silent=True)
```

(continues on next page)

(continued from previous page)

```

modelname_mt = dirname + "_mt"
mt = flopy.mt3d.Mt3dms(
    modelname=modelname_mt,
    model_ws=model_ws,
    exe_name=exe_name_mt,
    modflowmodel=mf,
)
btn = flopy.mt3d.Mt3dBtn(mt, icbund=1, prsity=prsity, sconc=0)
dceps = 1.0e-5
nplane = 1
npl = 0
nph = 16
npmin = 2
npmax = 32
dchmoc = 1.0e-3
nlsink = nplane
npsink = nph
adv = flopy.mt3d.Mt3dAdv(
    mt,
    mixelm=mixelm,
    dceps=dceps,
    nplane=nplane,
    npl=npl,
    nph=nph,
    npmin=npmin,
    npmax=npmax,
    nlsink=nlsink,
    npsink=npsink,
    percel=0.5,
)
dsp = flopy.mt3d.Mt3dDsp(mt, al=al, trpt=trpt)
spd = {0: [0, 15, 15, c0, 2]}
ssm = flopy.mt3d.Mt3dSsm(mt, stress_period_data=spd)
gcg = flopy.mt3d.Mt3dGcg(mt)
mt.write_input()
fname = os.path.join(model_ws, "MT3D001.UCN")
if os.path.isfile(fname):
    os.remove(fname)
mt.run_model(silent=True)

fname = os.path.join(model_ws, "MT3D001.UCN")
ucnobj = flopy.utils.UcnFile(fname)
times = ucnobj.get_times()
conc = ucnobj.get_alldata()

fname = os.path.join(model_ws, "MT3D001.OBS")
if os.path.isfile(fname):
    cvt = mt.load_obs(fname)
else:
    cvt = None

fname = os.path.join(model_ws, "MT3D001.MAS")

```

(continues on next page)

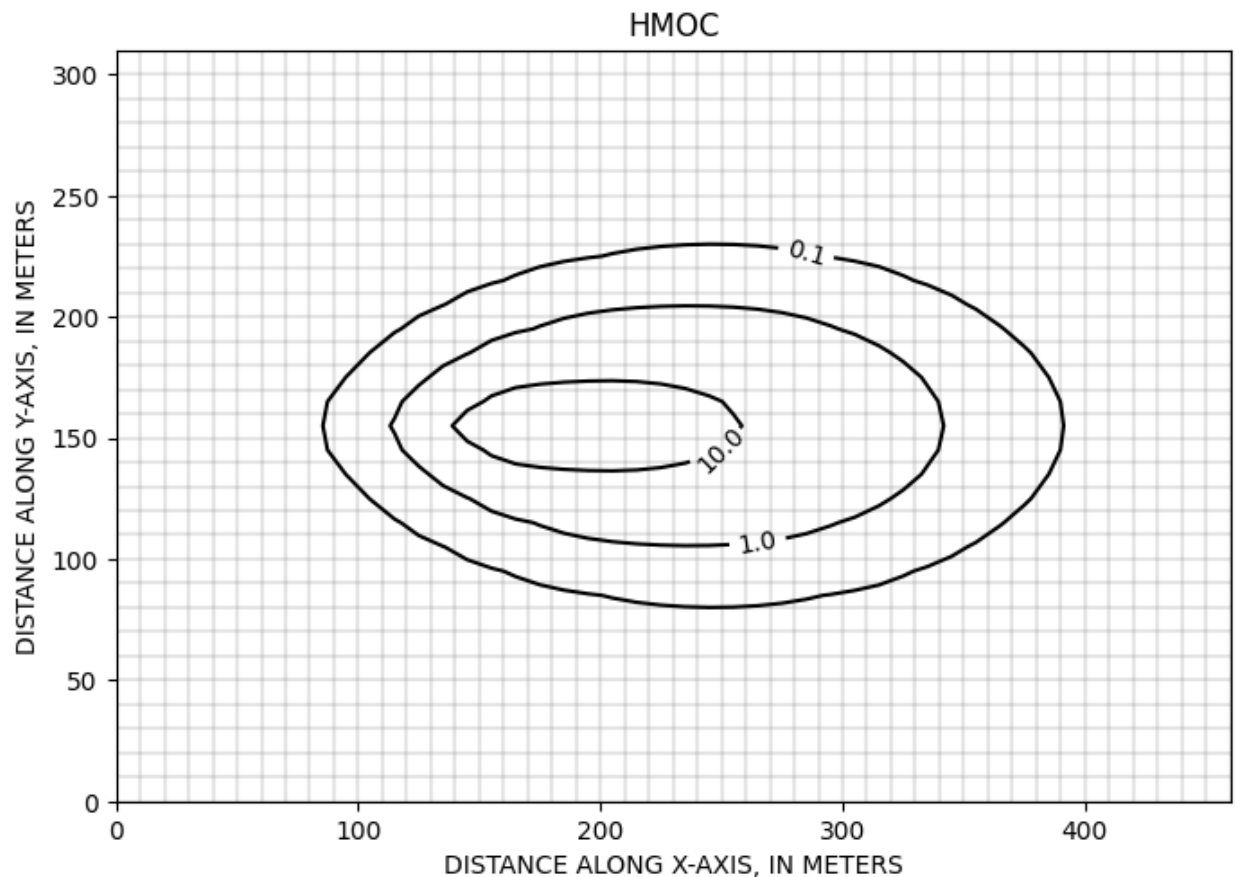
(continued from previous page)

```
mvt = mt.load_mas(fname)

return mf, mt, conc, cvt, mvt
```

```
[12]: ax = plt.subplot(1, 1, 1, aspect="equal")
mf, mt, conc, cvt, mvt = p03("p03", 3)
conc = conc[0, :, :, :]
pmv = flopy.plot.PlotMapView(model=mf)
pmv.plot_grid(color=".5", alpha=0.2)
cs = pmv.contour_array(conc, levels=[0.1, 1.0, 10.0, 50.0], colors="k")
plt.clabel(cs)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("HMOC")
```

```
[12]: Text(0.5, 1.0, 'HMOC')
```



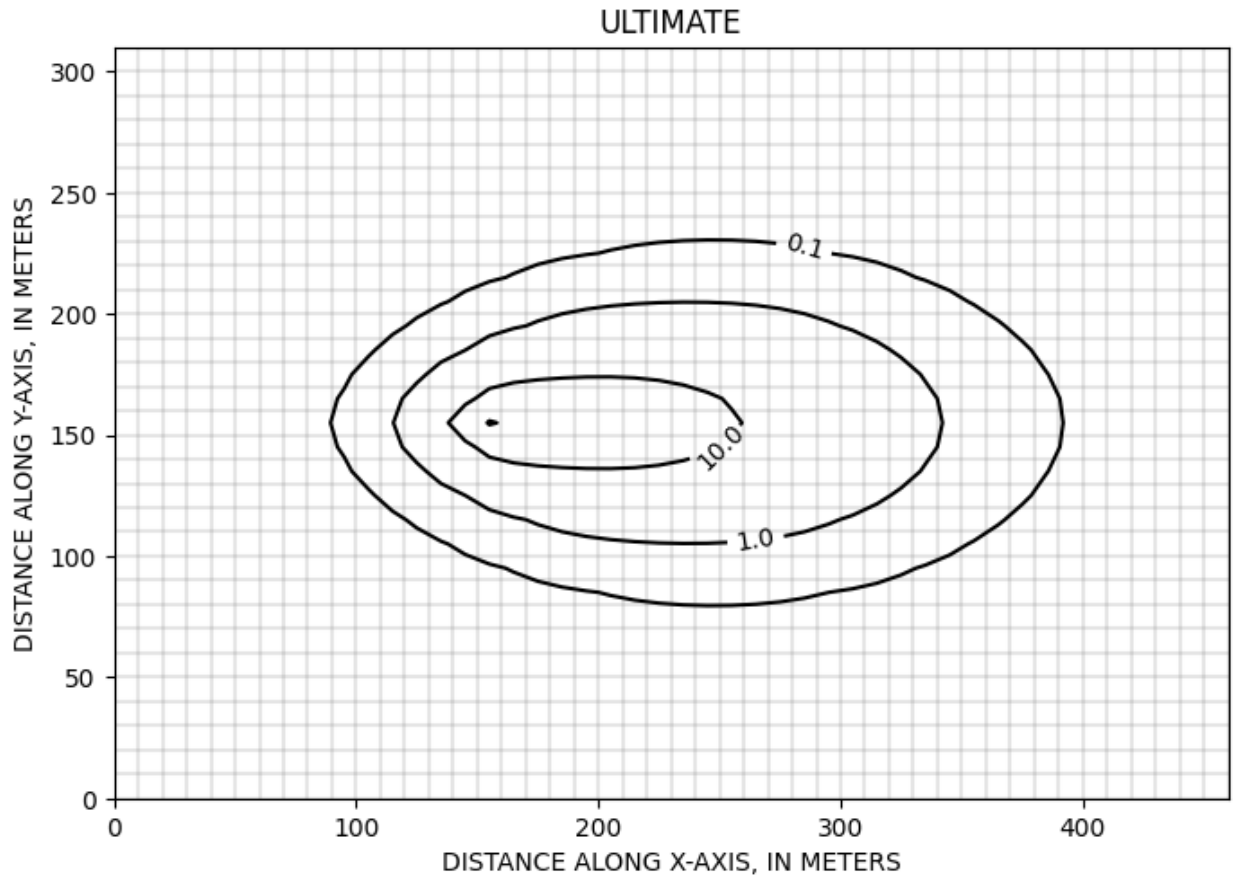
```
[13]: ax = plt.subplot(1, 1, 1, aspect="equal")
mf, mt, conc, cvt, mvt = p03("p03", -1)
conc = conc[0, :, :, :]
pmv = flopy.plot.PlotMapView(model=mf)
pmv.plot_grid(color=".5", alpha=0.2)
cs = pmv.contour_array(conc, levels=[0.1, 1.0, 10.0, 50.0], colors="k")
```

(continues on next page)

(continued from previous page)

```
plt.clabel(cs)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("ULTIMATE")
```

```
[13]: Text(0.5, 1.0, 'ULTIMATE')
```



Example 4. Two-Dimensional Transport in a Diagonal Flow Field

```
[14]: def p04(dirname, mixelm):
    model_ws = os.path.join(workdir, dirname)
    nlay = 1
    nrow = 100
    ncol = 100
    delr = 10
    delc = 10
    delv = 1
    Lx = (ncol - 1) * delr
    Ly = (nrow - 1) * delc
    Ls = np.sqrt(Lx**2 + Ly**2)
    v = 1.0
    prsity = 0.14
```

(continues on next page)

(continued from previous page)

```

q = v * prsity
al = 2.0
trpt = 0.1
q0 = 0.01
c0 = 1000.0

perlen_mf = 1000.0
perlen_mt = 1000.0
hk = 1.0
laytyp = 0

modelname_mf = dirname + "_mf"
mf = flopy.modflow.Modflow(
    modelname=modelname_mf, model_ws=model_ws, exe_name=exe_name_mf
)
dis = flopy.modflow.ModflowDis(
    mf,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=delr,
    delc=delc,
    top=0.0,
    botm=[0 - delv],
    perlen=perlen_mf,
)
ibound = np.ones((nlay, nrow, ncol), dtype=int) * -1
ibound[:, 1 : nrow - 1, 1 : ncol - 1] = 1

# set strt as a linear gradient at a 45 degree angle
h1 = q * Ls
x = mf.modelgrid.xcellcenters
y = mf.modelgrid.ycellcenters
a = -1
b = -1
c = 1
d = abs(a * x + b * y + c) / np.sqrt(2)
strt = h1 - d / Ls * h1

bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
lpf = flopy.modflow.ModflowLpf(mf, hk=hk, laytyp=laytyp)
wel = flopy.modflow.ModflowWel(mf, stress_period_data=[[0, 79, 20, q0]])
pcg = flopy.modflow.ModflowPcg(mf)
lmt = flopy.modflow.ModflowLmt(mf)
mf.write_input()
mf.run_model(silent=True)

modelname_mt = dirname + "_mt"
mt = flopy.mt3d.Mt3dms(
    modelname=modelname_mt,
    model_ws=model_ws,
    exe_name=exe_name_mt,

```

(continues on next page)

(continued from previous page)

```

        modflowmodel=mf,
    )
    btn = flopy.mt3d.Mt3dBtn(mt, icbund=1, prsity=prsity, sconc=0)
    dceps = 1.0e-5
    nplane = 1
    npl = 0
    nph = 16
    npmin = 2
    npmax = 32
    dchmoc = 1.0e-3
    nlsink = nplane
    npsink = nph
    adv = flopy.mt3d.Mt3dAdv(
        mt,
        mixelm=mixelm,
        dceps=dceps,
        nplane=nplane,
        npl=npl,
        nph=nph,
        npmin=npmin,
        npmax=npmax,
        nlsink=nlsink,
        npsink=npsink,
        percel=0.5,
    )
    dsp = flopy.mt3d.Mt3dDsp(mt, al=al, trpt=trpt)
    spd = {0: [0, 79, 20, c0, 2]}
    ssm = flopy.mt3d.Mt3dSsm(mt, stress_period_data=spd)
    gcg = flopy.mt3d.Mt3dGcg(mt)
    mt.write_input()
    fname = os.path.join(model_ws, "MT3D001.UCN")
    if os.path.isfile(fname):
        os.remove(fname)
    mt.run_model(silent=True)

    fname = os.path.join(model_ws, "MT3D001.UCN")
    ucnoobj = flopy.utils.UcnFile(fname)
    times = ucnoobj.get_times()
    conc = ucnoobj.get_alldata()

    fname = os.path.join(model_ws, "MT3D001.OBS")
    if os.path.isfile(fname):
        cvt = mt.load_obs(fname)
    else:
        cvt = None

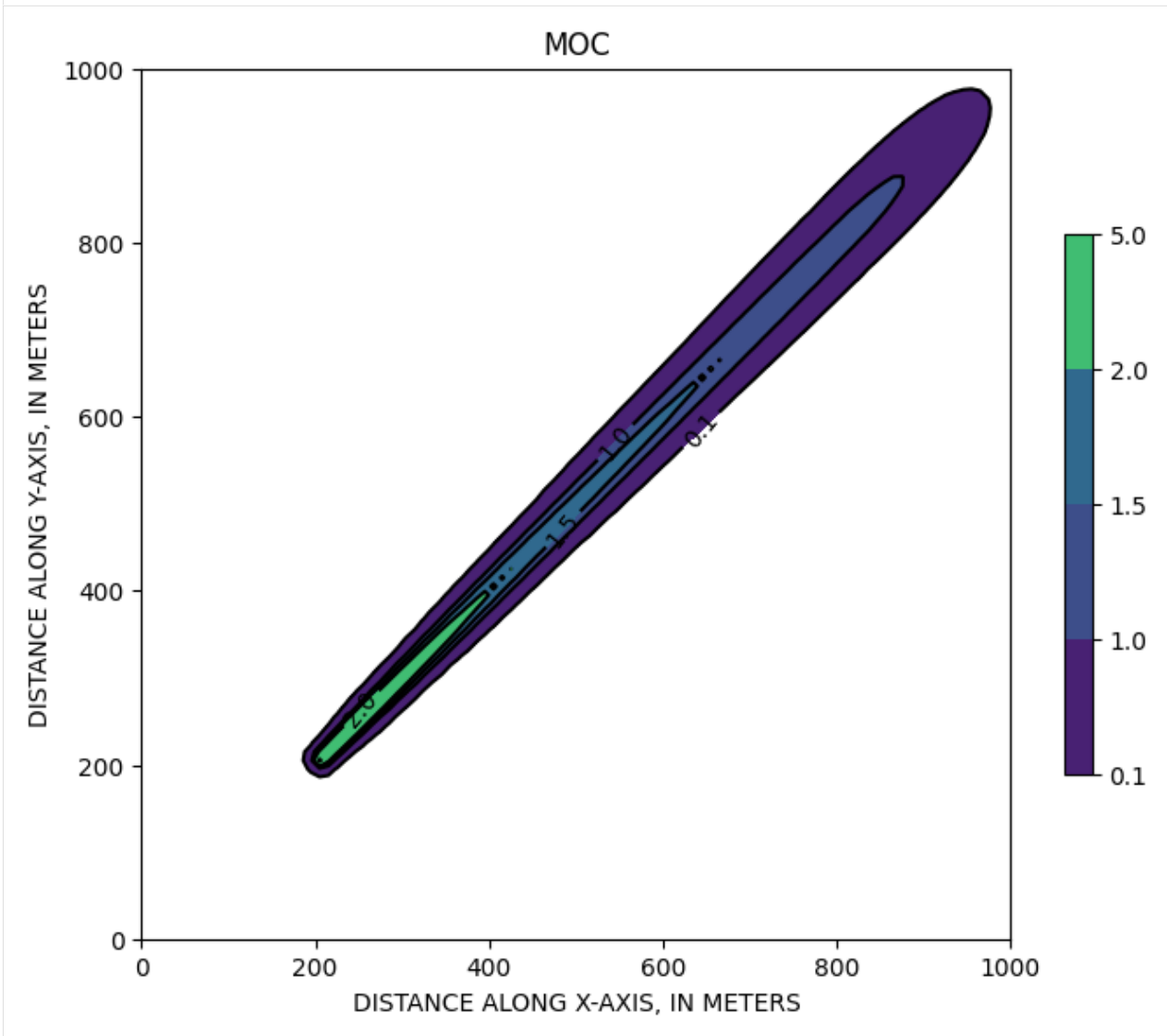
    fname = os.path.join(model_ws, "MT3D001.MAS")
    mvt = mt.load_mas(fname)

    return mf, mt, conc, cvt, mvt

```

```
[15]: ax = plt.subplot(1, 1, 1, aspect="equal")
mf, mt, conc, cvt, mvt = p04("p04", 1)
grid = mf.modelgrid
conc = conc[0, :, :, :]
levels = [0.1, 1.0, 1.5, 2.0, 5.0]
pmv = flopy.plot.PlotMapView(model=mf)
cf = plt.contourf(grid.xcellcenters, grid.ycellcenters, conc[0], levels=levels)
plt.colorbar(cf, shrink=0.5)
cs = pmv.contour_array(conc, levels=levels, colors="k")
plt.clabel(cs)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("MOC")
```

```
[15]: Text(0.5, 1.0, 'MOC')
```



```
[16]: ax = plt.subplot(1, 1, 1, aspect="equal")
mf, mt, conc, cvt, mvt = p04("p04", 0)
```

(continues on next page)

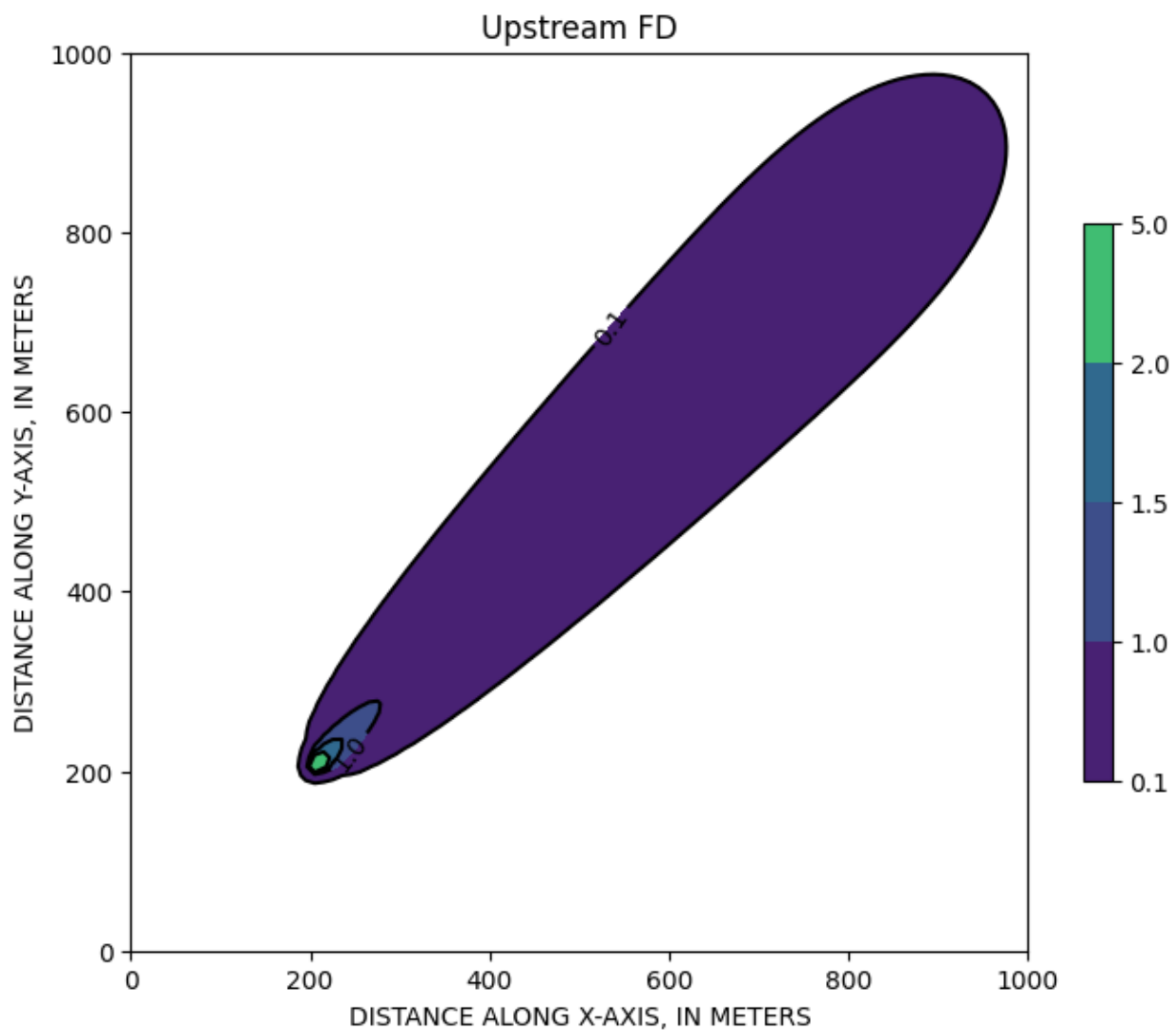
(continued from previous page)

```

grid = mf.modelgrid
conc = conc[0, :, :, :]
levels = [0.1, 1.0, 1.5, 2.0, 5.0]
pmv = flopy.plot.PlotMapView(model=mf)
cf = plt.contourf(grid.xcellcenters, grid.ycellcenters, conc[0], levels=levels)
plt.colorbar(cf, shrink=0.5)
cs = pmv.contour_array(conc, levels=levels, colors="k")
plt.clabel(cs)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("Upstream FD")

```

[16]: `Text(0.5, 1.0, 'Upstream FD')`



```

[17]: ax = plt.subplot(1, 1, 1, aspect="equal")
mf, mt, conc, cvt, mvt = p04("p04", -1)
grid = mf.modelgrid

```

(continues on next page)

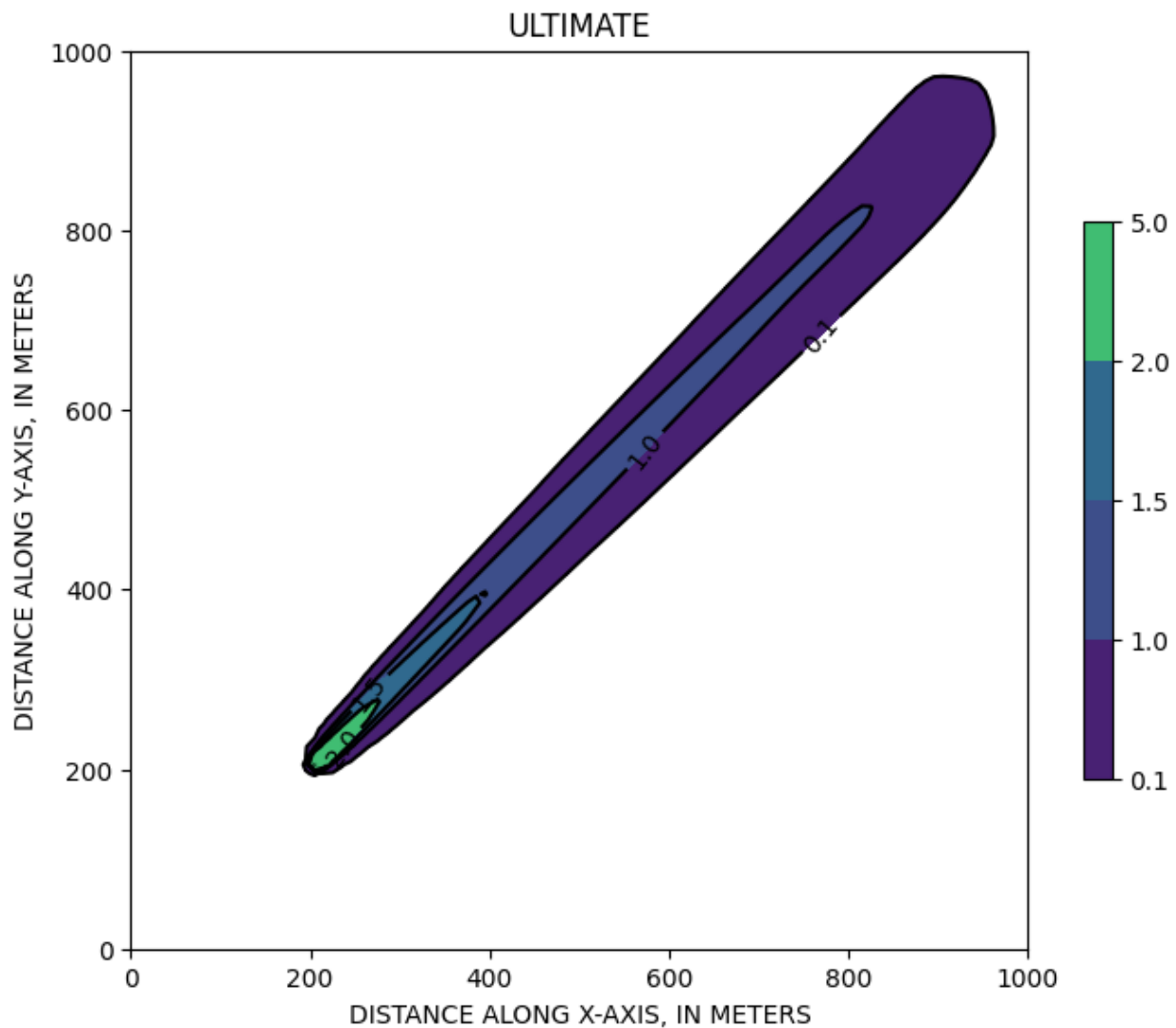
(continued from previous page)

```

conc = conc[0, :, :, :]
levels = [0.1, 1.0, 1.5, 2.0, 5.0]
pmv = flopy.plot.PlotMapView(model=mf)
cf = plt.contourf(grid.xcellcenters, grid.ycellcenters, conc[0], levels=levels)
plt.colorbar(cf, shrink=0.5)
cs = pmv.contour_array(conc, levels=levels, colors="k")
plt.clabel(cs)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("ULTIMATE")

```

```
[17]: Text(0.5, 1.0, 'ULTIMATE')
```



Example 5. Two-Dimensional Transport in a Radial Flow Field

```
[18]: def p05(dirname, mixelm, dt0, ttsmult):
    model_ws = os.path.join(workdir, dirname)
    nlay = 1
    nrow = 31
    ncol = 31
    delr = 10
    delc = 10
    delv = 1
    prsity = 0.30
    al = 10.0
    trpt = 1.0
    q0 = 100.0
    c0 = 1.0

    perlen_mf = 27.0
    perlen_mt = 27.0
    hk = 1.0
    laytyp = 0

    modelname_mf = dirname + "_mf"
    mf = flopy.modflow.Modflow(
        modelname=modelname_mf, model_ws=model_ws, exe_name=exe_name_mf
    )
    dis = flopy.modflow.ModflowDis(
        mf,
        nlay=nlay,
        nrow=nrow,
        ncol=ncol,
        delr=delr,
        delc=delc,
        top=0.0,
        botm=[0 - delv],
        perlen=perlen_mf,
    )
    ibound = np.ones((nlay, nrow, ncol), dtype=int) * -1
    ibound[:, 1 : nrow - 1, 1 : ncol - 1] = 1
    strt = 0.0

    bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
    lpf = flopy.modflow.ModflowLpf(mf, hk=hk, laytyp=laytyp)
    wel = flopy.modflow.ModflowWel(mf, stress_period_data=[[0, 15, 15, q0]])
    sip = flopy.modflow.ModflowSip(mf)
    lmt = flopy.modflow.ModflowLmt(mf)
    mf.write_input()
    mf.run_model(silent=True)

    modelname_mt = dirname + "_mt"
    mt = flopy.mt3d.Mt3dms(
        modelname=modelname_mt,
        model_ws=model_ws,
        exe_name=exe_name_mt,
```

(continues on next page)

(continued from previous page)

```

        modflowmodel=mf,
    )
    btn = flopy.mt3d.Mt3dBtn(
        mt, icbund=1, prsity=prsity, sconc=0, dt0=dt0, ttsmult=ttsmult
    )
    dceps = 1.0e-5
    nplane = 1
    npl = 0
    nph = 16
    npmin = 2
    npmax = 32
    dchmoc = 1.0e-3
    nlsink = nplane
    npsink = nph
    adv = flopy.mt3d.Mt3dAdv(
        mt,
        mixelm=mixelm,
        dceps=dceps,
        nplane=nplane,
        npl=npl,
        nph=nph,
        npmin=npmin,
        npmax=npmax,
        nlsink=nlsink,
        npsink=npsink,
        percel=0.5,
    )
    dsp = flopy.mt3d.Mt3dDsp(mt, al=al, trpt=trpt)
    spd = {0: [0, 15, 15, c0, -1]}
    ssm = flopy.mt3d.Mt3dSsm(mt, stress_period_data=spd)
    gcg = flopy.mt3d.Mt3dGcg(mt)
    mt.write_input()
    fname = os.path.join(model_ws, "MT3D001.UCN")
    if os.path.isfile(fname):
        os.remove(fname)
    mt.run_model(silent=True)

    fname = os.path.join(model_ws, "MT3D001.UCN")
    ucobj = flopy.utils.UcnFile(fname)
    times = ucobj.get_times()
    conc = ucobj.get_alldata()

    fname = os.path.join(model_ws, "MT3D001.OBS")
    if os.path.isfile(fname):
        cvt = mt.load_obs(fname)
    else:
        cvt = None

    fname = os.path.join(model_ws, "MT3D001.MAS")
    mvt = mt.load_mas(fname)

    return mf, mt, conc, cvt, mvt

```

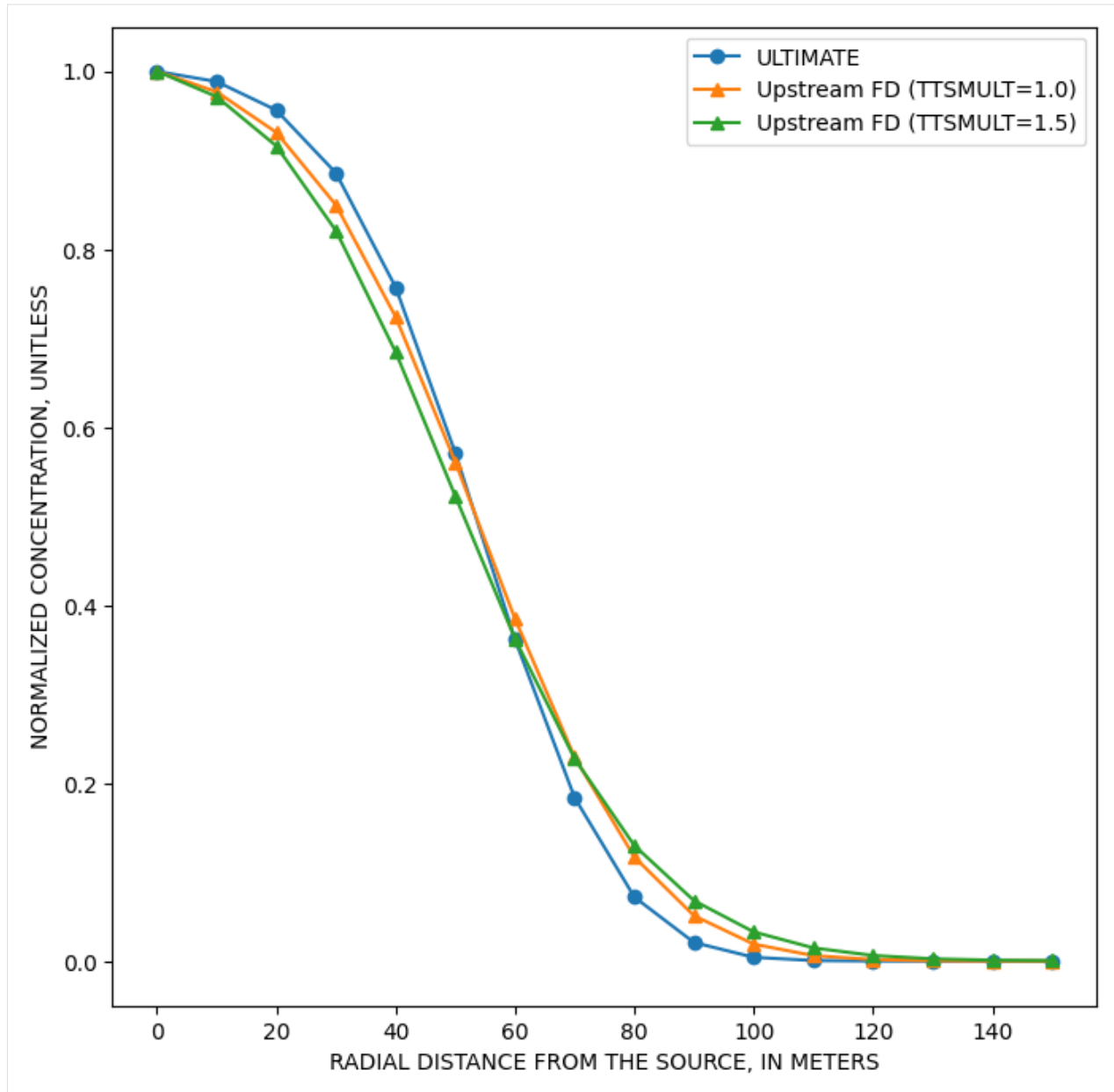
```
[19]: mf, mt, conc, cvt, mvt = p05("p05", -1, 0.3, 1.0)
      grid = mf.modelgrid
      conc = conc[0, 0, :, :]
      x = grid.xcellcenters[15, 15:] - grid.xcellcenters[15, 15]
      y = conc[15, 15:]
      plt.plot(x, y, label="ULTIMATE", marker="o")

      mf, mt, conc, cvt, mvt = p05("p05", 0, 0.3, 1.0)
      conc = conc[0, 0, :, :]
      x = grid.xcellcenters[15, 15:] - grid.xcellcenters[15, 15]
      y = conc[15, 15:]
      plt.plot(x, y, label="Upstream FD (TSMULT=1.0)", marker="^")

      mf, mt, conc, cvt, mvt = p05("p05", 0, 0.3, 1.5)
      conc = conc[0, 0, :, :]
      x = grid.xcellcenters[15, 15:] - grid.xcellcenters[15, 15]
      y = conc[15, 15:]
      plt.plot(x, y, label="Upstream FD (TSMULT=1.5)", marker="^")

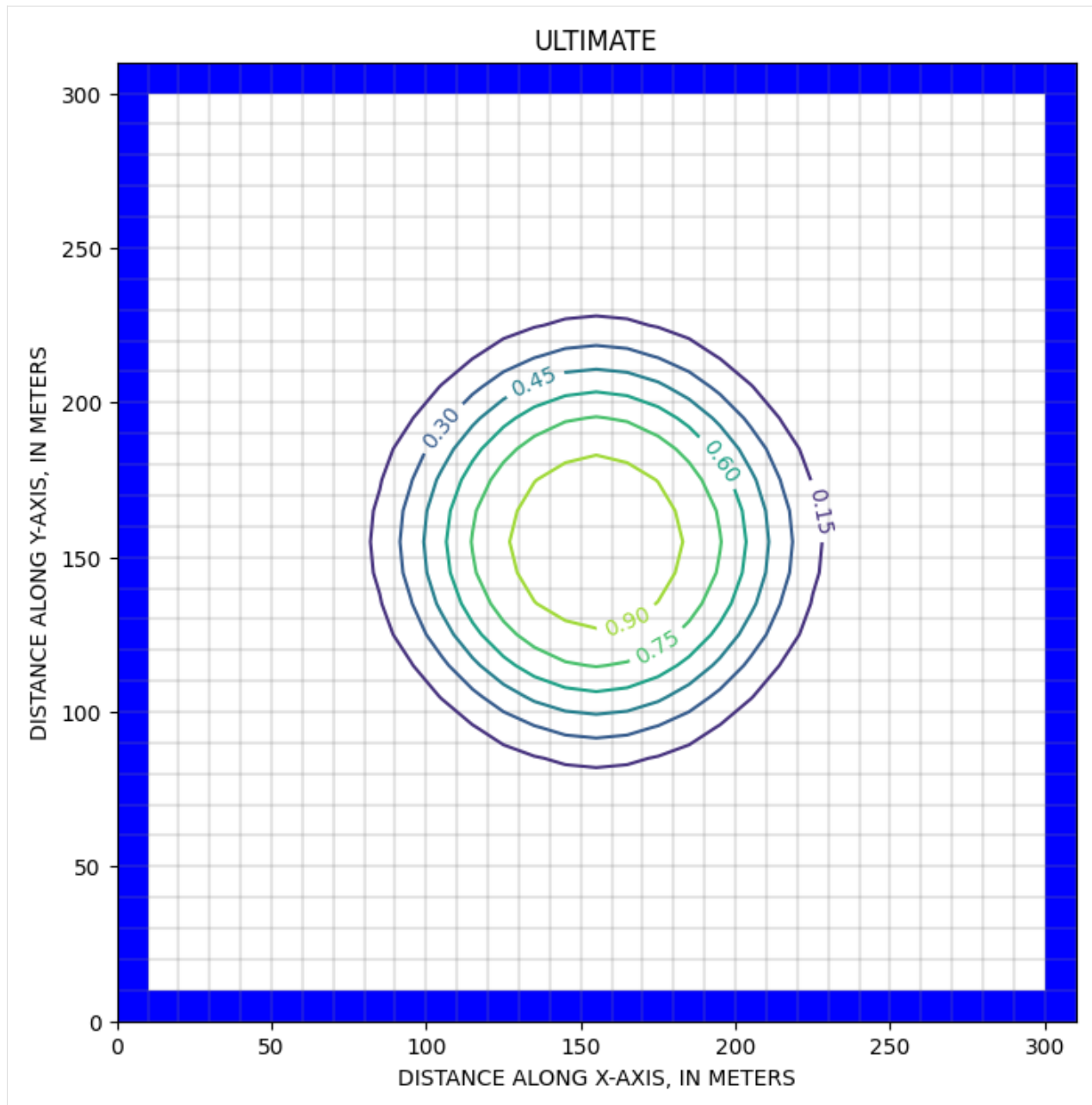
      plt.xlabel("RADIAL DISTANCE FROM THE SOURCE, IN METERS")
      plt.ylabel("NORMALIZED CONCENTRATION, UNITLESS")
      plt.legend()
```

```
[19]: <matplotlib.legend.Legend at 0x7f053c4a9f40>
```



```
[20]: ax = plt.subplot(1, 1, 1, aspect="equal")
mf, mt, conc, cvt, mvt = p05("p05", -1, 0.3, 1.0)
pmv = flopy.plot.PlotMapView(model=mf)
pmv.plot_grid(color=".5", alpha=0.2)
pmv.plot_ibound()
cs = pmv.contour_array(conc[0])
plt.clabel(cs)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("ULTIMATE")
```

```
[20]: Text(0.5, 1.0, 'ULTIMATE')
```

Example 6. Concentration at an Injection/Extraction Well

```
[21]: def p06(dirname, mixelm, dt0):
    model_ws = os.path.join(workdir, dirname)
    nlay = 1
    nrow = 31
    ncol = 31
    delr = 900
    delc = 900
    delv = 20
```

(continues on next page)

(continued from previous page)

```

prcity = 0.30
al = 100.0
trpt = 1.0
q0 = 86400.0
c0 = 100.0

perlen_mf = [912.5, 2737.5]
perlen_mt = perlen_mf
hk = 0.005 * 86400
laytyp = 0

modelname_mf = dirname + "_mf"
mf = flopy.modflow.Modflow(
    modelname=modelname_mf, model_ws=model_ws, exe_name=exe_name_mf
)
dis = flopy.modflow.ModflowDis(
    mf,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=delr,
    delc=delc,
    top=0.0,
    botm=[0 - delv],
    nper=2,
    perlen=perlen_mf,
)
ibound = np.ones((nlay, nrow, ncol), dtype=int) * -1
ibound[:, 1 : nrow - 1, 1 : ncol - 1] = 1
strt = 0.0

bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
lpf = flopy.modflow.ModflowLpf(mf, hk=hk, laytyp=laytyp)
welspd = {0: [[0, 15, 15, q0]], 1: [[0, 15, 15, -q0]]}
wel = flopy.modflow.ModflowWel(mf, stress_period_data=welspd)
sip = flopy.modflow.ModflowSip(mf)
lmt = flopy.modflow.ModflowLmt(mf)
mf.write_input()
mf.run_model(silent=True)

modelname_mt = dirname + "_mt"
mt = flopy.mt3d.Mt3dms(
    modelname=modelname_mt,
    model_ws=model_ws,
    exe_name=exe_name_mt,
    modflowmodel=mf,
)
btn = flopy.mt3d.Mt3dBtn(
    mt,
    icbund=1,
    prcity=prcity,
    scon=0,

```

(continues on next page)

(continued from previous page)

```

        nper=2,
        perlen=perlen_mt,
        dt0=dt0,
        obs=[(0, 15, 15)],
    )
    dceps = 1.0e-5
    nplane = 1
    npl = 16
    nph = 16
    npmin = 4
    npmax = 32
    dchmoc = 1.0e-3
    nlsink = nplane
    npsink = nph
    adv = flopy.mt3d.Mt3dAdv(
        mt,
        mixelm=mixelm,
        dceps=dceps,
        nplane=nplane,
        npl=npl,
        nph=nph,
        npmin=npmin,
        npmax=npmax,
        nlsink=nlsink,
        npsink=npsink,
        percel=0.5,
    )
    dsp = flopy.mt3d.Mt3dDsp(mt, al=al, trpt=trpt)
    spd = {0: [0, 15, 15, c0, 2], 1: [0, 15, 15, 0.0, 2]}
    ssm = flopy.mt3d.Mt3dSsm(mt, stress_period_data=spd)
    gcg = flopy.mt3d.Mt3dGcg(mt)
    mt.write_input()
    fname = os.path.join(model_ws, "MT3D001.UCN")
    if os.path.isfile(fname):
        os.remove(fname)
    mt.run_model(silent=True)

    fname = os.path.join(model_ws, "MT3D001.UCN")
    ucobj = flopy.utils.UcnFile(fname)
    times = ucobj.get_times()
    conc = ucobj.get_alldata()

    fname = os.path.join(model_ws, "MT3D001.OBS")
    if os.path.isfile(fname):
        cvt = mt.load_obs(fname)
    else:
        cvt = None

    fname = os.path.join(model_ws, "MT3D001.MAS")
    mvt = mt.load_mas(fname)

    return mf, mt, conc, cvt, mvt

```

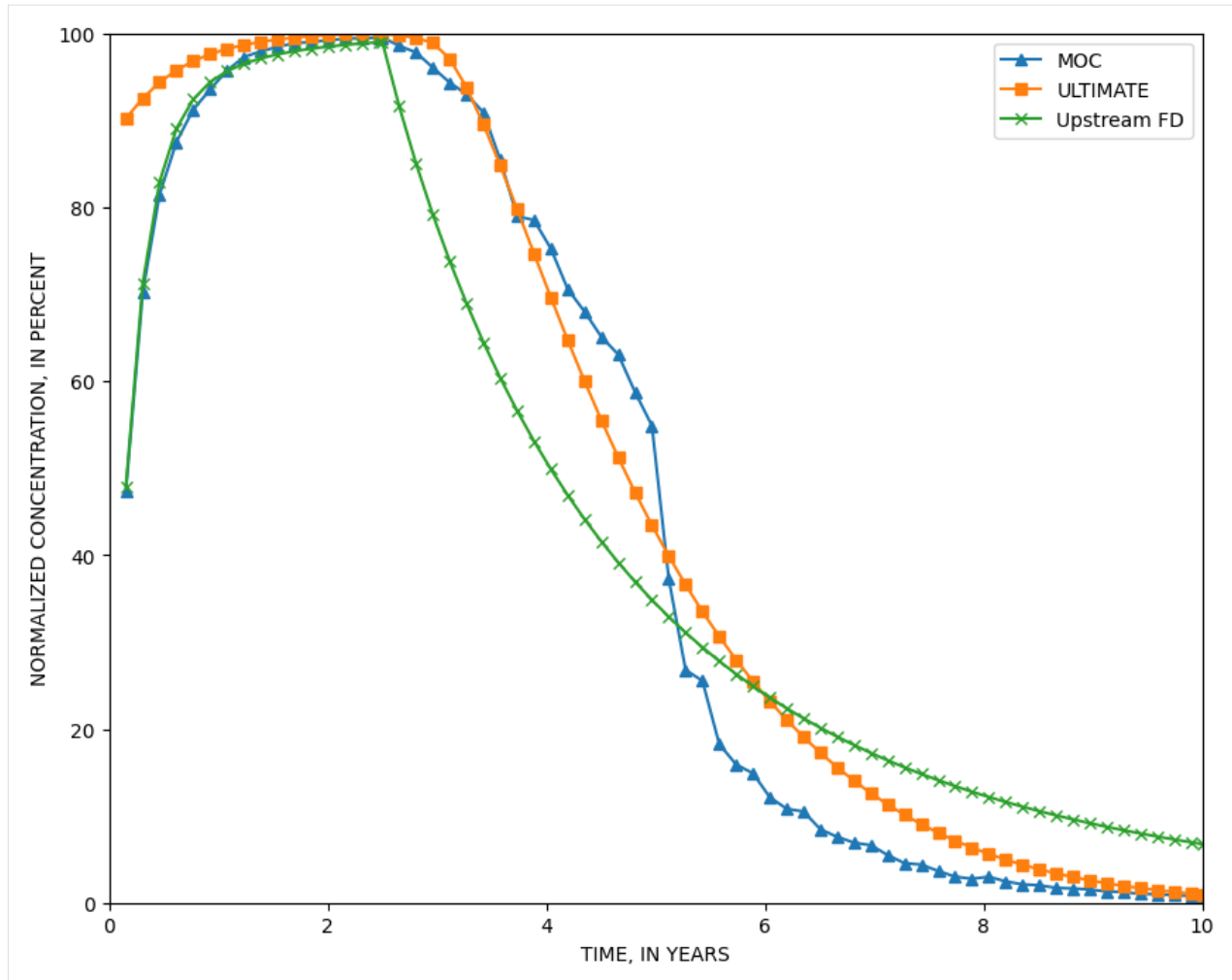
```
[22]: fig = plt.figure(figsize=(10, 8))
      ax = fig.add_subplot(1, 1, 1)
      mf, mt, conc, cvt, mvt = p06("p06", 1, 56.25)
      x = cvt["time"] / 365.0
      y = cvt["(1, 16, 16)"]
      ax.plot(x, y, label="MOC", marker="^")

      mf, mt, conc, cvt, mvt = p06("p06", -1, 56.25)
      x = cvt["time"] / 365.0
      y = cvt["(1, 16, 16)"]
      ax.plot(x, y, label="ULTIMATE", marker="s")

      mf, mt, conc, cvt, mvt = p06("p06", 0, 56.25)
      x = cvt["time"] / 365.0
      y = cvt["(1, 16, 16)"]
      ax.plot(x, y, label="Upstream FD", marker="x")

      plt.xlim(0, 10)
      plt.ylim(0, 100.0)
      plt.xlabel("TIME, IN YEARS")
      plt.ylabel("NORMALIZED CONCENTRATION, IN PERCENT")
      plt.legend()

[22]: <matplotlib.legend.Legend at 0x7f053c314eb0>
```



Example 7. Three-Dimensional Transport in a Uniform Flow Field

```
[23]: def p07(dirname, mixelm):
    model_ws = os.path.join(workdir, dirname)
    nlay = 8
    nrow = 15
    ncol = 21
    delr = 10
    delc = 10
    delv = 10
    Lx = (ncol - 1) * delr
    v = 1.0 / 3.0
    prsity = 0.2
    q = v * prsity
    al = 10.0
    trpt = 0.3
    trpv = 0.3
    q0 = 0.5
    c0 = 100.0
```

(continues on next page)

(continued from previous page)

```

perlen_mf = 100.0
perlen_mt = 100.0
hk = 0.5
laytyp = 0

modelname_mf = dirname + "_mf"
mf = flopy.modflow.Modflow(
    modelname=modelname_mf, model_ws=model_ws, exe_name=exe_name_mf
)
dis = flopy.modflow.ModflowDis(
    mf,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=delr,
    delc=delc,
    top=0.0,
    botm=[-delv * k for k in range(1, nlay + 1)],
    perlen=perlen_mf,
)
ibound = np.ones((nlay, nrow, ncol), dtype=int)
ibound[:, :, 0] = -1
ibound[:, :, -1] = -1
strt = np.zeros((nlay, nrow, ncol), dtype=float)
h1 = q * Lx
strt[:, :, 0] = h1
bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
lpf = flopy.modflow.ModflowLpf(mf, hk=hk, laytyp=laytyp)
wel = flopy.modflow.ModflowWel(mf, stress_period_data=[[6, 7, 2, q0]])
pcg = flopy.modflow.ModflowPcg(mf)
lmt = flopy.modflow.ModflowLmt(mf)
mf.write_input()
mf.run_model(silent=True)

modelname_mt = dirname + "_mt"
mt = flopy.mt3d.Mt3dms(
    modelname=modelname_mt,
    model_ws=model_ws,
    exe_name=exe_name_mt,
    modflowmodel=mf,
)
btn = flopy.mt3d.Mt3dBtn(mt, icbund=1, prsity=prsity, sconc=0)
dceps = 1.0e-5
nplane = 1
npl = 0
nph = 16
npmin = 2
npmax = 32
dchmoc = 1.0e-3
nlsink = nplane
npsink = nph

```

(continues on next page)

(continued from previous page)

```

adv = flopy.mt3d.Mt3dAdv(
    mt,
    mixelm=mixelm,
    dceps=dceps,
    nplane=nplane,
    npl=npl,
    nph=nph,
    npmin=npmin,
    npmax=npmax,
    nlsink=nlsink,
    npsink=npsink,
    percel=0.5,
)
dsp = flopy.mt3d.Mt3dDsp(mt, al=al, trpt=trpt, trpv=trpv)
spd = {0: [6, 7, 2, c0, 2]}
ssm = flopy.mt3d.Mt3dSsm(mt, stress_period_data=spd)
gcg = flopy.mt3d.Mt3dGcg(mt)
mt.write_input()
fname = os.path.join(model_ws, "MT3D001.UCN")
if os.path.isfile(fname):
    os.remove(fname)
mt.run_model(silent=True)

fname = os.path.join(model_ws, "MT3D001.UCN")
ucnobj = flopy.utils.UcnFile(fname)
times = ucnobj.get_times()
conc = ucnobj.get_alldata()

fname = os.path.join(model_ws, "MT3D001.OBS")
if os.path.isfile(fname):
    cvt = mt.load_obs(fname)
else:
    cvt = None

fname = os.path.join(model_ws, "MT3D001.MAS")
mvt = mt.load_mas(fname)

return mf, mt, conc, cvt, mvt

```

```

[24]: fig = plt.figure(figsize=(10, 20))
mf, mt, conc, cvt, mvt = p07("p07", -1)
conc = conc[0]

ax = fig.add_subplot(3, 1, 1, aspect="equal")
ilay = 4
pmv = flopy.plot.PlotMapView(ax=ax, model=mf, layer=ilay)
pmv.plot_grid(color=".5", alpha=0.2)
pmv.plot_ibound()
cs = pmv.contour_array(conc, levels=[0.01, 0.05, 0.15, 0.50], colors="k")
plt.clabel(cs)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")

```

(continues on next page)

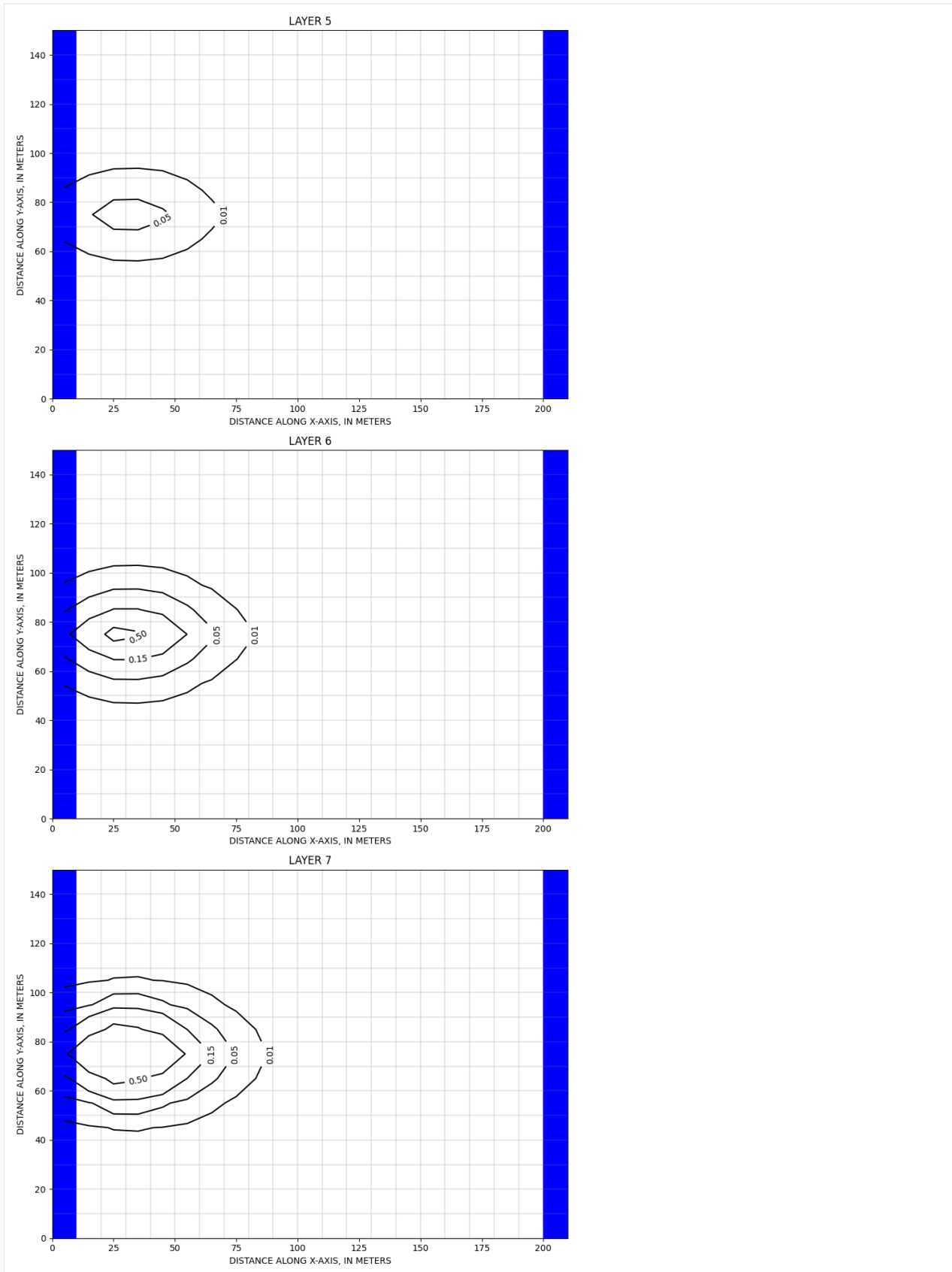
(continued from previous page)

```
plt.title("LAYER {}".format(ilay + 1))

ax = fig.add_subplot(3, 1, 2, aspect="equal")
ilay = 5
pmv = flopy.plot.PlotMapView(ax=ax, model=mf, layer=ilay)
pmv.plot_grid(color=".5", alpha=0.2)
pmv.plot_ibound()
cs = pmv.contour_array(conc, levels=[0.01, 0.05, 0.15, 0.50], colors="k")
plt.clabel(cs)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("LAYER {}".format(ilay + 1))

ax = fig.add_subplot(3, 1, 3, aspect="equal")
ilay = 6
pmv = flopy.plot.PlotMapView(ax=ax, model=mf, layer=ilay)
pmv.plot_grid(color=".5", alpha=0.2)
pmv.plot_ibound()
cs = pmv.contour_array(conc, levels=[0.01, 0.05, 0.15, 0.50], colors="k")
plt.clabel(cs)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("LAYER {}".format(ilay + 1))
plt.plot(grid.xcellcenters[7, 2], grid.ycellcenters[7, 2], "ko")

plt.tight_layout()
```

Example 8. Two-Dimensional, Vertical Transport in a Heterogeneous Aquifer

```
[25]: def p08(dirname, mixelm):
    model_ws = os.path.join(workdir, dirname)
    nlay = 27
    nrow = 1
    ncol = 50
    delr = 5
    delc = 1
    delv = 0.25
    prsity = 0.35
    al = 0.5
    trpt = 0.01
    trpv = 0.01
    dmcoef = 1.34e-5 / 100 / 100 * 86400
    rech = 0.1 / 365 # m/d

    perlen_mf = 1
    perlen_mt = [5 * 365, 15 * 365]

    k1 = 5e-4 / 100.0 * 86400 # m/d
    k2 = 1e-2 / 100.0 * 86400 # m/d
    hk = k1 * np.ones((nlay, nrow, ncol), dtype=float)
    hk[11:19, :, 0:24] = k2
    hk[11:19, :, 36:] = k2
    laytyp = 6 * [1] + 21 * [0]

    modelname_mf = dirname + "_mf"
    mf = flopy.modflow.Modflow(
        modelname=modelname_mf, model_ws=model_ws, exe_name=exe_name_mf
    )
    dis = flopy.modflow.ModflowDis(
        mf,
        nlay=nlay,
        nrow=nrow,
        ncol=ncol,
        delr=delr,
        delc=delc,
        top=6.75,
        botm=[6.75 - delv * k for k in range(1, nlay + 1)],
        perlen=perlen_mf,
    )
    f = open(os.path.join(datadir, "p08shead.dat"))
    strt = np.empty((nlay * ncol), dtype=float)
    strt = readId(f, strt).reshape((nlay, nrow, ncol))
    f.close()
    ibound = np.ones((nlay, nrow, ncol), dtype=int)
    ibound[5:, :, -1] = -1
    ibound[strt < 0] = 0

    bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
    lpf = flopy.modflow.ModflowLpf(mf, hk=hk, vka=hk, laytyp=laytyp)
    rch = flopy.modflow.ModflowRch(mf, rech=rech)
```

(continues on next page)

(continued from previous page)

```

pcg = flopy.modflow.ModflowPcg(mf)
lmt = flopy.modflow.ModflowLmt(mf)
mf.write_input()
mf.run_model(silent=True)

modelname_mt = dirname + "_mt"
mt = flopy.mt3d.Mt3dms(
    modelname=modelname_mt,
    model_ws=model_ws,
    exe_name=exe_name_mt,
    modflowmodel=mf,
)
btn = flopy.mt3d.Mt3dBtn(
    mt,
    icbund=1,
    prsity=prsity,
    sconc=0,
    nper=2,
    perlen=perlen_mt,
    timprs=[8 * 365, 12 * 365, 20 * 365],
)
percel = 1.0
itrack = 3
wd = 0.5
dceps = 1.0e-5
nplane = 0
npl = 0
nph = 10
npmin = 2
npmax = 20
dchmoc = 1.0e-3
nlsink = nplane
npsink = nph
adv = flopy.mt3d.Mt3dAdv(
    mt,
    mixelm=mixelm,
    dceps=dceps,
    nplane=nplane,
    npl=npl,
    nph=nph,
    npmin=npmin,
    npmax=npmax,
    nlsink=nlsink,
    npsink=npsink,
    percel=percel,
    itrack=itrack,
    wd=wd,
)
dsp = flopy.mt3d.Mt3dDsp(mt, al=al, trpt=trpt, trpv=trpv, dmcoef=dmcoef)
crch1 = np.zeros((nrow, ncol), dtype=float)
crch1[0, 9:18] = 1.0
cnc0 = [(0, 0, j, 1, -1) for j in range(8, 16)]

```

(continues on next page)

(continued from previous page)

```
cnc1 = [(0, 0, j, 0.0, -1) for j in range(8, 16)]
ssmspd = {0: cnc0, 1: cnc1}
ssm = flopy.mt3d.Mt3dSsm(mt, stress_period_data=ssmspd)
gcg = flopy.mt3d.Mt3dGcg(mt)
mt.write_input()
fname = os.path.join(model_ws, "MT3D001.UCN")
if os.path.isfile(fname):
    os.remove(fname)
mt.run_model(silent=True)

fname = os.path.join(model_ws, "MT3D001.UCN")
ucnobj = flopy.utils.UcnFile(fname)
times = ucnobj.get_times()
conc = ucnobj.get_alldata()

fname = os.path.join(model_ws, "MT3D001.OBS")
if os.path.isfile(fname):
    cvt = mt.load_obs(fname)
else:
    cvt = None

fname = os.path.join(model_ws, "MT3D001.MAS")
mvt = mt.load_mas(fname)

return mf, mt, conc, cvt, mvt
```

```
[26]: fig = plt.figure(figsize=(10, 15))
mf, mt, conc, cvt, mvt = p08("p08", 3)
hk = mf.lpf.hk.array

ax = fig.add_subplot(3, 1, 1)
mx = flopy.plot.PlotCrossSection(ax=ax, model=mf, line={"row": 0})
mx.plot_array(hk, masked_values=[hk[0, 0, 0]], alpha=0.2)
mx.plot_ibound()
mx.plot_grid(color="0.5", alpha=0.2)
cs = mx.contour_array(
    conc[1], levels=np.arange(0.05, 0.55, 0.05), masked_values=[1.0e30]
)
ax.set_title("TIME = 8 YEARS")

ax = fig.add_subplot(3, 1, 2)
mx = flopy.plot.PlotCrossSection(ax=ax, model=mf, line={"row": 0})
mx.plot_array(hk, masked_values=[hk[0, 0, 0]], alpha=0.2)
mx.plot_ibound()
mx.plot_grid(color="0.5", alpha=0.2)
cs = mx.contour_array(
    conc[2], levels=np.arange(0.05, 0.55, 0.05), masked_values=[1.0e30]
)
ax.set_title("TIME = 12 YEARS")

ax = fig.add_subplot(3, 1, 3)
mx = flopy.plot.PlotCrossSection(ax=ax, model=mf, line={"row": 0})
```

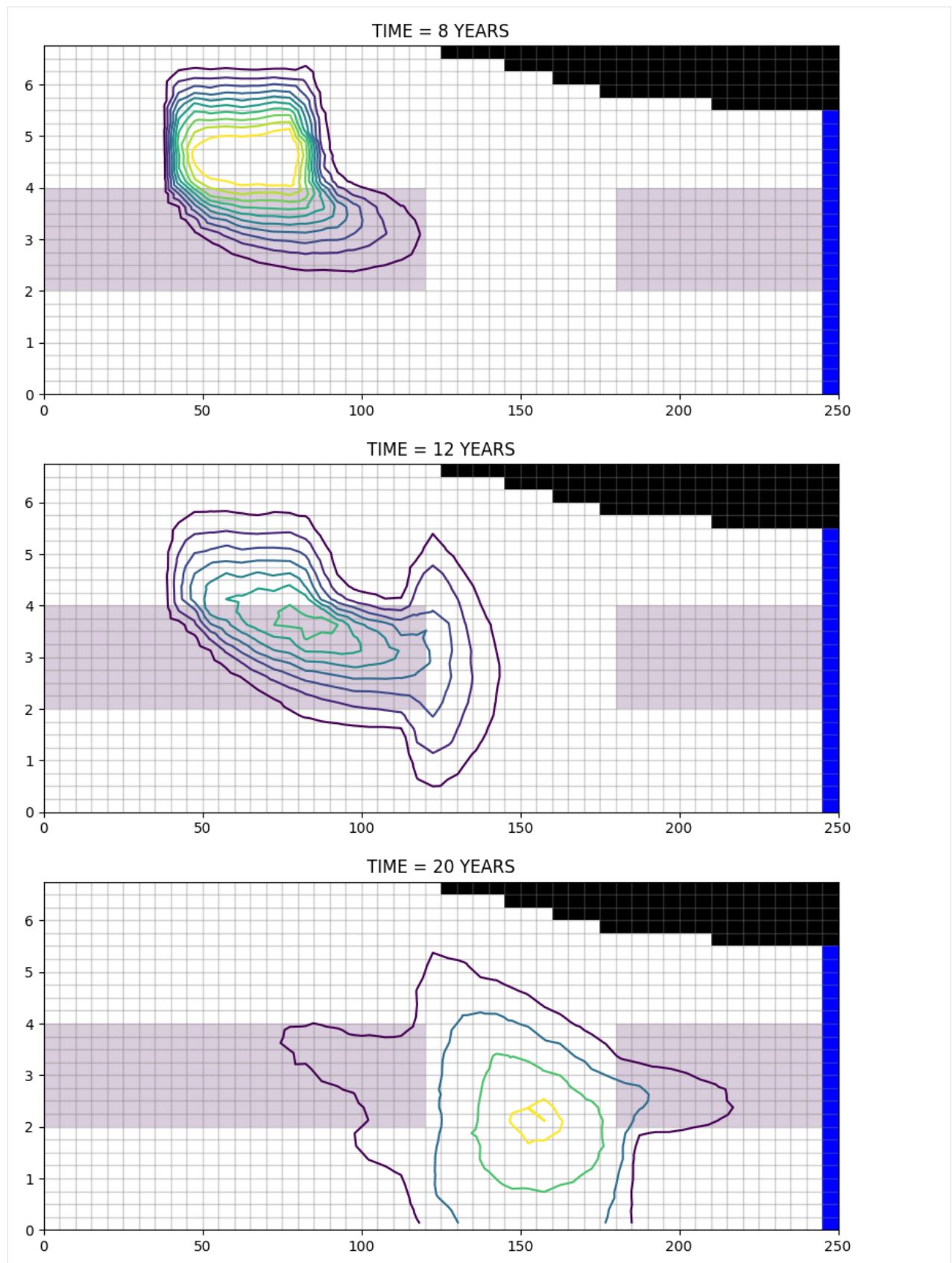
(continues on next page)

(continued from previous page)

```
mx.plot_array(hk, masked_values=[hk[0, 0, 0]], alpha=0.2)
mx.plot_ibound()
mx.plot_grid(color="0.5", alpha=0.2)
cs = mx.contour_array(
    conc[3], levels=[0.05, 0.1, 0.15, 0.19], masked_values=[1.0e30]
)
ax.set_title("TIME = 20 YEARS")
```

```
found 'rch' in modflow model, resetting crch to 0.0
```

```
[26]: Text(0.5, 1.0, 'TIME = 20 YEARS')
```



Example 9. Two-Dimensional Application Example

```
[27]: def p09(dirname, mixelm, nadvfd):
    model_ws = os.path.join(workdir, dirname)
    nlay = 1
    nrow = 18
    ncol = 14
    delr = 100
    delc = 100
    delv = 10
    prsity = 0.3
    al = 20.0
    trpt = 0.2

    perlen_mf = 1.0
    perlen_mt = [365.0 * 86400, 365.0 * 86400]
    laytyp = 0
    k1 = 1.474e-4
    k2 = 1.474e-7
    hk = k1 * np.ones((nlay, nrow, ncol), dtype=float)
    hk[:, 5:8, 1:8] = k2

    modelname_mf = dirname + "_mf"
    mf = flopy.modflow.Modflow(
        modelname=modelname_mf, model_ws=model_ws, exe_name=exe_name_mf
    )
    dis = flopy.modflow.ModflowDis(
        mf,
        nlay=nlay,
        nrow=nrow,
        ncol=ncol,
        delr=delr,
        delc=delc,
        top=0.0,
        botm=[0 - delv],
        perlen=perlen_mf,
    )
    ibound = np.ones((nlay, nrow, ncol), dtype=int)
    ibound[0, 0, :] = -1
    ibound[0, -1, :] = -1
    strt = np.zeros((nlay, nrow, ncol), dtype=float)
    strt[0, 0, :] = 250.0
    xc = mf.modelgrid.xcellcenters
    for j in range(ncol):
        strt[0, -1, j] = 20.0 + (xc[0, j] - xc[0, 0]) * 2.5 / 100
    bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
    lpf = flopy.modflow.ModflowLpf(mf, hk=hk, laytyp=laytyp)
    welspd = [[0, 3, 6, 0.001], [0, 10, 6, -0.0189]]
    wel = flopy.modflow.ModflowWel(mf, stress_period_data=welspd)
    pcg = flopy.modflow.ModflowPcg(mf)
    lmt = flopy.modflow.ModflowLmt(mf)
    mf.write_input()
    mf.run_model(silent=True)
```

(continues on next page)

(continued from previous page)

```

modelname_mt = dirname + "_mt"
mt = flopy.mt3d.Mt3dms(
    modelname=modelname_mt,
    model_ws=model_ws,
    exe_name=exe_name_mt,
    modflowmodel=mf,
)
btn = flopy.mt3d.Mt3dBtn(
    mt,
    icbund=1,
    prsity=prsity,
    sconc=0,
    nper=2,
    perlen=perlen_mt,
    obs=[[0, 10, 6]],
)
percel = 1.0
itrack = 2
dceps = 1.0e-5
nplane = 0
npl = 0
nph = 16
npmin = 0
npmax = 32
dchmoc = 1.0e-3
nlsink = nplane
npsink = nph
adv = flopy.mt3d.Mt3dAdv(
    mt,
    mixelm=mixelm,
    dceps=dceps,
    nplane=nplane,
    npl=npl,
    nph=nph,
    npmin=npmin,
    npmax=npmax,
    nlsink=nlsink,
    npsink=npsink,
    percel=percel,
    itrack=itrack,
    nadvfd=nadvfd,
)
dsp = flopy.mt3d.Mt3dDsp(mt, al=al, trpt=trpt)
spd = {
    0: [[0, 3, 6, 57.87, 2], [0, 10, 6, 0.0, 2]],
    1: [[0, 3, 6, 0.0, 2], [0, 10, 6, 0.0, 2]],
}
ssm = flopy.mt3d.Mt3dSsm(mt, stress_period_data=spd)
gcg = flopy.mt3d.Mt3dGcg(mt)
mt.write_input()
fname = os.path.join(model_ws, "MT3D001.UCN")

```

(continues on next page)

(continued from previous page)

```

if os.path.isfile(fname):
    os.remove(fname)
mt.run_model(silent=True)

fname = os.path.join(model_ws, "MT3D001.UCN")
ucnobj = flopy.utils.UcnFile(fname)
times = ucnobj.get_times()
conc = ucnobj.get_alldata()

fname = os.path.join(model_ws, "MT3D001.OBS")
if os.path.isfile(fname):
    cvt = mt.load_obs(fname)
else:
    cvt = None

fname = os.path.join(model_ws, "MT3D001.MAS")
mvt = mt.load_mas(fname)

return mf, mt, conc, cvt, mvt

```

```

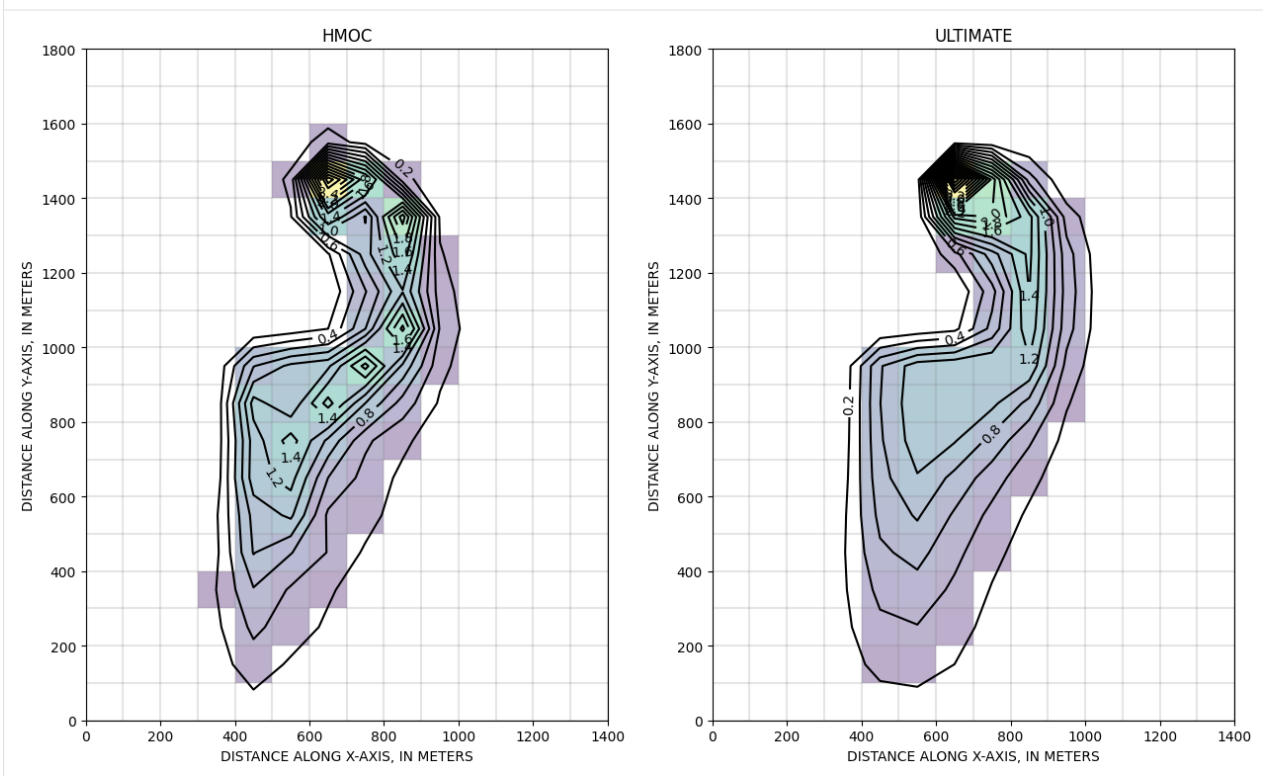
[28]: fig = plt.figure(figsize=(15, 10))
levels = np.arange(0.2, 10, 0.2)

ax = fig.add_subplot(1, 2, 1, aspect="equal")
mf, mt, conc, cvt, mvt = p09("p09", 3, 1)
cvt["time"] / 365.0 / 86400.0
y = cvt["(1, 11, 7)"]
conc = conc[:, 0, :, :]
cflood = np.ma.masked_less_equal(conc, 0.2)
pmv = flopy.plot.PlotMapView(ax=ax, model=mf)
pmv.plot_grid(color=".5", alpha=0.2)
cs = pmv.plot_array(cflood[0], alpha=0.2, vmin=0, vmax=3)
cs = pmv.contour_array(conc[0], colors="k", levels=levels)
plt.clabel(cs)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("HMOC")

ax = fig.add_subplot(1, 2, 2, aspect="equal")
mf, mt, conc, cvt, mvt = p09("p09", -1, 1)
cvt["time"] / 365.0 / 86400.0
y = cvt["(1, 11, 7)"]
conc = conc[:, 0, :, :]
cflood = np.ma.masked_less_equal(conc, 0.2)
pmv = flopy.plot.PlotMapView(ax=ax, model=mf)
pmv.plot_grid(color=".5", alpha=0.2)
cs = pmv.plot_array(cflood[0], alpha=0.2, vmin=0, vmax=3)
cs = pmv.contour_array(conc[0], colors="k", levels=levels)
plt.clabel(cs)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("ULTIMATE")

```

[28]: `Text(0.5, 1.0, 'ULTIMATE')`



```
[29]: fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(1, 1, 1)

mf, mt, conc, cvt, mvt = p09("p09", 3, 1)
x = cvt["time"] / 365.0 / 86400.0
y = cvt["(1, 11, 7)"]
plt.plot(x, y, label="HMOC", marker="d")

mf, mt, conc, cvt, mvt = p09("p09", -1, 1)
x = cvt["time"] / 365.0 / 86400.0
y = cvt["(1, 11, 7)"]
plt.plot(x, y, label="ULTIMATE", marker="^^")

mf, mt, conc, cvt, mvt = p09("p09", 0, 1)
x = cvt["time"] / 365.0 / 86400.0
y = cvt["(1, 11, 7)"]
plt.plot(x, y, label="Upstream FD", marker="^^")

mf, mt, conc, cvt, mvt = p09("p09", 0, 2)
x = cvt["time"] / 365.0 / 86400.0
y = cvt["(1, 11, 7)"]
plt.plot(x, y, label="Central FD", marker="^^")

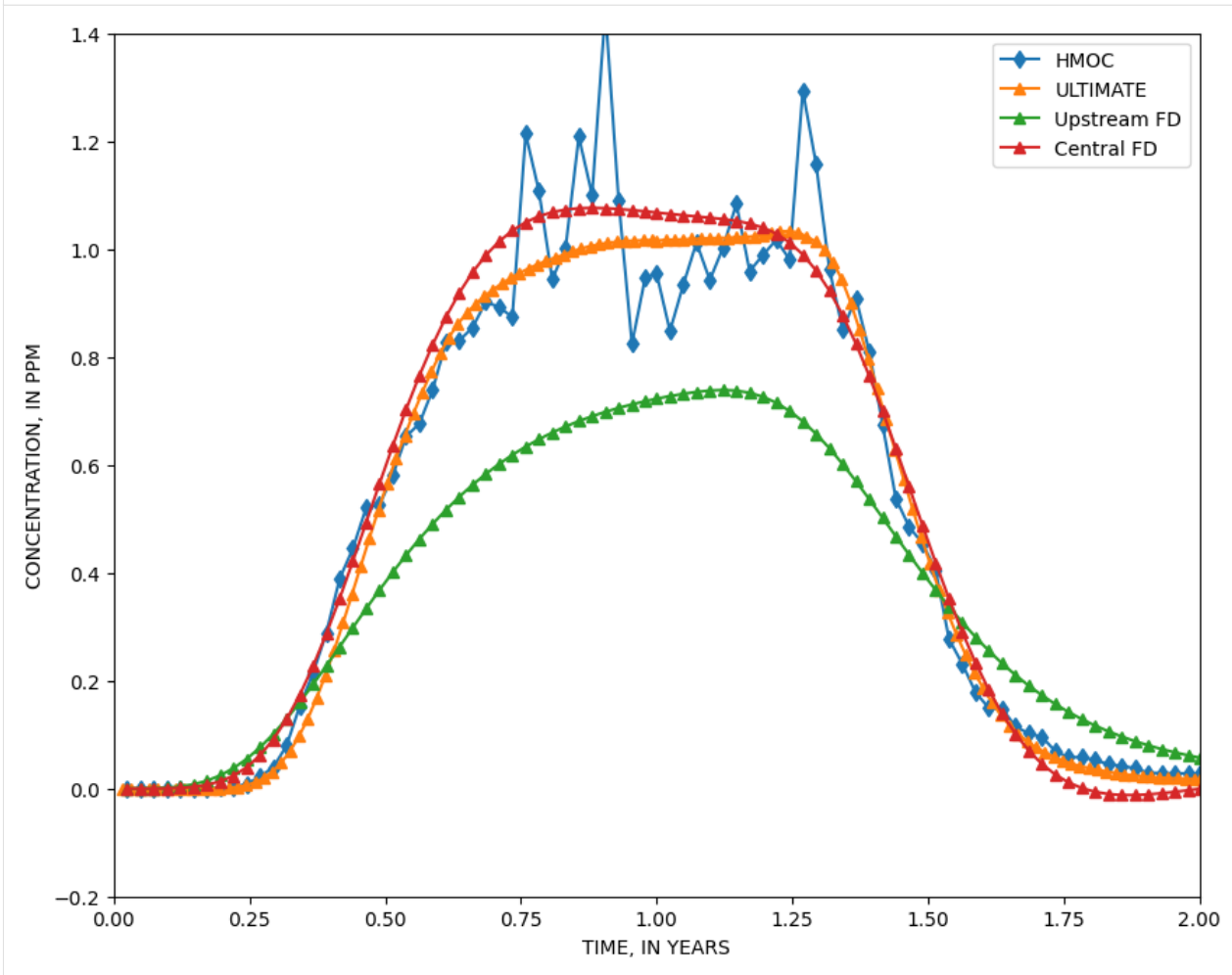
plt.xlim(0, 2)
plt.ylim(-0.2, 1.4)
plt.xlabel("TIME, IN YEARS")
```

(continues on next page)

(continued from previous page)

```
plt.ylabel("CONCENTRATION, IN PPM")
plt.legend()
```

[29]: <matplotlib.legend.Legend at 0x7f0535add790>



Example 10. Three-Dimensional Field Case Study

```
[30]: def p10(dirname, mixelm, perlen=1000, isothm=1, sp2=0.0, ttsmult=1.2):
    model_ws = os.path.join(workdir, dirname)
    nlay = 4
    nrow = 61
    ncol = 40
    delr = (
        [2000, 1600, 800, 400, 200, 100]
        + 28 * [50]
        + [100, 200, 400, 800, 1600, 2000]
    )
    delc = (
        [2000, 2000, 2000, 1600, 800, 400, 200, 100]
        + 45 * [50]
```

(continues on next page)

(continued from previous page)

```

    + [100, 200, 400, 800, 1600, 2000, 2000, 2000]
)
delv = 25.0
top = 780.0
botm = [top - delv * k for k in range(1, nlay + 1)]
prsim = 0.3
al = 10.0
trpt = 0.2
trpv = 0.2

perlen_mf = perlen
perlen_mt = perlen
hk = [60.0, 60.0, 520.0, 520.0]
vka = 0.1
laytyp = 0

modelname_mf = dirname + "_mf"
mf = flopy.modflow.Modflow(
    modelname=modelname_mf, model_ws=model_ws, exe_name=exe_name_mf
)
dis = flopy.modflow.ModflowDis(
    mf,
    nlay=nlay,
    nrow=nrow,
    ncol=ncol,
    delr=delr,
    delc=delc,
    top=top,
    botm=botm,
    perlen=perlen_mf,
)
ibound = np.ones((nlay, nrow, ncol), dtype=int)

# left side
ibound[:, :, 0] = -1
# right side
ibound[:, :, -1] = -1
# top
ibound[:, 0, :] = -1
# bottom
ibound[:, -1, :] = -1

f = open(os.path.join(datadir, "p10shead.dat"))
s0 = np.empty((nrow * ncol), dtype=float)
s0 = readld(f, s0).reshape((nrow, ncol))
f.close()
strt = np.zeros((nlay, nrow, ncol), dtype=float)
for k in range(nlay):
    strt[k] = s0
bas = flopy.modflow.ModflowBas(mf, ibound=ibound, strt=strt)
lpf = flopy.modflow.ModflowLpf(mf, hk=hk, layvka=1, vka=vka, laytyp=laytyp)
welspd = [

```

(continues on next page)

(continued from previous page)

```

        [3 - 1, 11 - 1, 29 - 1, -19230.00],
        [3 - 1, 19 - 1, 26 - 1, -19230.00],
        [3 - 1, 26 - 1, 23 - 1, -19230.00],
        [3 - 1, 33 - 1, 20 - 1, -19230.00],
        [3 - 1, 40 - 1, 17 - 1, -19230.00],
        [3 - 1, 48 - 1, 14 - 1, -19230.00],
        [3 - 1, 48 - 1, 9 - 1, -15384.00],
        [3 - 1, 52 - 1, 17 - 1, -17307.00],
    ]
    wel = flopy.modflow.ModflowWel(mf, stress_period_data=welspd)
    rch = flopy.modflow.ModflowRch(mf, rech=1.14e-3)
    pcg = flopy.modflow.ModflowPcg(mf)
    lmt = flopy.modflow.ModflowLmt(mf)
    mf.write_input()
    fname = os.path.join(model_ws, "MT3D001.UCN")
    if os.path.isfile(fname):
        os.remove(fname)
    mf.run_model(silent=True)

    modelname_mt = dirname + "_mt"
    mt = flopy.mt3d.Mt3dms(
        modelname=modelname_mt,
        model_ws=model_ws,
        exe_name=exe_name_mt,
        modflowmodel=mf,
    )
    f = open(os.path.join(datadir, "p10cinit.dat"))
    c0 = np.empty((nrow * ncol), dtype=float)
    c0 = readld(f, c0).reshape((nrow, ncol))
    f.close()
    sconc = np.zeros((nlay, nrow, ncol), dtype=float)
    sconc[1] = 0.2 * c0
    sconc[2] = c0
    obs = [
        [3 - 1, 11 - 1, 29 - 1],
        [3 - 1, 19 - 1, 26 - 1],
        [3 - 1, 26 - 1, 23 - 1],
        [3 - 1, 33 - 1, 20 - 1],
        [3 - 1, 40 - 1, 17 - 1],
        [3 - 1, 48 - 1, 14 - 1],
        [3 - 1, 48 - 1, 9 - 1],
        [3 - 1, 52 - 1, 17 - 1],
    ]
    btn = flopy.mt3d.Mt3dBtn(
        mt,
        icbund=1,
        prsity=prsity,
        sconc=sconc,
        timprs=[500, 750, 1000],
        dt0=2.25,
        ttsmult=ttsmult,
        obs=obs,
    )

```

(continues on next page)

(continued from previous page)

```

)
dceps = 1.0e-5
nplane = 0
npl = 0
nph = 16
npmin = 2
npmax = 32
dchmoc = 0.01
nlsink = nplane
npsink = nph
adv = flopy.mt3d.Mt3dAdv(
    mt,
    mixelm=mixelm,
    dceps=dceps,
    nplane=nplane,
    npl=npl,
    nph=nph,
    npmin=npmin,
    npmax=npmax,
    nlsink=nlsink,
    npsink=npsink,
    percel=1.0,
)
dsp = flopy.mt3d.Mt3dDsp(mt, al=al, trpt=trpt, trpv=trpv)
ssm = flopy.mt3d.Mt3dSsm(mt, crch=0.0)
rct = flopy.mt3d.Mt3dRct(
    mt, isothm=isothm, igetsc=0, rhob=1.7, sp1=0.176, sp2=sp2
)
mxiter = 1
if isothm == 4:
    mxiter = 50
gcg = flopy.mt3d.Mt3dGcg(mt, mxiter=mxiter, iter1=500)
mt.write_input()
fname = os.path.join(model_ws, "MT3D001.UCN")
if os.path.isfile(fname):
    os.remove(fname)
mt.run_model(silent=True)

fname = os.path.join(model_ws, "MT3D001.UCN")
ucnobj = flopy.utils.UcnFile(fname)
times = ucnobj.get_times()
conc = ucnobj.get_alldata()

fname = os.path.join(model_ws, "MT3D001.OBS")
if os.path.isfile(fname):
    cvt = mt.load_obs(fname)
else:
    cvt = None

fname = os.path.join(model_ws, "MT3D001.MAS")
mvt = mt.load_mas(fname)

```

(continues on next page)

(continued from previous page)

```
return mf, mt, conc, cvt, mvt
```

```
[31]: mf, mt, conctvd, cvttvd, mvttvd = p10("p10", -1)
mf, mt, conchmoc, cvthmoc, mvthmoc = p10("p10", 3)
mf, mt, concupfd, cvtupfd, mvtupfd = p10("p10", 0, ttsmult=1.0)
grid = mf.modelgrid
```

```
[32]: fig = plt.figure(figsize=(10, 15))
ax = fig.add_subplot(2, 2, 1, aspect="equal")
cinit = mt.btn.sconc[0].array[2]
pmv = flopy.plot.PlotMapView(model=mf)
pmv.plot_grid(color=".5", alpha=0.2)
cs = pmv.contour_array(cinit, levels=np.arange(20, 200, 20))
plt.xlim(5100, 5100 + 28 * 50)
plt.ylim(9100, 9100 + 45 * 50)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("LAYER {} INITIAL CONCENTRATION".format(3))
for k, i, j, q in mf.wel.stress_period_data[0]:
    plt.plot(grid.xcellcenters[i, j], grid.ycellcenters[i, j], "ks")

ax = fig.add_subplot(2, 2, 2, aspect="equal")
c = conctvd[0, 2]
chmoc = conchmoc[0, 2]
pmv = flopy.plot.PlotMapView(model=mf)
pmv.plot_grid(color=".5", alpha=0.2)
cs = pmv.contour_array(c, levels=np.arange(20, 200, 20))
cs = pmv.contour_array(chmoc, linestyle=":", levels=np.arange(20, 200, 20))
plt.xlim(5100, 5100 + 28 * 50)
plt.ylim(9100, 9100 + 45 * 50)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("LAYER {} TIME = 500 DAYS".format(3))
for k, i, j, q in mf.wel.stress_period_data[0]:
    plt.plot(grid.xcellcenters[i, j], grid.ycellcenters[i, j], "ks")

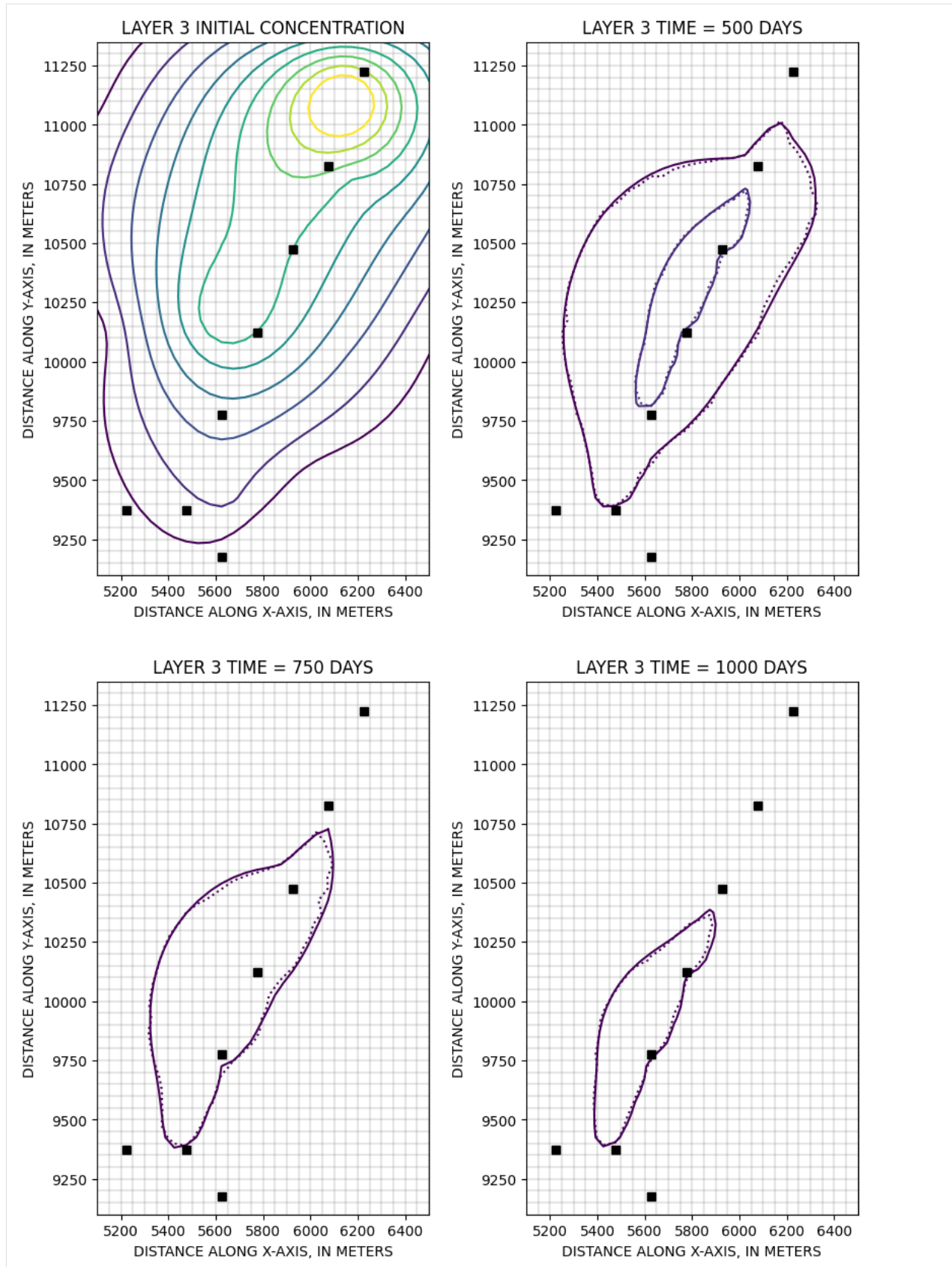
ax = fig.add_subplot(2, 2, 3, aspect="equal")
c = conctvd[1, 2]
chmoc = conchmoc[1, 2]
pmv = flopy.plot.PlotMapView(model=mf)
pmv.plot_grid(color=".5", alpha=0.2)
cs = pmv.contour_array(c, levels=np.arange(20, 200, 20))
cs = pmv.contour_array(chmoc, linestyle=":", levels=np.arange(20, 200, 20))
plt.xlim(5100, 5100 + 28 * 50)
plt.ylim(9100, 9100 + 45 * 50)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("LAYER {} TIME = 750 DAYS".format(3))
for k, i, j, q in mf.wel.stress_period_data[0]:
    plt.plot(grid.xcellcenters[i, j], grid.ycellcenters[i, j], "ks")
```

(continues on next page)

(continued from previous page)

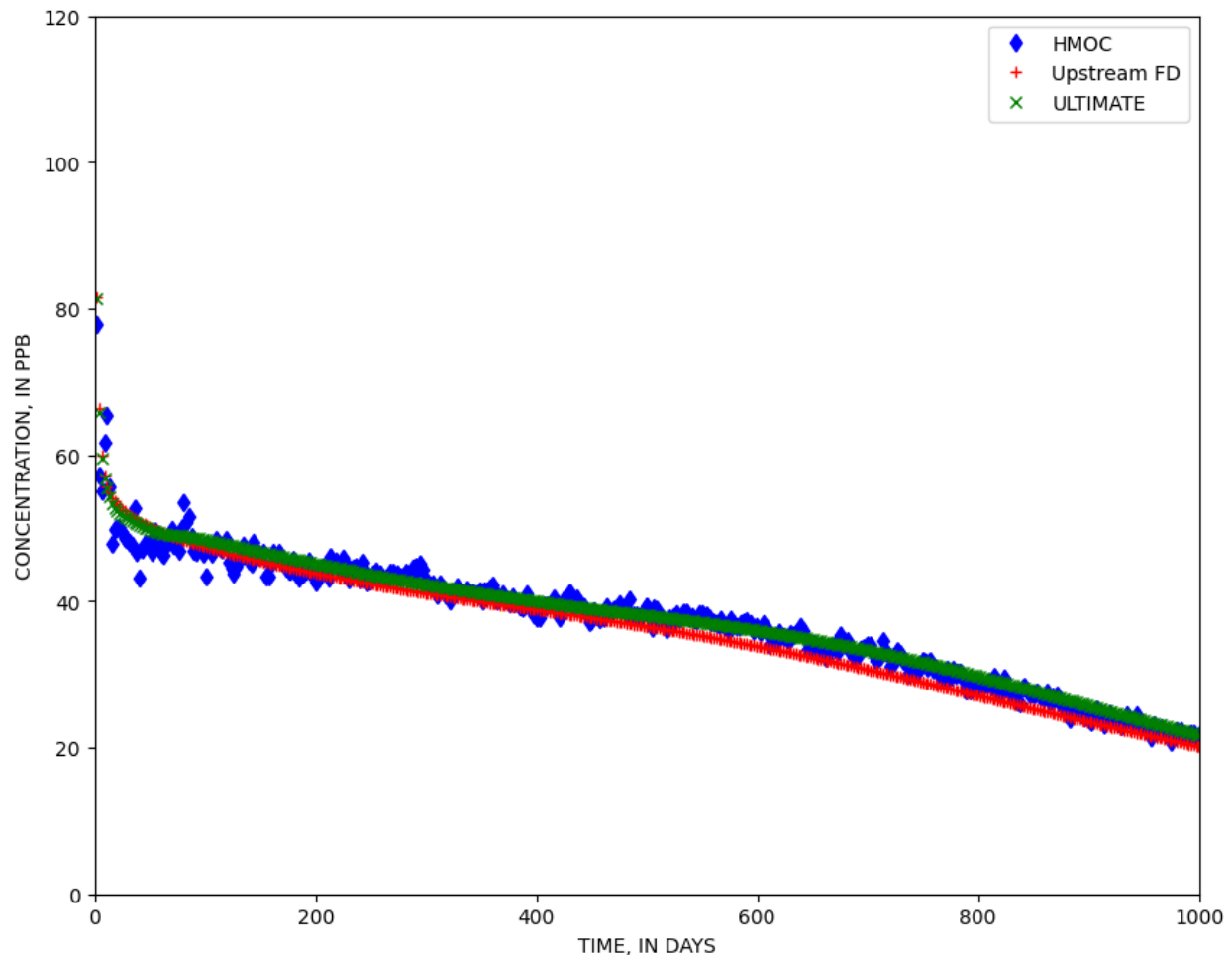
```
ax = fig.add_subplot(2, 2, 4, aspect="equal")
c = conctvd[2, 2]
chmoc = conchmoc[2, 2]
pmv = flopy.plot.PlotMapView(model=mf)
pmv.plot_grid(color=".5", alpha=0.2)
cs = pmv.contour_array(c, levels=np.arange(20, 200, 20))
cs = pmv.contour_array(chmoc, linestyle=":", levels=np.arange(20, 200, 20))
plt.xlim(5100, 5100 + 28 * 50)
plt.ylim(9100, 9100 + 45 * 50)
plt.xlabel("DISTANCE ALONG X-AXIS, IN METERS")
plt.ylabel("DISTANCE ALONG Y-AXIS, IN METERS")
plt.title("LAYER {} TIME = 1000 DAYS".format(3))
for k, i, j, q in mf.wel.stress_period_data[0]:
    plt.plot(grid.xcellcenters[i, j], grid.ycellcenters[i, j], "ks")

# plt.tight_layout()
```

```
[33]: fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(1, 1, 1)
w4 = "(3, 33, 20)"
ax.plot(cvthmoc["time"], cvthmoc[w4], "bd", label="HMOC")
ax.plot(cvtupfd["time"], cvtupfd[w4], "r+", label="Upstream FD")
ax.plot(cvttvd["time"], cvttvd[w4], "gx", label="ULTIMATE")
plt.xlim(0, 1000)
plt.ylim(0, 120)
plt.legend()
plt.xlabel("TIME, IN DAYS")
plt.ylabel("CONCENTRATION, IN PPB")
```

```
[33]: Text(0, 0.5, 'CONCENTRATION, IN PPB')
```



```
[34]: mf, mt, conctvd, cvttvd, mvt0 = p10("p10", 0, perlen=2000, isothm=0)
mf, mt, conctvd, cvttvd, mvt1 = p10("p10", 0, perlen=2000, isothm=1)
mf, mt, conctvd, cvttvd, mvt2 = p10("p10", 0, perlen=2000, isothm=4, sp2=0.1)
mf, mt, conctvd, cvttvd, mvt3 = p10(
    "p10", 0, perlen=2000, isothm=4, sp2=1.5e-4
)
mf, mt, conctvd, cvttvd, mvt4 = p10(
```

(continues on next page)

(continued from previous page)

```

    "p10", 0, perlen=2000, isothm=4, sp2=1.0e-6
)

```

```

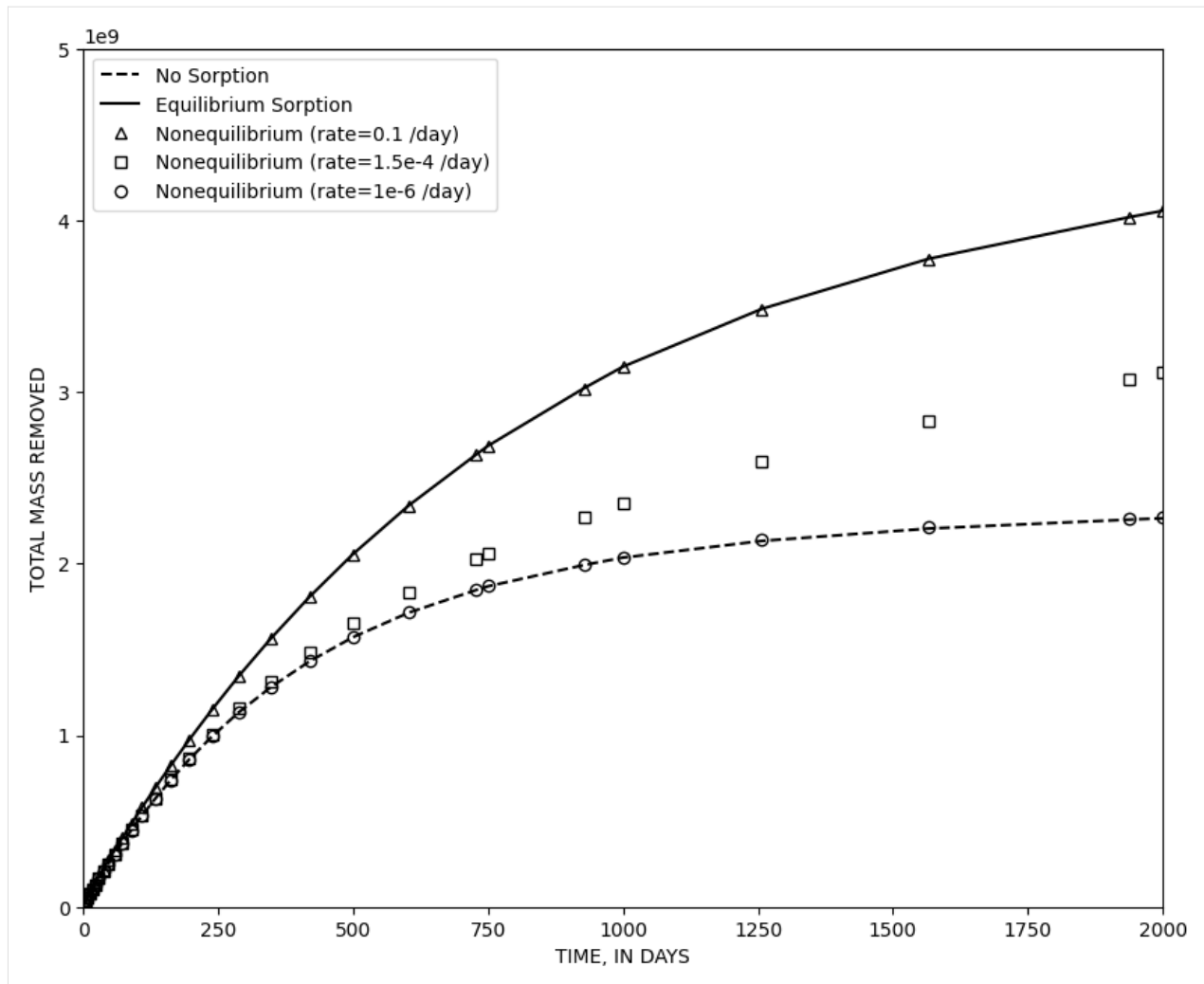
[35]: fig = plt.figure(figsize=(10, 8))
      ax = fig.add_subplot(1, 1, 1)
      ax.plot(mvt0["time"], -mvt0["sinks"], "k--", label="No Sorption")
      ax.plot(mvt1["time"], -mvt1["sinks"], "k-", label="Equilibrium Sorption")
      ax.plot(
          mvt2["time"],
          -mvt2["sinks"],
          "k^",
          fillstyle="none",
          label="Nonequilibrium (rate=0.1 /day)",
      )
      ax.plot(
          mvt3["time"],
          -mvt3["sinks"],
          "ks",
          fillstyle="none",
          label="Nonequilibrium (rate=1.5e-4 /day)",
      )
      ax.plot(
          mvt4["time"],
          -mvt4["sinks"],
          "ko",
          fillstyle="none",
          label="Nonequilibrium (rate=1e-6 /day)",
      )
      plt.xlim(0, 2000)
      plt.ylim(0, 5e9)
      plt.legend(loc=2)
      plt.xlabel("TIME, IN DAYS")
      plt.ylabel("TOTAL MASS REMOVED")

```

```

[35]: Text(0, 0.5, 'TOTAL MASS REMOVED')

```



```
[36]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass
```

6.10 Examples from Bakker and others (2016)

6.10.1 Basic Flopy example

From: Bakker, Mark, Post, Vincent, Langevin, C. D., Hughes, J. D., White, J. T., Starn, J. J. and Fienen, M. N., 2016, Scripting MODFLOW Model Development Using Python and FloPy: Groundwater, v. 54, p. 733–739, <https://doi.org/10.1111/gwat.12413>.

Import the modflow and utils subpackages of FloPy and give them the aliases `fpm` and `fpu`, respectively

```
[1]: import os
import sys
```

(continues on next page)

(continued from previous page)

```

import matplotlib as mpl
import numpy as np

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", "..", ".."))
    sys.path.append(fpth)
    import flopy

import flopy.modflow as fpm
import flopy.utils as fpu

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

```

```

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3

```

Create a MODFLOW model object. Here, the MODFLOW model object is stored in a Python variable called {model}, but this can be an arbitrary name. This object name is important as it will be used as a reference to the model in the remainder of the FloPy script. In addition, a {modelname} is specified when the MODFLOW model object is created. This {modelname} is used for all the files that are created by FloPy for this model.

```

[2]: exe = "mf2005"
ws = os.path.join("temp")
model = fpm.Modflow(modelname="gwexample", exe_name=exe, model_ws=ws)

```

The discretization of the model is specified with the discretization file (DIS) of MODFLOW. The aquifer is divided into 201 cells of length 10 m and width 1 m. The first input of the discretization package is the name of the model object. All other input arguments are self explanatory.

```

[3]: fpm.ModflowDis(
    model, nlay=1, nrow=1, ncol=201, delr=10, delc=1, top=50, botm=0
)

```

```

[3]:
    MODFLOW Discretization Package Class.

    Parameters
    -----
    model : model object
        The model object (of type :class:`flopy.modflow.Modflow`) to which
        this package will be added.

```

(continues on next page)

(continued from previous page)

```

nlay : int
    Number of model layers (the default is 1).
nrow : int
    Number of model rows (the default is 2).
ncol : int
    Number of model columns (the default is 2).
nper : int
    Number of model stress periods (the default is 1).
delr : float or array of floats (ncol), optional
    An array of spacings along a row (the default is 1.0).
delc : float or array of floats (nrow), optional
    An array of spacings along a column (the default is 0.0).
laycbd : int or array of ints (nlay), optional
    An array of flags indicating whether or not a layer has a Quasi-3D
    confining bed below it. 0 indicates no confining bed, and not zero
    indicates a confining bed. LAYCBD for the bottom layer must be 0. (the
    default is 0)
top : float or array of floats (nrow, ncol), optional
    An array of the top elevation of layer 1. For the common situation in
    which the top layer represents a water-table aquifer, it may be
    reasonable to set Top equal to land-surface elevation (the default is
    1.0)
botm : float or array of floats (nlay, nrow, ncol), optional
    An array of the bottom elevation for each model cell (the default is
    0.)
perlen : float or array of floats (nper)
    An array of the stress period lengths.
nstp : int or array of ints (nper)
    Number of time steps in each stress period (default is 1).
tsmult : float or array of floats (nper)
    Time step multiplier (default is 1.0).
steady : bool or array of bool (nper)
    true or False indicating whether or not stress period is steady state
    (default is True).
itmuni : int
    Time units, default is days (4)
lenuni : int
    Length units, default is meters (2)
extension : string
    Filename extension (default is 'dis')
unitnumber : int
    File unit number (default is None).
filenames : str or list of str
    Filenames to use for the package. If filenames=None the package name
    will be created using the model name and package extension. If a
    single string is passed the package will be set to the string.
    Default is None.
xul : float
    x coordinate of upper left corner of the grid, default is None, which
    means xul will be set to zero.
yul : float
    y coordinate of upper-left corner of the grid, default is None, which

```

(continues on next page)

(continued from previous page)

```

means yul will be calculated as the sum of the delc array. This
default, combined with the xul and rotation defaults will place the
lower-left corner of the grid at (0, 0).
rotation : float
    counter-clockwise rotation (in degrees) of the grid about the lower-
    left corner. default is 0.0
crs : pyproj.CRS, optional if `prjfile` is specified
    Coordinate reference system (CRS) for the model grid
    (must be projected; geographic CRS are not supported).
    The value can be anything accepted by
    :meth:`pyproj.CRS.from_user_input() <pyproj.crs.CRS.from_user_input>`,
    such as an authority string (eg "EPSG:26916") or a WKT string.
prjfile : str or pathlike, optional if `crs` is specified
    ESRI-style projection file with well-known text defining the CRS
    for the model grid (must be projected; geographic CRS are not supported).
start_datetime : str
    starting datetime of the simulation. default is '1/1/1970'

```

Attributes

```

heading : str
    Text string written to top of package input file.

```

Methods

See Also

Notes

Examples

```

>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> dis = flopy.modflow.ModflowDis(m)

    _name = DIS
    _parent = MODFLOW 1 layer(s) 1 row(s) 201 column(s) 1 stress period(s) ('flopy.modflow.
    ↪mf.Modflow)
    _totim = None ('NoneType)
    acceptable_dtypes (list, items = 3)
    allowDuplicates = False ('bool)
    botm = <flopy.utils.util_array.Util3d object at 0x7fc4d8d3bbb0> ('flopy.utils.util_
    ↪array.Util3d)
    delc = <flopy.utils.util_array.Util2d object at 0x7fc4d8d3b670> ('flopy.utils.util_
    ↪array.Util2d)
    delr = <flopy.utils.util_array.Util2d object at 0x7fc4d8d3b520> ('flopy.utils.util_
    ↪array.Util2d)
    file_name = gwexample.dis

```

(continues on next page)

(continued from previous page)

```

fn_path = /home/runner/work/flopy/flopy/.docs/Notebooks/temp/gwexample.dis ('str')
itmuni = 4 ('int')
itmuni_dict = {0: 'undefined', 1: 'seconds', 2: 'minutes', 3: 'hours', 4: 'days', 5:
↳ 'years'} ('dict')
laycbd = <flopy.utils.util_array.Util2d object at 0x7fc4d8d3b760> ('flopy.utils.util_
↳ array.Util2d')
lenuni = 2 ('int')
ncol = 201 ('int')
nlay = 1 ('int')
nper = 1 ('int')
nrow = 1 ('int')
nstp = <flopy.utils.util_array.Util2d object at 0x7fc4d8d3b640> ('flopy.utils.util_
↳ array.Util2d')
perlen = <flopy.utils.util_array.Util2d object at 0x7fc4d8d3b6a0> ('flopy.utils.util_
↳ array.Util2d')
start_datetime = 1-1-1970 ('str')
steady = <flopy.utils.util_array.Util2d object at 0x7fc4b6237370> ('flopy.utils.util_
↳ array.Util2d')
top = <flopy.utils.util_array.Util2d object at 0x7fc4d8d3b790> ('flopy.utils.util_array.
↳ Util2d')
tr = <flopy.utils.reference.TemporalReference object at 0x7fc4b6237430> ('flopy.utils.
↳ reference.TemporalReference')
tsmult = <flopy.utils.util_array.Util2d object at 0x7fc4b6237310> ('flopy.utils.util_
↳ array.Util2d')
unit_number = 11

```

Active cells and the like are defined with the Basic package (BAS), which is required for every MODFLOW model. It contains the {ibound} array, which is used to specify which cells are active (value is positive), inactive (value is 0), or fixed head (value is negative). The {numpy} package (aliased as {np}) can be used to quickly initialize the {ibound} array with values of 1, and then set the {ibound} value for the first and last columns to -1. The {numpy} package (and Python, in general) uses zero-based indexing and supports negative indexing so that row 1 and column 1, and row 1 and column 201, can be referenced as [0, 0], and [0, -1], respectively. Although this simulation is for steady flow, starting heads still need to be specified. They are used as the head for fixed-head cells (where {ibound} is negative), and as a starting point to compute the saturated thickness for cases of unconfined flow.

```

[4]: ibound = np.ones((1, 201))
ibound[0, 0] = ibound[0, -1] = -1
fpm.ModflowBas(model, ibound=ibound, strt=20)

```

[4]:

MODFLOW Basic Package Class.

Parameters

model : model object

The model object (of type :class:`flopy.modflow.mf.Modflow`) to which this package will be added.

ibound : array of ints, optional

The ibound array (the default is 1).

strt : array of floats, optional

(continues on next page)

(continued from previous page)

An array of starting heads (the default is 1.0).

`ifrefm` : bool, optional
Indication if data should be read using free format (the default is True).

`ixsec` : bool, optional
Indication of whether model is cross sectional or not (the default is False).

`ichflg` : bool, optional
Flag indicating that flows between constant head cells should be calculated (the default is False).

`stoper` : float
percent discrepancy that is compared to the budget percent discrepancy continue when the solver convergence criteria are not met. Execution will unless the budget percent discrepancy is greater than stoper (default is None). MODFLOW-2005 only

`hnoflo` : float
Head value assigned to inactive cells (default is -999.99).

`extension` : str, optional
File extension (default is 'bas').

`unitnumber` : int, optional
FORTRAN unit number for this package (default is None).

`filenames` : str or list of str
Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package name will be set to the string. Default is None.

Attributes

`heading` : str
Text string written to top of package input file.

`options` : list of str
Can be either or a combination of XSECTION, CHTOCH or FREE.

`ifrefm` : bool
Indicates whether or not packages will be written as free format.

Methods

See Also

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> bas = flopy.modflow.ModflowBas(m)
```

(continues on next page)

(continued from previous page)

```

    _name = BAS6
    _parent = MODFLOW 1 layer(s) 1 row(s) 201 column(s) 1 stress period(s) ('flopy.modflow.
    ↪mf.Modflow)
    acceptable_dtypes (list, items = 3)
    allowDuplicates = False ('bool)
    file_name = gwexample.bas
    fn_path = /home/runner/work/flopy/flopy/.docs/Notebooks/temp/gwexample.bas ('str)
    hnoflo = -999.99 ('float)
    ibound = <flopy.utils.util_array.Util3d object at 0x7fc4d8d044f0> ('flopy.utils.util_
    ↪array.Util3d)
    ichflg = False ('bool)
    ixsec = False ('bool)
    options = ('str)
    stoper = None ('NoneType)
    strt = <flopy.utils.util_array.Util3d object at 0x7fc4d8d47670> ('flopy.utils.util_
    ↪array.Util3d)
    unit_number = 13

```

The hydraulic properties of the aquifer are specified with the Layer Properties Flow (LPF) package (alternatively, the Block Centered Flow (BCF) package may be used). Only the hydraulic conductivity of the aquifer and the layer type ({laytyp}) need to be specified. The latter is set to 1, which means that MODFLOW will calculate the saturated thickness differently depending on whether or not the head is above the top of the aquifer.

```
[5]: fpm.ModflowLpf(model, hk=10, laytyp=1)
```

```
[5]:
```

MODFLOW Layer Property Flow Package Class.

Parameters

model : model object

The model object (of type :class:`flopy.modflow.mf.Modflow`) to which this package will be added.

ipakcb : int

A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0)

hdry : float

Is the head that is assigned to cells that are converted to dry during a simulation. Although this value plays no role in the model calculations, it is useful as an indicator when looking at the resulting heads that are output from the model. HDRY is thus similar to HNOFLO in the Basic Package, which is the value assigned to cells that are no-flow cells at the start of a model simulation. (default is -1.e30).

laytyp : int or array of ints (nlay)

Layer type, contains a flag for each layer that specifies the layer type.

0 confined

>0 convertible

<0 convertible unless the THICKSTRT option is in effect.

(continues on next page)

(continued from previous page)

(default is 0).

layavg : int or array of ints (nlay)
 Layer average
 0 is harmonic mean
 1 is logarithmic mean
 2 is arithmetic mean of saturated thickness and logarithmic mean of hydraulic conductivity
 (default is 0).

chani : float or array of floats (nlay)
 contains a value for each layer that is a flag or the horizontal anisotropy. If CHANI is less than or equal to 0, then variable HANI defines horizontal anisotropy. If CHANI is greater than 0, then CHANI is the horizontal anisotropy for the entire layer, and HANI is not read. If any HANI parameters are used, CHANI for all layers must be less than or equal to 0. Use as many records as needed to enter a value of CHANI for each layer. The horizontal anisotropy is the ratio of the hydraulic conductivity along columns (the Y direction) to the hydraulic conductivity along rows (the X direction).
 (default is 1).

layvka : int or array of ints (nlay)
 a flag for each layer that indicates whether variable VKA is vertical hydraulic conductivity or the ratio of horizontal to vertical hydraulic conductivity.
 0: VKA is vertical hydraulic conductivity
 not 0: VKA is the ratio of horizontal to vertical hydraulic conductivity
 (default is 0).

laywet : int or array of ints (nlay)
 contains a flag for each layer that indicates if wetting is active.
 0 wetting is inactive
 not 0 wetting is active
 (default is 0).

wetfct : float
 is a factor that is included in the calculation of the head that is initially established at a cell when it is converted from dry to wet.
 (default is 0.1).

iwetit : int
 is the iteration interval for attempting to wet cells. Wetting is attempted every IWETIT iteration. If using the PCG solver (Hill, 1990), this applies to outer iterations, not inner iterations. If IWETIT less than or equal to 0, it is changed to 1.
 (default is 1).

ihdwet : int
 is a flag that determines which equation is used to define the initial head at cells that become wet.
 (default is 0)

hk : float or array of floats (nlay, nrow, ncol)
 is the hydraulic conductivity along rows. HK is multiplied by horizontal anisotropy (see CHANI and HANI) to obtain hydraulic conductivity along columns.
 (default is 1.0).

hani : float or array of floats (nlay, nrow, ncol)
 is the ratio of hydraulic conductivity along columns to hydraulic

(continues on next page)

(continued from previous page)

```

conductivity along rows, where HK of item 10 specifies the hydraulic
conductivity along rows. Thus, the hydraulic conductivity along
columns is the product of the values in HK and HANI.
(default is 1.0).
vka : float or array of floats (nlay, nrow, ncol)
is either vertical hydraulic conductivity or the ratio of horizontal
to vertical hydraulic conductivity depending on the value of LAYVKA.
(default is 1.0).
ss : float or array of floats (nlay, nrow, ncol)
is specific storage unless the STORAGECOEFFICIENT option is used.
When STORAGECOEFFICIENT is used, Ss is confined storage coefficient.
(default is 1.e-5).
sy : float or array of floats (nlay, nrow, ncol)
is specific yield.
(default is 0.15).
vkcb : float or array of floats (nlay, nrow, ncol)
is the vertical hydraulic conductivity of a Quasi-three-dimensional
confining bed below a layer. (default is 0.0). Note that if an array
is passed for vkcb it must be of size (nlay, nrow, ncol) even though
the information for the bottom layer is not needed.
wetdry : float or array of floats (nlay, nrow, ncol)
is a combination of the wetting threshold and a flag to indicate
which neighboring cells can cause a cell to become wet.
(default is -0.01).
storagecoefficient : boolean
indicates that variable Ss and SS parameters are read as storage
coefficient rather than specific storage.
(default is False).
constantcv : boolean
indicates that vertical conductance for an unconfined cell is
computed from the cell thickness rather than the saturated thickness.
The CONSTANTCV option automatically invokes the NOCVCORRECTION
option. (default is False).
thickstrt : boolean
indicates that layers having a negative LAYTYP are confined, and their
cell thickness for conductance calculations will be computed as
STRT-BOT rather than TOP-BOT. (default is False).
nocvcorrection : boolean
indicates that vertical conductance is not corrected when the vertical
flow correction is applied. (default is False).
novfc : boolean
turns off the vertical flow correction under dewatered conditions.
This option turns off the vertical flow calculation described on p.
5-8 of USGS Techniques and Methods Report 6-A16 and the vertical
conductance correction described on p. 5-18 of that report.
(default is False).
extension : string
Filename extension (default is 'lpf')
unitnumber : int
File unit number (default is None).
filenames : str or list of str
Filenames to use for the package and the output files. If

```

(continues on next page)

(continued from previous page)

filenames=None the package name will be created using the model name and package extension and the cbc output name will be created using the model name and .cbc extension (for example, modflowtest.cbc), if ipakcbc is a number greater than zero. If a single string is passed the package will be set to the string and cbc output name will be created using the model name and .cbc extension, if ipakcbc is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

add_package : bool

Flag to add the initialised package object to the parent model object. Default is True.

Attributes

Methods

See Also

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lpf = flopy.modflow.ModflowLpf(m)

_name = LPF
_parent = MODFLOW 1 layer(s) 1 row(s) 201 column(s) 1 stress period(s) ('flopy.modflow.
↪mf.Modflow)
acceptable_dtypes (list, items = 3)
allowDuplicates = False ('bool)
chani = <flopy.utils.util_array.Util2d object at 0x7fc4ded637c0> ('flopy.utils.util_
↪array.Util2d)
file_name = gwexample.lpf
fn_path = /home/runner/work/flopy/flopy/.docs/Notebooks/temp/gwexample.lpf ('str)
hani = <flopy.utils.util_array.Util3d object at 0x7fc4d8d4cf10> ('flopy.utils.util_
↪array.Util3d)
hdry = -1e+30 ('float)
hk = <flopy.utils.util_array.Util3d object at 0x7fc4b6226430> ('flopy.utils.util_array.
↪Util3d)
ihdwet = 0 ('int)
ikcflag = 0 ('int)
ipakcb = 0 ('int)
iwetit = 1 ('int)
layavg = <flopy.utils.util_array.Util2d object at 0x7fc4dedddc40> ('flopy.utils.util_
↪array.Util2d)
```

(continues on next page)

(continued from previous page)

```
laytyp = <flopy.utils.util_array.Util2d object at 0x7fc4deddd700> ('flopy.utils.util_
↪array.Util2d)
layvka = <flopy.utils.util_array.Util2d object at 0x7fc4ded63400> ('flopy.utils.util_
↪array.Util2d)
laywet = <flopy.utils.util_array.Util2d object at 0x7fc4b7f76640> ('flopy.utils.util_
↪array.Util2d)
nplpf = 0 ('int)
options = ('str)
ss = <flopy.utils.util_array.Util3d object at 0x7fc4d8d4cc70> ('flopy.utils.util_array.
↪Util3d)
sy = <flopy.utils.util_array.Util3d object at 0x7fc4d8d4c9a0> ('flopy.utils.util_array.
↪Util3d)
unit_number = 15
vka = <flopy.utils.util_array.Util3d object at 0x7fc4d8d4ce50> ('flopy.utils.util_array.
↪Util3d)
vkcb = <flopy.utils.util_array.Util3d object at 0x7fc4d8d4c910> ('flopy.utils.util_
↪array.Util3d)
wetdry = <flopy.utils.util_array.Util3d object at 0x7fc4d8d4c0d0> ('flopy.utils.util_
↪array.Util3d)
wetfct = 0.1 ('float)
```

Aquifer recharge is simulated with the Recharge package (RCH) and the extraction of water at the two ditches is simulated with the Well package (WEL). The latter requires specification of the layer, row, column, and injection rate of the well for each stress period. The layers, rows, columns, and the stress period are numbered (consistent with Python's zero-based numbering convention) starting at 0. The required data are stored in a Python dictionary ({lrcQ} in the code below), which is used in FloPy to store data that can vary by stress period. The {lrcQ} dictionary specifies that two wells (one in cell 1, 1, 51 and one in cell 1, 1, 151), each with a rate of $-1 \text{ m}^3/\text{m/d}$, will be active for the first stress period. Because this is a steady-state model, there is only one stress period and therefore only one entry in the dictionary.

```
[6]: fpm.ModflowRch(model, rech=0.001)
lrcQ = {0: [[0, 0, 50, -1], [0, 0, 150, -1]]}
fpm.ModflowWel(model, stress_period_data=lrcQ)
```

[6]:

MODFLOW Well Package Class.

Parameters

model : model object

The model object (of type :class:`flopy.modflow.mf.Modflow`) to which this package will be added.

ipakcb : int

A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0).

stress_period_data : list of boundaries, or recarray of boundaries, or dictionary of boundaries

Each well is defined through definition of layer (int), row (int), column (int), flux (float).

The simplest form is a dictionary with a lists of boundaries for each

(continues on next page)

(continued from previous page)

stress period, where each list of boundaries itself is a list of boundaries. Indices of the dictionary are the numbers of the stress period. This gives the form of:

```
stress_period_data =
{0: [
    [lay, row, col, flux],
    [lay, row, col, flux],
    [lay, row, col, flux]
],
1: [
    [lay, row, col, flux],
    [lay, row, col, flux],
    [lay, row, col, flux]
], ...
kper:
[
    [lay, row, col, flux],
    [lay, row, col, flux],
    [lay, row, col, flux]
]
}
```

Note that if the number of lists is smaller than the number of stress periods, then the last list of wells will apply until the end of the simulation. Full details of all options to specify `stress_period_data` can be found in the `flopy3` boundaries Notebook in the `basic` subdirectory of the `examples` directory

`dtype` : custom datatype of `stress_period_data`.

If `None` the default well datatype will be applied (default is `None`).

`extension` : string

Filename extension (default is `'wel'`)

`options` : list of strings

Package options (default is `None`).

`unitnumber` : int

File unit number (default is `None`).

`filenames` : str or list of str

Filenames to use for the package and the output files. If `filenames=None` the package name will be created using the model name and package extension and the cbc output name will be created using the model name and `.cbc` extension (for example, `modflowtest.cbc`), if `ipakcbc` is a number greater than zero. If a single string is passed the package will be set to the string and cbc output names will be created using the model name and `.cbc` extension, if `ipakcbc` is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is `None`.

`add_package` : bool

Flag to add the initialised package object to the parent model object. Default is `True`.

Attributes

(continues on next page)

(continued from previous page)

```

-----
mxactw : int
    Maximum number of wells for a stress period. This is calculated
    automatically by FloPy based on the information in
    stress_period_data.

Methods
-----

See Also
-----

Notes
-----
Parameters are not supported in FloPy.

Examples
-----

>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lrcq = {0:[[2, 3, 4, -100.]], 1:[[2, 3, 4, -100.]]}
>>> wel = flopy.modflow.ModflowWel(m, stress_period_data=lrcq)

_name = WEL
_parent = MODFLOW 1 layer(s) 1 row(s) 201 column(s) 1 stress period(s) ('flopy.modflow.
↳mf.Modflow)
acceptable_dtypes (list, items = 3)
allowDuplicates = False ('bool)
dtype = [('k', '<i8'), ('i', '<i8'), ('j', '<i8'), ('flux', '<f4')] ('numpy.dtype[void])
file_name = gwexample.wel
fn_path = /home/runner/work/flopy/flopy/.docs/Notebooks/temp/gwexample.wel ('str)
ipakcb = 0 ('int)
iunitramp = None ('NoneType)
np = 0 ('int)
options (list, items = 0)
phiramp = None ('NoneType)
specify = False ('bool)
stress_period_data = <flopy.utils.util_list.MfList object at 0x7fc4b6237be0> ('flopy.
↳utils.util_list.MfList)
unit_number = 20

```

The Preconditioned Conjugate-Gradient (PCG) solver, using the default settings, is specified to solve the model.

[7]: `fpm.ModflowPcg(model)`

[7]:

```

MODFLOW Pcg Package Class.

Parameters
-----
model : model object

```

(continues on next page)

(continued from previous page)

The model object (of type :class:`flopy.modflow.mf.Modflow`) to which this package will be added.

mxiter : int
maximum number of outer iterations. (default is 50)

iter1 : int
maximum number of inner iterations. (default is 30)

npcond : int
flag used to select the matrix conditioning method. (default is 1).
specify npcond = 1 for Modified Incomplete Cholesky.
specify npcond = 2 for Polynomial.

hclose : float
is the head change criterion for convergence. (default is 1e-5).

rclose : float
is the residual criterion for convergence. (default is 1e-5)

relax : float
is the relaxation parameter used with npcond = 1. (default is 1.0)

nbpol : int
is only used when npcond = 2 to indicate whether the estimate of the upper bound on the maximum eigenvalue is 2.0, or whether the estimate will be calculated. nbpol = 2 is used to specify the value is 2.0; for any other value of nbpol, the estimate is calculated. Convergence is generally insensitive to this parameter. (default is 0).

iprpcg : int
solver print out interval. (default is 0).

mutpcg : int
If mutpcg = 0, tables of maximum head change and residual will be printed each iteration.
If mutpcg = 1, only the total number of iterations will be printed.
If mutpcg = 2, no information will be printed.
If mutpcg = 3, information will only be printed if convergence fails. (default is 3).

damp : float
is the steady-state damping factor. (default is 1.)

damp1 : float
is the transient damping factor. (default is 1.)

ihcofadd : int
is a flag that determines what happens to an active cell that is surrounded by dry cells. (default is 0). If ihcofadd=0, cell converts to dry regardless of HCOF value. This is the default, which is the way PCG2 worked prior to the addition of this option. If ihcofadd<>0, cell converts to dry only if HCOF has no head-dependent stresses or storage terms.

extension : list string
Filename extension (default is 'pcg')

unitnumber : int
File unit number (default is None).

filenames : str or list of str
Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

(continues on next page)

(continued from previous page)

Attributes

Methods

See Also

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> pcg = flopy.modflow.ModflowPcg(m)

_name = PCG
_parent = MODFLOW 1 layer(s) 1 row(s) 201 column(s) 1 stress period(s) ('flopy.modflow.
mf.Modflow)
acceptable_dtypes (list, items = 3)
allowDuplicates = False ('bool)
damp = 1.0 ('float)
damp1 = 1.0 ('float)
file_name = gwexample.pcg
fn_path = /home/runner/work/flopy/flopy/.docs/Notebooks/temp/gwexample.pcg ('str)
hclose = 1e-05 ('float)
ihcofadd = 0 ('int)
iprpcg = 0 ('int)
iter1 = 30 ('int)
mutpcg = 3 ('int)
mxiter = 50 ('int)
nbpol = 0 ('int)
npcond = 1 ('int)
rclose = 1e-05 ('float)
relax = 1.0 ('float)
unit_number = 27
```

The frequency and type of output that MODFLOW writes to an output file is specified with the Output Control (OC) package. In this case, the budget is printed and heads are saved (the default), so no arguments are needed.

[8]: `fpm.ModflowOc(model)`

[8]:

MODFLOW Output Control Package Class.

Parameters

model : model object

The model object (of type :class:`flopy.modflow.mf.Modflow`) to which

(continues on next page)

(continued from previous page)

this package will be added.

ihedfm : int
is a code for the format in which heads will be printed.
(default is 0).

iddnfm : int
is a code for the format in which drawdown will be printed.
(default is 0).

chedfm : string
is a character value that specifies the format for saving heads.
The format must contain 20 characters or less and must be a valid Fortran format that is enclosed in parentheses. The format must be enclosed in apostrophes if it contains one or more blanks or commas. The optional word LABEL after the format is used to indicate that each layer of output should be preceded with a line that defines the output (simulation time, the layer being output, and so forth). If there is no record specifying CHEDFM, then heads are written to a binary (unformatted) file. Binary files are usually more compact than text files, but they are not generally transportable among different computer operating systems or different Fortran compilers.
(default is None)

cddnfm : string
is a character value that specifies the format for saving drawdown.
The format must contain 20 characters or less and must be a valid Fortran format that is enclosed in parentheses. The format must be enclosed in apostrophes if it contains one or more blanks or commas. The optional word LABEL after the format is used to indicate that each layer of output should be preceded with a line that defines the output (simulation time, the layer being output, and so forth). If there is no record specifying CDDNFM, then drawdowns are written to a binary (unformatted) file. Binary files are usually more compact than text files, but they are not generally transportable among different computer operating systems or different Fortran compilers.
(default is None)

cboufm : string
is a character value that specifies the format for saving ibound.
The format must contain 20 characters or less and must be a valid Fortran format that is enclosed in parentheses. The format must be enclosed in apostrophes if it contains one or more blanks or commas. The optional word LABEL after the format is used to indicate that each layer of output should be preceded with a line that defines the output (simulation time, the layer being output, and so forth). If there is no record specifying CBOUFM, then ibounds are written to a binary (unformatted) file. Binary files are usually more compact than text files, but they are not generally transportable among different computer operating systems or different Fortran compilers.
(default is None)

stress_period_data : dictionary of lists
Dictionary key is a tuple with the zero-based period and step (IPEROC, ITSOC) for each print/save option list. If stress_period_data is None, then heads are saved for the last time step of each stress period. (default is None)

(continues on next page)

(continued from previous page)

The list can have any valid MODFLOW OC print/save option:

```
PRINT HEAD
PRINT DRAWDOWN
PRINT BUDGET
SAVE HEAD
SAVE DRAWDOWN
SAVE BUDGET
SAVE IBOUND
```

The lists can also include (1) DDREFERENCE in the list to reset drawdown reference to the period and step and (2) a list of layers for PRINT HEAD, SAVE HEAD, PRINT DRAWDOWN, SAVE DRAWDOWN, and SAVE IBOUND.

`stress_period_data = {(0,1):['save head']}` would save the head for the second timestep in the first stress period.

`compact` : boolean

Save results in compact budget form. (default is True).

`extension` : list of strings

(default is ['oc', 'hds', 'ddn', 'cbc', 'ibo']).

`unitnumber` : list of ints

(default is [14, 51, 52, 53, 0]).

`filenames` : str or list of str

Filenames to use for the package and the head, drawdown, budget (not used), and ibound output files. If filenames=None the package name will be created using the model name and package extension and the output file names will be created using the model name and extensions. If a single string is passed the package will be set to the string and output names will be created using the model name and head, drawdown, budget, and ibound extensions. To define the names for all package files (input and output) the length of the list of strings should be 5. Default is None.

Attributes

Methods

See Also

Notes

The "words" method for specifying output control is the only option available. Also, the "compact" budget should normally be used as it produces files that are typically much smaller. The compact budget form is also a requirement for using the MODPATH particle tracking program.

Examples

(continues on next page)

(continued from previous page)

```

>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> spd = {(0, 0): ['print head'],
...        (0, 1): [],
...        (0, 249): ['print head'],
...        (0, 250): [],
...        (0, 499): ['print head', 'save ibound'],
...        (0, 500): [],
...        (0, 749): ['print head', 'ddreference'],
...        (0, 750): [],
...        (0, 999): ['print head']}
>>> oc = flopy.modflow.ModflowOc(m, stress_period_data=spd, cboufm='(20i5)')

    _name = OC
    _parent = MODFLOW 1 layer(s) 1 row(s) 201 column(s) 1 stress period(s) ('flopy.modflow.
    mf.Modflow)
    acceptable_dtypes (list, items = 3)
    allowDuplicates = False ('bool')
    cboufm = None ('NoneType')
    cddnfm = None ('NoneType')
    chedfm = None ('NoneType')
    compact = True ('bool')
    file_name = gwexample.oc
    fn_path = /home/runner/work/flopy/flopy/.docs/Notebooks/temp/gwexample.oc ('str')
    ibouun = 0 ('int')
    iddnfm = 0 ('int')
    ihedfm = 0 ('int')
    iubud = 0 ('int')
    iuddn = 0 ('int')
    iuhead = 51 ('int')
    iuibnd = 0 ('int')
    label = LABEL ('str')
    savebud = False ('bool')
    saveddn = False ('bool')
    savehead = True ('bool')
    saveibnd = False ('bool')
    stress_period_data = {(0, 0): ['save head']} ('dict')
    unit_number = 14

```

Finally the MODFLOW input files are written (eight files for this model) and the model is run. This requires, of course, that MODFLOW is installed on your computer and FloPy can find the executable in your path.

```

[9]: model.write_input()
    success, buff = model.run_model(silent=True, report=True)
    if success:
        for line in buff:
            print(line)
    else:
        raise ValueError("Failed to run.")

```

```

MODFLOW-2005
U.S. GEOLOGICAL SURVEY MODULAR FINITE-DIFFERENCE GROUND-WATER FLOW MODEL
Version 1.12.00 2/3/2017

```

```

Using NAME file: gwexample.nam
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:42:49

```

```

Solving: Stress period:      1    Time step:      1    Ground-Water Flow Eqn.
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:42:49
Elapsed run time: 0.001 Seconds

```

```

Normal termination of simulation

```

After MODFLOW has responded with the positive {Normal termination of simulation}, the calculated heads can be read from the binary output file. First a file object is created. As the modelname used for all MODFLOW files was specified as {gwexample} in step 1, the file with the heads is called {gwexample.hds}. FloPy includes functions to read data from the file object, including heads for specified layers or time steps, or head time series at individual cells. For this simple mode, all computed heads are read.

```

[10]: fpth = os.path.join(ws, "gwexample.hds")
      hfile = fpu.HeadFile(fpth)
      h = hfile.get_data(totim=1.0)

```

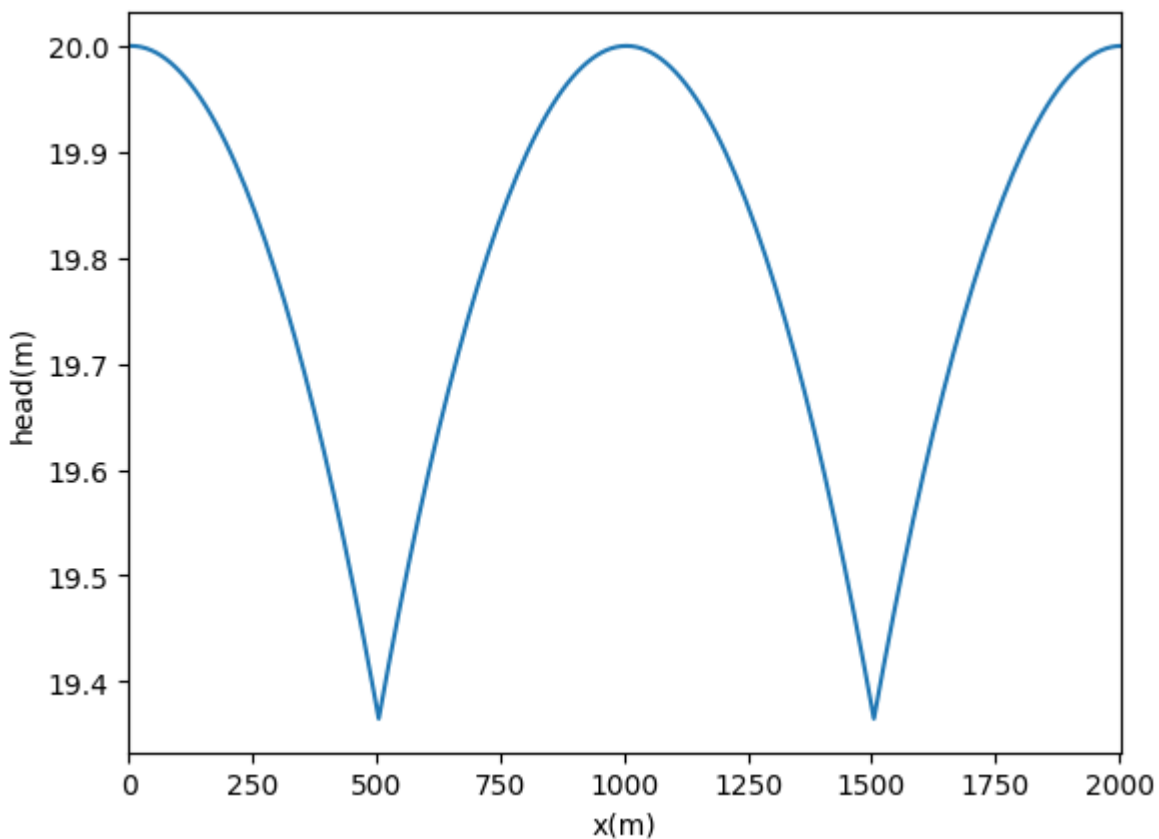
The heads are now stored in the Python variable {h}. FloPy includes powerful plotting functions to plot the grid, boundary conditions, head, etc. This functionality is demonstrated later. For this simple one-dimensional example, a plot is created with the matplotlib package

```

[11]: import matplotlib.pyplot as plt

      ax = plt.subplot(111)
      x = model.modelgrid.xcellcenters[0]
      ax.plot(x, h[0, 0, :])
      ax.set_xlim(0, x.max())
      ax.set_xlabel("x(m)")
      ax.set_ylabel("head(m)")
      plt.show()

```



6.10.2 Capture fraction example

From: Bakker, Mark, Post, Vincent, Langevin, C. D., Hughes, J. D., White, J. T., Starn, J. J. and Fienen, M. N., 2016, Scripting MODFLOW Model Development Using Python and FloPy: Groundwater, v. 54, p. 733-739, <https://doi.org/10.1111/gwat.12413>.

```
[1]: import os
import sys

import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import scipy.ndimage

try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", "..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
```

(continues on next page)

(continued from previous page)

```
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))
```

```
3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[2]: ws = os.path.join("temp")
     if not os.path.exists(ws):
         os.makedirs(ws)
```

```
[3]: fn = os.path.join(
     ..,
     "groundwater_paper",
     "uspb",
     "results",
     "USPB_capture_fraction_04_01.dat",
 )
     cf = np.loadtxt(fn)
     print(cf.shape)

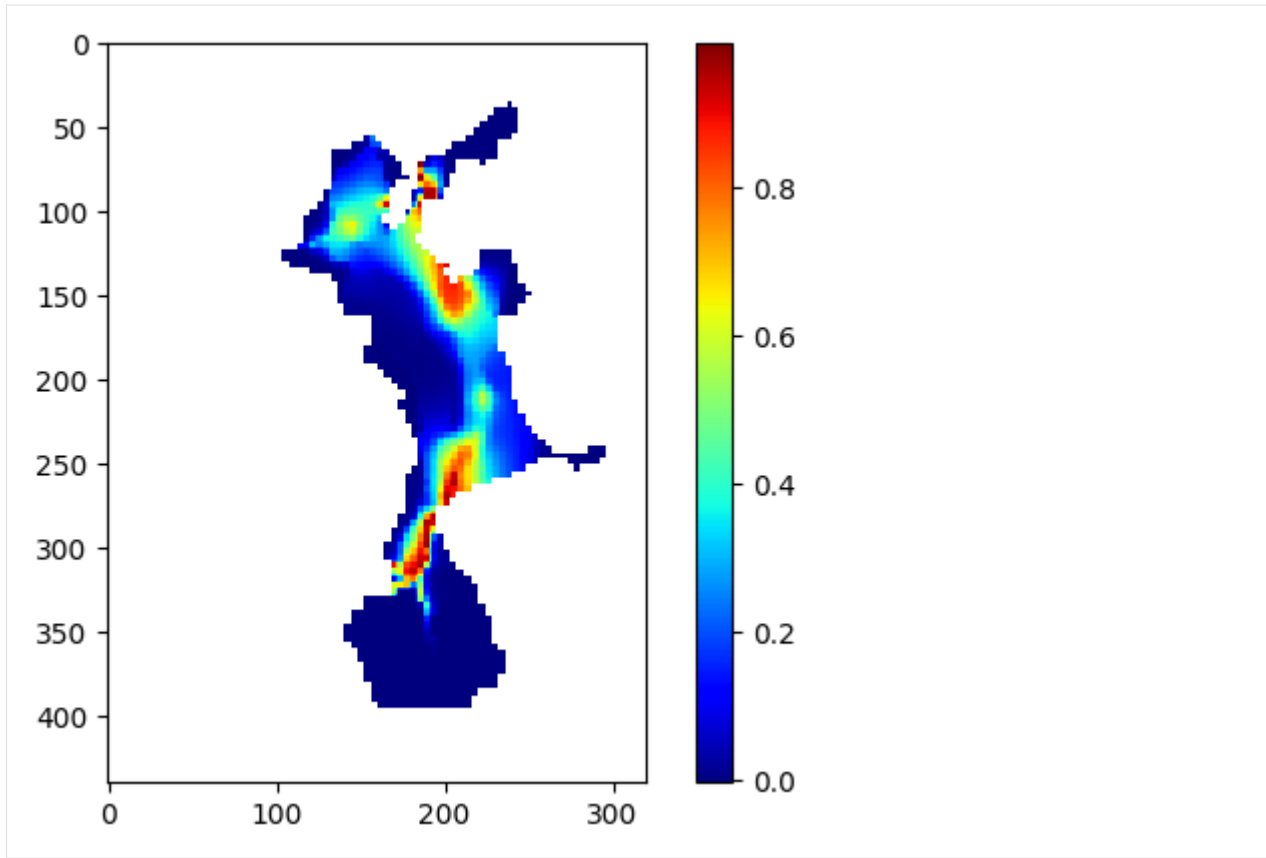
(110, 80)
```

```
[4]: cf2 = scipy.ndimage.zoom(cf, 4, order=0)
     print(cf2.shape)

(440, 320)
```

```
[5]: c = plt.imshow(cf2, cmap="jet")
     plt.colorbar(c)
```

```
[5]: <matplotlib.colorbar.Colorbar at 0x7f384e7d7880>
```

```
[6]: wsl = os.path.join(".", "groundwater_paper", "uspb", "flopy")
ml = flopy.modflow.Modflow.load("DG.nam", model_ws=wsl, verbose=False)

[7]: nlay, nrow, ncol = ml.nlay, ml.dis.nrow, ml.dis.ncol
xmax, ymax = ncol * 250.0, nrow * 250.0

[8]: plt.rcParams.update({"font.size": 6})
fig = plt.figure(figsize=(3.25, 4.47))
ax1 = plt.gca()
ax1.set_aspect("equal")
mm1 = flopy.plot.PlotMapView(model=ml, layer=4)
plt.xlim(0, xmax)
plt.ylim(0, ymax)
mm1.plot_inactive(color_noflow="0.75")
c = plt.imshow(cf2, cmap="jet", extent=[0, ncol * 250.0, 0, nrow * 250.0])
cb = plt.colorbar(c, shrink=0.5)
cb.ax.set_ylabel("Layer 4 capture fraction")
mm1.plot_bc(ftype="STR", plotAll=True)
plt.plot(
    [-10000],
    [-10000],
    marker="s",
    ms=10,
    lw=0.0,
    mec="0.2",
```

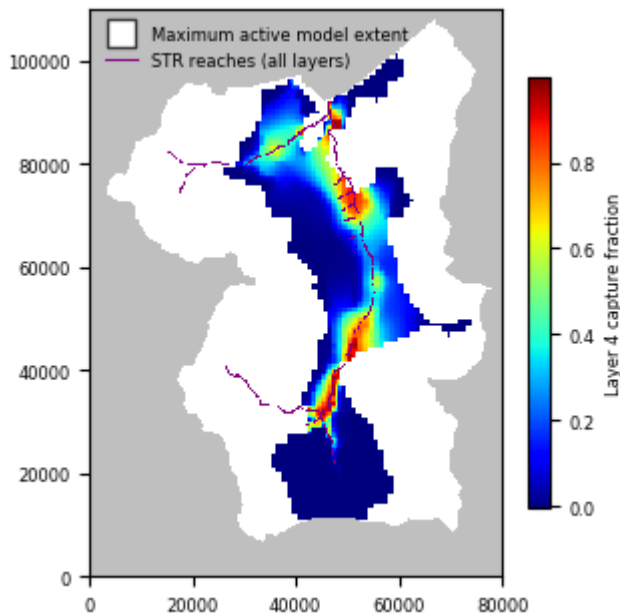
(continues on next page)

(continued from previous page)

```

    mfc="white",
    label="Maximum active model extent",
)
plt.plot(
    [-10000, 0],
    [-10000, 0],
    color="purple",
    lw=0.75,
    label="STR reaches (all layers)",
)
leg = plt.legend(loc="upper left", numpoints=1, prop={"size": 6})
leg.draw_frame(False)
plt.xticks([0, 20000, 40000, 60000, 80000])
plt.tight_layout()
plt.savefig(os.path.join(ws, "capture_fraction_010y.png"), dpi=300)

```



Rerun the model after changing workspace and writing input files

```

[9]: ml.change_model_ws(ws)
    ml.exe_name = "mf2005dbl"
    ml.write_input()
    ml.run_model(silent=True)

[9]: (True, [])

[10]: hedObj = flopy.utils.HeadFile(os.path.join(ws, "DG.hds"), precision="double")
    h = hedObj.get_data(kstpker=(0, 0))
    cbcObj = flopy.utils.CellBudgetFile(
        os.path.join(ws, "DG.cbc"), precision="double"
    )

    frf = cbcObj.get_data(kstpker=(0, 0), text="FLOW RIGHT FACE")[0]

```

(continues on next page)

(continued from previous page)

```

fff = cbcObj.get_data(kstpkper=(0, 0), text="FLOW FRONT FACE")[0]
qx, qy, qz = flopy.utils.postprocessing.get_specific_discharge(
    (frf, fff, None), ml
)

```

```

[11]: cnt = np.arange(1200, 1700, 100)
f, (ax1, ax2) = plt.subplots(
    1, 2, figsize=(6.75, 4.47), constrained_layout=True
)
ax1.set_xlim(0, xmax)
ax1.set_ylim(0, ymax)
ax2.set_xlim(0, xmax)
ax2.set_ylim(0, ymax)
ax1.set_aspect("equal")
ax2.set_aspect("equal")

mm1 = flopy.plot.PlotMapView(model=ml, ax=ax1, layer=3)
h1 = mm1.plot_array(h, masked_values=[-888, -999], vmin=1100, vmax=1700)
mm1.plot_inactive(color_noflow="0.75")
mm1.plot_bc(ftype="STR")
q1 = mm1.plot_vector(
    qx,
    qy,
    istep=5,
    jstep=5,
    normalize=True,
    color="0.4",
    scale=70,
    headwidth=3,
    headlength=3,
    headaxislength=3,
)
c1 = mm1.contour_array(
    h, masked_values=[-888, -999], colors="black", levels=cnt, linewidths=0.5
)
ax1.clabel(c1, fmt="%.0f", inline_spacing=0.5)

mm2 = flopy.plot.PlotMapView(model=ml, ax=ax2, layer=4)
h2 = mm2.plot_array(h, masked_values=[-888, -999], vmin=1100, vmax=1700)
mm2.plot_inactive(color_noflow="0.75")
mm2.plot_bc(ftype="STR")
q2 = mm2.plot_vector(
    qx,
    qy,
    istep=5,
    jstep=5,
    normalize=True,
    color="0.4",
    scale=70,
    headwidth=3,
    headlength=3,
    headaxislength=3,
)

```

(continues on next page)

(continued from previous page)

```

)
c2 = mm2.contour_array(
    h, masked_values=[-888, -999], colors="black", levels=cnt, linewidths=0.5
)
ax2.clabel(c2, fmt="%.0f", inline_spacing=0.5)

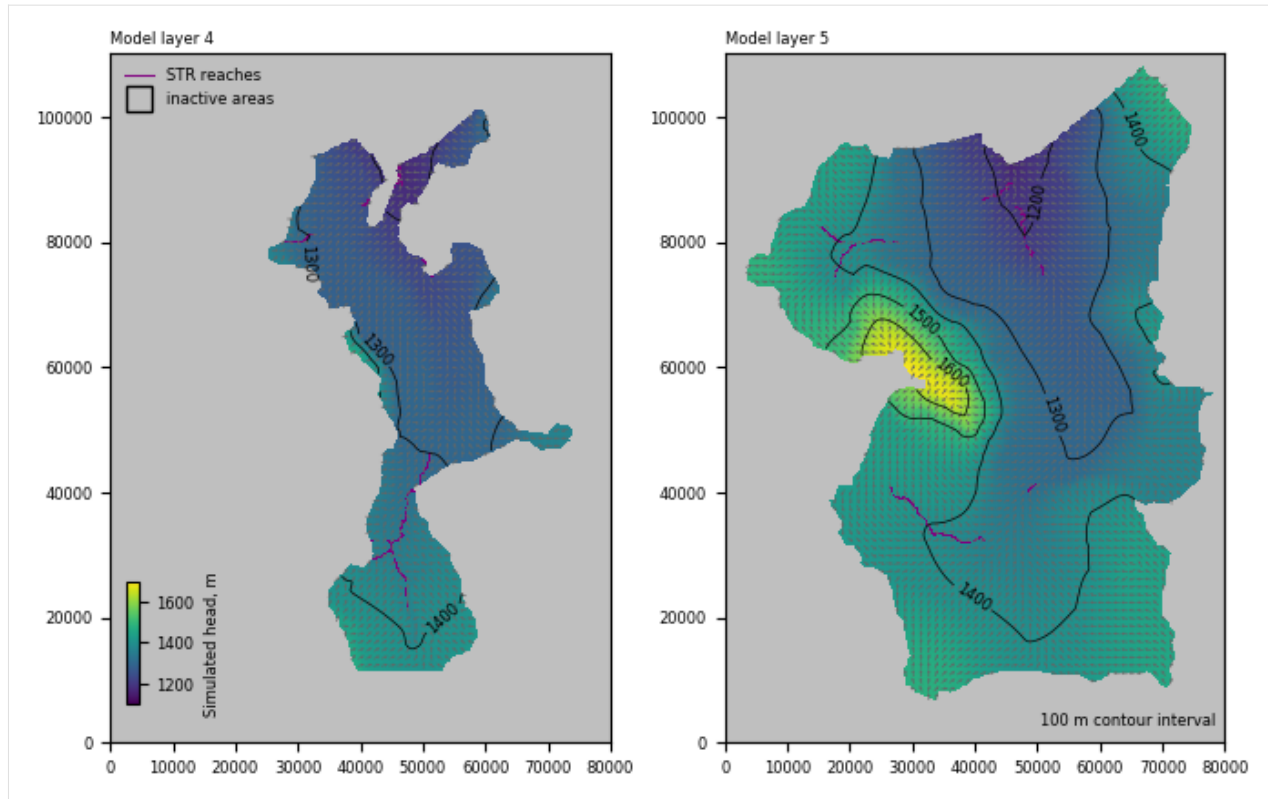
ax3 = f.add_axes([0.08, 0.125, 0.01, 0.15])
cb = plt.colorbar(h2, cax=ax3)
cb.ax.set_ylabel("Simulated head, m")

ax1.plot(
    [-10000, 0], [-10000, 0], color="purple", lw=0.75, label="STR reaches"
)
ax1.plot(
    [-10000],
    [-10000],
    marker="s",
    ms=10,
    lw=0.0,
    mec="black",
    mfc="None",
    label="inactive areas",
)
leg = ax1.legend(loc="upper left", numpoints=1, prop={"size": 6})
leg.draw_frame(False)

ax1.text(
    0.0, 1.01, "Model layer 4", ha="left", va="bottom", transform=ax1.transAxes
)
ax2.text(
    0.98,
    0.02,
    "100 m contour interval",
    ha="right",
    va="bottom",
    transform=ax2.transAxes,
)
ax2.text(
    0.0, 1.01, "Model layer 5", ha="left", va="bottom", transform=ax2.transAxes
)

plt.savefig(os.path.join(ws, "uspb_heads.png"), dpi=300)

```



```
[12]: fn = os.path.join(
    "..",
    "groundwater_paper",
    "uspb",
    "results",
    "USPB_capture_fraction_04_10.dat",
)
cf = np.loadtxt(fn)
cf2 = scipy.ndimage.zoom(cf, 4, order=0)
```

```
[13]: fig = plt.figure(figsize=(3.25, 4.47), constrained_layout=True)
ax1 = plt.gca()
ax1.set_aspect("equal")
mm1 = flopy.plot.PlotMapView(model=m1, layer=4)
plt.xlim(0, xmax)
plt.ylim(0, ymax)
mm1.plot_inactive(color_noflow="0.75")
c = plt.imshow(cf2, cmap="jet", extent=[0, ncol * 250.0, 0, nrow * 250.0])
cb = plt.colorbar(c, shrink=0.5)
cb.ax.set_ylabel("Layer 4 capture fraction")
mm1.plot_bc(ftype="STR", plotAll=True)
plt.plot(
    [-10000, 0],
    [-10000, 0],
    color="purple",
    lw=0.75,
    label="STR reaches (all layers)",
```

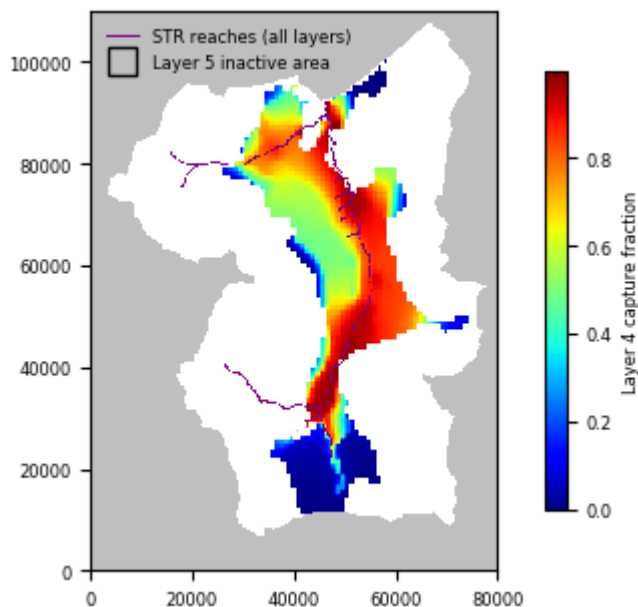
(continues on next page)

(continued from previous page)

```

)
plt.plot(
    [-10000],
    [-10000],
    marker="s",
    ms=10,
    lw=0.0,
    mec="black",
    mfc="None",
    label="Layer 5 inactive area",
)
leg = plt.legend(loc="upper left", numpoints=1, prop={"size": 6})
leg.draw_frame(False)
plt.xticks([0, 20000, 40000, 60000, 80000])
plt.savefig(os.path.join(ws, "capture_fraction_100y.png"), dpi=300)

```



6.11 Examples from Hughes and others (2023)

6.11.1 Watershed geoprocessing example

From Hughes, Joseph D., Langevin, Christian D., Paulinski, Scott R., Larsen, Joshua D., and Brakenhoff, David, 2023, FloPy Workflows for Creating Structured and Unstructured MODFLOW Models: Groundwater, <https://doi.org/10.1111/gwat.13327>

```

[1]: import os
import pathlib as pl
import sys

```

```
[2]: import matplotlib as mpl
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
import numpy as np
from shapely.geometry import LineString
```

```
[3]: import flopy
import flopy.plot.styles as styles
from flopy.discretization import StructuredGrid, VertexGrid
from flopy.utils.gridgen import Gridgen
from flopy.utils.triangle import Triangle
from flopy.utils.voronoi import VoronoiGrid
```

```
[4]: print(sys.version)
print("numpy version: {}".format(np.__version__))
print("matplotlib version: {}".format(mpl.__version__))
print("flopy version: {}".format(flopy.__version__))

3.8.18 (default, Aug 28 2023, 08:27:22)
[GCC 11.4.0]
numpy version: 1.24.4
matplotlib version: 3.7.3
flopy version: 3.4.3
```

```
[5]: # import all plot style information from defaults.py
sys.path.append("../common")
from groundwater2023_utils import (
    densify_geometry,
    geometries,
    set_idomain,
    string2geom,
)
```

```
[6]: # basic figure size
figwidth = 180 # 90 # mm
figwidth = figwidth / 10 / 2.54 # inches
figheight = figwidth
figsize = (figwidth, figheight)
```

```
[7]: # figure settings
levels = np.arange(10, 110, 10)
contour_color = "black"
contour_style = "--"
contour_dict = {
    "levels": levels,
    "linewidths": 0.5,
    "colors": contour_color,
    "linestyles": contour_style,
}
clabel_dict = {
    "inline": True,
    "fmt": "%1.0f",
```

(continues on next page)

(continued from previous page)

```

    "fontsize": 6,
    "inline_spacing": 0.5,
}
font_dict = {
    "fontsize": 5,
    "color": "black",
}
grid_dict = {
    "lw": 0.25,
    "color": "0.5",
}
refinement_dict = {
    "color": "magenta",
    "ls": ":",
    "lw": 1.0,
}
river_dict = {
    "color": "blue",
    "linestyle": "-",
    "linewidth": 1,
}
intersection_cmap = "Pastel2"
temp_cmap = mpl.colormaps[intersection_cmap]
intersection_rgba = temp_cmap(0)

```

```

[8]: # make a temporary working directory for gridgen output
temp_path = "./temp"
if not os.path.isdir(temp_path):
    os.mkdir(temp_path)

```

```

[9]: # Load the fine topography that will be sampled
ascii_file = pl.Path("../examples/data/geospatial/fine_topo.asc")
fine_topo = flopy.utils.Raster.load(ascii_file)

```

Define the problem size and extents

```

[10]: Lx = 180000
Ly = 100000
extent = (0, Lx, 0, Ly)
vmin, vmax = 0.0, 100.0

```

Create boundary and river data that will be used to refine the unstructured grids

```

[11]: boundary_polygon = string2geom geometries["boundary"])
bp = np.array(boundary_polygon)

sgs = [string2geom(geometries[f"streamseg{i}"]) for i in range(1, 5)]

fig = plt.figure(figsize=figsize)
ax = fig.add_subplot()
ax.set_aspect("equal")

```

(continues on next page)

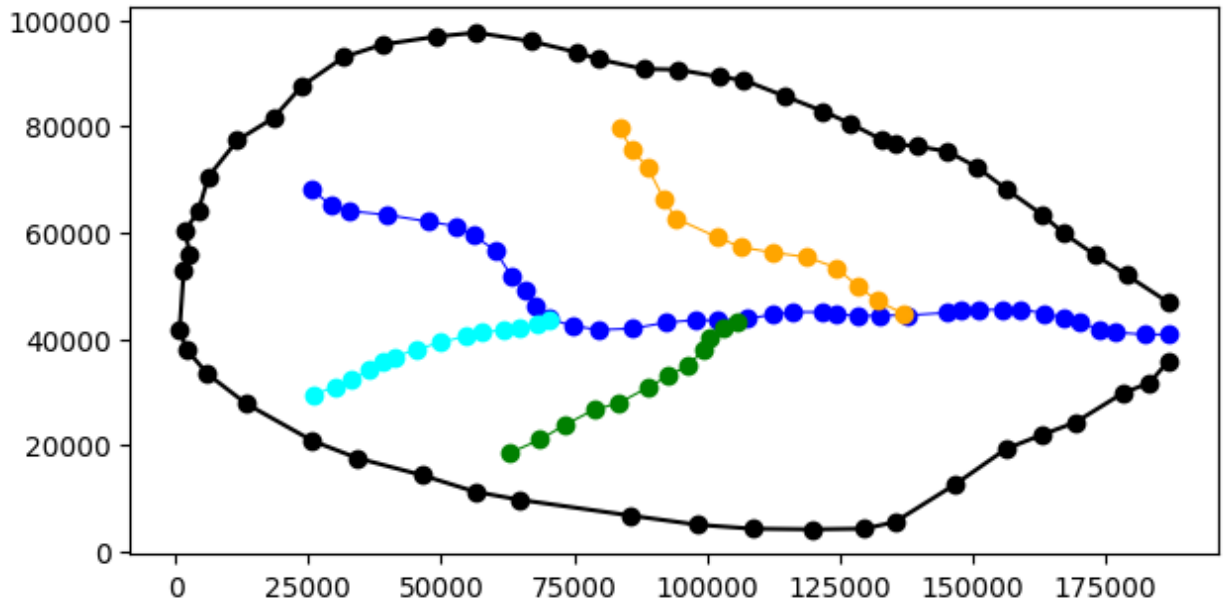
(continued from previous page)

```

riv_colors = ("blue", "cyan", "green", "orange", "red")

ax.plot(bp[:, 0], bp[:, 1], "ko-")
for idx, sg in enumerate(sgs):
    sa = np.array(sg)
    ax.plot(sa[:, 0], sa[:, 1], color=riv_colors[idx], lw=0.75, marker="o")

```



Structured Grids

Structured grid with constant row and column spacing

```

[12]: dx = dy = 1666.66666667
      nlay = 1
      nrow = int(Ly / dy) + 1
      ncol = int(Lx / dx) + 1
      delr = np.array(ncol * [dx])
      delc = np.array(nrow * [dy])
      top = np.ones((nrow, ncol)) * 1000.0
      botm = np.ones((nlay, nrow, ncol)) * -100.0
      struct_grid = StructuredGrid(
          nlay=nlay, delr=delr, delc=delc, xoff=0.0, yoff=0.0, top=top, botm=botm
      )

      set_idomain(struct_grid, boundary_polygon)

```

```

[13]: top_sg = fine_topo.resample_to_grid(
      struct_grid,
      band=fine_topo.bands[0],
      method="linear",

```

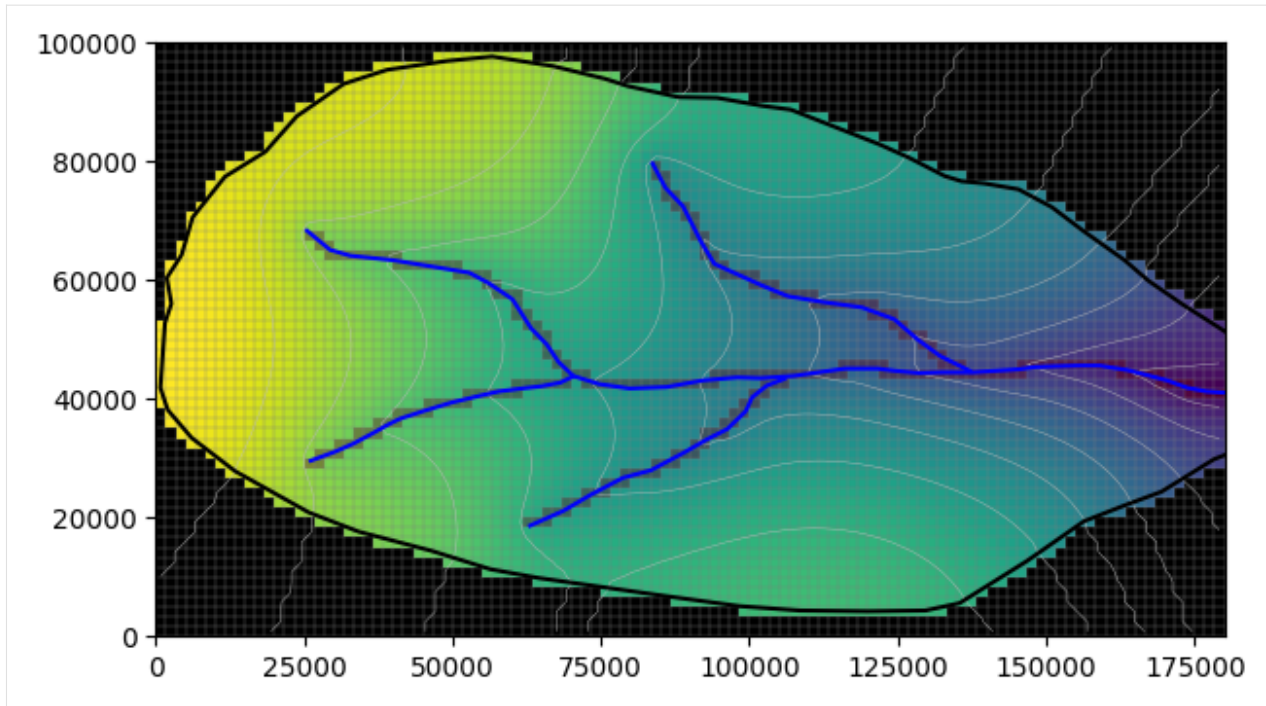
(continues on next page)

(continued from previous page)

```
    extrapolate_edges=True,  
)
```

```
[14]: ix = flopy.utils.GridIntersect(struct_grid, method="structured")  
      cellids = []  
      for sg in sgs:  
          v = ix.intersect(LineString(sg), sort_by_cellid=True)  
          cellids += v["cellids"].tolist()  
      intersection_sg = np.zeros(struct_grid.shape[1:])  
      for loc in cellids:  
          intersection_sg[loc] = 1
```

```
[15]: fig = plt.figure(figsize=figsize)  
      ax = fig.add_subplot()  
      pmv = flopy.plot.PlotMapView(modelgrid=struct_grid)  
      ax.set_aspect("equal")  
      pmv.plot_array(top_sg)  
      pmv.plot_array(  
          intersection_sg,  
          masked_values=[  
              0,  
          ],  
          alpha=0.2,  
          cmap="Reds_r",  
      )  
      pmv.plot_grid(lw=0.25, color="0.5")  
      cg = pmv.contour_array(top_sg, levels=levels, linewidths=0.3, colors="0.75")  
      pmv.plot_inactive()  
  
      ax.plot(bp[:, 0], bp[:, 1], "k-")  
      for sg in sgs:  
          sa = np.array(sg)  
          ax.plot(sa[:, 0], sa[:, 1], "b-")
```



Structured grid with variable row and column spacing

```
[16]: dx = dy = 5000

# create transition cells
multiplier = 1.175
transition = 20000.0
ncells = 7
smoothr = [
    transition * (multiplier - 1.0) / (multiplier ** float(ncells) - 1.0)
]
for i in range(ncells - 1):
    smoothr.append(smoothr[i] * multiplier)
smooth = smoothr.copy()
smooth.reverse()
```

```
[17]: # build the structured grid with variable row and column spacing
dx = 9 * [5000] + smooth + 15 * [1666.66666667] + smoothr + 14 * [5000]
dy = 4 * [5000] + smooth + 12 * [1666.66666667] + smoothr + 4 * [5000]

nlay = 1
ncol = len(dx)
nrow = len(dy)

top = np.ones((nrow, ncol)) * 1000.0
botm = np.ones((nlay, nrow, ncol)) * -100.0

delr = np.array(dx)
```

(continues on next page)

(continued from previous page)

```

delc = np.array(dy)
struct_vrc_grid = StructuredGrid(
    nlay=nlay,
    delr=delr,
    delc=delc,
    xoff=0.0,
    yoff=0.0,
    top=top,
    botm=botm,
)
set_idomain(struct_vrc_grid, boundary_polygon)

```

```

[18]: top_sg_vrc = fine_topo.resample_to_grid(
    struct_vrc_grid,
    band=fine_topo.bands[0],
    method="linear",
    extrapolate_edges=True,
)

```

```

[19]: ix = flopy.utils.GridIntersect(struct_vrc_grid, method="structured")
cellids = []
for sg in sgs:
    v = ix.intersect(LineString(sg), sort_by_cellid=True)
    cellids += v["cellids"].tolist()
intersection_sg_vrc = np.zeros(struct_vrc_grid.shape[1:])
for loc in cellids:
    intersection_sg_vrc[loc] = 1

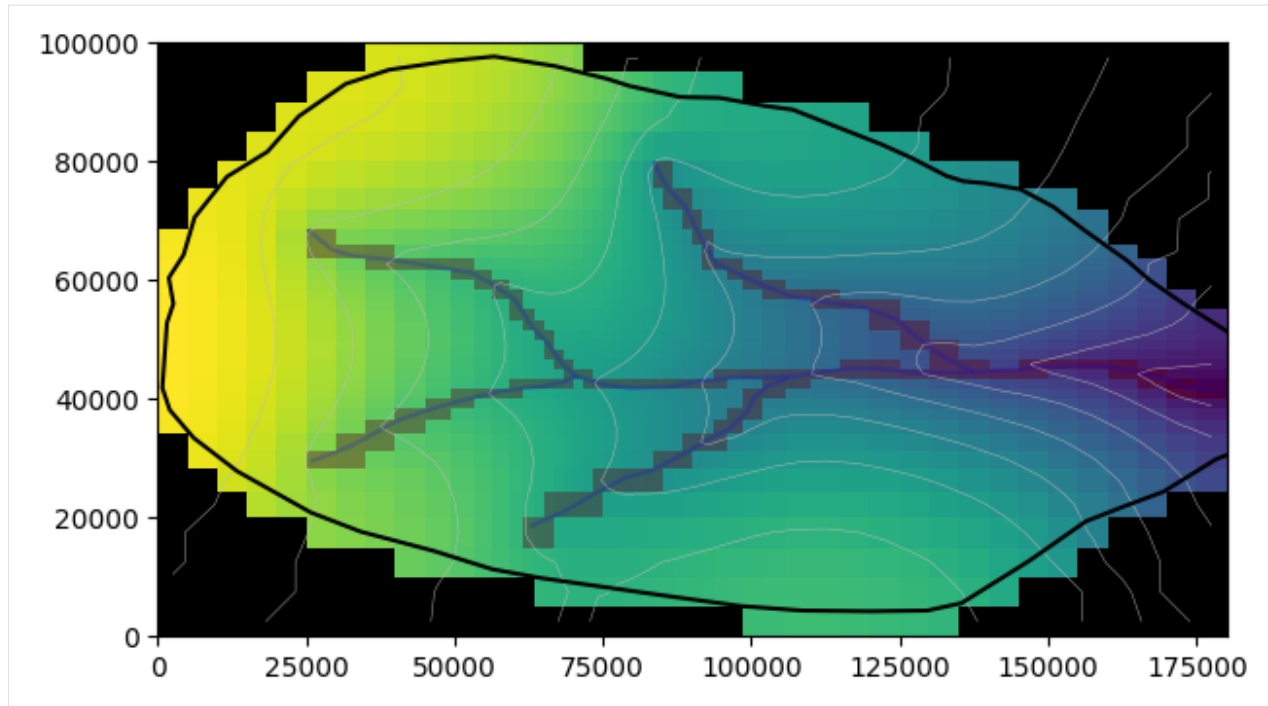
```

```

[20]: fig = plt.figure(figsize=figsize)
ax = fig.add_subplot()
pmv = flopy.plot.PlotMapView(modelgrid=struct_vrc_grid)
ax.set_aspect("equal")
pmv.plot_array(top_sg_vrc)
pmv.plot_array(
    intersection_sg_vrc,
    masked_values=[
        0,
    ],
    alpha=0.2,
    cmap="Reds_r",
)
cg = pmv.contour_array(
    top_sg_vrc, levels=levels, linewidths=0.3, colors="0.75"
)
pmv.plot_inactive()

ax.plot(bp[:, 0], bp[:, 1], "k-")
for sg in sgs:
    sa = np.array(sg)
    ax.plot(sa[:, 0], sa[:, 1], "b-", alpha=0.2)

```



Local grid refinement grid

```
[21]: from flopy.utils.lgrutil import Lgr

nlayp = 1
dx = 5000
nrowp = int(Ly / dx)
ncolp = int(Lx / dx)
delrp = dx
delcp = dx
# topp = 1000.0
# botmp = [-100.]
topp = np.ones((nrowp, ncolp)) * 1000.0
botmp = np.ones((nlayp, nrowp, ncolp)) * -100.0

idomainp = np.ones((nlayp, nrowp, ncolp), dtype=int)
idomainp[0, 8:12, 13:18] = 0

ncpp = 3
ncpp1 = [1]

lgr = Lgr(
    nlayp,
    nrowp,
    ncolp,
    delrp,
    delcp,
    topp,
```

(continues on next page)

(continued from previous page)

```

    botmp,
    idomainp,
    ncpp=ncpp,
    ncppl=ncppl,
    xllp=0.0,
    yllp=0.0,
)

delr = np.array(ncolp * [dx])
delc = np.array(nrowp * [dx])
struct_gridp = StructuredGrid(
    nlay=1, delr=delr, delc=delc, idomain=idomainp, top=topp, botm=botmp
)
set_idomain(struct_gridp, boundary_polygon)

delr_child, delc_child = lgr.get_delr_delc()
xoff, yoff = lgr.get_lower_left()
nrowc, ncolc = delc_child.shape[0], delr_child.shape[0]
topc = np.ones((nrowc, ncolc)) * 1000.0
botmc = np.ones((nlayp, nrowc, ncolc)) * -100.0
struct_gridc = StructuredGrid(
    delr=delr_child,
    delc=delc_child,
    xoff=xoff,
    yoff=yoff,
    idomain=idomainp,
    top=topc,
    botm=botmc,
)

extent_child = struct_gridc.extent

nested_grid = [struct_gridp, struct_gridc]

```

```

[22]: top_ngp = fine_topo.resample_to_grid(
    struct_gridp,
    band=fine_topo.bands[0],
    method="linear",
    extrapolate_edges=True,
)
top_ngc = fine_topo.resample_to_grid(
    struct_gridc,
    band=fine_topo.bands[0],
    method="linear",
    extrapolate_edges=True,
)
top_nested_grid = [top_ngp, top_ngc]

```

```

[23]: ix = flopy.utils.GridIntersect(struct_gridp, method="structured")
cellids = []
for sg in sgs:

```

(continues on next page)

(continued from previous page)

```

    v = ix.intersection(LineString(sg), sort_by_cellid=True)
    cellids += v["cellids"].tolist()
intersection_ngp = np.zeros(struct_gridp.shape[1:])
for loc in cellids:
    intersection_ngp[loc] = 1
intersection_ngp[idomainp[0] == 0] = 0

ixs = flopy.utils.GridIntersect(struct_gridc, method="structured")
cellids = []
for sg in sgs:
    v = ixs.intersection(LineString(sg), sort_by_cellid=True)
    cellids += v["cellids"].tolist()
intersection_ngc = np.zeros(struct_gridc.shape[1:])
for loc in cellids:
    intersection_ngc[loc] = 1

intersection_nested_grid = [intersection_ngp, intersection_ngc]

```

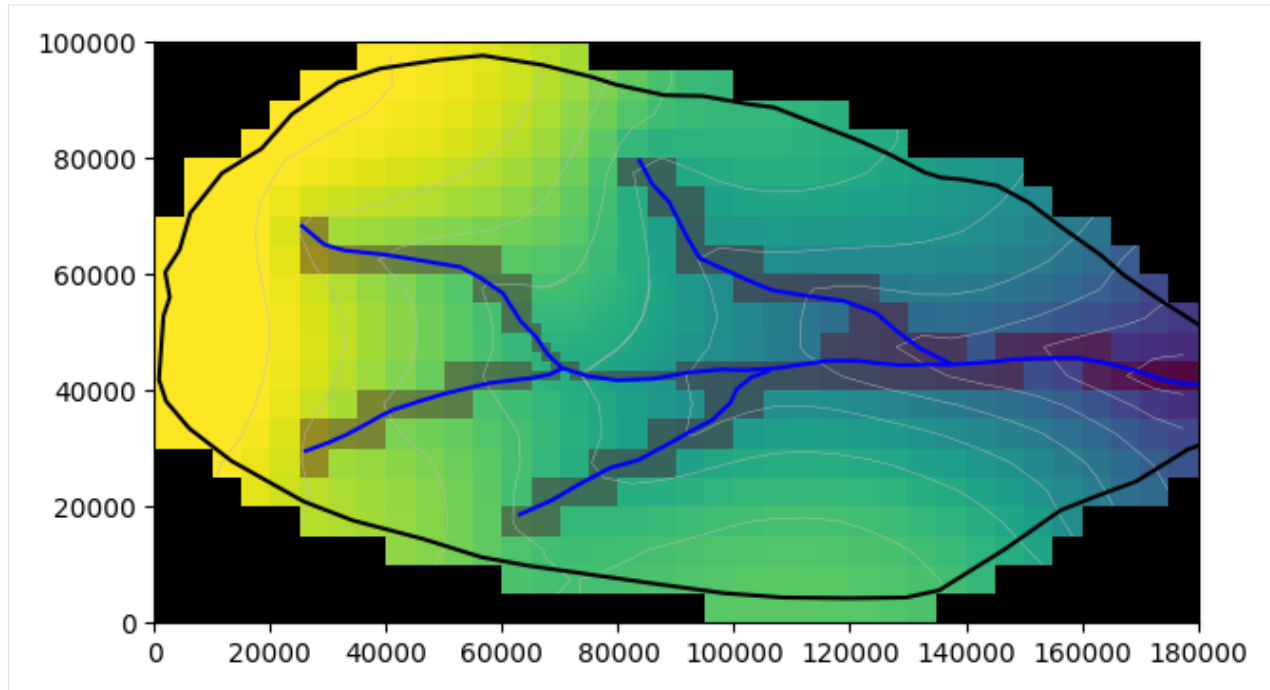
```

[24]: fig = plt.figure(figsize=figsize)
ax = fig.add_subplot()
pmv = flopy.plot.PlotMapView(modelgrid=struct_gridp, extent=extent)
pmv.plot_inactive()
pmv.plot_array(top_ngp, vmin=vmin, vmax=vmax)
pmv.plot_array(
    intersection_nested_grid[0],
    masked_values=[
        0,
    ],
    alpha=0.2,
    cmap="Reds_r",
)
cgp = pmv.contour_array(top_ngp, levels=levels, linewidths=0.3, colors="0.75")
pmv.plot_inactive(zorder=100)
ax.set_aspect("equal")

pmvc = flopy.plot.PlotMapView(modelgrid=struct_gridc, ax=ax, extent=extent)
# pmvc.plot_grid()
pmvc.plot_array(top_ngc, vmin=vmin, vmax=vmax)
pmvc.plot_array(
    intersection_nested_grid[1],
    masked_values=[
        0,
    ],
    alpha=0.2,
    cmap="Reds_r",
)
cgc = pmvc.contour_array(top_ngc, levels=levels, linewidths=0.3, colors="0.75")

ax.plot(bp[:, 0], bp[:, 1], "k-")
for sg in sgs:
    sa = np.array(sg)
    ax.plot(sa[:, 0], sa[:, 1], "b-")

```



Unstructured grids

```
[25]: # create a polygon of the common refinement area for the quadtree grid
lgr_poly = [
    [
        (extent_child[0], extent_child[2]),
        (extent_child[0], extent_child[3]),
        (extent_child[1], extent_child[3]),
        (extent_child[1], extent_child[2]),
        (extent_child[0], extent_child[2]),
    ]
]
```

Quadtree grid

```
[26]: sim = flopy.mf6.MFSimulation()
gwf = gwf = flopy.mf6.ModflowGwf(sim)
dx = dy = 5000.0
nr = int(Ly / dy)
nc = int(Lx / dx)
dis6 = flopy.mf6.ModflowGwfdis(
    gwf,
    nrow=nr,
    ncol=nc,
    delr=dy,
    delc=dx,
)
```

(continues on next page)

(continued from previous page)

```

g = Gridgen(gwf.modelgrid, model_ws=temp_path)
adpoly = [
    [(1000, 1000), (3000, 1000), (3000, 2000), (1000, 2000), (1000, 1000)]]
]
adpoly = boundary_polygon + [boundary_polygon[0]]
adpoly = [[adpoly]]
g.add_refinement_features([lgr_poly], "polygon", 2, range(1))

refine_line = sgs
g.add_refinement_features(refine_line, "line", 2, range(1))
g.build(verbose=False)

gridprops_vg = g.get_gridprops_vertexgrid()
quadtree_grid = flopy.discretization.VertexGrid(**gridprops_vg)
set_idomain(quadtree_grid, boundary_polygon)

```

```

[27]: top_qg = fine_topo.resample_to_grid(
        quadtree_grid,
        band=fine_topo.bands[0],
        method="linear",
        extrapolate_edges=True,
    )

```

```

[28]: ix = flopy.utils.GridIntersect(quadtree_grid, method="vertex")
cellids = []
for sg in sgs:
    v = ix.intersect(LineString(sg), sort_by_cellid=True)
    cellids += v["cellids"].tolist()
intersection_qg = np.zeros(quadtree_grid.shape[1:])
for loc in cellids:
    intersection_qg[loc] = 1

```

```

[29]: fig = plt.figure(figsize=figsize)
ax = fig.add_subplot()
pmv = flopy.plot.PlotMapView(modelgrid=quadtree_grid)
pmv.plot_array(top_qg, ec="0.75")
pmv.plot_array(
    intersection_qg,
    masked_values=[
        0,
    ],
    alpha=0.2,
    cmap="Reds_r",
)
cg = pmv.contour_array(top_qg, levels=levels, linewidths=0.3, colors="white")
pmv.plot_inactive(zorder=100)
ax.set_aspect("equal")

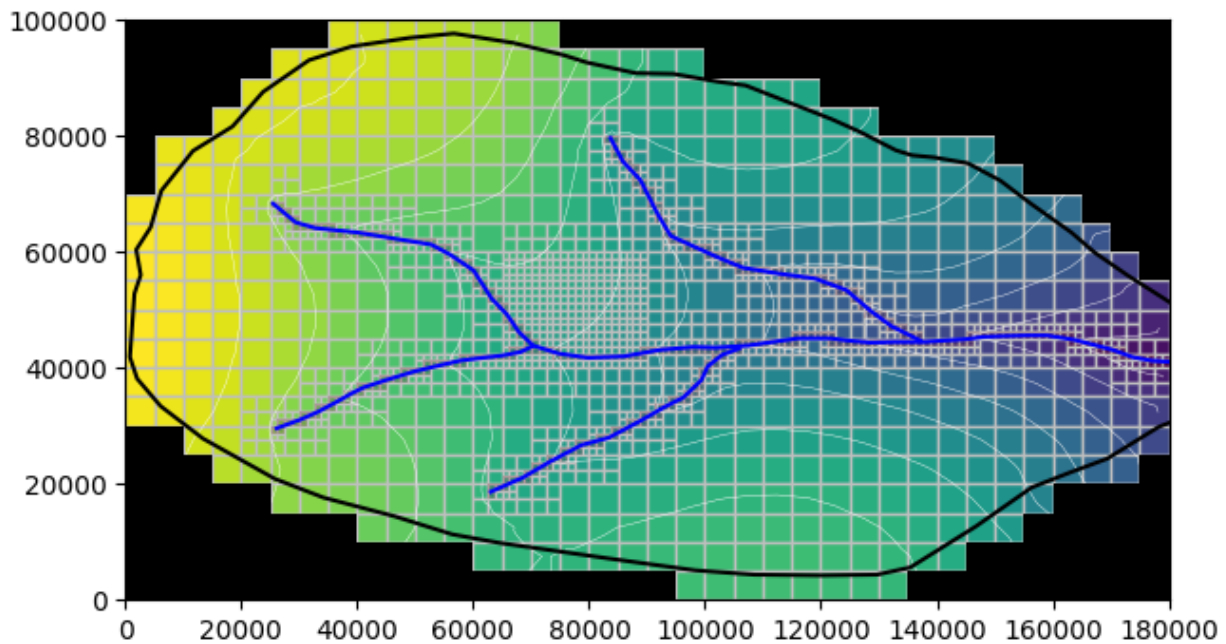
ax.plot(bp[:, 0], bp[:, 1], "k-")
for sg in sgs:
    sa = np.array(sg)

```

(continues on next page)

(continued from previous page)

```
ax.plot(sa[:, 0], sa[:, 1], "b-")
```



Triangular grid

```
[30]: nodes = []
      for sg in sgs:
          sg_densify = densify_geometry(sg, 2000)
          nodes.extend(sg_densify)
      for x in struct_gridc.get_xvertices_for_layer(0)[0, :]:
          for y in struct_gridc.get_yvertices_for_layer(0)[:, 0]:
              nodes.append((x, y))
      nodes = np.array(nodes)

[31]: tri = Triangle(
        maximum_area=5000 * 5000, angle=30, nodes=nodes, model_ws=temp_path
    )
    poly = bp
    tri.add_polygon(poly)
    tri.build(verbose=False)

    cell2d = tri.get_cell2d()
    vertices = tri.get_vertices()

    top = np.ones(tri.ncpl) * 1000.0
    botm = np.ones((1, tri.ncpl)) * -100.0
    idomain = np.ones((1, tri.ncpl), dtype=int)
```

(continues on next page)

(continued from previous page)

```
triangular_grid = VertexGrid(
    vertices=vertices,
    cell2d=cell2d,
    idomain=idomain,
    nlay=1,
    ncpl=tri.ncpl,
    top=top,
    botm=botm,
)
```

```
[32]: top_tg = fine_topo.resample_to_grid(
    triangular_grid,
    band=fine_topo.bands[0],
    method="linear",
    extrapolate_edges=True,
)
```

```
[33]: ix = flopy.utils.GridIntersect(triangular_grid) # , method="vertex")
cellids = []
for sg in sgs:
    v = ix.intersect(
        LineString(sg),
        return_all_intersections=True,
        keepzerolengths=False,
        sort_by_cellid=True,
    )
    cellids += v["cellids"].tolist()
intersection_tg = np.zeros(triangular_grid.shape[1:])
for loc in cellids:
    intersection_tg[loc] = 1
```

```
[34]: fig = plt.figure(figsize=figsize)
ax = fig.add_subplot()
ax.set_aspect("equal")

pmv = flopy.plot.PlotMapView(modelgrid=triangular_grid)

pmv.plot_array(top_tg, ec="0.75")
pmv.plot_array(
    intersection_tg,
    masked_values=[
        0,
    ],
    alpha=0.2,
    cmap="Reds_r",
)
cg = pmv.contour_array(top_tg, levels=levels, linewidths=0.3, colors="white")
ax.clabel(cg, cg.levels, inline=True, fmt="%1.0f", fontsize=10)

pmv.plot_inactive(zorder=100)
```

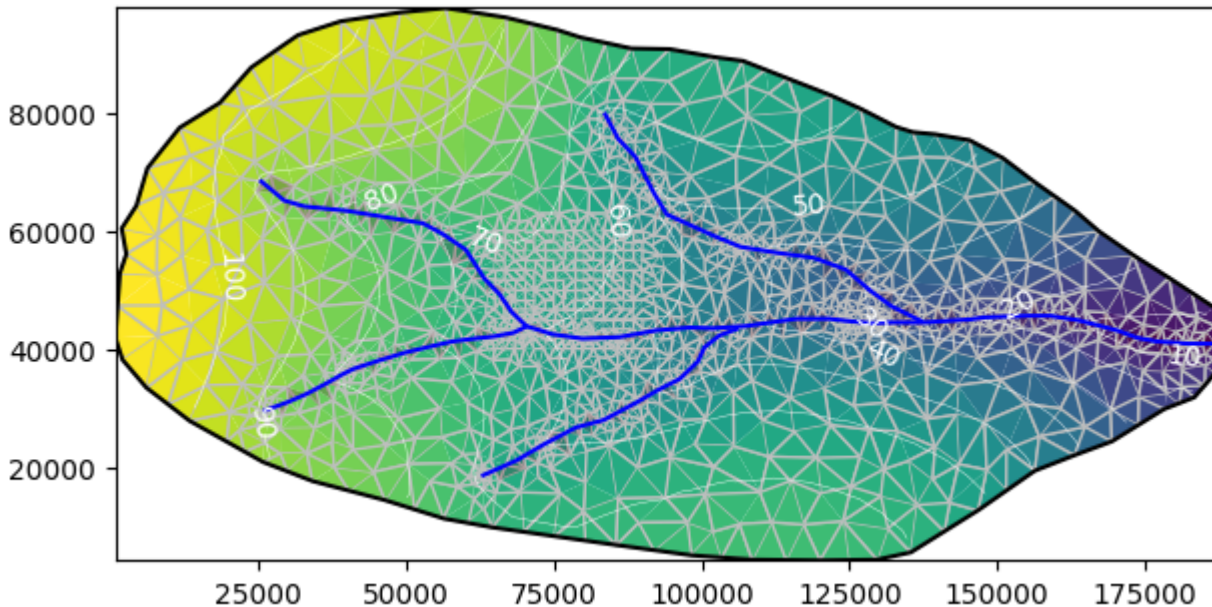
(continues on next page)

(continued from previous page)

```

if True:
    ax.plot(bp[:, 0], bp[:, 1], "k-")
    for sg in sgs:
        sa = np.array(sg)
        ax.plot(sa[:, 0], sa[:, 1], "b-")

```



Voronoi Grid from the Triangle object

```

[35]: # create vor object and VertexGrid from the triangle object tri
vor = VoronoiGrid(tri)
gridprops = vor.get_gridprops_vertexgrid()
idomain = np.ones((1, vor.ncpl), dtype=int)
voronoi_grid = VertexGrid(**gridprops, nlay=1, idomain=idomain)

```

```

[36]: top_vg = fine_topo.resample_to_grid(
    voronoi_grid,
    band=fine_topo.bands[0],
    method="linear",
    extrapolate_edges=True,
)

```

```

[37]: ix = flopy.utils.GridIntersect(voronoi_grid, method="vertex")
cellids = []
for sg in sgs:
    v = ix.intersect(
        LineString(sg),
        return_all_intersections=True,
        keepzerolengths=False,
        sort_by_cellid=True,
    )

```

(continues on next page)

(continued from previous page)

```

    )
    cellids += v["cellids"].tolist()
intersection_vg = np.zeros(voronoi_grid.shape[1:])
for loc in cellids:
    intersection_vg[loc] = 1

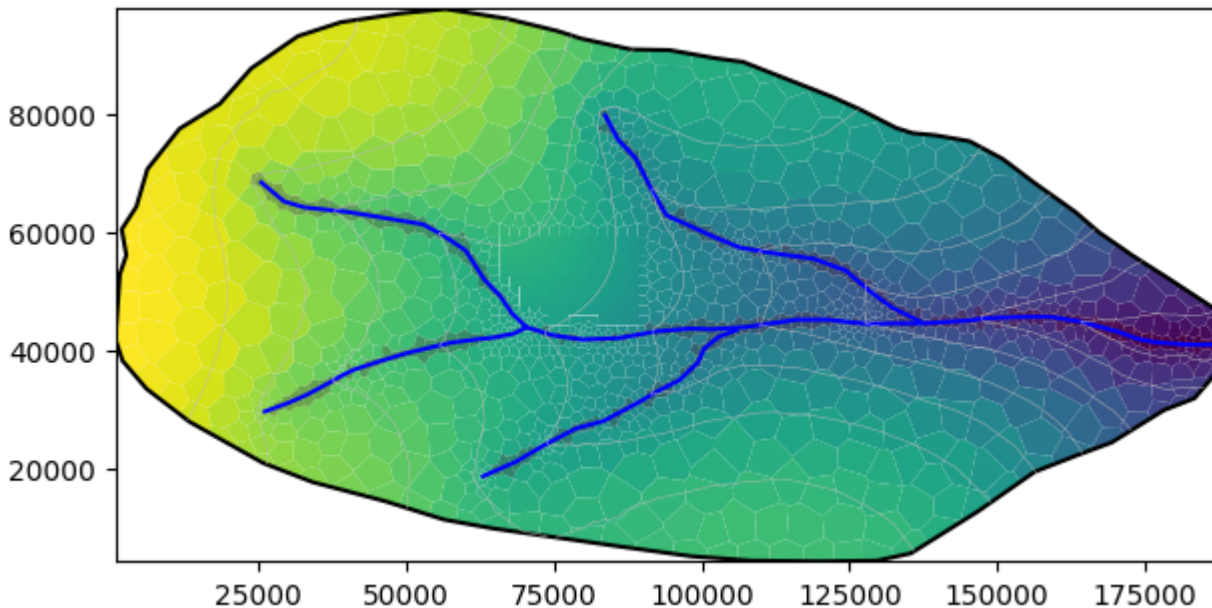
```

```

[38]: fig = plt.figure(figsize=figsize)
ax = fig.add_subplot()
pmv = flopy.plot.PlotMapView(modelgrid=voronoi_grid)
ax.set_aspect("equal")
pmv.plot_array(top_vg)
pmv.plot_array(
    intersection_vg,
    masked_values=[
        0,
    ],
    alpha=0.2,
    cmap="Reds_r",
)
pmv.plot_inactive()
ax.plot(bp[:, 0], bp[:, 1], "k-")
for sg in sgs:
    sa = np.array(sg)
    ax.plot(sa[:, 0], sa[:, 1], "b-")

cg = pmv.contour_array(top_vg, levels=levels, linewidths=0.3, colors="0.75")

```



Plot all six grids

Create a polyline for Local Grid Refinement area

```
[39]: lgr_poly_array = np.array(lgr_poly).squeeze()
```

Plot the grids

```
[40]: # Make a plot of the six grids
figwidth = 17.15 / 2.54
figheight = 3.66 * (Ly / Lx) * 8.25 / 2.54
grids = [
    struct_grid,
    struct_vrc_grid,
    nested_grid,
    quadtree_grid,
    triangular_grid,
    voronoi_grid,
    None,
]
topo_grids = [top_sg, top_sg_vrc, top_nested_grid, top_qg, top_tg, top_vg]
extent = (0, 180000, 0, 100000)
cbar_axis = [0.0, 0.1, 0.25, 0.9]

with styles.USGSMap():
    fig = plt.figure(figsize=(figwidth, figheight), constrained_layout=True)
    gs = gridspec.GridSpec(ncols=16, nrows=17, figure=fig)
    axs = [fig.add_subplot(gs[:5, :8])]
    axs.append(fig.add_subplot(gs[:5, 8:]))
    axs.append(fig.add_subplot(gs[5:10, :8]))
    axs.append(fig.add_subplot(gs[5:10, 8:]))
    axs.append(fig.add_subplot(gs[10:15, :8]))
    axs.append(fig.add_subplot(gs[10:15, 8:]))
    axs.append(fig.add_subplot(gs[15:, :]))

    for idx, ax in enumerate(axs[:-1]):
        g = grids[idx]
        t = topo_grids[idx]
        if g is not None:
            if isinstance(g, list):
                g = g[0]
                t = t[0]

        pmv = flopy.plot.PlotMapView(modelgrid=g, ax=ax)

        v = pmv.plot_array(t)
        pmv.plot_grid(**grid_dict)
        cg = pmv.contour_array(t, **contour_dict)
        pmv.ax.clabel(cg, cg.levels, **clabel_dict)
        pmv.plot_inactive(color_noflow="gray", zorder=100)
```

(continues on next page)

(continued from previous page)

```

if isinstance(grids[idx], list):
    gg = grids[idx]
    tt = topo_grids[idx]
    for g, t in zip(gg[1:], tt[1:]):
        pmvc = flopy.plot.PlotMapView(
            modelgrid=g, ax=ax, extent=extent
        )
        pmvc.plot_array(top_ngc, vmin=vmin, vmax=vmax)
        pmvc.plot_grid(**grid_dict)
        cgc = pmvc.contour_array(top_ngc, **contour_dict)

# plot lgr polyline
ax.plot(
    lgr_poly_array[:, 0],
    lgr_poly_array[:, 1],
    zorder=101,
    **refinement_dict,
)

ax.set_aspect("equal")
ax.set_axisbelow(False)

ax.set_xlim(extent[0], extent[1])
ax.set_xticks(np.arange(0, 200000, 50000))
if idx in (4, 5):
    ax.set_xticklabels(np.arange(0, 200, 50))
    ax.set_xlabel("x position (km)")
else:
    ax.set_xticklabels([])

ax.set_ylim(extent[2], extent[3])
ax.set_yticks(np.arange(0, 150000, 50000))
if idx in (0, 2, 4):
    ax.set_yticklabels(np.arange(0, 150, 50))
    ax.set_ylabel("y position (km)")
else:
    ax.set_yticklabels([])

styles.heading(ax, idx=idx)
if True:
    ax.plot(bp[:, 0], bp[:, 1], "k-", lw=2.0)
    for sg in sgs:
        sa = np.array(sg)
        ax.plot(sa[:, 0], sa[:, 1], **river_dict)

# legend
ax = axs[-1]
xy0 = (-100, -100)
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.set_axis_off()

```

(continues on next page)

(continued from previous page)

```

ax.axhline(
    xy0[0],
    color="black",
    lw=2,
    label="Basin boundary",
)
ax.axhline(
    xy0[0],
    **river_dict,
    label="River",
)
ax.axhline(
    xy0[0],
    color=contour_color,
    lw=0.5,
    ls="--",
    label="Elevation contour",
)
ax.axhline(
    xy0[0],
    label="Grid refinement area",
    **refinement_dict,
)
ax.axhline(
    xy0[0],
    marker="s",
    mec="0.5",
    mfc="white",
    markeredgewidth=0.25,
    lw=0,
    label="Grid cell",
)
ax.axhline(
    xy0[0],
    marker="s",
    mec="0.5",
    mfc="gray",
    markeredgewidth=0.25,
    lw=0,
    label="Inactive area",
)
styles.graph_legend(
    ax,
    ncol=3,
    loc="lower center",
    labelspace=0.5,
    columnspacing=0.6,
    handletextpad=0.3,
)

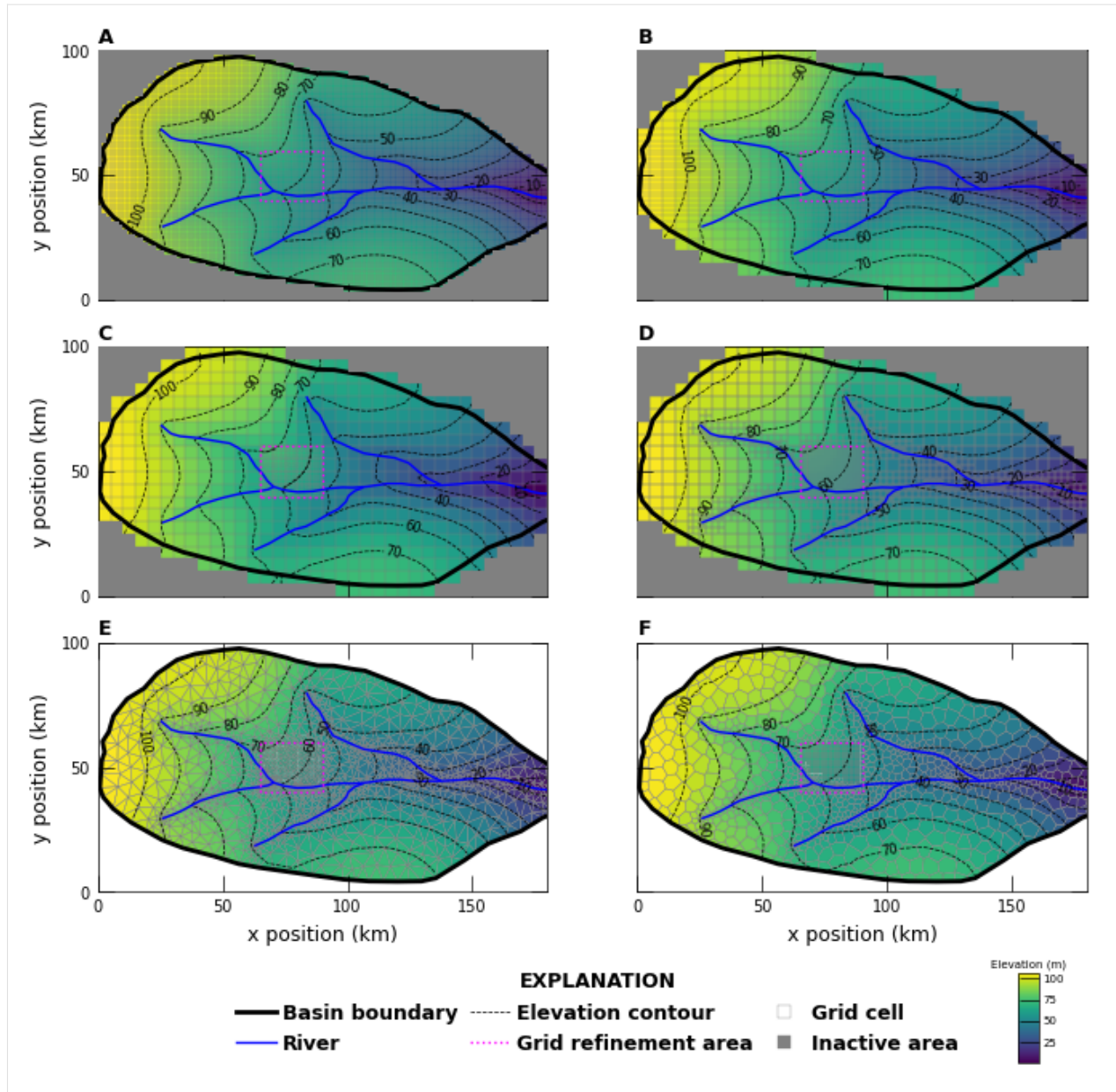
# colorbar
ax = fig.add_subplot(gs[15:, 14:])

```

(continues on next page)

(continued from previous page)

```
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.set_axis_off()
cax = ax.inset_axes(
    cbar_axis,
)
#     cax.set_axisbelow(False)
cbar = plt.colorbar(
    v,
    orientation="vertical",
    cax=cax,
    ticks=[25, 50, 75, 100],
)
cbar.ax.tick_params(
    labelsize=5,
    labelcolor="black",
    color="black",
    length=9,
    pad=2,
)
cbar.ax.set_title(
    "Elevation (m)", pad=2.5, loc="center", fontdict=font_dict
)
```



Plot the river intersection for the six grids

```
[41]: figwidth = 17.15 / 2.54
figheight = 3.66 * (Ly / Lx) * 8.25 / 2.54
grids = [
    struct_grid,
    struct_vrc_grid,
    nested_grid,
    quadtree_grid,
    triangular_grid,
    voronoi_grid,
```

(continues on next page)

(continued from previous page)

```

    None,
]
intersections = [
    intersection_sg,
    intersection_sg_vrc,
    intersection_nested_grid,
    intersection_qg,
    intersection_tg,
    intersection_vg,
    None,
]
extent = list(extent_child)
de = 10000.0
extent[0] -= 2.0 * de
extent[1] += 2.5 * de
extent[2] -= de
extent[3] += de

with styles.USGSMap():
    fig = plt.figure(figsize=(figwidth, figheight), constrained_layout=True)
    gs = gridspec.GridSpec(ncols=2, nrows=16, figure=fig)
    axs = [fig.add_subplot(gs[:5, 0])]
    axs.append(fig.add_subplot(gs[:5, 1]))
    axs.append(fig.add_subplot(gs[5:10, 0]))
    axs.append(fig.add_subplot(gs[5:10, 1]))
    axs.append(fig.add_subplot(gs[10:15, 0]))
    axs.append(fig.add_subplot(gs[10:15, 1]))
    axs.append(fig.add_subplot(gs[15:, :]))

    for idx, ax in enumerate(axs[:-1]):
        g = grids[idx]
        t = intersections[idx]
        if g is not None:
            if isinstance(g, list):
                g = g[0]
                t = t[0]

        pmv = flopy.plot.PlotMapView(modelgrid=g, ax=ax, extent=extent)

        v = pmv.plot_array(t, masked_values=(0,), cmap=intersection_cmap)
        pmv.plot_grid(**grid_dict)

        pmv.plot_inactive(color_noflow="gray", zorder=100)

        if isinstance(grids[idx], list):
            gg = grids[idx]
            tt = intersections[idx]
            for g, t in zip(gg[1:], tt[1:]):
                pmvc = flopy.plot.PlotMapView(
                    modelgrid=g, ax=ax, extent=extent
                )

```

(continues on next page)

(continued from previous page)

```

        pmvc.plot_array(
            t, masked_values=(0,), cmap=intersection_cmap
        )
        pmvc.plot_grid(**grid_dict)

# plot lgr polyline
ax.plot(
    lgr_poly_array[:, 0],
    lgr_poly_array[:, 1],
    zorder=101,
    **refinement_dict,
)

ax.set_aspect("equal")
ax.set_axisbelow(False)

ax.set_xlim(extent[0], extent[1])
ax.set_xticks(np.arange(50000, 120000, 10000))
if idx in (4, 5):
    ax.set_xticklabels(np.arange(50, 120, 10))
    ax.set_xlabel("x position (km)")
else:
    ax.set_xticklabels([])

ax.set_ylim(extent[2], extent[3])
ax.set_yticks(np.arange(35000, 70000, 10000))
if idx in (0, 2, 4):
    ax.set_yticklabels(np.arange(35, 75, 10))
    ax.set_ylabel("y position (km)")
else:
    ax.set_yticklabels([])

styles.heading(ax, idx=idx)
if True:
    ax.plot(bp[:, 0], bp[:, 1], "k-", lw=2.0)
    for sg in sgs:
        sa = np.array(sg)
        ax.plot(sa[:, 0], sa[:, 1], **river_dict)

# legend
ax = axs[-1]
xy0 = (-100, -100)
ax.set_xlim(0, 1)
ax.set_ylim(0, 1)
ax.set_axis_off()

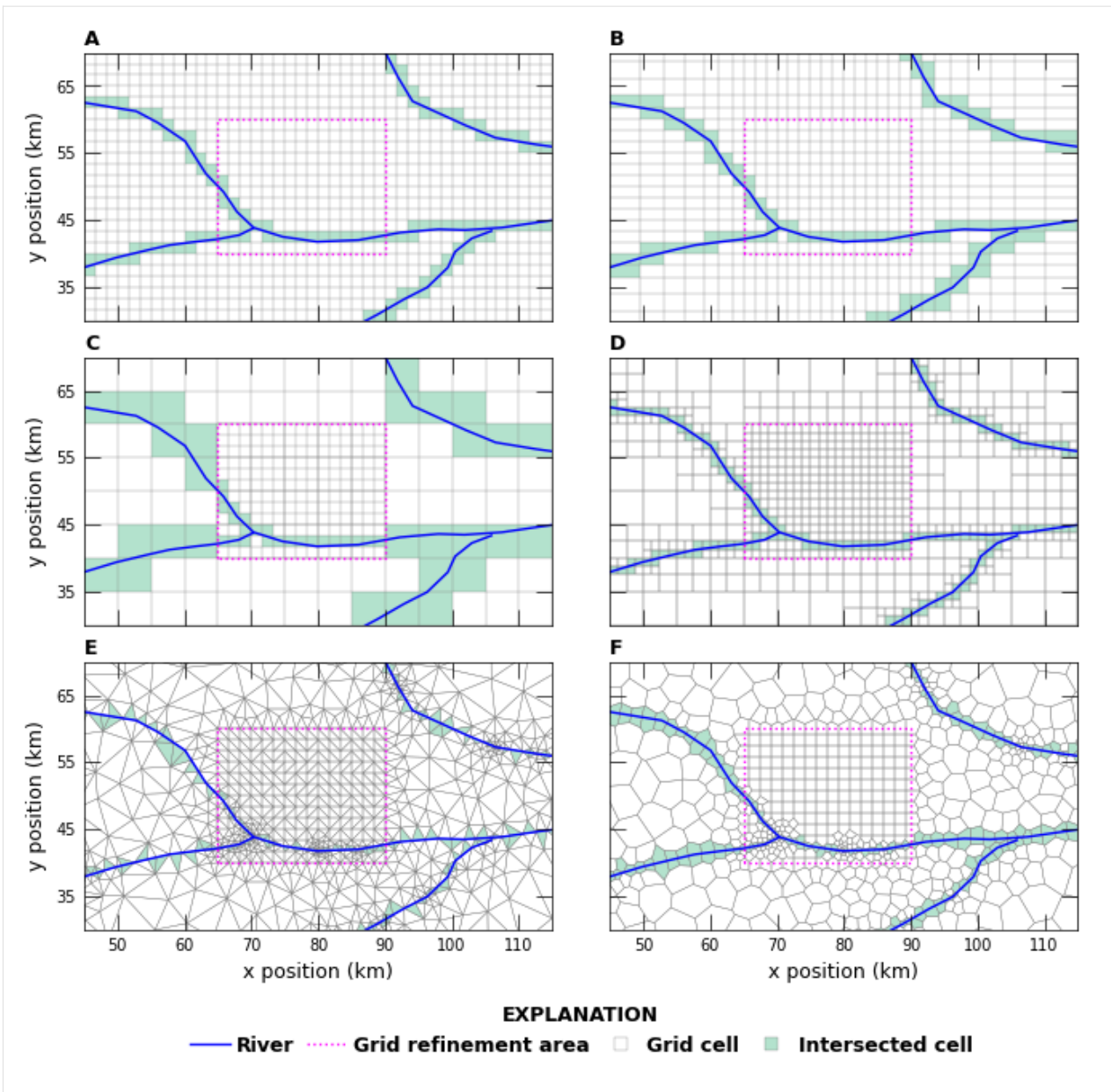
ax.axhline(xy0[0], **river_dict, label="River")
ax.axhline(
    xy0[0],
    label="Grid refinement area",
    **refinement_dict,
)

```

(continues on next page)

(continued from previous page)

```
ax.axhline(
    xy0[0],
    marker="s",
    mec="0.5",
    mfc="white",
    markeredgewidth=0.25,
    lw=0,
    label="Grid cell",
)
ax.axhline(
    xy0[0],
    marker="s",
    mfc=intersection_rgba,
    mec="0.5",
    markeredgewidth=0.25,
    lw=0,
    label="Intersected cell",
)
styles.graph_legend(
    ax,
    ncol=4,
    loc="lower center",
    labelspaceing=0.5,
    columnspacing=0.6,
    handletextpad=0.3,
)
```



6.12 Miscellaneous examples

6.12.1 Henry Saltwater Intrusion Problem

In this notebook, we will use FloPy to create, run, and post process the Henry saltwater intrusion problem using SEAWAT Version 4.

```
[1]: import os
import sys
from tempfile import TemporaryDirectory
```

(continues on next page)

(continued from previous page)

```
import numpy as np

# run installed version of flopy or add local path
try:
    import flopy
except:
    fpth = os.path.abspath(os.path.join("..", ".."))
    sys.path.append(fpth)
    import flopy

print(sys.version)
print("numpy version: {}".format(np.__version__))
print("flopy version: {}".format(flopy.__version__))
```

3.8.18 (default, Aug 28 2023, 08:27:22)
 [GCC 11.4.0]
 numpy version: 1.24.4
 flopy version: 3.4.3

```
[2]: # temporary directory
temp_dir = TemporaryDirectory()
workspace = temp_dir.name
```

```
[3]: # Input variables for the Henry Problem
Lx = 2.0
Lz = 1.0
nlay = 50
nrow = 1
ncol = 100
delr = Lx / ncol
delc = 1.0
delv = Lz / nlay
henry_top = 1.0
henry_botm = np.linspace(henry_top - delv, 0.0, nlay)
qinflow = 5.702 # m3/day
dmcoef = 0.57024 # m2/day Could also try 1.62925 as another case of the Henry problem
hk = 864.0 # m/day
```

```
[4]: # Create the basic MODFLOW model structure
modelname = "henry"
swt = flopy.seawat.Seawat(modelname, exe_name="swtv4", model_ws=workspace)
print(swt.namefile)

# save cell fluxes to unit 53
ipakcb = 53

# Add DIS package to the MODFLOW model
dis = flopy.modflow.ModflowDis(
    swt,
    nlay,
    nrow,
    ncol,
```

(continues on next page)

(continued from previous page)

```

    nper=1,
    delr=delr,
    delc=delc,
    laycbd=0,
    top=henry_top,
    botm=henry_botm,
    perlen=1.5,
    nstp=15,
)

# Variables for the BAS package
ibound = np.ones((nlay, nrow, ncol), dtype=np.int32)
ibound[:, :, -1] = -1
bas = flopy.modflow.ModflowBas(swt, ibound, 0)

# Add LPF package to the MODFLOW model
lpf = flopy.modflow.ModflowLpf(swt, hk=hk, vka=hk, ipakcb=ipakcb)

# Add PCG Package to the MODFLOW model
pcg = flopy.modflow.ModflowPcg(swt, hclose=1.0e-8)

# Add OC package to the MODFLOW model
oc = flopy.modflow.ModflowOc(
    swt,
    stress_period_data={(0, 0): ["save head", "save budget"]},
    compact=True,
)

# Create WEL and SSM data
itype = flopy.mt3d.Mt3dSsm.itype_dict()
wel_data = {}
ssm_data = {}
wel_sp1 = []
ssm_sp1 = []
for k in range(nlay):
    wel_sp1.append([k, 0, 0, qinflow / nlay])
    ssm_sp1.append([k, 0, 0, 0.0, itype["WEL"]])
    ssm_sp1.append([k, 0, ncol - 1, 35.0, itype["BAS6"]])
wel_data[0] = wel_sp1
ssm_data[0] = ssm_sp1
wel = flopy.modflow.ModflowWel(swt, stress_period_data=wel_data, ipakcb=ipakcb)

henry.nam

```

```

[5]: # Create the basic MT3DMS model structure
# mt = flopy.mt3d.Mt3dms(modelname, 'nam_mt3dms', mf, model_ws=workspace)
btn = flopy.mt3d.Mt3dBtn(
    swt,
    nprs=-5,
    prsity=0.35,
    sconc=35.0,
    ifmtcn=0,

```

(continues on next page)

(continued from previous page)

```

    chkmas=False,
    nprobs=10,
    nprmas=10,
    dt0=0.001,
)
adv = flopy.mt3d.Mt3dAdv(swt, mixelm=0)
dsp = flopy.mt3d.Mt3dDsp(swt, al=0.0, trpt=1.0, trpv=1.0, dmcoef=dmcoef)
gcg = flopy.mt3d.Mt3dGcg(swt, iter1=500, mxiter=1, isolve=1, cclose=1e-7)
ssm = flopy.mt3d.Mt3dSsm(swt, stress_period_data=ssm_data)

# Create the SEAWAT model structure
# mswt = flopy.seawat.Seawat(modelname, 'nam_swt', mf, mt, model_ws=workspace, exe_name=
#     'swtv4')
vdf = flopy.seawat.SeawatVdf(
    swt,
    iwttable=0,
    densemin=0,
    densemax=0,
    denseref=1000.0,
    denseslp=0.7143,
    firstdt=1e-3,
)

```

```

[6]: # Write the input files
     swt.write_input()

```

```

[7]: # Try to delete the output files, to prevent accidental use of older files
     try:
         os.remove(os.path.join(workspace, "MT3D001.UCN"))
         os.remove(os.path.join(workspace, modelname + ".hds"))
         os.remove(os.path.join(workspace, modelname + ".cbc"))
     except:
         pass

```

```

[8]: v = swt.run_model(silent=True, report=True)
     for idx in range(-3, 0):
         print(v[1][idx])

```

Elapsed run time: 12.183 Seconds

Normal termination of SEAWAT

```

[9]: # Post-process the results
     import numpy as np

     import flopy.utils.binaryfile as bf

     # Load data
     ucobj = bf.UcnFile(os.path.join(workspace, "MT3D001.UCN"), model=swt)
     times = ucobj.get_times()
     concentration = ucobj.get_data(totim=times[-1])

```

(continues on next page)

(continued from previous page)

```

cbbobj = bf.CellBudgetFile(os.path.join(workspace, "henry.cbc"))
times = cbbobj.get_times()
qx = cbbobj.get_data(text="flow right face", totim=times[-1])[0]
qz = cbbobj.get_data(text="flow lower face", totim=times[-1])[0]

# Average flows to cell centers
qx_avg = np.empty(qx.shape, dtype=qx.dtype)
qx_avg[:, :, 1:] = 0.5 * (qx[:, :, 0 : ncol - 1] + qx[:, :, 1:ncol])
qx_avg[:, :, 0] = 0.5 * qx[:, :, 0]
qz_avg = np.empty(qz.shape, dtype=qz.dtype)
qz_avg[1:, :, :] = 0.5 * (qz[0 : nlay - 1, :, :] + qz[1:nlay, :, :])
qz_avg[0, :, :] = 0.5 * qz[0, :, :]

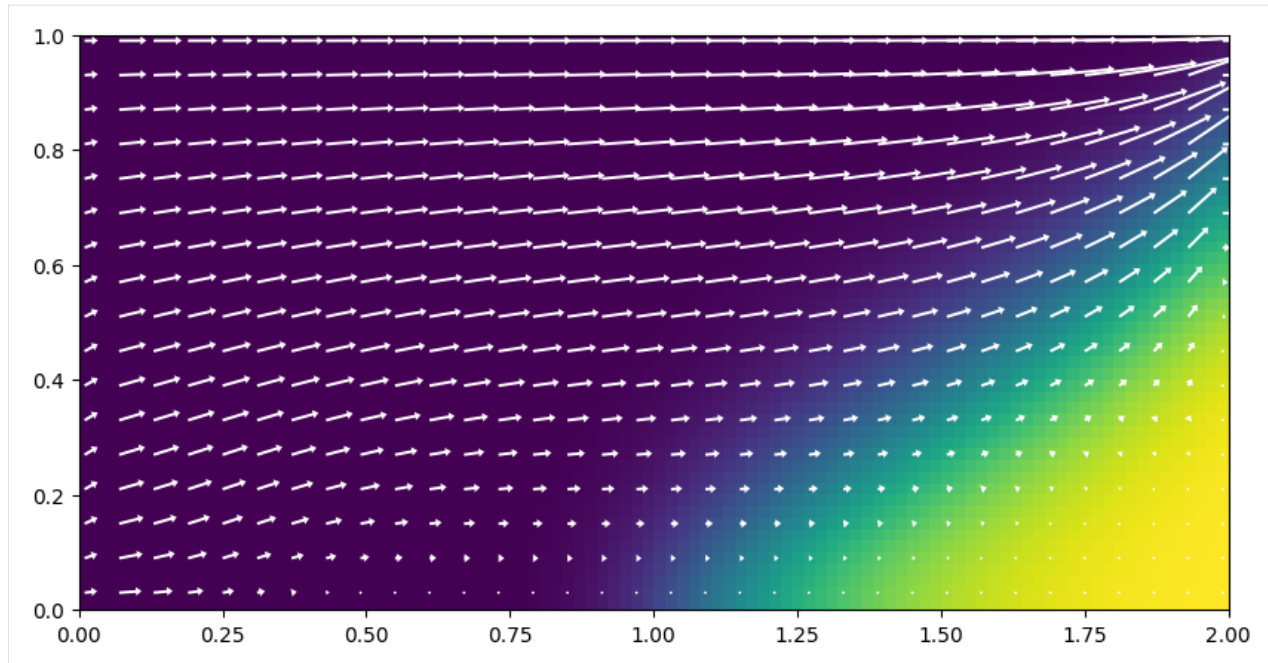
```

```

[10]: # Make the plot
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
ax.imshow(
    concentration[:, 0, :], interpolation="nearest", extent=(0, Lx, 0, Lz)
)
y, x, z = dis.get_node_coordinates()
X, Z = np.meshgrid(x, z[:, 0, 0])
iskip = 3
ax.quiver(
    X[:, :iskip, ::iskip],
    Z[:, :iskip, ::iskip],
    qx_avg[:, :iskip, 0, ::iskip],
    -qz_avg[:, :iskip, 0, ::iskip],
    color="w",
    scale=5,
    headwidth=3,
    headlength=2,
    headaxislength=2,
    width=0.0025,
)
plt.savefig(os.path.join(workspace, "henry.png"))
plt.show()

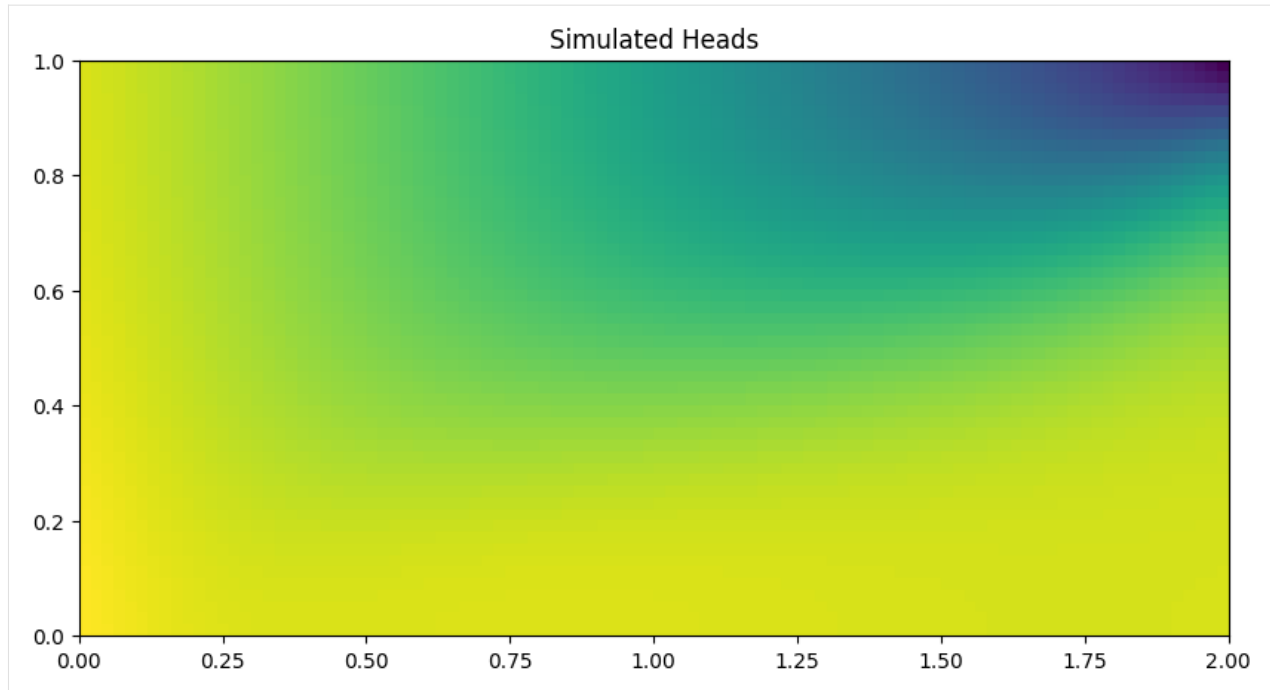
```



```
[11]: # Extract the heads
      fname = os.path.join(workspace, "henry.hds")
      headobj = bf.HeadFile(fname)
      times = headobj.get_times()
      head = headobj.get_data(totim=times[-1])

[12]: # Make a simple head plot
      fig = plt.figure(figsize=(10, 10))
      ax = fig.add_subplot(1, 1, 1, aspect="equal")
      im = ax.imshow(head[:, 0, :], interpolation="nearest", extent=(0, Lx, 0, Lz))
      ax.set_title("Simulated Heads")

[12]: Text(0.5, 1.0, 'Simulated Heads')
```



Change the format of several arrays and rerun the model

```
[13]: swt.btn.prsity.how = "constant"
      swt.btn.prsity[0].how = "internal"
      swt.btn.prsity[1].how = "external"
      swt.btn.sconc[0].how = "external"
      swt.btn.prsity[0].fmtin = "(100E15.6)"
      swt.lpf.hk[0].fmtin = "(BINARY)"
```

```
[14]: swt.write_input()
      v = swt.run_model(silent=True, report=True)
      for idx in range(-3, 0):
          print(v[1][idx])
```

Util2d:hk layer 1: resetting 'how' to external
Elapsed run time: 12.036 Seconds

Normal termination of SEAWAT

```
[15]: try:
      # ignore PermissionError on Windows
      temp_dir.cleanup()
      except:
          pass
```

6.12.2 ZoneBudget Example

This notebook demonstrates how to use the ZoneBudget class to extract budget information from the cell by cell budget file using an array of zones.

First set the path and import the required packages. The flopy path doesn't have to be set if you install flopy from a binary installer. If you want to run this notebook, you have to set the path to your own flopy path.

```
[1]: import sys
      from pathlib import Path
      from tempfile import TemporaryDirectory

      import matplotlib as mpl
      import matplotlib.pyplot as plt
      import numpy as np
      import pandas as pd

      proj_root = Path.cwd().parent.parent

      # run installed version of flopy or add local path
      try:
          import flopy
      except:
          sys.path.append(proj_root)
          import flopy

      print(sys.version)
      print("numpy version: {}".format(np.__version__))
      print("matplotlib version: {}".format(mpl.__version__))
      print("pandas version: {}".format(pd.__version__))
      print("flopy version: {}".format(flopy.__version__))

      3.8.18 (default, Aug 28 2023, 08:27:22)
      [GCC 11.4.0]
      numpy version: 1.24.4
      matplotlib version: 3.7.3
      pandas version: 2.0.3
      flopy version: 3.4.3
```

```
[2]: # temporary workspace
      temp_dir = TemporaryDirectory()
      workspace = Path(temp_dir.name)

      # Set path to example datafiles
      loadpth = proj_root / "examples" / "data" / "zonbud_examples"
      cbc_f = loadpth / "freyberg.gitcbc"
```

Read File Containing Zones

Using the `ZoneBudget.read_zone_file()` utility, we can import zonebudget-style array files.

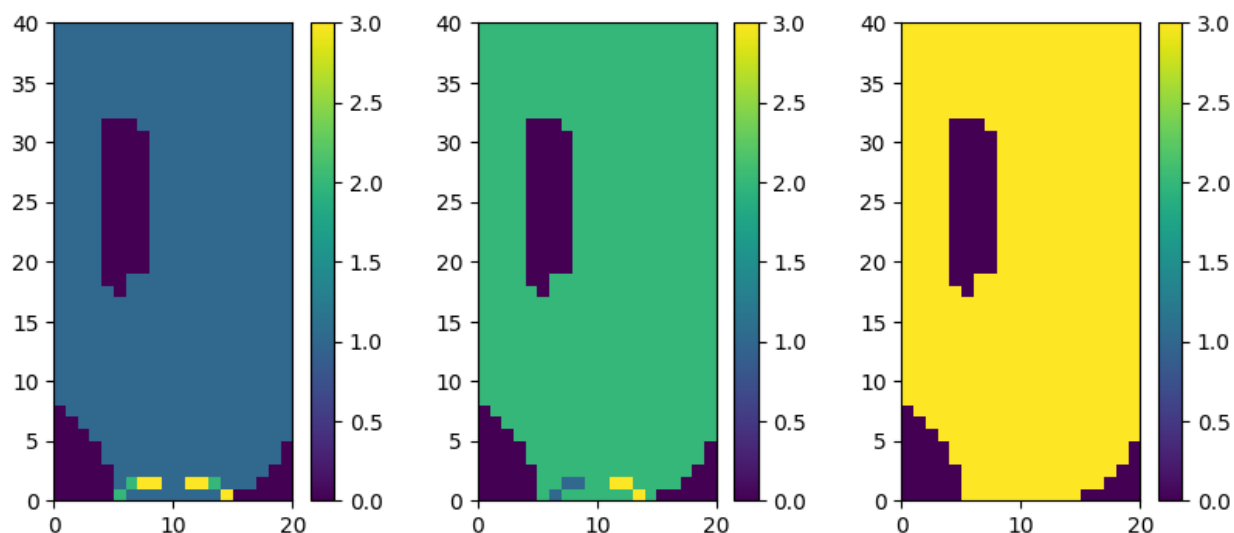
```
[3]: from flopy.utils import ZoneBudget

zone_file = loadpth / "zonef_mlt.zbr"
zon = ZoneBudget.read_zone_file(zone_file)
nlay, nrow, ncol = zon.shape

fig = plt.figure(figsize=(10, 4))

for lay in range(nlay):
    ax = fig.add_subplot(1, nlay, lay + 1)
    im = ax.pcolormesh(zon[lay, :-1, :])
    cbar = plt.colorbar(im)
    plt.gca().set_aspect("equal")

plt.show()
```



Extract Budget Information from ZoneBudget Object

At the core of the `ZoneBudget` object is a numpy structured array. The class provides some wrapper functions to help us interrogate the array and save it to disk.

```
[4]: # Create a ZoneBudget object and get the budget record array
zb = flopy.utils.ZoneBudget(cbc_f, zon, kstpkper=(0, 1096))
zb.get_budget()

[4]: array([(1097., 0, 1096, 'FROM_STORAGE', 0., 0.00000000e+00, 0.00000000e+00, 0.
    ↪ 00000000e+00),
    (1097., 0, 1096, 'FROM_CONSTANT_HEAD', 0., 0.00000000e+00, 2.3156659e+02, 8.
    ↪ 6217201e+01),
    (1097., 0, 1096, 'FROM_WELLS', 0., 0.00000000e+00, 0.00000000e+00, 0.00000000e+00),
    (continues on next page)
```

(continued from previous page)

```

(1097., 0, 1096, 'FROM_DRAINS', 0., 0.0000000e+00, 0.0000000e+00, 0.0000000e+00),
(1097., 0, 1096, 'FROM_RECHARGE', 0., 5.1455815e+03, 1.4936376e+01, 2.
↪9872751e+01),
(1097., 0, 1096, 'FROM_ZONE_0', 0., 0.0000000e+00, 0.0000000e+00, 0.0000000e+00),
(1097., 0, 1096, 'FROM_ZONE_1', 0., 0.0000000e+00, 3.4751235e+03, 1.3860045e+02),
(1097., 0, 1096, 'FROM_ZONE_2', 0., 3.2693188e+03, 0.0000000e+00, 1.7646553e+03),
(1097., 0, 1096, 'FROM_ZONE_3', 0., 1.9218604e+02, 1.5280482e+03, 0.0000000e+00),
(1097., 0, 1096, 'TOTAL_IN', 0., 8.6070859e+03, 5.2496748e+03, 2.0193457e+03),
(1097., 0, 1096, 'TO_STORAGE', 0., 0.0000000e+00, 0.0000000e+00, 0.0000000e+00),
(1097., 0, 1096, 'TO_CONSTANT_HEAD', 0., 2.3054836e+02, 2.1570151e+02, 2.
↪9911380e+02),
(1097., 0, 1096, 'TO_WELLS', 0., 4.7627998e+03, 0.0000000e+00, 0.0000000e+00),
(1097., 0, 1096, 'TO_DRAINS', 0., 0.0000000e+00, 0.0000000e+00, 0.0000000e+00),
(1097., 0, 1096, 'TO_RECHARGE', 0., 0.0000000e+00, 0.0000000e+00, 0.0000000e+00),
(1097., 0, 1096, 'TO_ZONE_0', 0., 0.0000000e+00, 0.0000000e+00, 0.0000000e+00),
(1097., 0, 1096, 'TO_ZONE_1', 0., 0.0000000e+00, 3.2693188e+03, 1.9218604e+02),
(1097., 0, 1096, 'TO_ZONE_2', 0., 3.4751235e+03, 0.0000000e+00, 1.5280482e+03),
(1097., 0, 1096, 'TO_ZONE_3', 0., 1.3860045e+02, 1.7646553e+03, 0.0000000e+00),
(1097., 0, 1096, 'TOTAL_OUT', 0., 8.6070723e+03, 5.2496758e+03, 2.0193480e+03),
(1097., 0, 1096, 'IN-OUT', 0., 1.3671875e-02, 9.7656250e-04, 2.3193359e-03),
(1097., 0, 1096, 'PERCENT_DISCREPANCY', nan, 1.5884454e-04, 1.8602341e-05, 1.
↪1485574e-04)],
dtype=[('totim', '<f4'), ('time_step', '<i4'), ('stress_period', '<i4'), ('name', '
↪<U50'), ('ZONE_0', '<f4'), ('ZONE_1', '<f4'), ('ZONE_2', '<f4'), ('ZONE_3', '<f4')]]

```

```
[5]: # Get a list of the unique budget record names
```

```
zb.get_record_names()
```

```
[5]: array(['FROM_CONSTANT_HEAD', 'FROM_DRAINS', 'FROM_RECHARGE',
'FROM_STORAGE', 'FROM_WELLS', 'FROM_ZONE_0', 'FROM_ZONE_1',
'FROM_ZONE_2', 'FROM_ZONE_3', 'IN-OUT', 'PERCENT_DISCREPANCY',
'TOTAL_IN', 'TOTAL_OUT', 'TO_CONSTANT_HEAD', 'TO_DRAINS',
'TO_RECHARGE', 'TO_STORAGE', 'TO_WELLS', 'TO_ZONE_0', 'TO_ZONE_1',
'TO_ZONE_2', 'TO_ZONE_3'], dtype='<U50')
```

```
[6]: # Look at a subset of fluxes
```

```
names = ["FROM_RECHARGE", "FROM_ZONE_1", "FROM_ZONE_3"]
```

```
zb.get_budget(names=names)
```

```
[6]: array([(1097., 0, 1096, 'FROM_RECHARGE', 0., 5145.5815 , 14.936376, 29.872751),
(1097., 0, 1096, 'FROM_ZONE_1', 0., 0. , 3475.1235 , 138.60045 ),
(1097., 0, 1096, 'FROM_ZONE_3', 0., 192.18604, 1528.0482 , 0. )],
dtype=[('totim', '<f4'), ('time_step', '<i4'), ('stress_period', '<i4'), ('name', '
↪<U50'), ('ZONE_0', '<f4'), ('ZONE_1', '<f4'), ('ZONE_2', '<f4'), ('ZONE_3', '<f4')]]

```

```
[7]: # Look at fluxes in from zone 2
```

```
names = ["FROM_RECHARGE", "FROM_ZONE_1", "FROM_ZONE_3"]
```

```
zones = ["ZONE_2"]
```

```
zb.get_budget(names=names, zones=zones)
```

```
[7]: array([(1097., 0, 1096, 'FROM_RECHARGE', 14.936376),
(1097., 0, 1096, 'FROM_ZONE_1', 3475.1235 ),
(1097., 0, 1096, 'FROM_ZONE_3', 1528.0482 )],

```

(continues on next page)

(continued from previous page)

```
dtype={'names': ['totim', 'time_step', 'stress_period', 'name', 'ZONE_2'], 'formats': ['<f4', '<i4', '<i4', '<U50', '<f4'], 'offsets': [0, 4, 8, 12, 220], 'itemsizes': [228]}
```

```
[8]: # Look at all of the mass-balance records
names = ["TOTAL_IN", "TOTAL_OUT", "IN-OUT", "PERCENT_DISCREPANCY"]
zb.get_budget(names=names)

[8]: array([(1097., 0, 1096, 'TOTAL_IN', 0., 8.6070859e+03, 5.249675e+03, 2.0193457e+03),
          (1097., 0, 1096, 'TOTAL_OUT', 0., 8.6070723e+03, 5.249676e+03, 2.0193480e+03),
          (1097., 0, 1096, 'IN-OUT', 0., 1.3671875e-02, 9.765625e-04, 2.3193359e-03),
          (1097., 0, 1096, 'PERCENT_DISCREPANCY', nan, 1.5884454e-04, 1.860234e-05, 1.
          ↪1485574e-04)],
          dtype=[('totim', '<f4'), ('time_step', '<i4'), ('stress_period', '<i4'), ('name', '
          ↪<U50'), ('ZONE_0', '<f4'), ('ZONE_1', '<f4'), ('ZONE_2', '<f4'), ('ZONE_3', '<f4')])
```

Convert Units

The ZoneBudget class supports the use of mathematical operators and returns a new copy of the object.

```
[9]: cmd = flopy.utils.ZoneBudget(cbc_f, zon, kstpker=(0, 0))
cfd = cmd / 35.3147
inyr = (cfd / (250 * 250)) * 365 * 12

cmdbud = cmd.get_budget()
cfdbud = cfd.get_budget()
inyrbud = inyr.get_budget()

names = ["FROM_RECHARGE"]
rowidx = np.in1d(cmdbud["name"], names)
colidx = "ZONE_1"

print("{:,.1f} cubic meters/day".format(cmdbud[rowidx][colidx][0]))
print("{:,.1f} cubic feet/day".format(cfdbud[rowidx][colidx][0]))
print("{:,.1f} inches/year".format(inyrbud[rowidx][colidx][0]))

6,222.7 cubic meters/day
176.2 cubic feet/day
12.3 inches/year
```

```
[10]: cmd is cfd
```

```
[10]: False
```


Alias Names

A dictionary of {zone: "alias"} pairs can be passed to replace the typical "ZONE_X" fieldnames of the ZoneBudget structured array with more descriptive names.

```
[11]: aliases = {1: "SURF", 2: "CONF", 3: "UFA"}
zb = flopy.utils.ZoneBudget(cbc_f, zon, totim=[1097.0], aliases=aliases)
zb.get_budget()

[11]: array([(1097., 0, 1096, 'FROM_STORAGE', 0., 0.00000000e+00, 0.00000000e+00, 0.
→00000000e+00),
      (1097., 0, 1096, 'FROM_CONSTANT_HEAD', 0., 0.00000000e+00, 2.3156659e+02, 8.
→6217201e+01),
      (1097., 0, 1096, 'FROM_WELLS', 0., 0.00000000e+00, 0.00000000e+00, 0.00000000e+00),
      (1097., 0, 1096, 'FROM_DRAINS', 0., 0.00000000e+00, 0.00000000e+00, 0.00000000e+00),
      (1097., 0, 1096, 'FROM_RECHARGE', 0., 5.1455815e+03, 1.4936376e+01, 2.
→9872751e+01),
      (1097., 0, 1096, 'FROM_ZONE_0', 0., 0.00000000e+00, 0.00000000e+00, 0.00000000e+00),
      (1097., 0, 1096, 'FROM_SURF', 0., 0.00000000e+00, 3.4751235e+03, 1.3860045e+02),
      (1097., 0, 1096, 'FROM_CONF', 0., 3.2693188e+03, 0.00000000e+00, 1.7646553e+03),
      (1097., 0, 1096, 'FROM_UFA', 0., 1.9218604e+02, 1.5280482e+03, 0.00000000e+00),
      (1097., 0, 1096, 'TOTAL_IN', 0., 8.6070859e+03, 5.2496748e+03, 2.0193457e+03),
      (1097., 0, 1096, 'TO_STORAGE', 0., 0.00000000e+00, 0.00000000e+00, 0.00000000e+00),
      (1097., 0, 1096, 'TO_CONSTANT_HEAD', 0., 2.3054836e+02, 2.1570151e+02, 2.
→9911380e+02),
      (1097., 0, 1096, 'TO_WELLS', 0., 4.7627998e+03, 0.00000000e+00, 0.00000000e+00),
      (1097., 0, 1096, 'TO_DRAINS', 0., 0.00000000e+00, 0.00000000e+00, 0.00000000e+00),
      (1097., 0, 1096, 'TO_RECHARGE', 0., 0.00000000e+00, 0.00000000e+00, 0.00000000e+00),
      (1097., 0, 1096, 'TO_ZONE_0', 0., 0.00000000e+00, 0.00000000e+00, 0.00000000e+00),
      (1097., 0, 1096, 'TO_SURF', 0., 0.00000000e+00, 3.2693188e+03, 1.9218604e+02),
      (1097., 0, 1096, 'TO_CONF', 0., 3.4751235e+03, 0.00000000e+00, 1.5280482e+03),
      (1097., 0, 1096, 'TO_UFA', 0., 1.3860045e+02, 1.7646553e+03, 0.00000000e+00),
      (1097., 0, 1096, 'TOTAL_OUT', 0., 8.6070723e+03, 5.2496758e+03, 2.0193480e+03),
      (1097., 0, 1096, 'IN-OUT', 0., 1.3671875e-02, 9.7656250e-04, 2.3193359e-03),
      (1097., 0, 1096, 'PERCENT_DISCREPANCY', nan, 1.5884454e-04, 1.8602341e-05, 1.
→1485574e-04)],
      dtype=[('totim', '<f4'), ('time_step', '<i4'), ('stress_period', '<i4'), ('name', '
→<U50'), ('ZONE_0', '<f4'), ('SURF', '<f4'), ('CONF', '<f4'), ('UFA', '<f4')])
```

Return the Budgets as a Pandas DataFrame

Set kstpkper and totim keyword args to None (or omit) to return all times. The get_dataframes() method will return a DataFrame multi-indexed on totim and name.

```
[12]: aliases = {1: "SURF", 2: "CONF", 3: "UFA"}
times = list(range(1092, 1097 + 1))
zb = flopy.utils.ZoneBudget(cbc_f, zon, totim=times, aliases=aliases)
zb.get_dataframes()
```

```
[12]:
```

		ZONE_0	SURF	CONF	UFA	
totim	name					
	1092.0	FROM_STORAGE	0.0	393.480286	230.476242	228.273621
		FROM_CONSTANT_HEAD	0.0	0.000000	13.225761	5.042325

(continues on next page)

(continued from previous page)

```

FROM_WELLS      0.0      0.000000      0.000000      0.000000
FROM_DRAINS     0.0      0.000000      0.000000      0.000000
FROM_RECHARGE   0.0  6018.483887      17.470200      34.940399
...
1097.0 TO_CONF      0.0  3475.123535      0.000000      1528.048218
      TO_UFA        0.0   138.600449     1764.655273      0.000000
      TOTAL_OUT     0.0  8607.072266     5249.675781     2019.348022
      IN-OUT        0.0    0.013672      0.000977      0.002319
      PERCENT_DISCREPANCY  NaN    0.000159      0.000019      0.000115

```

[132 rows x 4 columns]

Slice the multi-index dataframe to retrieve a subset of the budget. NOTE: We can pass “names” directly to the `get_dataframes()` method to return a subset of records. By omitting the “FROM_” or “TO_” prefix we get both.

```

[13]: dateidx1 = 1095.0
      dateidx2 = 1097.0
      names = ["FROM_RECHARGE", "TO_WELLS", "CONSTANT_HEAD"]
      zones = ["SURF", "CONF"]
      df = zb.get_dataframes(names=names)
      df.loc[(slice(dateidx1, dateidx2), slice(None)), :][zones]

```

```

[13]:
      totim  name      SURF      CONF
1095.0 FROM_CONSTANT_HEAD  0.000000  14.531923
      FROM_RECHARGE    5010.149414  14.543249
      TO_CONSTANT_HEAD  664.912598  493.175781
      TO_WELLS        794.582886   0.000000
1096.0 FROM_CONSTANT_HEAD  0.000000   6.501888
      FROM_RECHARGE    6115.663086  17.752289
      TO_CONSTANT_HEAD  690.230591  511.618652
      TO_WELLS        1373.782715   0.000000
1097.0 FROM_CONSTANT_HEAD  0.000000  231.566589
      FROM_RECHARGE    5145.581543  14.936376
      TO_CONSTANT_HEAD  230.548355  215.701508
      TO_WELLS        4762.799805   0.000000

```

Look at pumpage (TO_WELLS) as a percentage of recharge (FROM_RECHARGE)

```

[14]: dateidx1 = 1095.0
      dateidx2 = 1097.0
      zones = ["SURF"]

      # Pull out the individual records of interest
      rech = df.loc[(slice(dateidx1, dateidx2), ["FROM_RECHARGE"]), :][zones]
      pump = df.loc[(slice(dateidx1, dateidx2), ["TO_WELLS"]), :][zones]

      # Remove the "record" field from the index so we can
      # take the difference of the two DataFrames
      rech = rech.reset_index()
      rech = rech.set_index(["totim"])
      rech = rech[zones]

```

(continues on next page)

(continued from previous page)

```

pump = pump.reset_index()
pump = pump.set_index(["totim"])
pump = pump[zones] * -1

```

```

# Compute pumping as a percentage of recharge

```

```

(pump / rech) * 100.0

```

```

[14]:          SURF
totim
1095.0 -15.859466
1096.0 -22.463348
1097.0 -92.560966

```

Pass `start_datetime` and `timeunit` keyword arguments to return a dataframe with a datetime multi-index

```

[15]: dateidx1 = pd.Timestamp("1972-12-29")
dateidx2 = pd.Timestamp("1972-12-30")
names = ["FROM_RECHARGE", "TO_WELLS", "CONSTANT_HEAD"]
zones = ["SURF", "CONF"]
df = zb.get_dataframes(start_datetime="1970-01-01", timeunit="D", names=names)
df.loc[(slice(dateidx1, dateidx2), slice(None)), :][zones]

```

```

[15]:          SURF          CONF
datetime  name
1972-12-29 FROM_CONSTANT_HEAD    0.000000  16.905813
          FROM_RECHARGE    4203.679199  12.202263
          TO_CONSTANT_HEAD    655.814514  487.094849
          TO_WELLS    1930.483154    0.000000
1972-12-30 FROM_CONSTANT_HEAD    0.000000  18.877954
          FROM_RECHARGE    4047.502441  11.748919
          TO_CONSTANT_HEAD    650.441589  482.853638
          TO_WELLS    1279.166382    0.000000

```

Pass `index_key` to indicate which fields to use in the multi-index (default is "totim"; valid keys are "totim" and "kstpker")

```

[16]: df = zb.get_dataframes(index_key="kstpker")
df.head()

```

```

[16]:          ZONE_0          SURF          CONF  \
time_step stress_period name
0          1091  FROM_STORAGE    0.0  393.480286  230.476242
          FROM_CONSTANT_HEAD    0.0    0.000000  13.225761
          FROM_WELLS    0.0    0.000000    0.000000
          FROM_DRAINS    0.0    0.000000    0.000000
          FROM_RECHARGE    0.0  6018.483887  17.470200

          UFA
time_step stress_period name
0          1091  FROM_STORAGE  228.273621
          FROM_CONSTANT_HEAD    5.042325
          FROM_WELLS    0.000000
          FROM_DRAINS    0.000000

```

(continues on next page)

(continued from previous page)

FROM_RECHARGE	34.940399
---------------	-----------

Write Budget Output to CSV

We can write the resulting recarray to a csv file with the `.to_csv()` method of the `ZoneBudget` object.

```
[17]: zb = flopy.utils.ZoneBudget(cbc_f, zon, kstpker=[(0, 0), (0, 1096)])
f_out = workspace / "Example_output.csv"
zb.to_csv(f_out)
```

```
# Read the file in to see the contents
```

```
try:
    import pandas as pd

    print(pd.read_csv(f_out).to_string(index=False))
except:
    with open(f_out, "r") as f:
        for line in f.readlines():
            print("\t".join(line.split(",")))
```

totim	time_step	stress_period	name	ZONE_0	ZONE_1	ZONE_2
↪ ZONE_3						
1.0	0	0	FROM_STORAGE	0.0	0.000000	0.000000
↪ 0.000000						
1.0	0	0	FROM_CONSTANT_HEAD	0.0	0.000000	0.000000
↪ 0.000000						
1.0	0	0	FROM_WELLS	0.0	0.000000	0.000000
↪ 0.000000						
1.0	0	0	FROM_DRAINS	0.0	0.000000	0.000000
↪ 0.000000						
1.0	0	0	FROM_RECHARGE	0.0	6222.673300	18.062912
↪ 36.125824						
1.0	0	0	FROM_ZONE_0	0.0	0.000000	0.000000
↪ 0.000000						
1.0	0	0	FROM_ZONE_1	0.0	0.000000	4275.257300
↪ 491.945070						
1.0	0	0	FROM_ZONE_2	0.0	2744.821800	0.000000
↪ 2115.654000						
1.0	0	0	FROM_ZONE_3	0.0	451.545720	1215.952300
↪ 0.000000						
1.0	0	0	TOTAL_IN	0.0	9419.041000	5509.272500
↪ 2643.725000						
1.0	0	0	TO_STORAGE	0.0	0.000000	0.000000
↪ 0.000000						
1.0	0	0	TO_CONSTANT_HEAD	0.0	821.283200	648.806100
↪ 976.233600						
1.0	0	0	TO_WELLS	0.0	0.000000	0.000000
↪ 0.000000						
1.0	0	0	TO_DRAINS	0.0	3832.150000	0.000000
↪ 0.000000						

(continues on next page)

(continued from previous page)

1.0	0	0	TO_RECHARGE	0.0	0.000000	0.000000	┐
→0.000000							
1.0	0	0	TO_ZONE_0	0.0	0.000000	0.000000	┐
→0.000000							
1.0	0	0	TO_ZONE_1	0.0	0.000000	2744.821800	┐
→451.545720							
1.0	0	0	TO_ZONE_2	0.0	4275.257300	0.000000	┐
→1215.952300							
1.0	0	0	TO_ZONE_3	0.0	491.945070	2115.654000	┐
→0.000000							
1.0	0	0	TOTAL_OUT	0.0	9420.636000	5509.282000	┐
→2643.731400							
1.0	0	0	IN-OUT	0.0	1.594727	0.009766	┐
→0.006348							
1.0	0	0	PERCENT_DISCREPANCY	NaN	0.016929	0.000177	┐
→0.000240							
1097.0	0	1096	FROM_STORAGE	0.0	0.000000	0.000000	┐
→0.000000							
1097.0	0	1096	FROM_CONSTANT_HEAD	0.0	0.000000	231.566590	┐
→86.217200							
1097.0	0	1096	FROM_WELLS	0.0	0.000000	0.000000	┐
→0.000000							
1097.0	0	1096	FROM_DRAINS	0.0	0.000000	0.000000	┐
→0.000000							
1097.0	0	1096	FROM_RECHARGE	0.0	5145.581500	14.936376	┐
→29.872751							
1097.0	0	1096	FROM_ZONE_0	0.0	0.000000	0.000000	┐
→0.000000							
1097.0	0	1096	FROM_ZONE_1	0.0	0.000000	3475.123500	┐
→138.600450							
1097.0	0	1096	FROM_ZONE_2	0.0	3269.318800	0.000000	┐
→1764.655300							
1097.0	0	1096	FROM_ZONE_3	0.0	192.186040	1528.048200	┐
→0.000000							
1097.0	0	1096	TOTAL_IN	0.0	8607.086000	5249.675000	┐
→2019.345700							
1097.0	0	1096	TO_STORAGE	0.0	0.000000	0.000000	┐
→0.000000							
1097.0	0	1096	TO_CONSTANT_HEAD	0.0	230.548360	215.701500	┐
→299.113800							
1097.0	0	1096	TO_WELLS	0.0	4762.800000	0.000000	┐
→0.000000							
1097.0	0	1096	TO_DRAINS	0.0	0.000000	0.000000	┐
→0.000000							
1097.0	0	1096	TO_RECHARGE	0.0	0.000000	0.000000	┐
→0.000000							
1097.0	0	1096	TO_ZONE_0	0.0	0.000000	0.000000	┐
→0.000000							
1097.0	0	1096	TO_ZONE_1	0.0	0.000000	3269.318800	┐
→192.186040							
1097.0	0	1096	TO_ZONE_2	0.0	3475.123500	0.000000	┐
→1528.048200							

(continues on next page)

(continued from previous page)

1097.0	0	1096	TO_ZONE_3	0.0	138.600450	1764.655300	↵
↵0.000000							
1097.0	0	1096	TOTAL_OUT	0.0	8607.072000	5249.676000	↵
↵2019.348000							
1097.0	0	1096	IN-OUT	0.0	0.013672	0.000977	↵
↵0.002319							
1097.0	0	1096	PERCENT_DISCREPANCY	NaN	0.000159	0.000019	↵
↵0.000115							

Net Budget

Using the “net” keyword argument, we can request a net budget for each zone/record name or for a subset of zones and record names. Note that we can identify the record names we want without the added “_IN” or “_OUT” string suffix.

```
[18]: zon = np.ones((nlay, nrow, ncol), int)
      zon[1, :, :] = 2
      zon[2, :, :] = 3

      aliases = {1: "SURF", 2: "CONF", 3: "UFA"}
      times = list(range(1092, 1097 + 1))
      zb = flopy.utils.ZoneBudget(cbc_f, zon, totim=times, aliases=aliases)
      zb.get_budget(names=["STORAGE", "WELLS"], zones=["SURF", "UFA"], net=True)
```

```
[18]: array([(1092., 0, 1091, 'STORAGE', -386.15085, -425.65155),
      (1092., 0, 1091, 'WELLS', -2829.812, 0. ),
      (1093., 0, 1092, 'STORAGE', -12.50354, -151.3594 ),
      (1093., 0, 1092, 'WELLS', -1930.4832, 0. ),
      (1094., 0, 1093, 'STORAGE', -198.92935, -270.8744 ),
      (1094., 0, 1093, 'WELLS', -1279.1664, 0. ),
      (1095., 0, 1094, 'STORAGE', -718.4885, -604.8537 ),
      (1095., 0, 1094, 'WELLS', -794.5829, 0. ),
      (1096., 0, 1095, 'STORAGE', -855.1075, -622.88477),
      (1096., 0, 1095, 'WELLS', -1373.7827, 0. ),
      (1097., 0, 1096, 'STORAGE', 0. , 0. ),
      (1097., 0, 1096, 'WELLS', -4762.8, 0. )],
      dtype={'names': ['totim', 'time_step', 'stress_period', 'name', 'SURF', 'UFA'],
      ↵'formats': ['<f4', '<i4', '<i4', '<U50', '<f4', '<f4'], 'offsets': [0, 4, 8, 12, 212, ↵
      ↵220], 'itemsize': 224})
```

```
[19]: df = zb.get_dataframes(
      names=["STORAGE", "WELLS"], zones=["SURF", "UFA"], net=True
      )
      df.head(6)
```

```
[19]:
```

		SURF	UFA
totim	name		
1092.0	STORAGE	-386.150848	-425.651550
	WELLS	-2829.812012	0.000000
1093.0	STORAGE	-12.503540	-151.359406
	WELLS	-1930.483154	0.000000

(continues on next page)

(continued from previous page)

```
1094.0 STORAGE -198.929352 -270.874390
      WELLS -1279.166382 0.000000
```

Plot Budget Components

The following is a function that can be used to better visualize the budget components using matplotlib.

```
[20]: def tick_label_formatter_comma_sep(x, pos):
      return "{:,.0f}".format(x)

def volumetric_budget_bar_plot(values_in, values_out, labels, **kwargs):
    if "ax" in kwargs:
        ax = kwargs.pop("ax")
    else:
        ax = plt.gca()

    x_pos = np.arange(len(values_in))
    rects_in = ax.bar(x_pos, values_in, align="center", alpha=0.5)

    x_pos = np.arange(len(values_out))
    rects_out = ax.bar(x_pos, values_out, align="center", alpha=0.5)

    plt.xticks(list(x_pos), labels)
    ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=90)
    ax.get_yaxis().set_major_formatter(
        mpl.ticker.FuncFormatter(tick_label_formatter_comma_sep)
    )

    ymin, ymax = ax.get_ylim()
    if ymax != 0:
        if abs(ymin) / ymax < 0.33:
            ymin = -(ymax * 0.5)
        else:
            ymin *= 1.35
    else:
        ymin *= 1.35
    plt.ylim([ymin, ymax * 1.25])

    for i, rect in enumerate(rects_in):
        label = "{:,.0f}".format(values_in[i])
        height = values_in[i]
        x = rect.get_x() + rect.get_width() / 2
        y = height + (0.02 * ymax)
        vertical_alignment = "bottom"
        horizontal_alignment = "center"
        ax.text(
            x,
            y,
            label,
```

(continues on next page)

(continued from previous page)

```

        ha=horizontal_alignment,
        va=vertical_alignment,
        rotation=90,
    )

    for i, rect in enumerate(rects_out):
        label = "{:,.0f}".format(values_out[i])
        height = values_out[i]
        x = rect.get_x() + rect.get_width() / 2
        y = height + (0.02 * ymin)
        vertical_alignment = "top"
        horizontal_alignment = "center"
        ax.text(
            x,
            y,
            label,
            ha=horizontal_alignment,
            va=vertical_alignment,
            rotation=90,
        )

    # horizontal line indicating zero
    ax.plot(
        [
            rects_in[0].get_x() - rects_in[0].get_width() / 2,
            rects_in[-1].get_x() + rects_in[-1].get_width(),
        ],
        [0, 0],
        "k",
    )

    return rects_in, rects_out

```

```

[21]: fig = plt.figure(figsize=(16, 5))

times = [2.0, 500.0, 1000.0, 1095.0]

for idx, t in enumerate(times):
    ax = fig.add_subplot(1, len(times), idx + 1)

    zb = flopy.utils.ZoneBudget(
        cbc_f, zon, kstpker=None, totim=t, aliases=aliases
    )

    recname = "STORAGE"
    values_in = zb.get_dataframes(names="FROM_{}".format(recname)).T.squeeze()
    values_out = (
        zb.get_dataframes(names="TO_{}".format(recname)).T.squeeze() * -1
    )
    labels = values_in.index.tolist()

    rects_in, rects_out = volumetric_budget_bar_plot(

```

(continues on next page)

(continued from previous page)

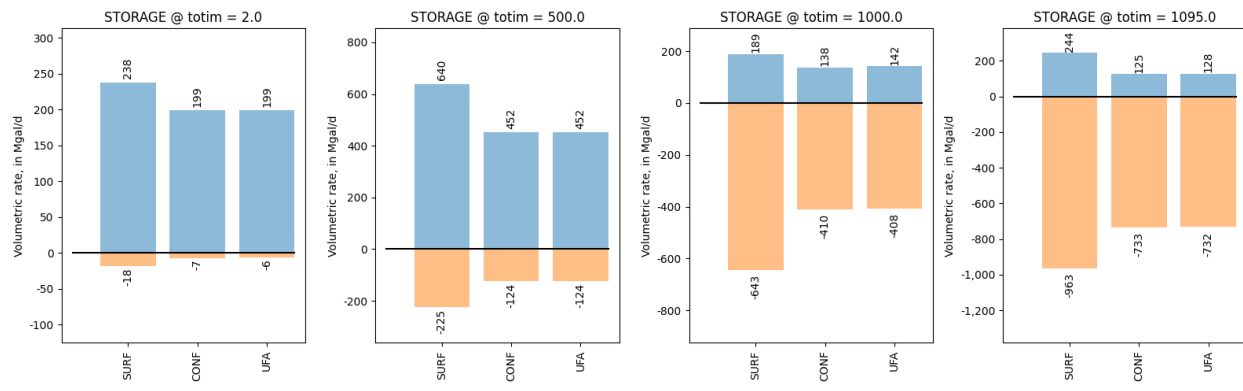
```

        values_in, values_out, labels, ax=ax
    )

    plt.ylabel("Volumetric rate, in Mgal/d")
    plt.title("{} @ totim = {}".format(recname, t))

plt.tight_layout()
plt.show()

```



Zonebudget for Modflow 6 (ZoneBudget6)

This section shows how to build and run a Zonebudget when working with a MODFLOW 6 model.

First let's load a model

```

[22]: mf6_exe = "mf6"
      zb6_exe = "zbud6"

sim_ws = proj_root / "examples" / "data" / "mf6-freyberg"
cpth = workspace / "zbud6"
cpth.mkdir()

sim = flopy.mf6.MFSimulation.load(sim_ws=sim_ws, exe_name=mf6_exe)
sim.simulation_data.mfpath.set_sim_path(cpth)
sim.write_simulation()
success, buff = sim.run_simulation(silent=True, report=True)
assert success, "Failed to run"
for line in buff:
    print(line)

loading simulation...
  loading simulation name file...
  loading tdis package...
  loading model gwf6...
  loading package dis...
  loading package ic...
WARNING: Block "options" is not a valid block name for file type ic.
  loading package oc...
  loading package npf...

```

(continues on next page)

(continued from previous page)

```

loading package sto...
loading package chd...
loading package riv...
loading package wel...
loading package rch...
loading solution package freyberg...
WARNING: MFFileMgt's set_sim_path has been deprecated. Please use MFSimulation's set_
↪sim_path in the future.
writing simulation...
writing simulation name file...
writing simulation tdis package...
writing solution package freyberg...
writing model freyberg...
writing model name file...
writing package dis...
writing package ic...
writing package oc...
writing package npf...
writing package sto...
writing package chd-1...
writing package riv-1...
writing package wel-1...
writing package rch-1...

```

```

MODFLOW 6
U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
VERSION 6.4.2 06/28/2023

```

```

MODFLOW 6 compiled Jul 05 2023 20:29:14 with Intel(R) Fortran Intel(R) 64
Compiler Classic for applications running on Intel(R) 64, Version 2021.7.0
Build 20220726_000000

```

This software has been approved for release by the U.S. Geological Survey (USGS). Although the software has been subjected to rigorous review, the USGS reserves the right to update the software as needed pursuant to further analysis and review. No warranty, expressed or implied, is made by the USGS or the U.S. Government as to the functionality of the software and related material nor shall the fact of release constitute any such warranty. Furthermore, the software is released on condition that neither the USGS nor the U.S. Government shall be held liable for any damages resulting from its authorized or unauthorized use. Also refer to the USGS Water Resources Software User Rights Notice for complete use, copyright, and distribution information.

```
Run start date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:51:43
```

```
Writing simulation list file: mfsim.lst
Using Simulation name file: mfsim.nam
```

```
Solving: Stress period:      1      Time step:      1
```

(continues on next page)

(continued from previous page)

```
Run end date and time (yyyy/mm/dd hh:mm:ss): 2023/09/30 20:51:43
Elapsed run time: 0.069 Seconds

Normal termination of simulation.
```

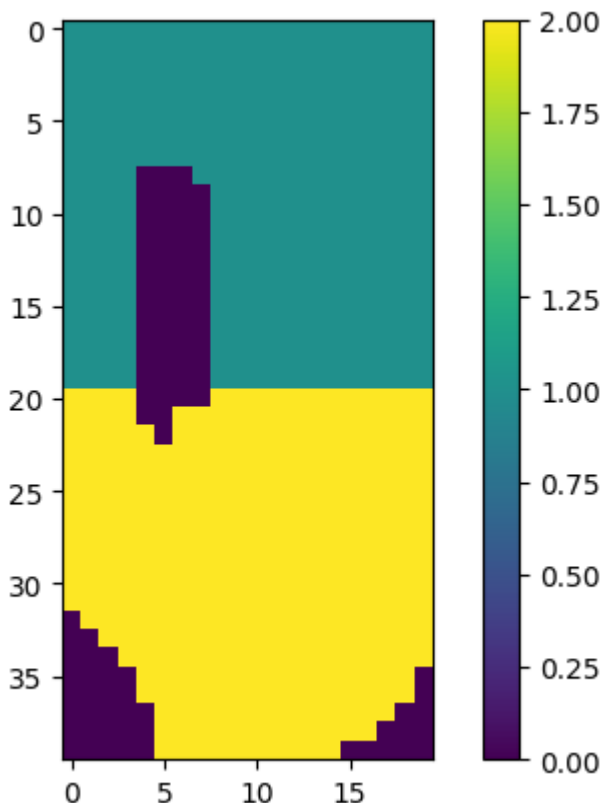
Use the the .output model attribute to create a zonebudget model

The .output attribute allows the user to access model output and create zonebudget models easily. The user only needs to pass in a zone array to create a zonebudget model!

```
[23]: # let's get our idomain array from the model, split it into two zones, and use it as a
      ↪ zone array
ml = sim.get_model("freyberg")
zones = ml.modelgrid.idomain
zones[0, 20:] = np.where(zones[0, 20:] != 0, 2, 0)

plt.imshow(zones[0])
plt.colorbar()
```

```
[23]: <matplotlib.colorbar.Colorbar at 0x7fe46790cbb0>
```



```
[24]: # now let's build a zonebudget model and run it!
zonbud = ml.output.zonebudget(zones)
zonbud.change_model_ws(cpth)
```

(continues on next page)

(continued from previous page)

```
zonbud.write_input()
success, buff = zonbud.run_model(exe_name=zb6_exe, silent=True)
```

Getting the zonebudget output

We can then get the output as a recarray using the `.get_budget()` method or as a pandas dataframe using the `.get_dataframes()` method.

```
[25]: zonbud.get_budget()
```

```
[25]: rec.array([(10., 0, 0, 'STO_SS_IN', 0., 0.),
                 (10., 0, 0, 'STO_SY_IN', 0., 0.),
                 (10., 0, 0, 'DATA_SPDIS_IN', 0., 0.),
                 (10., 0, 0, 'WEL_IN', 0., 0.),
                 (10., 0, 0, 'RIV_IN', 0.00419403, 0.),
                 (10., 0, 0, 'RCHA_IN', 0.0353, 0.0342),
                 (10., 0, 0, 'CHD_IN', 0., 0.00017814),
                 (10., 0, 0, 'STO_SS_OUT', 0., 0.),
                 (10., 0, 0, 'STO_SY_OUT', 0., 0.),
                 (10., 0, 0, 'DATA_SPDIS_OUT', 0., 0.),
                 (10., 0, 0, 'WEL_OUT', 0.0162, 0.00585),
                 (10., 0, 0, 'RIV_OUT', 0.02102198, 0.02637233),
                 (10., 0, 0, 'RCHA_OUT', 0., 0.),
                 (10., 0, 0, 'CHD_OUT', 0., 0.00442785),
                 (10., 0, 0, 'FROM_ZONE_0', 0., 0.),
                 (10., 0, 0, 'FROM_ZONE_1', 0., 0.00405829),
                 (10., 0, 0, 'FROM_ZONE_2', 0.00178624, 0.),
                 (10., 0, 0, 'TO_ZONE_0', 0., 0.),
                 (10., 0, 0, 'TO_ZONE_1', 0., 0.00178624),
                 (10., 0, 0, 'TO_ZONE_2', 0.00405829, 0.)],
          dtype=[('totim', '<f8'), ('time_step', '<i8'), ('stress_period', '<i8'), ('name', '<f8'), ('ZONE_1', '<f8'), ('ZONE_2', '<f8')])
```

```
[26]: # get the net flux using net=True flag
zonbud.get_dataframes(net=True)
```

```
[26]:
```

		ZONE_1	ZONE_2
totim	name		
10.0	STO_SS	0.000000	0.000000
	STO_SY	0.000000	0.000000
	DATA_SPDIS	0.000000	0.000000
	WEL	-0.016200	-0.005850
	RIV	-0.016828	-0.026372
	RCHA	0.035300	0.034200
	CHD	0.000000	-0.004250
	ZONE_0	0.000000	0.000000
	ZONE_1	0.000000	0.002272
	ZONE_2	-0.002272	0.000000

```
[27]: # we can also pivot the data into a spreadsheet like format
zonbud.get_dataframes(net=True, pivot=True)
```

```
[27]:
```

	totim	kper	kstp	zone	CHD	DATA_SPDIS	RCHA	RIV	STO_SS	\
0	10.0	0	0	1	0.000000	0.0	0.0353	-0.016828	0.0	
1	10.0	0	0	2	-0.00425	0.0	0.0342	-0.026372	0.0	

	STO_SY	WEL	ZONE_0	ZONE_1	ZONE_2
0	0.0	-0.01620	0.0	0.000000	-0.002272
1	0.0	-0.00585	0.0	0.002272	0.000000

```
[28]: # or get a volumetric budget by supplying modeltime
mt = ml.modeltime

# budget recarray must be pivoted to get volumetric budget!
zonbud.get_volumetric_budget(
    mt, recarray=zonbud.get_budget(net=True, pivot=True)
)
```

```
[28]:
```

	totim	kper	kstp	zone	CHD	DATA_SPDIS	RCHA	RIV	STO_SS	\
0	10.0	0	0	1	0.000000	0.0	0.353	-0.168280	0.0	
1	10.0	0	0	2	-0.042497	0.0	0.342	-0.263723	0.0	

	STO_SY	WEL	ZONE_0	ZONE_1	ZONE_2
0	0.0	-0.1620	0.0	0.000000	-0.022721
1	0.0	-0.0585	0.0	0.022721	0.000000

```
[29]: try:
        # ignore PermissionError on Windows
        temp_dir.cleanup()
    except:
        pass
```


FREQUENTLY ASKED QUESTIONS

7.1 Executable does not exist

One common error is missing executables. FloPy can drive MODFLOW and a number of related programs. In order for you to run a model you have created or loaded using FloPy, the desired executable must be available in your path. Executable paths must either be provided explicitly, or if an executable name is provided, the binary must be on the system path.

You can test if the executable is available using the *which* function. For example, to test if the *mf2005* executable is available in your path, use:

```
flopy.which("mf2005") # equivalent to shutil.which("mf2005")
```

If the executable is found, its path is returned, otherwise the function returns an empty string. If you receive the latter, you should:

1. Check that you have spelled the executable correctly.
2. If you have spelled the executable correctly then you need to move the executable into your working directory or into a directory in your path.
3. **If you have spelled the executable correctly but don't have the executable. Your options are:**
 - Download a precompiled version of the executable. Precompiled versions of MODFLOW-based codes are available from the U.S. Geological Survey for the Windows operating system.
 - Compile the source code (available from the U.S. Geological Survey) for the Windows, OS X, Linux, and UNIX operating systems and place the compiled executable in the working directory or a directory contained in your path (for example, */Users/jdhughes/.local/bin/* as indicated above).

You can get a list of the directories in your system path using:

```
os.getenv("PATH")
```

There is a *get-modflow* command line utility that can be used to download MODFLOW executables. See the [get-modflow documentation](#) for more information.

API REFERENCE

8.1 MODFLOW 6

FloPy for MODFLOW 6 allows for the construction of multi-model simulations. In order to construct a MODFLOW 6 simulation using FloPy, first construct a simulation (MFSimulation) object. Then construct the MODFLOW 6 models (Modflowgwf and Modflowgwt) and the packages, like TDIS, that are associated with the simulation. Finally, construct the packages that are associated with each of your models.

8.1.1 MODFLOW 6 Base Classes

FloPy for MODFLOW 6 is object oriented code that uses inheritance. The FloPy classes used to define different types models and packages share common code that is defined in these base classes.

Contents:

flopy.mf6.mfbase module

Base classes for Modflow 6

```
class ExtFileAction(value, names=None, *, module=None, qualname=None, type=None, start=1,
                    boundary=None)
```

Bases: [Enum](#)

Defines what to do with external files when the simulation or model's path change.

copy_all = 1

copy_none = 2

copy_relative_paths = 3

```
exception FlopyException(error, location="")
```

Bases: [Exception](#)

General FloPy exception

```
exception MFDataException(model=None, package=None, path=None, current_process=None,
                           data_element=None, method_caught_in=None, org_type=None,
                           org_value=None, org_traceback=None, message=None, debug=None,
                           mfdata_except=None)
```

Bases: `Exception`

Exception with MODFLOW data. Exception includes detailed error information.

class `MFFileMgmt`(*path*: `str` | `PathLike`, *mfsim*=None)

Bases: `object`

Class containing MODFLOW path data

Parameters

path (`str` or `PathLike`) - Path on disk to the simulation

model_relative_path

Dictionary of relative paths to each model folder

Type

`dict`

add_ext_file(*file_path*, *model_name*)

Add an external file to the path list. For internal FloPy use, not intended for end user.

copy_files(*copy_relative_only*=True)

Copy files external to updated path.

Parameters

copy_relative_only (`bool`) - Only copy files with relative paths.

get_model_path(*key*, *last_loaded_path*=False)

Returns the model working path for the model key.

Parameters

- **key** (`str`) - Model name whose path flopy will retrieve
- **last_loaded_path** (`bool`) - Get the last path loaded by FloPy which may not be the most recent path.

Returns

model path

Return type

`str`

get_sim_path(*last_loaded_path*=False)

Get the simulation path.

get_updated_path(*external_file_path*, *model_name*, *ext_file_action*)

For internal FloPy use, not intended for end user.

resolve_path(*path*, *model_name*, *last_loaded_path*=False, *move_abs_paths*=False)

Resolve a simulation or model path. For internal FloPy use, not intended for end user.

set_last_accessed_model_path()

Set the last accessed model path to the current model path. For internal FloPy use, not intended for end user.

set_last_accessed_path()

Set the last accessed simulation path to the current simulation path. For internal FloPy use, not intended for end user.

set_sim_path(path: *str* | *PathLike*, internal_use=False)

Set the file path to the simulation files. Internal use only, call MFSimulation's set_sim_path method instead.

Parameters

path (*str* or *PathLike*) - Path to simulation folder

Return type

None

Examples

```
self.simulation_data.mfdata.set_sim_path('path/to/workspace')
```

strip_model_relative_path(model_name, path) → *str*

Strip out the model relative path part of *path*. For internal FloPy use, not intended for end user.

static unique_file_name(file_name, lookup)

Generate a unique file name. For internal FloPy use, not intended for end user.

class MFFilePath(file_path, model_name)

Bases: *object*

Class that stores a single file path along with the associated model name.

isabs()**exception MFInvalidTransientBlockHeaderException**

Bases: *Exception*

Exception occurs when parsing a transient block header

class PackageContainer(simulation_data, name)

Bases: *object*

Base class for any class containing packages.

Parameters

- **simulation_data** (*SimulationData*) - The simulation's *SimulationData* object
- **name** (*str*) - Name of the package container object

package_type_dict

Dictionary of packages by package type

Type

dictionary

package_name_dict

Dictionary of packages by package name

Type

dictionary

static `get_module_val(module, item, attrb)`

Static method that returns a python class module value. For internal FloPy use only, not intended for end users.

get_package(*name=None*)

Finds a package by package name, package key, package type, or partial package name. returns either a single package, a list of packages, or None.

Parameters

name (*str*) - Name of the package, 'RIV', 'LPF', etc.

Returns

pp

Return type

Package object

static `model_factory(model_type)`

Static method that returns the appropriate model type object based on the `model_type` string. For internal FloPy use only, not intended for end users.

Parameters

model_type (*str*) - Type of model that package is a part of

Returns

model

Return type

MFModel subclass

```
models_by_type = {'gwf': <class 'flopy.mf6.modflow.mfgwf.ModflowGwf'>, 'gwt': <class
'flopy.mf6.modflow.mfgwt.ModflowGwt'>}
```

```
modflow_models = [<class 'flopy.mf6.modflow.mfgwf.ModflowGwf'>, <class
'flopy.mf6.modflow.mfgwt.ModflowGwt'>]
```

```

modflow_packages = [<class 'flopy.mf6.modflow.mfnam.ModflowNam'>, <class
'flopy.mf6.modflow.mftdis.ModflowTdis'>, <class
'flopy.mf6.modflow.mfems.ModflowEms'>, <class 'flopy.mf6.modflow.mfgnc.ModflowGnc'>,
<class 'flopy.mf6.modflow.mfgnc.GncPackages'>, <class
'flopy.mf6.modflow.mfgwfapi.ModflowGwfapi'>, <class
'flopy.mf6.modflow.mfgwfbuy.ModflowGwfbuy'>, <class
'flopy.mf6.modflow.mfgwfchd.ModflowGwfchd'>, <class
'flopy.mf6.modflow.mfgwfcsb.ModflowGwfcsb'>, <class
'flopy.mf6.modflow.mfgwfdis.ModflowGwfdis'>, <class
'flopy.mf6.modflow.mfgwfdisu.ModflowGwfdisu'>, <class
'flopy.mf6.modflow.mfgwfdisv.ModflowGwfdisv'>, <class
'flopy.mf6.modflow.mfgwfdrn.ModflowGwfdrn'>, <class
'flopy.mf6.modflow.mfgwfevt.ModflowGwfevt'>, <class
'flopy.mf6.modflow.mfgwfevta.ModflowGwfevta'>, <class
'flopy.mf6.modflow.mfgwfgfb.ModflowGwfgfb'>, <class
'flopy.mf6.modflow.mfgwfgnc.ModflowGwfgnc'>, <class
'flopy.mf6.modflow.mfgwfgnc.GwfgncPackages'>, <class
'flopy.mf6.modflow.mfgwfgwf.ModflowGwfgwf'>, <class
'flopy.mf6.modflow.mfgwfgwt.ModflowGwfgwt'>, <class
'flopy.mf6.modflow.mfgwfhfb.ModflowGwfhfb'>, <class
'flopy.mf6.modflow.mfgwfic.ModflowGwfic'>, <class
'flopy.mf6.modflow.mfgwflak.ModflowGwflak'>, <class
'flopy.mf6.modflow.mfgwfmaw.ModflowGwfmaw'>, <class
'flopy.mf6.modflow.mfgwfmvr.ModflowGwfmvr'>, <class
'flopy.mf6.modflow.mfgwfmvr.GwfmvrPackages'>, <class
'flopy.mf6.modflow.mfgwfnam.ModflowGwfnam'>, <class
'flopy.mf6.modflow.mfgwfnpf.ModflowGwfnpf'>, <class
'flopy.mf6.modflow.mfgwfoc.ModflowGwfoc'>, <class
'flopy.mf6.modflow.mfgwfrch.ModflowGwfrch'>, <class
'flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha'>, <class
'flopy.mf6.modflow.mfgwfriv.ModflowGwfriv'>, <class
'flopy.mf6.modflow.mfgwfsfr.ModflowGwfsfr'>, <class
'flopy.mf6.modflow.mfgwfsto.ModflowGwfsto'>, <class
'flopy.mf6.modflow.mfgwfuzf.ModflowGwfuzf'>, <class
'flopy.mf6.modflow.mfgwfvsc.ModflowGwfvsc'>, <class
'flopy.mf6.modflow.mfgfwel.ModflowGfwel'>, <class
'flopy.mf6.modflow.mfgwtadv.ModflowGwtadv'>, <class
'flopy.mf6.modflow.mfgwtapi.ModflowGwtapi'>, <class
'flopy.mf6.modflow.mfgwtcnc.ModflowGwtcnc'>, <class
'flopy.mf6.modflow.mfgwtdis.ModflowGwtdis'>, <class
'flopy.mf6.modflow.mfgwtdisu.ModflowGwtdisu'>, <class
'flopy.mf6.modflow.mfgwtdisv.ModflowGwtdisv'>, <class
'flopy.mf6.modflow.mfgwt dsp.ModflowGwt dsp'>, <class
'flopy.mf6.modflow.mfgwt fmi.ModflowGwt fmi'>, <class
'flopy.mf6.modflow.mfgwt gwt.ModflowGwt gwt'>, <class
'flopy.mf6.modflow.mfgwt ic.ModflowGwt ic'>, <class
'flopy.mf6.modflow.mfgwt ist.ModflowGwt ist'>, <class
'flopy.mf6.modflow.mfgwt lkt.ModflowGwt lkt'>, <class
'flopy.mf6.modflow.mfgwt mst.ModflowGwt mst'>, <class
'flopy.mf6.modflow.mfgwt mvt.ModflowGwt mvt'>, <class
'flopy.mf6.modflow.mfgwt mvt.Gwt mvt Packages'>, <class
'flopy.mf6.modflow.mfgwt mwt.ModflowGwt mwt'>, <class
'flopy.mf6.modflow.mfgwt nam.ModflowGwt nam'>, <class
'flopy.mf6.modflow.mfgwt oc.ModflowGwt oc'>, <class
'flopy.mf6.modflow.mfgwt sft.ModflowGwt sft'>, <class
'flopy.mf6.modflow.mfgwt src.ModflowGwt src'>, <class
'flopy.mf6.modflow.mfgwt ssm.ModflowGwt ssm'>, <class
'flopy.mf6.modflow.mfgwt uzt.ModflowGwt uzt'>, <class
'flopy.mf6.modflow.mfims.ModflowIms'>, <class 'flopy.mf6.modflow.mfmvr.ModflowMvr'>,
<class 'flopy.mf6.modflow.mfmvr.MvrPackages'>, <class

```

property package_dict

Returns a copy of the package name dictionary.

static package_factory(package_type: *str*, model_type: *str*)

Static method that returns the appropriate package type object based on the package_type and model_type strings. For internal FloPy use only, not intended for end users.

Parameters

- **package_type** (*str*) - Type of package to create
- **model_type** (*str*) - Type of model that package is a part of

Returns

package

Return type

MFPackage subclass

property package_key_dict**static package_list()**

Static method that returns the list of available packages. For internal FloPy use only, not intended for end users.

Returns a list of MFPackage subclasses

property package_names

Returns a list of package names.

```

packages_by_abbr = {'ems': <class 'flopy.mf6.modflow.mfems.ModflowEms'>, 'gnc':
<class 'flopy.mf6.modflow.mfgnc.ModflowGnc'>, 'gncpackages': <class
'flopy.mf6.modflow.mfgnc.GncPackages'>, 'gwfapi': <class
'flopy.mf6.modflow.mfgwfapi.ModflowGwfapi'>, 'gwfbuy': <class
'flopy.mf6.modflow.mfgwfbuy.ModflowGwfbuy'>, 'gwfchd': <class
'flopy.mf6.modflow.mfgwfhhd.ModflowGwfhhd'>, 'gwfcsub': <class
'flopy.mf6.modflow.mfgwfcsub.ModflowGwfcsub'>, 'gwfdis': <class
'flopy.mf6.modflow.mfgwfdis.ModflowGwfdis'>, 'gwfdisu': <class
'flopy.mf6.modflow.mfgwfdisu.ModflowGwfdisu'>, 'gwfdisv': <class
'flopy.mf6.modflow.mfgwfdisv.ModflowGwfdisv'>, 'gwfdrn': <class
'flopy.mf6.modflow.mfgwfdrn.ModflowGwfdrn'>, 'gwfevt': <class
'flopy.mf6.modflow.mfgwfevt.ModflowGwfevt'>, 'gwfevta': <class
'flopy.mf6.modflow.mfgwfevta.ModflowGwfevta'>, 'gwfgfb': <class
'flopy.mf6.modflow.mfgwfgfb.ModflowGwfgfb'>, 'gwfgnc': <class
'flopy.mf6.modflow.mfgwfgnc.ModflowGwfgnc'>, 'gwfgncpackages': <class
'flopy.mf6.modflow.mfgwfgnc.GwfgncPackages'>, 'gwfgwf': <class
'flopy.mf6.modflow.mfgwfgwf.ModflowGwfgwf'>, 'gwfgwt': <class
'flopy.mf6.modflow.mfgwfgwt.ModflowGwfgwt'>, 'gwfhfb': <class
'flopy.mf6.modflow.mfgwfhfb.ModflowGwfhfb'>, 'gwfic': <class
'flopy.mf6.modflow.mfgwfic.ModflowGwfic'>, 'gwflak': <class
'flopy.mf6.modflow.mfgwflak.ModflowGwflak'>, 'gwfmaw': <class
'flopy.mf6.modflow.mfgwfmaw.ModflowGwfmaw'>, 'gwfmvr': <class
'flopy.mf6.modflow.mfgwfmvr.ModflowGwfmvr'>, 'gwfmvrpackages': <class
'flopy.mf6.modflow.mfgwfmvr.GwfmvrPackages'>, 'gwfnam': <class
'flopy.mf6.modflow.mfgwfnam.ModflowGwfnam'>, 'gwfnpf': <class
'flopy.mf6.modflow.mfgwfnpf.ModflowGwfnpf'>, 'gwfoc': <class
'flopy.mf6.modflow.mfgwfoc.ModflowGwfoc'>, 'gwfrch': <class
'flopy.mf6.modflow.mfgwfrch.ModflowGwfrch'>, 'gwfrcha': <class
'flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha'>, 'gwfriv': <class
'flopy.mf6.modflow.mfgwfriv.ModflowGwfriv'>, 'gwfsfr': <class
'flopy.mf6.modflow.mfgwfsfr.ModflowGwfsfr'>, 'gwfsto': <class
'flopy.mf6.modflow.mfgwfsto.ModflowGwfsto'>, 'gwfuze': <class
'flopy.mf6.modflow.mfgwfuze.ModflowGwfuze'>, 'gwfvsc': <class
'flopy.mf6.modflow.mfgwfvsc.ModflowGwfvsc'>, 'gwfwel': <class
'flopy.mf6.modflow.mfgwfwel.ModflowGwfwel'>, 'gwtadv': <class
'flopy.mf6.modflow.mfgwtadv.ModflowGwtadv'>, 'gwtapi': <class
'flopy.mf6.modflow.mfgwtapi.ModflowGwtapi'>, 'gwtcnc': <class
'flopy.mf6.modflow.mfgwtcnc.ModflowGwtcnc'>, 'gwtdis': <class
'flopy.mf6.modflow.mfgwtdis.ModflowGwtdis'>, 'gwtdisu': <class
'flopy.mf6.modflow.mfgwtdisu.ModflowGwtdisu'>, 'gwtdisv': <class
'flopy.mf6.modflow.mfgwtdisv.ModflowGwtdisv'>, 'gwtdsp': <class
'flopy.mf6.modflow.mfgwtdsp.ModflowGwtdsp'>, 'gwtfmi': <class
'flopy.mf6.modflow.mfgwtfmi.ModflowGwtfmi'>, 'gwtgtw': <class
'flopy.mf6.modflow.mfgwtgtw.ModflowGwtgtw'>, 'gwtic': <class
'flopy.mf6.modflow.mfgwtic.ModflowGwtic'>, 'gwtist': <class
'flopy.mf6.modflow.mfgwtist.ModflowGwtist'>, 'gwtlkt': <class
'flopy.mf6.modflow.mfgwtlkt.ModflowGwtlkt'>, 'gwtmst': <class
'flopy.mf6.modflow.mfgwtmst.ModflowGwtmst'>, 'gwtmvt': <class
'flopy.mf6.modflow.mfgwtmvt.ModflowGwtmvt'>, 'gwtmvtpackages': <class
'flopy.mf6.modflow.mfgwtmvt.GwtmvtPackages'>, 'gwtmwt': <class
'flopy.mf6.modflow.mfgwtmwt.ModflowGwtmwt'>, 'gwtnam': <class
'flopy.mf6.modflow.mfgwtoc.ModflowGwtoc'>, 'gwtsft': <class
'flopy.mf6.modflow.mfgwtsft.ModflowGwtsft'>, 'gwtsrc': <class
'flopy.mf6.modflow.mfgwtsrc.ModflowGwtsrc'>, 'gwtssm': <class
'flopy.mf6.modflow.mfgwtssm.ModflowGwtssm'>, 'gwtuiz': <class
'flopy.mf6.modflow.mfgwtuiz.ModflowGwtuiz'>, 'ims': <class
'flopy.mf6.modflow.mfims.ModflowIms'>, 'mvr': <class
'flopy.mf6.modflow.mfmvr.ModflowMvr'>, 'mvrpackages': <class
'flopy.mf6.modflow.mfmvr.MvrPackages'>, 'mvt': <class

```

register_package(package)

Base method for registering a package. Should be overridden.

class PackageContainerType(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: [Enum](#)

Determines whether a package container is a simulation, model, or package.

model = 2

package = 3

simulation = 1

exception ReadAsArraysException

Bases: [Exception](#)

Exception occurs when loading ReadAsArrays package as non-ReadAsArrays package.

exception StructException(error, location)

Bases: [Exception](#)

Exception with the package file structure

class VerbosityLevel(value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None)

Bases: [Enum](#)

Determines how much information FloPy writes to the console

normal = 2

quiet = 1

verbose = 3

flopy.mf6.mfmodel module

class MFModel(simulation, model_type='gwf6', modelname='model', model_nam_file=None, version='mf6', exe_name='mf6', add_to_simulation=True, structure=None, model_rel_path='.', verbose=False, **kwargs)

Bases: [PackageContainer](#), [ModelInterface](#)

MODFLOW-6 model base class. Represents a single model in a simulation.

Parameters

- **simulation_data** ([MFSimulationData](#)) - Simulation data object of the simulation this model will belong to
- **structure** ([MFModelStructure](#)) - Structure of this type of model
- **modelname** ([str](#)) - Name of the model
- **model_nam_file** ([str](#)) - Relative path to the model name file from model working folder
- **version** ([str](#)) - Version of modflow

- **exe_name** (*str*) - Model executable name
- **model_ws** (*str*) - Model working folder path
- **disfile** (*str*) - Relative path to dis file from model working folder
- **grid_type** (*str*) - Type of grid the model will use (structured, unstructured, vertices)
- **verbose** (*bool*) - Verbose setting for model operations (default False)

name

Name of the model

Type*str***exe_name**

Model executable name

Type*str***packages**

Dictionary of model packages

Typedict of *MFPackage***check**(*f=None, verbose=True, level=1*)

Check model data for common errors.

Parameters

- **f** (*str* or *file handle*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If f is None, check method does not write results to a summary file. (default is None)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.

Returns

success

Return type*bool***Examples**

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.check()
```

property exename

MODFLOW executable name

export(*f*, ****kwargs**)

Method to export a model to a shapefile or netcdf file

Parameters

- **f** (*str*) - File name (“*.nc*” for netcdf or “*.shp*” for shapefile) or dictionary of
- ****kwargs** (*keyword arguments*) -
 - modelgrid**: `flopy.discretization.Grid`
User supplied modelgrid object which will supercede the built in modelgrid object
 - epsg**
[int] EPSG projection code
 - prj**
[str] The prj file name
 - if *fmt* is set to ‘*vtk*’, parameters of `vtk.export_model`

get_grid_type()

Return the type of grid used by model ‘*model_name*’ in simulation containing simulation data ‘*simulation_data*’.

Returns

grid type

Return type

`DiscretizationType`

get_ims_package()

Get the IMS package associated with this model.

Returns

IMS package

Return type

`ModflowIms`

get_steadystate_list()

Returns a list of stress periods that are steady state.

Returns

steady state list

Return type

list

property hdry

Dry cell value

property hnoflo

No-flow cell value

inspect_cells(*cell_list*, *stress_period=None*, *output_file_path=None*,
inspect_budget=True, *inspect_dependent_var=True*)

Inspect model cells. Returns model data associated with cells.

Parameters

- **cell_list** (*list of tuples*) - List of model cells. Each model cell is a tuple of integers. ex: [(1,1,1), (2,4,3)]
- **stress_period** (*int*) - For transient data only return data from this stress period. If not specified or None, all stress period data will be returned.
- **output_file_path** (*str*) - Path to output file that will contain the inspection results
- **inspect_budget** (*bool*) - Inspect budget file
- **inspect_dependent_var** (*bool*) - Inspect head file

Returns

output - Dictionary containing inspection results

Return type

`dict`

Examples

```
>>> import flopy
>>> sim = flopy.mf6.MFSimulation.load("name", "mf6", "mf6", ".")
>>> model = sim.get_model()
>>> inspect_list = [(2, 3, 2), (0, 4, 2), (0, 2, 4)]
>>> out_file = os.path.join("temp", "inspect_AdvGW_tidal.csv")
>>> model.inspect_cells(inspect_list, output_file_path=out_file)
```

is_valid()

Checks the validity of the model and all of its packages

Returns

valid

Return type

`bool`

property laycbd

Quasi-3D confining bed. Not supported in MODFLOW-6.

Returns

None

Return type

None

property laytyp

Layering type

```
classmethod load_base(simulation, structure, modelname='NewModel',
                      model_nam_file='modflowtest.nam', mtype='gwf', version='mf6',
                      exe_name: str | PathLike = 'mf6', strict=True,
                      model_rel_path='.', load_only=None)
```

Class method that loads an existing model.

Parameters

- **simulation** (`MFSimulation`) - simulation object that this model is a part of

- **simulation_data** ([MFSimulationData](#)) - simulation data object
- **structure** ([MFModelStructure](#)) - structure of this type of model
- **model_name** (*str*) - name of the model
- **model_name_file** (*str*) - relative path to the model name file from model working folder
- **version** (*str*) - version of modflow
- **exe_name** (*str* or *PathLike*) - model executable name or path
- **strict** (*bool*) - strict mode when loading files
- **model_rel_path** (*str*) - relative path of model folder to simulation folder
- **load_only** (*list*) - list of package abbreviations or package names corresponding to packages that flopy will load. default is None, which loads all packages. the discretization packages will load regardless of this setting. subpackages, like time series and observations, will also load regardless of this setting. example list: ['ic', 'maw', 'npf', 'oc', 'my_well_package_1']

Returns

model

Return type

[MFModel](#)

Examples

```
load_package(ftype, fname, pname, strict, ref_path, dict_package_name=None,
             parent_package=None)
```

Loads a package from a file. This method is used internally by FloPy and is not intended for the end user.

Parameters

- **ftype** (*str*) - the file type
- **fname** (*str*) - the name of the file containing the package input
- **pname** (*str*) - the user-defined name for the package
- **strict** (*bool*) - strict mode when loading the file
- **ref_path** (*str*) - path to the file. uses local path if set to None
- **dict_package_name** (*str*) - package name for dictionary lookup
- **parent_package** ([MFPackage](#)) - parent package

Examples

`match_array_cells(cell_list, data_shape, array_data, key, data_output)`

property `model_ws`

Model file path.

property `modeldiscrit`

Basic model spatial discretization information. This is used internally prior to model spatial discretization information being fully loaded.

Returns

model grid - FloPy object containing basic spatial discretization information for the model.

Return type

Grid subclass

property `modelgrid`

Model spatial discretization information.

Returns

model grid - FloPy object containing spatial discretization information for the model.

Return type

Grid subclass

property `modeltime`

Model time discretization information.

Returns

modeltime - FloPy object containing time discretization information for the simulation.

Return type

ModelTime

property `namefile`

Model namefile object.

property `nper`

Number of stress periods.

Returns

nper - Number of stress periods in the simulation.

Return type

int

property `output`

property `packagelist`

List of model packages.

`plot(SelPackList=None, **kwargs)`

Plot 2-D, 3-D, transient 2-D, and stress period list (MfList) model input data from a model instance

Parameters

- **model** - Flopy model instance
 - **SelPackList** - (list) list of package names to plot, if none all packages will be plotted
 - ****kwargs** - dict filename_base : str
Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)
- file_extension**
[str] Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')
- mflay**
[int] MODFLOW zero-based layer number to return. If None, then all all layers will be included. (default is None)
- kper**
[int] MODFLOW zero-based stress period number to return. (default is zero)
- key**
[str] MfList dictionary key. (default is None)

Returns

list

Empty list is returned if filename_base is not None. Otherwise a list of matplotlib.pyplot.axis are returned.

Return type

axes

register_package(package, add_to_package_list=True, set_package_name=True, set_package_filename=True)

Registers a package with the model. This method is used internally by FloPy and is not intended for use by the end user.

Parameters

- **package** (MFPackage) - Package to register
- **add_to_package_list** (bool) - Add package to lookup list
- **set_package_name** (bool) - Produce a package name for this package
- **set_package_filename** (bool) - Produce a filename for this package

Returns

path, package structure

Return type

tuple, MFPackageStructure

remove_package(package_name)

Removes package and all child packages from the model. *package_name* can be the package's name, type, or package object to be removed from the model.

Parameters

package_name (*str*) - Package name, package type, or package object to be removed from the model.

rename_all_packages(*name*)

Renames all package files in the model.

Parameters

name (*str*) - Prefix of package names. Packages files will be named <name>.<package ext>.

set_all_data_external(*check_data=True, external_data_folder=None*)

Sets the model's list and array data to be stored externally.

Parameters

- **check_data** (*bool*) - Determines if data error checking is enabled during this process.
- **external_data_folder** - Folder, relative to the simulation path or model relative path (see `use_model_relative_path` parameter), where external data will be stored

set_all_data_internal(*check_data=True*)

Sets the model's list and array data to be stored externally.

Parameters

check_data (*bool*) - Determines if data error checking is enabled during this process.

set_model_relative_path(*model_ws*)

Sets the file path to the model folder relative to the simulation folder and updates all model file paths, placing them in the model folder.

Parameters

model_ws (*str*) - Model working folder relative to simulation working folder

property solver_tols

Returns the solver inner hclose and rclose values.

Returns

inner_hclose, rclose

Return type

float, float

update_package_filename(*package, new_name*)

Updates the filename for a package. For internal flopy use only.

Parameters

- **package** (*MFPackage*) - Package object
- **new_name** (*str*) - New package name

property verbose

Verbose setting for model operations (True/False)

property version

Version of MODFLOW

```
write(ext_file_action=ExtFileAction.copy_relative_paths)
```

Writes out model's package files.

Parameters

ext_file_action ([ExtFileAction](#)) - Defines what to do with external files when the simulation path has changed. defaults to `copy_relative_paths` which copies only files with relative paths, leaving files defined by absolute paths fixed.

flopy.mf6.mfpackage module

```
class MFBlock(simulation_data, dimensions, structure, path, model_or_sim,
              container_package)
```

Bases: [object](#)

Represents a block in a MF6 input file. This class is used internally by FloPy and use by users of the FloPy library is not recommended.

Parameters

- **simulation_data** ([MFSimulationData](#)) - Data specific to this simulation
- **dimensions** ([MFDimensions](#)) - Describes model dimensions including model grid and simulation time
- **structure** ([MFVariableStructure](#)) - Structure describing block
- **path** ([tuple](#)) - Unique path to block

block_headers

Block header text (BEGIN/END), header variables, comments in the header

Type

[MFBlockHeader](#)

structure

Structure describing block

Type

[MFBlockStructure](#)

path

Unique path to block

Type

[tuple](#)

datasets

Dictionary of dataset objects with keys that are the name of the dataset

Type

[OrderedDict](#)

datasets_keyword

Dictionary of dataset objects with keys that are key words to identify start of dataset

Type

[dict](#)

enabled

If block is being used in the simulation

Type

`bool`

add_dataset(dataset_struct, data, var_path)

Add data to this block.

data_factory(sim_data, model_or_sim, structure, enable, path, dimensions, data=None, package=None)

Creates the appropriate data child object derived from MFData.

header_exists(key, data_path=None)

is_allowed()

Determine if block is valid based on the values of dependant MODFLOW variables.

is_empty()

Returns true if this block is empty.

is_valid()

Returns true if the block is valid.

load(block_header, fd, strict=True)

Loads block from file object. file object must be advanced to beginning of block before calling.

Parameters

- **block_header** (`MFBlockHeader`) - Block header for block being loaded.
- **fd** (`file`) - File descriptor of file being loaded
- **strict** (`bool`) - Enforce strict MODFLOW 6 file format.

set_all_data_external(base_name, check_data=True, external_data_folder=None)

Sets the block's list and array data to be stored externally, base_name is external file name's prefix, check_data determines if data error checking is enabled during this process.

Parameters

- **base_name** (`str`) - Base file name of external files where data will be written to.
- **check_data** (`bool`) - Whether to do data error checking.
- **external_data_folder** - Folder where external data will be stored

set_all_data_internal(check_data=True)

Sets the block's list and array data to be stored internally, check_data determines if data error checking is enabled during this process.

Parameters

- **check_data** (`bool`) - Whether to do data error checking.

set_model_relative_path(model_ws)

Sets `model_ws` as the model path relative to the simulation's path.

Parameters

model_ws (*str*) - Model path relative to the simulation's path.

write(*fd*, *ext_file_action*=*ExtFileAction.copy_relative_paths*)

Writes block to a file object.

Parameters

fd (*file object*) - File object to write to.

class MFBlockHeader(*name*, *variable_strings*, *comment*, *simulation_data*=None, *path*=None, *block*=None)

Bases: *object*

Represents the header of a block in a MF6 input file. This class is used internally by FloPy and its direct use by a user of this library is not recommend.

Parameters

- **name** (*str*) - Block name
- **variable_strings** (*list*) - List of strings that appear after the block name
- **comment** (*MFComment*) - Comment text in the block header

name

Block name

Type

str

variable_strings

List of strings that appear after the block name

Type

list

comment

Comment text in the block header

Type

MFComment

data_items

List of MFVariable of the variables contained in this block

Type

list

add_data_item(*new_data*, *data*)

Adds data to the block.

build_header_variables(*simulation_data*, *block_header_structure*, *block_path*, *data*, *dimensions*)

Builds data objects to hold header variables.

connect_to_dict(*simulation_data*, *path*, *comment*=None)

Add comment to the simulation dictionary

get_comment()

Get block header comment

get_transient_key(*data_path=None*)

Get transient key associated with this block header.

is_same_header(*block_header*)

Checks if *block_header* is the same header as this header.

write_footer(*fd*)

Writes block footer to file object *fd*.

Parameters

fd (*file object*) - File object to write block footer to.

write_header(*fd*)

Writes block header to file object *fd*.

Parameters

fd (*file object*) - File object to write block header to.

class MFChildPackages(*model_or_sim*, *parent*, *pkg_type*, *filerecord*, *package=None*,
package_class=None)

Bases: [object](#)

Behind the scenes code for creating an interface to access child packages from a parent package. This class is automatically constructed by the FloPy library and is for internal library use only.

init_package(*package*, *fname*, *remove_packages=True*)

class MFPackage(*parent*, *package_type*, *filename=None*, *pname=None*, *loading_package=False*,
***kwargs*)

Bases: [PackageContainer](#), [PackageInterface](#)

Provides an interface for the user to specify data to build a package.

Parameters

- **parent** ([MFModel](#), [MFSimulation](#), or [MFPackage](#)) - The parent model, simulation, or package containing this package
- **package_type** (*str*) - String defining the package type
- **filename** (*str* or *PathLike*) - Name or path of file where this package is stored
- **quoted_filename** (*str*) - Filename with quotes around it when there is a space in the name
- **pname** (*str*) - Package name
- **loading_package** (*bool*) - Whether or not to add this package to the parent container's package list during initialization

blocks

Dictionary of blocks contained in this package by block name

Type

[dict](#)

path

Data dictionary path to this package

Type
tuple

structure

Describes the blocks and data contain in this package

Type
PackageStructure

dimensions

Resolves data dimensions for data within this package

Type
PackageDimension

build_child_package(pkg_type, data, parameter_name, filerecord)

Builds a child package. This method is only intended for FloPy internal use.

build_child_packages_container(pkg_type, filerecord)

Builds a container object for any child packages. This method is only intended for FloPy internal use.

build_mfdata(var_name, data=None)

Returns the appropriate data type object (mfdata list, mfdata array, or mfdata scalar) given that object the appropriate structure (looked up based on var_name) and any data supplied. This method is for internal FloPy library use only.

Parameters

- **var_name** (*str*) - Variable name
- **data** (*many supported types*) - Data contained in this object

Returns

data object

Return type

MFDData subclass

check(f=None, verbose=True, level=1, checktype=None)

Data check, returns True on success.

create_package_dimensions()

Creates a package dimensions object. For internal FloPy library use.

Returns

package dimensions

Return type

PackageDimensions

property data_list

List of data in this package.

export(f, **kwargs)

Method to export a package to netcdf or shapefile based on the extension of the file name (.shp for shapefile, .nc for netcdf)

Parameters

- **f** (*str*) - Filename

- **kwargs** (*keyword arguments*) -

modelgrid

[flopy.discretization.Grid instance] User supplied modelgrid which can be used for exporting in lieu of the modelgrid associated with the model object

Return type

None or Netcdf object

property filename

Package's file name.

get_file_path()

Returns the package file's path.

Returns

file path

Return type

`str`

inspect_cells(*cell_list*, *stress_period=None*)

Inspect model cells. Returns package data associated with cells.

Parameters

- **cell_list** (*list of tuples*) - List of model cells. Each model cell is a tuple of integers. ex: [(1,1,1), (2,4,3)]
- **stress_period** (*int*) - For transient data, only return data from this stress period. If not specified or None, all stress period data will be returned.

Returns

output - Array containing inspection results

Return type

array

is_valid()

Returns whether or not this package is valid.

Returns

is valid

Return type

`bool`

load(*strict=True*)

Loads the package from file.

Parameters

strict (*bool*) - Enforce strict checking of data.

Returns

success

Return type

`bool`

property name

Name of package

property output

Method to get output associated with a specific package

Return type

MF6Output object

property package_type

String describing type of package

property parent

Parent package

plot(kwargs)**

Plot 2-D, 3-D, transient 2-D, and stress period list (MfList) package input data

Parameters

****kwargs** (*dict*) -

filename_base

[str] Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)

file_extension

[str] Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')

mflay

[int] MODFLOW zero-based layer number to return. If None, then all layers will be included. (default is None)

kper

[int] MODFLOW zero-based stress period number to return. (default is zero)

key

[str] MfList dictionary key. (default is None)

Returns

axes - Empty list is returned if filename_base is not None. Otherwise a list of matplotlib.pyplot.axis are returned.

Return type

list

property plottable

If package is plottable

property quoted_filename

Package's file name with quotes if there is a space.

remove()

Removes this package from the simulation/model it is currently a part of.

set_all_data_external(check_data=True, external_data_folder=None)

Sets the package's list and array data to be stored externally.

Parameters

- **check_data** (*bool*) - Determine if data error checking is enabled
- **external_data_folder** - Folder where external data will be stored

set_all_data_internal(*check_data=True*)

Sets the package's list and array data to be stored internally.

Parameters

- check_data** (*bool*) - Determine if data error checking is enabled

set_model_relative_path(*model_ws*)

Sets the model path relative to the simulation's path.

Parameters

- model_ws** (*str*) - Model path relative to the simulation's path.

write(*ext_file_action=ExtFileAction.copy_relative_paths*)

Writes the package to a file.

Parameters

- ext_file_action** (*ExtFileAction*) - How to handle pathing of external data files.

8.1.2 MODFLOW 6 Simulation

MODFLOW 6 allows you to create simulations that can contain multiple models and packages. The FloPy for MODFLOW 6 simulation classes define functionality that applies to the entire MODFLOW 6 simulation. When using FloPy for MODFLOW 6 the first object you will most likely create is a simulation (MFSimulation) object.

Contents:

flopy.mf6.modflow.mfsimulation module

```
class MFSimulation(sim_name='sim', version='mf6', exe_name: str | PathLike = 'mf6', sim_ws:  
                  str | PathLike = '.', verbosity_level=1, continue_=None, nocheck=None,  
                  memory_print_option=None, write_headers=True, lazy_io=False)
```

Bases: *PackageContainer*

Entry point into any MODFLOW simulation.

MFSimulation is used to load, build, and/or save a MODFLOW 6 simulation. A MFSimulation object must be created before creating any of the MODFLOW 6 model objects.

Parameters

- **sim_name** (*str*) - Name of the simulation.
- **version** (*str*) - Version of MODFLOW 6 executable
- **exe_name** (*str*) - Path to MODFLOW 6 executable
- **sim_ws** (*str*) - Path to MODFLOW 6 simulation working folder. This is the folder containing the simulation name file.
- **verbosity_level** (*int*) - Verbosity level of standard output from 0 to 2. When 0 is specified no standard output is written. When 1 is specified standard error/warning messages with some

informational messages are written. When 2 is specified full error/warning/informational messages are written (this is ideal for debugging).

- **continue** (*bool*) - Sets the continue option in the simulation name file. The continue option is a keyword flag to indicate that the simulation should continue even if one or more solutions do not converge.
- **nocheck** (*bool*) - Sets the nocheck option in the simulation name file. The nocheck option is a keyword flag to indicate that the model input check routines should not be called prior to each time step. Checks are performed by default.
- **memory_print_option** (*str*) - Sets memory_print_option in the simulation name file. Memory_print_option is a flag that controls printing of detailed memory manager usage to the end of the simulation list file. NONE means do not print detailed information. SUMMARY means print only the total memory for each simulation component. ALL means print information for each variable stored in the memory manager. NONE is default if memory_print_option is not specified.
- **write_headers** (*bool*) - When true flopy writes a header to each package file indicating that it was created by flopy.
- **lazy_io** (*bool*) - When true flopy only reads external data when the data is requested and only writes external data if the data has changed. This option automatically overrides the verify_data and auto_set_sizes, turning both off.

Examples

```
>>> s = MFSimulation.load('my simulation', 'simulation.nam')
```

sim_name

Name of the simulation

Type

str

name_file

Simulation name file package

Type

MFPackage

check(*f*: *str* | *PathLike* | *None* = *None*, *verbose*=*True*, *level*=1)

Check model data for common errors.

Parameters

- **f** (*str* or *PathLike*, *optional*) - String defining file name or file handle for summary file of check method output. If *str* or *pathlike*, a file handle is created. If *None*, the method does not write results to a summary file. (default is *None*)

- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.

Returns

check list - Python list containing simulation check results

Return type

list

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.check()
```

delete_output_files()

Deletes simulation output files.

property exchange_files

Return list of exchange files associated with this simulation.

Returns

list

Return type

list of exchange names

get_exchange_file(filename)

Get a specified exchange file.

Parameters

filename (*str*) - Name of exchange file to get

Returns

exchange package

Return type

MFPackage

get_file(filename)

Get a specified file.

Parameters

filename (*str*) - Name of mover file to get

Returns

mover package

Return type

MFPackage

get_gnc_file(filename)

Get a specified gnc file.

Parameters

filename (*str*) - Name of gnc file to get

Returns**gnc package****Return type***MFPackage***get_ims_package(key)****get_model(model_name=None)**

Returns the models in the simulation with a given model name, name file name, or model type.

Parameters

model_name (*str*) - Name of the model to get. Passing in None or an empty list will get the first model.

Returns**model****Return type***MFModel***get_mvr_file(filename)**

Get a specified mover file.

Parameters

filename (*str*) - Name of mover file to get

Returns**mover package****Return type***MFPackage***get_mvt_file(filename)**

Get a specified mvt file.

Parameters

filename (*str*) - Name of mover transport file to get

Returns**mover transport package****Return type***MFPackage***get_solution_package(key)**

Get the solution package with the specified key.

Parameters

key (*str*) - solution package file name

Returns**solution_package****Return type***MFPackage***is_valid()**

Checks the validity of the solution and all of its models and packages. Returns true if the solution is valid, false if it is not.

Returns

valid - Whether this is a valid simulation

Return type

`bool`

```
classmethod load(sim_name='modflowsim', version='mf6', exe_name: str | PathLike =
                 'mf6', sim_ws: str | PathLike = '.', strict=True, verbosity_level=1,
                 load_only=None, verify_data=False, write_headers=True,
                 lazy_io=False)
```

Load an existing model.

Parameters

- **sim_name** (`str`) - Name of the simulation.
- **version** (`str`) - MODFLOW version
- **exe_name** (`str` or `PathLike`) - Path to MODFLOW executable (relative to the simulation workspace or absolute)
- **sim_ws** (`str` or `PathLike`) - Path to simulation workspace
- **strict** (`bool`) - Strict enforcement of file formatting
- **verbosity_level** (`int`) -

Verbosity level of standard output

0: No standard output 1: Standard error/warning messages with some informational

messages

2: Verbose mode with full error/warning/informational messages. This is ideal for debugging.

- **load_only** (`list`) - List of package abbreviations or package names corresponding to packages that flopy will load. default is None, which loads all packages. the discretization packages will load regardless of this setting. subpackages, like time series and observations, will also load regardless of this setting. example list: ['ic', 'maw', 'npf', 'oc', 'ims', 'gwf6-gwf6']
- **verify_data** (`bool`) - Verify data when it is loaded. this can slow down loading
- **write_headers** (`bool`) - When true flopy writes a header to each package file indicating that it was created by flopy
- **lazy_io** (`bool`) - When true flopy only reads external data when the data is requested and only writes external data if the data has changed. This option automatically overrides the `verify_data` and `auto_set_sizes`, turning both off.

Returns

`sim`

Return type

MFSimulation object

Examples

```
>>> s = flopy.mf6.mfsimulation.load('my simulation')
```

load_package(ftype, fname: *str* | *PathLike*, pname, strict, ref_path: *str* | *PathLike*, dict_package_name=None, parent_package=None)

Load a package from a file.

Parameters

- **ftype** (*str*) - the file type
- **fname** (*str* or *PathLike*) - the path of the file containing the package input
- **pname** (*str*) - the user-defined name for the package
- **strict** (*bool*) - strict mode when loading the file
- **ref_path** (*str*) - path to the file. uses local path if set to None
- **dict_package_name** (*str*) - package name for dictionary lookup
- **parent_package** (*MFPackage*) - parent package

property model_dict

Return a dictionary of models associated with this simulation.

Returns

model dict - dictionary of models

Return type

dict

property model_names

Return a list of model names associated with this simulation.

Returns

list

Return type

list of model names

plot(model_list: *str* | *List[str]* | *None* = None, SelPackList=None, **kwargs)

Plot simulation or models.

Method to plot a whole simulation or a series of models that are part of a simulation.

Parameters

- **model_list** (*list*, optional) - List of model names to plot, if none all models will be plotted
- **SelPackList** (*list*, optional) - List of package names to plot, if none all packages will be plotted
- **kwargs** -
filename_base
[*str*] Base file name that will be used to automatically generate file names for output image files. Plots will

be exported as image files if `file_name_base` is not `None`.
(default is `None`)

file_extension

[str] Valid matplotlib.pyplot file extension for `savefig()`.
Only used if `filename_base` is not `None`. (default is 'png')

mflay

[int] MODFLOW zero-based layer number to return. If `None`,
then all layers will be included. (default is `None`)

kper

[int] MODFLOW zero-based stress period number to return.
(default is zero)

key

[str] MFList dictionary key. (default is `None`)

Returns

axes - matplotlib.pyplot.axes objects

Return type

(list)

register_exchange_file(package)

Register an exchange package file with the simulation. This is a call-back
method made from the package and should not be called directly.

Parameters

package (MFPackage) - Exchange package object to register

register_ims_package(solution_file: MFPackage, model_list: str | List[str])

register_model(model, model_type, model_name, model_namefile)

Add a model to the simulation. This is a call-back method made from the
package and should not be called directly.

Parameters

- **model** (MFModel) - Model object to add to simulation
- **sln_group** (str) - Solution group of model

Returns

model_structure_object

Return type

MFModelStructure

**register_package(package, add_to_package_list=True, set_package_name=True,
set_package_filename=True)**

Register a package file with the simulation. This is a call-back method made
from the package and should not be called directly.

Parameters

- **package** (MFPackage) - Package to register
- **add_to_package_list** (bool) - Add package to lookup list
- **set_package_name** (bool) - Produce a package name for this
package

- **set_package_filename** (*bool*) - Produce a filename for this package

Returns
(*path*)

Return type
tuple, package structure : MFPackageStructure)

register_solution_package(*solution_file*: MFPackage, *model_list*: *str* | *List[str]*)

Register a solution package with the simulation.

Parameters

solution_file
[MFPackage] solution package to register

model_list
[list of strings] list of models using the solution package to be registered

remove_exchange_file(*package*)

Removes the exchange file “package”. This is for internal flopy library use only.

Parameters

package (MFPackage) - Exchange package to be removed

remove_model(*model_name*)

Remove model with name *model_name* from the simulation

Parameters

model_name (*str*) - Model name to remove from simulation

remove_package(*package_name*)

Removes package from the simulation. *package_name* can be the package’s name, type, or package object to be removed from the model.

Parameters

package_name (*str*) - Name of package to be removed

rename_all_packages(*name*)

Rename all packages with name as prefix.

Parameters

name (*str*) - Prefix of package names

rename_model_namefile(*model*, *new_namefile*)

Rename a model’s namefile. For internal flopy library use only.

Parameters

- **model** (MFModel) - Model object whose namefile to rename
- **new_namefile** (*str*) - Name of the new namefile

run_simulation(*silent*=None, *pause*=False, *report*=False, *processors*=None, *normal_msg*='normal termination', *use_async*=False, *cargs*=None)

Run the simulation.

Parameters

- **silent** (*bool*) - Run in silent mode

- **pause** (*bool*) - Pause at end of run
- **report** (*bool*) - Save stdout lines to a list (buff)
- **processors** (*int*) - Number of processors. Parallel simulations are only supported for MODFLOW 6 simulations. (default is None)
- **normal_msg** (*str* or *list*) - Normal termination message used to determine if the run terminated normally. More than one message can be provided using a list. (default is 'normal termination')
- **use_async** (*bool*) - Asynchronously read model stdout and report with timestamps. good for models that take long time to run. not good for models that run really fast
- **cargs** (*str* or *list of strings*) - Additional command line arguments to pass to the executable. default is None

Returns

- **success** (*bool*)
- **buff** (*list of lines of stdout*)

set_all_data_external(*check_data=True, external_data_folder=None*)

Sets the simulation's list and array data to be stored externally.

Parameters

- **check_data** (*bool*) - Determines if data error checking is enabled during this process. Data error checking can be slow on large datasets.
- **external_data_folder** (*str* or *PathLike*) - Path relative to the simulation path or model relative path (see `use_model_relative_path` parameter), where external data will be stored

set_all_data_internal(*check_data=True*)

set_sim_path(*path: str | PathLike*)

Return a list of output data keys.

Parameters

path (*str*) - Relative or absolute path to simulation root folder.

property sim_package_list

List of all "simulation level" packages

property sim_path: *Path*

update_package_filename(*package, new_name*)

Updates internal arrays to be consistent with a new file name. This is for internal flopy library use only.

Parameters

- **package** (*MFPackage*) - Package with new name
- **new_name** (*str*) - Package's new name

write_simulation(*ext_file_action*=*ExtFileAction.copy_relative_paths*, *silent*=*False*)

Write the simulation to files.

Parameters

ext_file_action

[*ExtFileAction*] Defines what to do with external files when the simulation path has changed. Defaults to *copy_relative_paths* which copies only files with relative paths, leaving files defined by absolute paths fixed.

silent

[*bool*] Writes out the simulation in silent mode (*verbosity_level* = 0)

class MFSimulationData(*path*: *str* | *PathLike*, *mfsim*)

Bases: *object*

Class containing MODFLOW simulation data and file formatting data. Use *MFSimulationData* to set simulation-wide settings which include data formatting and file location settings.

Parameters

path (*str*) - path on disk to the simulation

indent_string

String used to define how much indent to use (file formatting)

Type

str

internal_formatting

List defining string to use for internal formatting

Type

list

external_formatting

List defining string to use for external formatting

Type

list

open_close_formatting

List defining string to use for open/close

Type

list

max_columns_of_data

Maximum columns of data before line wraps. For structured grids this is set to *ncol* by default. For all other grids the default is 20.

Type

int

wrap_multidim_arrays

Whether to wrap line for multi-dimensional arrays at the end of a row/column/layer

Type

bool

_float_precision

Number of decimal points to write for a floating point number

Type

`int`

_float_characters

Number of characters a floating point number takes up

Type

`int`

write_headers

When true flopy writes a header to each package file indicating that it was created by flopy

Type

`bool`

sci_note_upper_thres

Numbers greater than this threshold are written in scientific notation

Type

`float`

sci_note_lower_thres

Numbers less than this threshold are written in scientific notation

Type

`float`

mfp_path

File path location information for the simulation

Type

MFFileMgmt

model_dimensions

Dictionary containing discretization information for each model

Type

`dict`

mfd_data

Custom dictionary containing all model data for the simulation

Type

SimulationDict

property float_characters

Gets max characters used in floating point numbers.

property float_precision

Gets precision of floating point numbers.

property lazy_io**property max_columns_of_data**

set_sci_note_lower_thres(value)

Sets threshold number where any number smaller than threshold is represented in scientific notation.

Parameters

value (*float*) - threshold value

set_sci_note_upper_thres(value)

Sets threshold number where any number larger than threshold is represented in scientific notation.

Parameters

value (*float*) - threshold value

class SimulationDict(path=None)

Bases: *dict*

Class containing custom dictionary for MODFLOW simulations. Dictionary contains model data. Dictionary keys are “paths” to the data that include the model and package containing the data.

Behaves as an dict with some additional features described below.

Parameters

path (*MFFileMgmt*) - Object containing path information for the simulation

find_in_path(key_path, key_leaf)

Attempt to find key_leaf in a partial key path key_path.

Parameters

- **key_path** (*str*) - partial path to the data
- **key_leaf** (*str*) - name of the data

Returns

- **Data** (*MFData*,)
- **index** (*int*)

input_keys()

Return a list of input data keys.

Returns

input keys

Return type

list

keys()

Return a list of all keys.

Returns

all keys

Return type

list

observation_keys()

Return a list of observation keys.

Returns
 observation keys

Return type
 list

output_keys(*print_keys=True*)

Return a list of output data keys supported by the dictionary.

Parameters
print_keys (*bool*) - print keys to console

Returns
 output keys

Return type
 list

8.1.3 MODFLOW 6 Simulation Packages

MODFLOW 6 simulation packages are the packages that are not necessarily tied to a specific model and can apply to the entire simulation or a group of models in the simulation.

Contents:

flopy.mf6.modflow.mfgnc module

```
class GncPackages(model_or_sim, parent, pkg_type, filerecord, package=None,
                  package_class=None)
```

Bases: *MFChildPackages*

GncPackages is a container class for the ModflowGnc class.

initialize()

Initializes a new ModflowGnc package removing any sibling child packages attached to the same parent package. See ModflowGnc init documentation for definition of parameters.

append_package()

Adds a new ModflowGwfgnc package to the container. See ModflowGwfgnc init documentation for definition of parameters.

```
append_package(print_input=None, print_flows=None, explicit=None, numgnc=None,
                numalphaj=None, gncdata=None, filename=None, pname=None)
```

```
initialize(print_input=None, print_flows=None, explicit=None, numgnc=None,
            numalphaj=None, gncdata=None, filename=None, pname=None)
```

```
package_abbr = 'gncpackages'
```

```
class ModflowGnc(simulation, loading_package=False, print_input=None, print_flows=None,
                  explicit=None, numgnc=None, numalphaj=None, gncdata=None, filename=None,
                  pname=None, **kwargs)
```

Bases: *MFPackage*

ModflowGnc defines a gnc package.

Parameters

- **simulation** ([MFSimulation](#)) - Simulation that this package is a part of. Package is automatically added to simulation when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **print_input** (*boolean*) -
 - `print_input` (*boolean*) keyword to indicate that the list of GNC information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - `print_flows` (*boolean*) keyword to indicate that the list of GNC flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **explicit** (*boolean*) -
 - `explicit` (*boolean*) keyword to indicate that the ghost node correction is applied in an explicit manner on the right-hand side of the matrix. The explicit approach will likely require additional outer iterations. If the keyword is not specified, then the correction will be applied in an implicit manner on the left-hand side. The implicit approach will likely converge better, but may require additional memory. If the EXPLICIT keyword is not specified, then the BICGSTAB linear acceleration option should be specified within the LINEAR block of the Sparse Matrix Solver.
- **numgnc** (*integer*) -
 - `numgnc` (*integer*) is the number of GNC entries.
- **numalphaj** (*integer*) -
 - `numalphaj` (*integer*) is the number of contributing factors.
- **gncdata** (*[cellidn, cellidm, cellidsj, alphasj]*) -
 - `cellidn` (*(integer, ...)*) is the cellid of the cell, *n*, in which the ghost node is located. For a structured grid that uses the DIS input file, CELLIDN is the layer, row, and column numbers of the cell. For a grid that uses the DISV input file, CELLIDN is the layer number and CELL2D number for the two cells. If the model uses the unstructured discretization (DISU) input file, then CELLIDN is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - `cellidm` (*(integer, ...)*) is the cellid of the connecting cell, *m*, to which flow occurs from the ghost node. For a

structured grid that uses the DIS input file, CELLIDM is the layer, row, and column numbers of the cell. For a grid that uses the DISV input file, CELLIDM is the layer number and CELL2D number for the two cells. If the model uses the unstructured discretization (DISU) input file, then CELLIDM is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.

- `cellidsj` ((integer, ...)) is the array of CELLIDS for the contributing `j` cells, which contribute to the interpolated head value at the ghost node. This item contains one CELLID for each of the contributing cells of the ghost node. Note that if the number of actual contributing cells needed by the user is less than NUMALPHAJ for any ghost node, then a dummy CELLID of zero(s) should be inserted with an associated contributing factor of zero. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column numbers of the cell. For a grid that uses the DISV input file, CELLID is the layer number and cell2d number for the two cells. If the model uses the unstructured discretization (DISU) input file, then CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
- `alphasj` (double) is the contributing factors for each contributing node in CELLIDSJ. Note that if the number of actual contributing cells is less than NUMALPHAJ for any ghost node, then dummy CELLIDS should be inserted with an associated contributing factor of zero. The sum of ALPHASJ should be less than one. This is because one minus the sum of ALPHASJ is equal to the alpha term (alpha n in equation 4-61 of the GWF Model report) that is multiplied by the head in cell n .
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

```
dfn = [['header'], ['block options', 'name print_input', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name print_flows', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name explicit', 'type
keyword', 'tagged true', 'reader urword', 'optional true'], ['block dimensions',
'name numgnc', 'type integer', 'reader urword', 'optional false'], ['block
dimensions', 'name numalphaj', 'type integer', 'reader urword', 'optional false'],
['block gncdata', 'name gncdata', 'type recarray cellidn cellidm cellidsj alphasj',
'shape (maxbound)', 'reader urword'], ['block gncdata', 'name cellidn', 'type
integer', 'shape', 'tagged false', 'in_record true', 'reader urword', 'numeric_index
true'], ['block gncdata', 'name cellidm', 'type integer', 'shape', 'tagged false',
'in_record true', 'reader urword', 'numeric_index true'], ['block gncdata', 'name
cellidsj', 'type integer', 'shape (numalphaj)', 'tagged false', 'in_record true',
'reader urword', 'numeric_index true'], ['block gncdata', 'name alphasj', 'type
double precision', 'shape (numalphaj)', 'tagged false', 'in_record true', 'reader
urword']]

dfn_file_name = 'gwf-gnc.dfn'

gncdata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gnc'
```

flopy.mf6.modflow.mfims module

```
class ModflowIms(simulation, loading_package=False, print_option=None, complexity=None,
    csv_output_filerecord=None, csv_outer_output_filerecord=None,
    csv_inner_output_filerecord=None, no_ptcrecord=None,
    ats_outer_maximum_fraction=None, outer_hclose=None, outer_dvclose=None,
    outer_rclosebnd=None, outer_maximum=None, under_relaxation=None,
    under_relaxation_gamma=None, under_relaxation_theta=None,
    under_relaxation_kappa=None, under_relaxation_momentum=None,
    backtracking_number=None, backtracking_tolerance=None,
    backtracking_reduction_factor=None, backtracking_residual_limit=None,
    inner_maximum=None, inner_hclose=None, inner_dvclose=None,
    rcloserecord=None, linear_acceleration=None, relaxation_factor=None,
    preconditioner_levels=None, preconditioner_drop_tolerance=None,
    number_orthogonalizations=None, scaling_method=None,
    reordering_method=None, filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowIms defines a ims package.

Parameters

- **simulation** ([MFSimulation](#)) - Simulation that this package is a part of. Package is automatically added to simulation when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **print_option** (*string*) -
 - print_option (*string*) is a flag that controls printing of convergence information from the solver. NONE means print nothing. SUMMARY means print only the total number of

iterations and nonlinear residual reduction summaries. ALL means print linear matrix solver convergence information to the solution listing file and model specific linear matrix solver convergence information to each model listing file in addition to SUMMARY information. NONE is default if PRINT_OPTION is not specified.

- **complexity** (*string*) -

- complexity (*string*) is an optional keyword that defines default non- linear and linear solver parameters. SIMPLE - indicates that default solver input values will be defined that work well for nearly linear models. This would be used for models that do not include nonlinear stress packages and models that are either confined or consist of a single unconfined layer that is thick enough to contain the water table within a single layer. MODERATE - indicates that default solver input values will be defined that work well for moderately nonlinear models. This would be used for models that include nonlinear stress packages and models that consist of one or more unconfined layers. The MODERATE option should be used when the SIMPLE option does not result in successful convergence. COMPLEX - indicates that default solver input values will be defined that work well for highly nonlinear models. This would be used for models that include nonlinear stress packages and models that consist of one or more unconfined layers representing complex geology and surface- water/groundwater interaction. The COMPLEX option should be used when the MODERATE option does not result in successful convergence. Non- linear and linear solver parameters assigned using a specified complexity can be modified in the NONLINEAR and LINEAR blocks. If the COMPLEXITY option is not specified, NONLINEAR and LINEAR variables will be assigned the simple complexity values.

- **csv_output_filerecord** (*[csvfile]*) -

- csvfile (*string*) name of the ascii comma separated values output file to write solver convergence information. If PRINT_OPTION is NONE or SUMMARY, comma separated values output includes maximum head change convergence information at the end of each outer iteration for each time step. If PRINT_OPTION is ALL, comma separated values output includes maximum head change and maximum residual convergence information for the solution and each model (if the solution includes more than one model) and linear acceleration information for each inner iteration.

- **csv_outer_output_filerecord** (*[outer_csvfile]*) -

- outer_csvfile (*string*) name of the ascii comma separated values output file to write maximum dependent-variable (for example, head) change convergence information at the end of each outer iteration for each time step.

- **csv_inner_output_filerecord** (*[inner_csvfile]*) -

- inner_csvfile (*string*) name of the ascii comma separated values output file to write solver convergence information. Comma separated values output includes maximum dependent-variable

(for example, head) change and maximum residual convergence information for the solution and each model (if the solution includes more than one model) and linear acceleration information for each inner iteration.

- **no_ptcrecord** (*[no_ptc_option]*) -
 - **no_ptc_option** (string) is an optional keyword that is used to define options for disabling pseudo-transient continuation (PTC). **FIRST** is an optional keyword to disable PTC for the first stress period, if steady-state and one or more model is using the Newton-Raphson formulation. **ALL** is an optional keyword to disable PTC for all steady-state stress periods for models using the Newton-Raphson formulation. If **NO_PTC_OPTION** is not specified, the **NO_PTC ALL** option is used.
- **ats_outer_maximum_fraction** (*double*) -
 - **ats_outer_maximum_fraction** (double) real value defining the fraction of the maximum allowable outer iterations used with the Adaptive Time Step (ATS) capability if it is active. If this value is set to zero by the user, then this solution will have no effect on ATS behavior. This value must be greater than or equal to zero and less than or equal to 0.5 or the program will terminate with an error. If it is not specified by the user, then it is assigned a default value of one third. When the number of outer iterations for this solution is less than the product of this value and the maximum allowable outer iterations, then ATS will increase the time step length by a factor of **DTADJ** in the ATS input file. When the number of outer iterations for this solution is greater than the maximum allowable outer iterations minus the product of this value and the maximum allowable outer iterations, then the ATS (if active) will decrease the time step length by a factor of $1 / \text{DTADJ}$.
- **outer_hclose** (*double*) -
 - **outer_hclose** (double) real value defining the head change criterion for convergence of the outer (nonlinear) iterations, in units of length. When the maximum absolute value of the head change at all nodes during an iteration is less than or equal to **OUTER_HCLOSE**, iteration stops. Commonly, **OUTER_HCLOSE** equals 0.01. The **OUTER_HCLOSE** option has been deprecated in favor of the more general **OUTER_DVCLOSE** (for dependent variable), however either one can be specified in order to maintain backward compatibility.
- **outer_dvclose** (*double*) -
 - **outer_dvclose** (double) real value defining the dependent-variable (for example, head) change criterion for convergence of the outer (nonlinear) iterations, in units of the dependent-variable (for example, length for head). When the maximum absolute value of the dependent-variable change at all nodes during an iteration is less than or equal to **OUTER_DVCLOSE**, iteration stops. Commonly, **OUTER_DVCLOSE** equals 0.01. The keyword, **OUTER_HCLOSE** can be still be specified instead of **OUTER_DVCLOSE** for backward compatibility with

previous versions of MODFLOW 6 but eventually OUTER_HCLOSE will be deprecated and specification of OUTER_HCLOSE will cause MODFLOW 6 to terminate with an error.

- **outer_rclosebnd** (*double*) -
 - outer_rclosebnd (double) real value defining the residual tolerance for convergence of model packages that solve a separate equation not solved by the IMS linear solver. This value represents the maximum allowable residual between successive outer iterations at any single model package element. An example of a model package that would use OUTER_RCLOSEBND to evaluate convergence is the SFR package which solves a continuity equation for each reach. The OUTER_RCLOSEBND option is deprecated and has no effect on simulation results as of version 6.1.1. The keyword, OUTER_RCLOSEBND can be still be specified for backward compatibility with previous versions of MODFLOW 6 but eventually specification of OUTER_RCLOSEBND will cause MODFLOW 6 to terminate with an error.
- **outer_maximum** (*integer*) -
 - outer_maximum (integer) integer value defining the maximum number of outer (nonlinear) iterations - that is, calls to the solution routine. For a linear problem OUTER_MAXIMUM should be 1.
- **under_relaxation** (*string*) -
 - under_relaxation (string) is an optional keyword that defines the nonlinear under-relaxation schemes used. Under-relaxation is also known as dampening, and is used to reduce the size of the calculated dependent variable before proceeding to the next outer iteration. Under-relaxation can be an effective tool for highly nonlinear models when there are large and often counteracting changes in the calculated dependent variable between successive outer iterations. By default under-relaxation is not used. NONE - under-relaxation is not used (default). SIMPLE - Simple under-relaxation scheme with a fixed relaxation factor (UNDER_RELAXATION_GAMMA) is used. COOLEY - Cooley under-relaxation scheme is used. DBD - delta-bar-delta under- relaxation is used. Note that the under-relaxation schemes are often used in conjunction with problems that use the Newton-Raphson formulation, however, experience has indicated that they also work well for non-Newton problems, such as those with the wet/dry options of MODFLOW 6.
- **under_relaxation_gamma** (*double*) -
 - under_relaxation_gamma (double) real value defining either the relaxation factor for the SIMPLE scheme or the history or memory term factor of the Cooley and delta-bar-delta algorithms. For the SIMPLE scheme, a value of one indicates that there is no under-relaxation and the full head change is applied. This value can be gradually reduced from one as a way to improve convergence; for well behaved problems, using a value less than one can increase the number of outer iterations required for convergence and needlessly increase

run times. `UNDER_RELAXATION_GAMMA` must be greater than zero for the `SIMPLE` scheme or the program will terminate with an error. For the Cooley and delta-bar-delta schemes, `UNDER_RELAXATION_GAMMA` is a memory term that can range between zero and one. When `UNDER_RELAXATION_GAMMA` is zero, only the most recent history (previous iteration value) is maintained. As `UNDER_RELAXATION_GAMMA` is increased, past history of iteration changes has greater influence on the memory term. The memory term is maintained as an exponential average of past changes. Retaining some past history can overcome granular behavior in the calculated function surface and therefore helps to overcome cyclic patterns of non-convergence. The value usually ranges from 0.1 to 0.3; a value of 0.2 works well for most problems. `UNDER_RELAXATION_GAMMA` only needs to be specified if `UNDER_RELAXATION` is not `NONE`.

- **`under_relaxation_theta`** (*double*) -
 - `under_relaxation_theta` (*double*) real value defining the reduction factor for the learning rate (under-relaxation term) of the delta- bar-delta algorithm. The value of `UNDER_RELAXATION_THETA` is between zero and one. If the change in the dependent-variable (for example, head) is of opposite sign to that of the previous iteration, the under-relaxation term is reduced by a factor of `UNDER_RELAXATION_THETA`. The value usually ranges from 0.3 to 0.9; a value of 0.7 works well for most problems. `UNDER_RELAXATION_THETA` only needs to be specified if `UNDER_RELAXATION` is `DBD`.
- **`under_relaxation_kappa`** (*double*) -
 - `under_relaxation_kappa` (*double*) real value defining the increment for the learning rate (under-relaxation term) of the delta-bar-delta algorithm. The value of `UNDER_RELAXATION_kappa` is between zero and one. If the change in the dependent-variable (for example, head) is of the same sign to that of the previous iteration, the under- relaxation term is increased by an increment of `UNDER_RELAXATION_KAPPA`. The value usually ranges from 0.03 to 0.3; a value of 0.1 works well for most problems. `UNDER_RELAXATION_KAPPA` only needs to be specified if `UNDER_RELAXATION` is `DBD`.
- **`under_relaxation_momentum`** (*double*) -
 - `under_relaxation_momentum` (*double*) real value defining the fraction of past history changes that is added as a momentum term to the step change for a nonlinear iteration. The value of `UNDER_RELAXATION_MOMENTUM` is between zero and one. A large momentum term should only be used when small learning rates are expected. Small amounts of the momentum term help convergence. The value usually ranges from 0.0001 to 0.1; a value of 0.001 works well for most problems. `UNDER_RELAXATION_MOMENTUM` only needs to be specified if `UNDER_RELAXATION` is `DBD`.
- **`backtracking_number`** (*integer*) -
 - `backtracking_number` (*integer*) integer value defining the maximum number of backtracking iterations allowed for residual reduction

computations. If BACKTRACKING_NUMBER = 0 then the backtracking iterations are omitted. The value usually ranges from 2 to 20; a value of 10 works well for most problems.

- **backtracking_tolerance** (*double*) -
 - backtracking_tolerance (*double*) real value defining the tolerance for residual change that is allowed for residual reduction computations. BACKTRACKING_TOLERANCE should not be less than one to avoid getting stuck in local minima. A large value serves to check for extreme residual increases, while a low value serves to control step size more severely. The value usually ranges from 1.0 to 10; a value of 10 works well for most problems but lower values like 1.1 may be required for harder problems. BACKTRACKING_TOLERANCE only needs to be specified if BACKTRACKING_NUMBER is greater than zero.
- **backtracking_reduction_factor** (*double*) -
 - backtracking_reduction_factor (*double*) real value defining the reduction in step size used for residual reduction computations. The value of BACKTRACKING_REDUCTION_FACTOR is between zero and one. The value usually ranges from 0.1 to 0.3; a value of 0.2 works well for most problems. BACKTRACKING_REDUCTION_FACTOR only needs to be specified if BACKTRACKING_NUMBER is greater than zero.
- **backtracking_residual_limit** (*double*) -
 - backtracking_residual_limit (*double*) real value defining the limit to which the residual is reduced with backtracking. If the residual is smaller than BACKTRACKING_RESIDUAL_LIMIT, then further backtracking is not performed. A value of 100 is suitable for large problems and residual reduction to smaller values may only slow down computations. BACKTRACKING_RESIDUAL_LIMIT only needs to be specified if BACKTRACKING_NUMBER is greater than zero.
- **inner_maximum** (*integer*) -
 - inner_maximum (*integer*) integer value defining the maximum number of inner (linear) iterations. The number typically depends on the characteristics of the matrix solution scheme being used. For nonlinear problems, INNER_MAXIMUM usually ranges from 60 to 600; a value of 100 will be sufficient for most linear problems.
- **inner_hclose** (*double*) -
 - inner_hclose (*double*) real value defining the head change criterion for convergence of the inner (linear) iterations, in units of length. When the maximum absolute value of the head change at all nodes during an iteration is less than or equal to INNER_HCLOSE, the matrix solver assumes convergence. Commonly, INNER_HCLOSE is set equal to or an order of magnitude less than the OUTER_HCLOSE value specified for the NONLINEAR block. The INNER_HCLOSE keyword has been deprecated in favor of the more general INNER_DVCLOSE (for dependent variable), however

either one can be specified in order to maintain backward compatibility.

- **inner_dvclose** (*double*) -

- inner_dvclose (double) real value defining the dependent-variable (for example, head) change criterion for convergence of the inner (linear) iterations, in units of the dependent-variable (for example, length for head). When the maximum absolute value of the dependent- variable change at all nodes during an iteration is less than or equal to INNER_DVCLOSE, the matrix solver assumes convergence. Commonly, INNER_DVCLOSE is set equal to or an order of magnitude less than the OUTER_DVCLOSE value specified for the NONLINEAR block. The keyword, INNER_HCLOSE can be still be specified instead of INNER_DVCLOSE for backward compatibility with previous versions of MODFLOW 6 but eventually INNER_HCLOSE will be deprecated and specification of INNER_HCLOSE will cause MODFLOW 6 to terminate with an error.

- **rcloserecord** ([*inner_rclose*, *rclose_option*]) -

- inner_rclose (double) real value that defines the flow residual tolerance for convergence of the IMS linear solver and specific flow residual criteria used. This value represents the maximum allowable residual at any single node. Value is in units of length cubed per time, and must be consistent with MODFLOW 6 length and time units. Usually a value of 1.0×10^{-1} is sufficient for the flow-residual criteria when meters and seconds are the defined MODFLOW 6 length and time.
- rclose_option (string) an optional keyword that defines the specific flow residual criterion used. STRICT-an optional keyword that is used to specify that INNER_RCLOSE represents a infinity-Norm (absolute convergence criteria) and that the dependent-variable (for example, head) and flow convergence criteria must be met on the first inner iteration (this criteria is equivalent to the criteria used by the MODFLOW-2005 PCG package (Hill, 1990)). L2NORM_RCLOSE-an optional keyword that is used to specify that INNER_RCLOSE represents a L-2 Norm closure criteria instead of a infinity-Norm (absolute convergence criteria). When L2NORM_RCLOSE is specified, a reasonable initial INNER_RCLOSE value is 0.1 times the number of active cells when meters and seconds are the defined MODFLOW 6 length and time. RELATIVE_RCLOSE-an optional keyword that is used to specify that INNER_RCLOSE represents a relative L-2 Norm reduction closure criteria instead of a infinity-Norm (absolute convergence criteria). When RELATIVE_RCLOSE is specified, a reasonable initial INNER_RCLOSE value is 1.0×10^{-4} and convergence is achieved for a given inner (linear) iteration when $\Delta h \leq \text{INNER_DVCLOSE}$ and the current L-2 Norm is \leq the product of the RELATIVE_RCLOSE and the initial L-2 Norm for the current inner (linear) iteration. If RCLOSE_OPTION is not specified, an absolute residual (infinity-norm) criterion is used.

- **linear_acceleration** (*string*) -

- linear_acceleration (string) a keyword that defines the linear acceleration method used by the default IMS linear solvers. CG - preconditioned conjugate gradient method. BICGSTAB - preconditioned bi-conjugate gradient stabilized method.
- **relaxation_factor** (double) -
 - relaxation_factor (double) optional real value that defines the relaxation factor used by the incomplete LU factorization preconditioners (MILU(0) and MILUT). RELAXATION_FACTOR is unitless and should be greater than or equal to 0.0 and less than or equal to 1.0. RELAXATION_FACTOR values of about 1.0 are commonly used, and experience suggests that convergence can be optimized in some cases with relax values of 0.97. A RELAXATION_FACTOR value of 0.0 will result in either ILU(0) or ILUT preconditioning (depending on the value specified for PRECONDITIONER_LEVELS and/or PRECONDITIONER_DROP_TOLERANCE). By default, RELAXATION_FACTOR is zero.
- **preconditioner_levels** (integer) -
 - preconditioner_levels (integer) optional integer value defining the level of fill for ILU decomposition used in the ILUT and MILUT preconditioners. Higher levels of fill provide more robustness but also require more memory. For optimal performance, it is suggested that a large level of fill be applied (7 or 8) with use of a drop tolerance. Specification of a PRECONDITIONER_LEVELS value greater than zero results in use of the ILUT preconditioner. By default, PRECONDITIONER_LEVELS is zero and the zero-fill incomplete LU factorization preconditioners (ILU(0) and MILU(0)) are used.
- **preconditioner_drop_tolerance** (double) -
 - preconditioner_drop_tolerance (double) optional real value that defines the drop tolerance used to drop preconditioner terms based on the magnitude of matrix entries in the ILUT and MILUT preconditioners. A value of 10^{-4} works well for most problems. By default, PRECONDITIONER_DROP_TOLERANCE is zero and the zero-fill incomplete LU factorization preconditioners (ILU(0) and MILU(0)) are used.
- **number_orthogonalizations** (integer) -
 - number_orthogonalizations (integer) optional integer value defining the interval used to explicitly recalculate the residual of the flow equation using the solver coefficient matrix, the latest dependent- variable (for example, head) estimates, and the right hand side. For problems that benefit from explicit recalculation of the residual, a number between 4 and 10 is appropriate. By default, NUMBER_ORTHOGONALIZATIONS is zero.
- **scaling_method** (string) -
 - scaling_method (string) an optional keyword that defines the matrix scaling approach used. By default, matrix scaling is not applied. NONE - no matrix scaling applied. DIAGONAL - symmetric matrix scaling using the POLCG preconditioner scaling

method in Hill (1992). L2NORM - symmetric matrix scaling using the L2 norm.

- **reordering_method** (*string*) -
 - **reordering_method** (*string*) an optional keyword that defines the matrix reordering approach used. By default, matrix reordering is not applied. NONE - original ordering. RCM - reverse Cuthill McKee ordering. MD - minimum degree ordering.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfullaktab package must have a mfgwflak package parent_file.

```
csv_inner_output_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator  
object>
```

```
csv_outer_output_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator  
object>
```

```
csv_output_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```

dfn = [['header', ['solution_package', '*']], ['block options', 'name print_option',
'type string', 'reader urword', 'optional true'], ['block options', 'name
complexity', 'type string', 'reader urword', 'optional true'], ['block options',
'name csv_output_filerecord', 'type record csv_output fileout csvfile', 'shape',
'reader urword', 'tagged true', 'optional true', 'deprecated 6.1.1'], ['block
options', 'name csv_output', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false', 'deprecated 6.1.1'], ['block options',
'name csvfile', 'type string', 'preserve_case true', 'shape', 'in_record true',
'reader urword', 'tagged false', 'optional false', 'deprecated 6.1.1'], ['block
options', 'name csv_outer_output_filerecord', 'type record csv_outer_output fileout
outer_csvfile', 'shape', 'reader urword', 'tagged true', 'optional true'], ['block
options', 'name csv_outer_output', 'type keyword', 'shape', 'in_record true',
'reader urword', 'tagged true', 'optional false'], ['block options', 'name fileout',
'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name outer_csvfile', 'type string', 'preserve_case
true', 'shape', 'in_record true', 'reader urword', 'tagged false', 'optional
false'], ['block options', 'name csv_inner_output_filerecord', 'type record
csv_inner_output fileout inner_csvfile', 'shape', 'reader urword', 'tagged true',
'optional true'], ['block options', 'name csv_inner_output', 'type keyword',
'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'],
['block options', 'name inner_csvfile', 'type string', 'preserve_case true',
'shape', 'in_record true', 'reader urword', 'tagged false', 'optional false'],
['block options', 'name no_ptcrecord', 'type record no_ptc no_ptc_option', 'reader
urword', 'optional true'], ['block options', 'name no_ptc', 'type keyword',
'in_record true', 'reader urword', 'optional false', 'tagged true'], ['block
options', 'name no_ptc_option', 'type string', 'in_record true', 'reader urword',
'optional true', 'tagged false'], ['block options', 'name
ats_outer_maximum_fraction', 'type double precision', 'reader urword', 'optional
true'], ['block nonlinear', 'name outer_hclose', 'type double precision', 'reader
urword', 'optional true', 'deprecated 6.1.1'], ['block nonlinear', 'name
outer_dvclose', 'type double precision', 'reader urword', 'optional false'], ['block
nonlinear', 'name outer_rclosebnd', 'type double precision', 'reader urword',
'optional true', 'deprecated 6.1.1'], ['block nonlinear', 'name outer_maximum',
'type integer', 'reader urword', 'optional false'], ['block nonlinear', 'name
under_relaxation', 'type string', 'reader urword', 'optional true'], ['block
nonlinear', 'name under_relaxation_gamma', 'type double precision', 'reader urword',
'optional true'], ['block nonlinear', 'name under_relaxation_theta', 'type double
precision', 'reader urword', 'optional true'], ['block nonlinear', 'name
under_relaxation_kappa', 'type double precision', 'reader urword', 'optional true'],
['block nonlinear', 'name under_relaxation_momentum', 'type double precision',
'reader urword', 'optional true'], ['block nonlinear', 'name backtracking_number',
'type integer', 'reader urword', 'optional true'], ['block nonlinear', 'name
backtracking_tolerance', 'type double precision', 'reader urword', 'optional true'],
['block nonlinear', 'name backtracking_reduction_factor', 'type double precision',
'reader urword', 'optional true'], ['block nonlinear', 'name
backtracking_residual_limit', 'type double precision', 'reader urword', 'optional
true'], ['block linear', 'name inner_maximum', 'type integer', 'reader urword',
'optional false'], ['block linear', 'name inner_hclose', 'type double precision',
'reader urword', 'optional true', 'deprecated 6.1.1'], ['block linear', 'name
inner_dvclose', 'type double precision', 'reader urword', 'optional false'], ['block
linear', 'name rcloserecord', 'type record inner_rclose rclose_option', 'reader
urword', 'optional false'], ['block linear', 'name inner_rclose', 'type double
precision', 'in_record true', 'reader urword', 'tagged true', 'optional false'],
['block linear', 'name rclose_option', 'type string', 'tagged false', 'in_record
true', 'reader urword', 'optional true'], ['block linear', 'name
linear_acceleration', 'type string', 'reader urword', 'optional false'], ['block
linear', 'name relaxation_factor', 'type double precision', 'reader urword',
'optional true'], ['block linear', 'name preconditioner_levels', 'type integer',
'reader urword', 'optional true'], ['block linear', 'name
preconditioner_drop_tolerance', 'type double precision', 'reader urword', 'optional

```

```
dfn_file_name = 'sln-ims.dfn'

no_ptcrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'ims'

rcloserecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfmvr module

```
class ModflowMvr(parent_model_or_package, loading_package=False, print_input=None,
                  print_flows=None, modelnames=None, budget_filerecord=None,
                  budgetcsv_filerecord=None, maxmvr=None, maxpackages=None, packages=None,
                  perioddata=None, filename=None, pname=None, **kwargs)
```

Bases: [*MFPackage*](#)

ModflowMvr defines a mvr package. This package can only be used to move water between two different models. To move water between two packages in the same model use the “model level” mover package (ex. ModflowGwfmvr).

Parameters

- **parent_model_or_package** (*MFModel/MFPackage*) -
Parent_model_or_package that this package is a part of. Package is automatically added to parent_model_or_package when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **print_input** (*boolean*) -
 - print_input (boolean) keyword to indicate that the list of MVR information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (boolean) keyword to indicate that the list of MVR flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **modelnames** (*boolean*) -
 - modelnames (boolean) keyword to indicate that all package names will be preceded by the model name for the package. Model names are required when the Mover Package is used with a GWF-GWF Exchange. The MODELNAME keyword should not be used for a Mover Package that is for a single GWF Model.
- **budget_filerecord** (*[budgetfile]*) -
 - budgetfile (string) name of the output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -

- `budgetcsvfile` (string) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **`maxmvr` (integer)** -
 - `maxmvr` (integer) integer value specifying the maximum number of water mover entries that will be specified for any stress period.
- **`maxpackages` (integer)** -
 - `maxpackages` (integer) integer value specifying the number of unique packages that are included in this water mover input file.
- **`packages` ([`mname`, `pname`])** -
 - `mname` (string) name of model containing the package. Model names are assigned by the user in the simulation name file.
 - `pname` (string) is the name of a package that may be included in a subsequent stress period block. The package name is assigned in the name file for the GWLF Model. Package names are optionally provided in the name file. If they are not provided by the user, then packages are assigned a default value, which is the package acronym followed by a hyphen and the package number. For example, the first Drain Package is named DRN-1. The second Drain Package is named DRN-2, and so forth.
- **`perioddata` ([`mname1`, `pname1`, `id1`, `mname2`, `pname2`, `id2`, `mvrtype`, `value`])** -
 - `mname1` (string) name of model containing the package, PNAME1.
 - `pname1` (string) is the package name for the provider. The package PNAME1 must be designated to provide water through the MVR Package by specifying the keyword "MOVER" in its OPTIONS block.
 - `id1` (integer) is the identifier for the provider. For the standard boundary packages, the provider identifier is the number of the boundary as it is listed in the package input file. (Note that the order of these boundaries may change by stress period, which must be accounted for in the Mover Package.) So the first well has an identifier of one. The second is two, and so forth. For the advanced packages, the identifier is the reach number (SFR Package), well number (MAW Package), or UZF cell number. For the Lake Package, ID1 is the lake outlet number. Thus, outflows from a single lake can be routed to different streams, for example. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.
 - `mname2` (string) name of model containing the package, PNAME2.
 - `pname2` (string) is the package name for the receiver. The package PNAME2 must be designated to receive water from the

MVR Package by specifying the keyword “MOVER” in its OPTIONS block.

- `id2` (integer) is the identifier for the receiver. The receiver identifier is the reach number (SFR Package), Lake number (LAK Package), well number (MAW Package), or UZF cell number. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.
 - `mvrtype` (string) is the character string signifying the method for determining how much water will be moved. Supported values are “FACTOR” “EXCESS” “THRESHOLD” and “UPTO”. These four options determine how the receiver flow rate, Q_R , is calculated. These options mirror the options defined for the `cprior` variable in the SFR package, with the term “FACTOR” being functionally equivalent to the “FRACTION” option for `cprior`.
 - `value` (double) is the value to be used in the equation for calculating the amount of water to move. For the “FACTOR” option, `VALUE` is the α factor. For the remaining options, `VALUE` is the specified flow rate, Q_S .
- `filename` (*String*) - File name for this package.
 - `pname` (*String*) - Package name for this package.
 - `parent_file` (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, `mfutllaktab` package must have a `mfgwflak` package `parent_file`.

```
budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```

dfn = [['header'], ['block options', 'name print_input', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name print_flows', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name modelnames', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name
budget_filerecord', 'type record budget fileout budgetfile', 'shape', 'reader
urword', 'tagged true', 'optional true'], ['block options', 'name budget', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name fileout', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
budgetfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budgetcsv_filerecord', 'type record budgetcsv fileout budgetcsvfile', 'shape',
'reader urword', 'tagged true', 'optional true'], ['block options', 'name
budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name budgetcsvfile', 'type string',
'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged false',
'optional false'], ['block dimensions', 'name maxmvr', 'type integer', 'reader
urword', 'optional false'], ['block dimensions', 'name maxpackages', 'type integer',
'reader urword', 'optional false'], ['block packages', 'name packages', 'type
recarray mname pname', 'reader urword', 'shape (npackages)', 'optional false'],
['block packages', 'name mname', 'type string', 'reader urword', 'shape', 'tagged
false', 'in_record true', 'optional true'], ['block packages', 'name pname', 'type
string', 'reader urword', 'shape', 'tagged false', 'in_record true', 'optional
false'], ['block period', 'name iper', 'type integer', 'block_variable True',
'in_record true', 'tagged false', 'shape', 'valid', 'reader urword', 'optional
false'], ['block period', 'name perioddata', 'type recarray mname1 pname1 id1 mname2
pname2 id2 mvrtype value', 'shape (maxbound)', 'reader urword'], ['block period',
'name mname1', 'type string', 'reader urword', 'shape', 'tagged false', 'in_record
true', 'optional true'], ['block period', 'name pname1', 'type string', 'shape',
'tagged false', 'in_record true', 'reader urword'], ['block period', 'name id1',
'type integer', 'shape', 'tagged false', 'in_record true', 'reader urword',
'numeric_index true'], ['block period', 'name mname2', 'type string', 'reader
urword', 'shape', 'tagged false', 'in_record true', 'optional true'], ['block
period', 'name pname2', 'type string', 'shape', 'tagged false', 'in_record true',
'reader urword'], ['block period', 'name id2', 'type integer', 'shape', 'tagged
false', 'in_record true', 'reader urword', 'numeric_index true'], ['block period',
'name mvrtype', 'type string', 'shape', 'tagged false', 'in_record true', 'reader
urword'], ['block period', 'name value', 'type double precision', 'shape', 'tagged
false', 'in_record true', 'reader urword']]

```

```
dfn_file_name = 'gwf-mvr.dfn'
```

```
package_abbrev = 'mvr'
```

```
packages = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
class MvrPackages(model_or_sim, parent, pkg_type, filerecord, package=None,
                  package_class=None)
```

Bases: [MFChildPackages](#)

MvrPackages is a container class for the ModflowMvr class.

```
initialize()
```

Initializes a new ModflowMvr package removing any sibling child packages

attached to the same parent package. See `ModflowMvr` init documentation for definition of parameters.

`append_package()`

Adds a new `ModflowGwfmvr` package to the container. See `ModflowGwfmvr` init documentation for definition of parameters.

```
append_package(print_input=None, print_flows=None, modelnames=None,
               budget_filerecord=None, budgetcsv_filerecord=None, maxmvr=None,
               maxpackages=None, packages=None, perioddata=None, filename=None,
               pname=None)
```

```
initialize(print_input=None, print_flows=None, modelnames=None,
           budget_filerecord=None, budgetcsv_filerecord=None, maxmvr=None,
           maxpackages=None, packages=None, perioddata=None, filename=None,
           pname=None)
```

```
package_abbr = 'mvrpackages'
```

`flopy.mf6.modflow.mfnam` module

```
class ModflowNam(simulation, loading_package=False, continue_=None, nocheck=None,
                 memory_print_option=None, maxerrors=None, print_input=None, tdis6=None,
                 models=None, exchanges=None, mxiter=None, solutiongroup=None,
                 filename=None, pname=None, **kwargs)
```

Bases: `MFPackage`

`ModflowNam` defines a `nam` package.

Parameters

- **`simulation`** (`MFSimulation`) - Simulation that this package is a part of. Package is automatically added to simulation when it is initialized.
- **`loading_package`** (`bool`) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **`continue`** (`boolean`) -
 - `continue` (`boolean`) keyword flag to indicate that the simulation should continue even if one or more solutions do not converge.
- **`nocheck`** (`boolean`) -
 - `nocheck` (`boolean`) keyword flag to indicate that the model input check routines should not be called prior to each time step. Checks are performed by default.
- **`memory_print_option`** (`string`) -
 - `memory_print_option` (`string`) is a flag that controls printing of detailed memory manager usage to the end of the simulation list file. `NONE` means do not print detailed information. `SUMMARY` means print only the total memory for each simulation component. `ALL` means print information for each variable stored in the memory manager. `NONE` is default if `MEMORY_PRINT_OPTION` is not specified.

- **maxerrors** (*integer*) -
 - maxerrors (*integer*) maximum number of errors that will be stored and printed.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to activate printing of simulation input summaries to the simulation list file (mfsim.lst). With this keyword, input summaries will be written for those packages that support newer input data model routines. Not all packages are supported yet by the newer input data model routines.
- **tdis6** (*string*) -
 - tdis6 (*string*) is the name of the Temporal Discretization (TDIS) Input File.
- **models** (*[mtype, mfname, mname]*) -
 - mtype (*string*) is the type of model to add to simulation.
 - mfname (*string*) is the file name of the model name file.
 - mname (*string*) is the user-assigned name of the model. The model name cannot exceed 16 characters and must not have blanks within the name. The model name is case insensitive; any lowercase letters are converted and stored as upper case letters.
- **exchanges** (*[exgtype, exgfile, exgmnamea, exgmnameb]*) -
 - exgtype (*string*) is the exchange type.
 - exgfile (*string*) is the input file for the exchange.
 - exgmnamea (*string*) is the name of the first model that is part of this exchange.
 - exgmnameb (*string*) is the name of the second model that is part of this exchange.
- **mxiter** (*integer*) -
 - mxiter (*integer*) is the maximum number of outer iterations for this solution group. The default value is 1. If there is only one solution in the solution group, then MXITER must be 1.
- **solutiongroup** (*[slntype, slnfname, slnmnames]*) -
 - slntype (*string*) is the type of solution. The Integrated Model Solution (IMS6) is the only supported option in this version.
 - slnfname (*string*) name of file containing solution input.
 - slnmnames (*string*) is the array of model names to add to this solution. The number of model names is determined by the number of model names the user provides on this line.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.

- `parent_file` ([MFPackage](#)) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutilaktab package must have a mfgwflak package `parent_file`.

```
dfn = [['header'], ['block options', 'name continue', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name nocheck', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name memory_print_option', 'type
string', 'reader urword', 'optional true', 'mf6internal prmem'], ['block options',
'name maxerrors', 'type integer', 'reader urword', 'optional true'], ['block
options', 'name print_input', 'type keyword', 'reader urword', 'optional true'],
['block timing', 'name tdis6', 'preserve_case true', 'type string', 'reader urword',
'optional'], ['block models', 'name models', 'type recarray mtype mfname mname',
'reader urword', 'optional'], ['block models', 'name mtype', 'in_record true', 'type
string', 'tagged false', 'reader urword'], ['block models', 'name mfname',
'in_record true', 'type string', 'preserve_case true', 'tagged false', 'reader
urword'], ['block models', 'name mname', 'in_record true', 'type string', 'tagged
false', 'reader urword'], ['block exchanges', 'name exchanges', 'type recarray
exgtype exgfile exgmnamea exgmnameb', 'reader urword', 'optional'], ['block
exchanges', 'name exgtype', 'in_record true', 'type string', 'tagged false', 'reader
urword'], ['block exchanges', 'name exgfile', 'in_record true', 'type string',
'preserve_case true', 'tagged false', 'reader urword'], ['block exchanges', 'name
exgmnamea', 'in_record true', 'type string', 'tagged false', 'reader urword'],
['block exchanges', 'name exgmnameb', 'in_record true', 'type string', 'tagged
false', 'reader urword'], ['block solutiongroup', 'name group_num', 'type integer',
'block_variable True', 'in_record true', 'tagged false', 'shape', 'reader urword'],
['block solutiongroup', 'name mxiter', 'type integer', 'reader urword', 'optional
true'], ['block solutiongroup', 'name solutiongroup', 'type recarray slntype
slnfname slnmnames', 'reader urword'], ['block solutiongroup', 'name slntype', 'type
string', 'valid ims6', 'in_record true', 'tagged false', 'reader urword'], ['block
solutiongroup', 'name slnfname', 'type string', 'preserve_case true', 'in_record
true', 'tagged false', 'reader urword'], ['block solutiongroup', 'name slnmnames',
'type string', 'in_record true', 'shape (:)', 'tagged false', 'reader urword']]
```

```
dfn_file_name = 'sim-nam.dfn'
```

```
exchanges = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
models = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'nam'
```

```
solutiongroup = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mftdis module

```
class ModflowTdis(simulation, loading_package=False, time_units=None,
                  start_date_time=None, ats_perioddata=None, nper=1, perioddata=((1.0, 1,
1.0)), filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowTdis defines a tdis package.

Parameters

- **simulation** (`MFSimulation`) - Simulation that this package is a part of. Package is automatically added to simulation when it is initialized.
- **loading_package** (`bool`) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **time_units** (`string`) -
 - `time_units` (`string`) is the time units of the simulation. This is a text string that is used as a label within model output files. Values for `time_units` may be “unknown”, “seconds”, “minutes”, “hours”, “days”, or “years”. The default time unit is “unknown”.
- **start_date_time** (`string`) -
 - `start_date_time` (`string`) is the starting date and time of the simulation. This is a text string that is used as a label within the simulation list file. The value has no effect on the simulation. The recommended format for the starting date and time is described at <https://www.w3.org/TR/NOTE-datetime>.
- **ats_perioddata** (`{varname:data}` or `perioddata data`) -
 - Contains data for the ats package. Data can be stored in a dictionary containing data for the ats package with variable names as keys and package data as values. Data just for the `ats_perioddata` variable is also acceptable. See ats package documentation for more information.
- **nper** (`integer`) -
 - `nper` (`integer`) is the number of stress periods for the simulation.
- **perioddata** (`[perlen, nstp, tsmult]`) -
 - `perlen` (`double`) is the length of a stress period.
 - `nstp` (`integer`) is the number of time steps in a stress period.
 - `tsmult` (`double`) is the multiplier for the length of successive time steps. The length of a time step is calculated by multiplying the length of the previous time step by TSMULT. The length of the first time step, Δt_1 , is related to PERLEN, NSTP, and TSMULT by the relation $\Delta t_1 = \text{perlen} \frac{\text{tsmult}-1}{\text{tsmult}^{\text{nstp}}-1}$.
- **filename** (`String`) - File name for this package.
- **pname** (`String`) - Package name for this package.
- **parent_file** (`MFPackage`) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfullaktab package must have a mfgwflak package `parent_file`.

`ats_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

```
dfn = [['header'], ['block options', 'name time_units', 'type string', 'reader
urword', 'optional true'], ['block options', 'name start_date_time', 'type string',
'reader urword', 'optional true'], ['block options', 'name ats_filerecord', 'type
record ats6 filein ats6_filename', 'shape', 'reader urword', 'tagged true',
'optional true', 'construct_package ats', 'construct_data perioddata',
'parameter_name ats_perioddata'], ['block options', 'name ats6', 'type keyword',
'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'],
['block options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name ats6_filename',
'type string', 'preserve_case true', 'in_record true', 'reader urword', 'optional
false', 'tagged false'], ['block dimensions', 'name nper', 'type integer', 'reader
urword', 'optional false', 'default_value 1'], ['block perioddata', 'name
perioddata', 'type recarray perlen nstp tsmult', 'reader urword', 'optional false',
'default_value ((1.0, 1, 1.0),)], ['block perioddata', 'name perlen', 'type double
precision', 'in_record true', 'tagged false', 'reader urword', 'optional false'],
['block perioddata', 'name nstp', 'type integer', 'in_record true', 'tagged false',
'reader urword', 'optional false'], ['block perioddata', 'name tsmult', 'type double
precision', 'in_record true', 'tagged false', 'reader urword', 'optional false']]

dfn_file_name = 'sim-tdis.dfn'

package_abbr = 'tdis'

perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

8.1.4 MODFLOW 6 Models

MODFLOW 6 supports both groundwater flow (`mfgwf.ModflowGwf`) and groundwater transport (`mfgwt.ModflowGwt`) models. FloPy for MODFLOW 6 model objects can be constructed after a FloPy simulation (`MFSimulation`) object has been constructed.

Contents:

`flopy.mf6.modflow.mfgwf` module

```
class ModflowGwf(simulation, modelname='model', model_nam_file=None, version='mf6',
                 exe_name='mf6', model_rel_path='.', list=None, print_input=None,
                 print_flows=None, save_flows=None, newtonoptions=None, packages=None,
                 **kwargs)
```

Bases: `MFModel`

`Modflowgwf` defines a `gwf` model

Parameters

- **modelname** (*string*) - name of the model
- **model_nam_file** (*string*) - relative path to the model name file from model working folder
- **version** (*string*) - version of modflow
- **exe_name** (*string*) - model executable name
- **model_ws** (*string*) - model working folder path

- **sim** (*MFSimulation*) - Simulation that this model is a part of. Model is automatically added to simulation when it is initialized.
- **list** (*string*) -
 - **list** (*string*) is name of the listing file to create for this GWF model. If not specified, then the name of the list file will be the basename of the GWF model name file and the '.lst' extension. For example, if the GWF name file is called "my.model.nam" then the list file will be called "my.model.lst".
- **print_input** (*boolean*) -
 - **print_input** (*boolean*) keyword to indicate that the list of all model stress package information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - **print_flows** (*boolean*) keyword to indicate that the list of all model package flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - **save_flows** (*boolean*) keyword to indicate that all model package flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **newtonoptions** (*[under_relaxation]*) -
 - **under_relaxation** (*string*) keyword that indicates whether the groundwater head in a cell will be under-relaxed when water levels fall below the bottom of the model below any given cell. By default, Newton-Raphson UNDER_RELAXATION is not applied.
- **packages** (*[ftype, fname, pname]*) -
 - **ftype** (*string*) is the file type, which must be one of the following character values shown in table in mf6io.pdf. Ftype may be entered in any combination of uppercase and lowercase.
 - **fname** (*string*) is the name of the file containing the package input. The path to the file should be included if the file is not located in the folder where the program was run.
 - **pname** (*string*) is the user-defined name for the package. PNAME is restricted to 16 characters. No spaces are allowed in PNAME. PNAME character values are read and stored by the program for stress packages only. These names may be useful for labeling purposes when multiple stress packages of the same type are located within a single GWF Model. If PNAME is specified for a stress package, then PNAME will be used in the flow budget table in the listing file; it will also be used for the text entry in the cell-by-cell budget file. PNAME is case insensitive and is stored in all upper case letters.

```
load : (simulation : MFSimulationData, model_name : string,  
       namfile : string, version : string, exe_name : string, model_ws : string,  
       strict : boolean) : MFSimulation a class method that loads a model from files  
  
classmethod load(simulation, structure, modelname='NewModel',  
                 model_nam_file='modflowtest.nam', version='mf6', exe_name='mf6',  
                 strict=True, model_rel_path='.', load_only=None)  
  
model_type = 'gwf'
```

flopy.mf6.modflow.mfgwt module

```
class ModflowGwt(simulation, modelname='model', model_nam_file=None, version='mf6',  
                 exe_name='mf6', model_rel_path='.', list=None, print_input=None,  
                 print_flows=None, save_flows=None, packages=None, **kwargs)
```

Bases: [MFModel](#)

Modflowgwt defines a gwt model

Parameters

- **modelname** (*string*) - name of the model
- **model_nam_file** (*string*) - relative path to the model name file from model working folder
- **version** (*string*) - version of modflow
- **exe_name** (*string*) - model executable name
- **model_ws** (*string*) - model working folder path
- **sim** ([MFSimulation](#)) - Simulation that this model is a part of. Model is automatically added to simulation when it is initialized.
- **list** (*string*) -
 - list (*string*) is name of the listing file to create for this GWT model. If not specified, then the name of the list file will be the basename of the GWT model name file and the '.lst' extension. For example, if the GWT name file is called "my.model.nam" then the list file will be called "my.model.lst".
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of all model stress package information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (*boolean*) keyword to indicate that the list of all model package flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -

- `save_flows` (boolean) keyword to indicate that all model package flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **packages** (*[ftype, fname, pname]*) -
 - `ftype` (string) is the file type, which must be one of the following character values shown in table in `mf6io.pdf`. `Ftype` may be entered in any combination of uppercase and lowercase.
 - `fname` (string) is the name of the file containing the package input. The path to the file should be included if the file is not located in the folder where the program was run.
 - `pname` (string) is the user-defined name for the package. `PNAME` is restricted to 16 characters. No spaces are allowed in `PNAME`. `PNAME` character values are read and stored by the program for stress packages only. These names may be useful for labeling purposes when multiple stress packages of the same type are located within a single GWT Model. If `PNAME` is specified for a stress package, then `PNAME` will be used in the flow budget table in the listing file; it will also be used for the text entry in the cell-by-cell budget file. `PNAME` is case insensitive and is stored in all upper case letters.

```
load : (simulation : MFSimulationData, model_name : string,
       namfile : string, version : string, exe_name : string, model_ws : string,
       strict : boolean) : MFSimulation a class method that loads a model from files

classmethod load(simulation, structure, modelname='NewModel',
                 model_nam_file='modflowtest.nam', version='mf6', exe_name='mf6',
                 strict=True, model_rel_path='.', load_only=None)

model_type = 'gwt'
```

8.1.5 MODFLOW 6 Groundwater Flow Model Packages

MODFLOW 6 groundwater flow models support a number of required and optional packages. Once a MODFLOW 6 groundwater flow model object (`mfgwf.ModflowGwf`) has been constructed various packages associated with the groundwater flow model can be constructed.

Contents:

`flopy.mf6.modflow.mfgwfapi` module

```
class ModflowGwfapi(model, loading_package=False, boundnames=None, print_input=None,
                    print_flows=None, save_flows=None, observations=None, mover=None,
                    maxbound=None, filename=None, pname=None, **kwargs)
```

Bases: `MFPackage`

`ModflowGwfapi` defines a api package within a `gwf6` model.

Parameters

- **model** (`MFModel`) - Model that this package is a part of. Package is automatically added to model when it is initialized.

- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **boundnames** (*boolean*) -
 - boundnames (*boolean*) keyword to indicate that boundary names may be provided with the list of api boundary cells.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of api boundary information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (*boolean*) keyword to indicate that the list of api boundary flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (*boolean*) keyword to indicate that api boundary flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **observations** (*{varname:data} or continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **mover** (*boolean*) -
 - mover (*boolean*) keyword to indicate that this instance of the api boundary Package can be used with the Water Mover (MVR) Package. When the MOVER option is specified, additional memory is allocated within the package to store the available, provided, and received water.
- **maxbound** (*integer*) -
 - maxbound (*integer*) integer value specifying the maximum number of api boundary cells that will be specified for use during any stress period.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

```

dfn = [['header'], ['block options', 'name boundnames', 'type keyword', 'shape',
'reader urword', 'optional true'], ['block options', 'name print_input', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name print_flows',
'type keyword', 'reader urword', 'optional true'], ['block options', 'name
save_flows', 'type keyword', 'reader urword', 'optional true'], ['block options',
'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape', 'reader
urword', 'tagged true', 'optional true', 'construct_package obs', 'construct_data
continuous', 'parameter_name observations'], ['block options', 'name obs6', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name filein', 'type keyword', 'shape', 'in_record true',
'reader urword', 'tagged true', 'optional false'], ['block options', 'name
obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block options', 'name mover', 'type
keyword', 'tagged true', 'reader urword', 'optional true'], ['block dimensions',
'name maxbound', 'type integer', 'reader urword', 'optional false']]

dfn_file_name = 'gwf-api.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfapi'

```

flopy.mf6.modflow.mfmgwfbuy module

```

class ModflowGwfbuy(model, loading_package=False, hhformulation_rhs=None,
    denseref=1000.0, density_filerecord=None, dev_efh_formulation=None,
    nrhospecies=None, packagedata=None, filename=None, pname=None,
    **kwargs)

```

Bases: [MFPackage](#)

ModflowGwfbuy defines a buy package within a gwf6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **hhformulation_rhs** (*boolean*) -
 - hhformulation_rhs (boolean) use the variable-density hydraulic head formulation and add off-diagonal terms to the right-hand. This option will prevent the BUY Package from adding asymmetric terms to the flow matrix.
- **denseref** (*double*) -
 - denseref (double) fluid reference density used in the equation of state. This value is set to 1000. if not specified as an option.
- **density_filerecord** (*[densityfile]*) -
 - densityfile (string) name of the binary output file to write density information. The density file has the same format as the head file. Density values will be written to the density

file whenever heads are written to the binary head file. The settings for controlling head output are contained in the Output Control option.

- **dev_efh_formulation** (*boolean*) -
 - **dev_efh_formulation** (*boolean*) use the variable-density equivalent freshwater head formulation instead of the hydraulic head head formulation. This dev option has only been implemented for confined aquifer conditions and should generally not be used.
- **nrhospecies** (*integer*) -
 - **nrhospecies** (*integer*) number of species used in density equation of state. This value must be one or greater if the BUY package is activated.
- **packagedata** (*[irhospec, drhodc, crhoref, modelname, auxspeciesname]*) -
 - **irhospec** (*integer*) integer value that defines the species number associated with the specified PACKAGEDATA data on the line. IRHOSPECIES must be greater than zero and less than or equal to NRHOSPECIES. Information must be specified for each of the NRHOSPECIES species or the program will terminate with an error. The program will also terminate with an error if information for a species is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - **drhodc** (*double*) real value that defines the slope of the density- concentration line for this species used in the density equation of state.
 - **crhoref** (*double*) real value that defines the reference concentration value used for this species in the density equation of state.
 - **modelname** (*string*) name of GWT model used to simulate a species that will be used in the density equation of state. This name will have no effect if the simulation does not include a GWT model that corresponds to this GWF model.
 - **auxspeciesname** (*string*) name of an auxiliary variable in a GWF stress package that will be used for this species to calculate a density value. If a density value is needed by the Buoyancy Package then it will use the concentration values in this AUXSPECIESNAME column in the density equation of state. For advanced stress packages (LAK, SFR, MAW, and UZF) that have an associated advanced transport package (LKT, SFT, MWT, and UZT), the FLOW_PACKAGE_AUXILIARY_NAME option in the advanced transport package can be used to transfer simulated concentrations into the flow package auxiliary variable. In this manner, the Buoyancy Package can calculate density values for lakes, streams, multi-aquifer wells, and unsaturated zone flow cells using simulated concentrations.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulaktab package must have a mfgwflak package parent_file.

```
density_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
dfn = [['header'], ['block options', 'name hhformulation_rhs', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name denseref', 'type double
precision', 'reader urword', 'optional true', 'default_value 1000.'], ['block
options', 'name density_filerecord', 'type record density fileout densityfile',
'shape', 'reader urword', 'tagged true', 'optional true'], ['block options', 'name
density', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name fileout', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name densityfile', 'type string', 'preserve_case true', 'shape',
'in_record true', 'reader urword', 'tagged false', 'optional false'], ['block
options', 'name dev_efh_formulation', 'type keyword', 'reader urword', 'optional
true'], ['block dimensions', 'name nrhospecies', 'type integer', 'reader urword',
'optional false'], ['block packagedata', 'name packagedata', 'type recarray irhospec
drhodc crhoref modelname auxspeciesname', 'shape (nrhospecies)', 'reader urword'],
['block packagedata', 'name irhospec', 'type integer', 'shape', 'tagged false',
'in_record true', 'reader urword', 'numeric_index true'], ['block packagedata',
'name drhodc', 'type double precision', 'shape', 'tagged false', 'in_record true',
'reader urword'], ['block packagedata', 'name crhoref', 'type double precision',
'shape', 'tagged false', 'in_record true', 'reader urword'], ['block packagedata',
'name modelname', 'type string', 'in_record true', 'tagged false', 'shape', 'reader
urword'], ['block packagedata', 'name auxspeciesname', 'type string', 'in_record
true', 'tagged false', 'shape', 'reader urword']]
```

```
dfn_file_name = 'gwf-buy.dfn'
```

```
package_abbr = 'gwfbuy'
```

```
packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfchd module

```
class ModflowGwfchd(model, loading_package=False, auxiliary=None, auxmultname=None,
                    boundnames=None, print_input=None, print_flows=None, save_flows=None,
                    timeseries=None, observations=None, maxbound=None,
                    stress_period_data=None, filename=None, pname=None, **kwargs)
```

Bases: *MFPackage*

ModflowGwfchd defines a chd package within a gwf6 model.

Parameters

- **model** (*MFModel*) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **auxiliary** (*[string]*) -

- auxiliary (string) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** (string) -
 - auxmultname (string) name of auxiliary variable to be used as multiplier of CHD head value.
- **boundnames** (boolean) -
 - boundnames (boolean) keyword to indicate that boundary names may be provided with the list of constant-head cells.
- **print_input** (boolean) -
 - print_input (boolean) keyword to indicate that the list of constant- head information will be written to the listing file immediately after it is read.
- **print_flows** (boolean) -
 - print_flows (boolean) keyword to indicate that the list of constant- head flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (boolean) -
 - save_flows (boolean) keyword to indicate that constant-head flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **timeseries** ({varname:data} or timeseries data) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** ({varname:data} or continuous data) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **maxbound** (integer) -

- maxbound (integer) integer value specifying the maximum number of constant-head cells that will be specified for use during any stress period.
- **stress_period_data** ([*cellid*, *head*, *aux*, *boundname*]) -
 - *cellid* ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *head* (double) is the head at the boundary. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - *aux* (double) represents the values of the auxiliary variables for each constant head. The values of auxiliary variables must be present for each constant head. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - *boundname* (string) name of the constant head boundary cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfullaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```
dfn = [['header', 'multi-package'], ['block options', 'name auxiliary', 'type
string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name
auxmultname', 'type string', 'shape', 'reader urword', 'optional true'], ['block
options', 'name boundnames', 'type keyword', 'shape', 'reader urword', 'optional
true'], ['block options', 'name print_input', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name print_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name save_flows', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name ts_filerecord', 'type
record ts6 filein ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional
true', 'construct_package ts', 'construct_data timeseries', 'parameter_name
timeseries'], ['block options', 'name ts6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
filein', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name ts6_filename', 'type string',
'preserve_case true', 'in_record true', 'reader urword', 'optional false', 'tagged
false'], ['block options', 'name obs_filerecord', 'type record obs6 filein
obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package obs', 'construct_data continuous', 'parameter_name
observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block dimensions', 'name maxbound',
'type integer', 'reader urword', 'optional false'], ['block period', 'name iper',
'type integer', 'block_variable True', 'in_record true', 'tagged false', 'shape',
'valid', 'reader urword', 'optional false'], ['block period', 'name
stress_period_data', 'type recarray cellid head aux boundname', 'shape (maxbound)',
'reader urword'], ['block period', 'name cellid', 'type integer', 'shape
(ncelldim)', 'tagged false', 'in_record true', 'reader urword'], ['block period',
'name head', 'type double precision', 'shape', 'tagged false', 'in_record true',
'reader urword', 'time_series true'], ['block period', 'name aux', 'type double
precision', 'in_record true', 'tagged false', 'shape (naux)', 'reader urword',
'optional true', 'time_series true'], ['block period', 'name boundname', 'type
string', 'shape', 'tagged false', 'in_record true', 'reader urword', 'optional
true']]
```

```
dfn_file_name = 'gwf-chd.dfn'
```

```
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'gwfchd'
```

```
stress_period_data = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flop.py.mf6.modflow.mfgwfcs sub module

```
class ModflowGwfcs sub(model, loading_package=False, boundnames=None, print_input=None,
                        save_flows=None, gammaw=9806.65, beta=4.6512e-10, head_based=None,
                        initial_preconsolidation_head=None, ndelaycells=None,
                        compression_indices=None, update_material_properties=None,
                        cell_fraction=None, specified_initial_interbed_state=None,
                        specified_initial_preconsolidation_stress=None,
                        specified_initial_delay_head=None, effective_stress_lag=None,
                        strainib_filerecord=None, straincg_filerecord=None,
                        compaction_filerecord=None, compaction_elastic_filerecord=None,
                        compaction_inelastic_filerecord=None,
                        compaction_interbed_filerecord=None,
                        compaction_coarse_filerecord=None, zdisplacement_filerecord=None,
                        package_convergence_filerecord=None, timeseries=None,
                        observations=None, ninterbeds=None, maxsig0=None, cg_ske_cr=1e-05,
                        cg_theta=0.2, sgm=None, sgs=None, packagedata=None,
                        stress_period_data=None, filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwfcs sub defines a csub package within a gw6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** ([bool](#)) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **boundnames** ([boolean](#)) -
 - boundnames ([boolean](#)) keyword to indicate that boundary names may be provided with the list of CSUB cells.
- **print_input** ([boolean](#)) -
 - print_input ([boolean](#)) keyword to indicate that the list of CSUB information will be written to the listing file immediately after it is read.
- **save_flows** ([boolean](#)) -
 - save_flows ([boolean](#)) keyword to indicate that cell-by-cell flow terms will be written to the file specified with "BUDGET SAVE FILE" in Output Control.
- **gammaw** ([double](#)) -
 - gammaw ([double](#)) unit weight of water. For freshwater, GAMMAW is 9806.65 Newtons/cubic meters or 62.48 lb/cubic foot in SI and English units, respectively. By default, GAMMAW is 9806.65 Newtons/cubic meters.
- **beta** ([double](#)) -
 - beta ([double](#)) compressibility of water. Typical values of BETA are 4.6512e-10 1/Pa or 2.2270e-8 lb/square foot in SI and English units, respectively. By default, BETA is 4.6512e-10 1/Pa.

- **head_based** (*boolean*) -
 - head_based (boolean) keyword to indicate the head-based formulation will be used to simulate coarse-grained aquifer materials and no- delay and delay interbeds. Specifying HEAD_BASED also specifies the INITIAL_PRECONSOLIDATION_HEAD option.
- **initial_preconsolidation_head** (*boolean*) -
 - initial_preconsolidation_head (boolean) keyword to indicate that preconsolidation heads will be specified for no-delay and delay interbeds in the PACKAGEDATA block. If the SPECIFIED_INITIAL_INTERBED_STATE option is specified in the OPTIONS block, user-specified preconsolidation heads in the PACKAGEDATA block are absolute values. Otherwise, user-specified preconsolidation heads in the PACKAGEDATA block are relative to steady-state or initial heads.
- **ndelaycells** (*integer*) -
 - ndelaycells (integer) number of nodes used to discretize delay interbeds. If not specified, then a default value of 19 is assigned.
- **compression_indices** (*boolean*) -
 - compression_indices (boolean) keyword to indicate that the recompression (CR) and compression (CC) indices are specified instead of the elastic specific storage (SSE) and inelastic specific storage (SSV) coefficients. If not specified, then elastic specific storage (SSE) and inelastic specific storage (SSV) coefficients must be specified.
- **update_material_properties** (*boolean*) -
 - update_material_properties (boolean) keyword to indicate that the thickness and void ratio of coarse-grained and interbed sediments (delay and no-delay) will vary during the simulation. If not specified, the thickness and void ratio of coarse-grained and interbed sediments will not vary during the simulation.
- **cell_fraction** (*boolean*) -
 - cell_fraction (boolean) keyword to indicate that the thickness of interbeds will be specified in terms of the fraction of cell thickness. If not specified, interbed thickness must be specified.
- **specified_initial_interbed_state** (*boolean*) -
 - specified_initial_interbed_state (boolean) keyword to indicate that absolute preconsolidation stresses (heads) and delay bed heads will be specified for interbeds defined in the PACKAGEDATA block. The SPECIFIED_INITIAL_INTERBED_STATE option is equivalent to specifying the SPECIFIED_INITIAL_PRECONSOLITATION_STRESS and SPECIFIED_INITIAL_DELAY_HEAD. If SPECIFIED_INITIAL_INTERBED_STATE is not specified then preconsolidation stress (head) and delay bed head values specified in the PACKAGEDATA block are relative

to simulated values of the first stress period if steady-state or initial stresses and GWF heads if the first stress period is transient.

- **specified_initial_preconsolidation_stress** (*boolean*) -
 - `specified_initial_preconsolidation_stress` (*boolean*) keyword to indicate that absolute preconsolidation stresses (heads) will be specified for interbeds defined in the PACKAGEDATA block. If SPECIFIED_INITIAL_PRECONSOLITATION_STRESS and SPECIFIED_INITIAL_INTERBED_STATE are not specified then preconsolidation stress (head) values specified in the PACKAGEDATA block are relative to simulated values if the first stress period is steady-state or initial stresses (heads) if the first stress period is transient.
- **specified_initial_delay_head** (*boolean*) -
 - `specified_initial_delay_head` (*boolean*) keyword to indicate that absolute initial delay bed head will be specified for interbeds defined in the PACKAGEDATA block. If SPECIFIED_INITIAL_DELAY_HEAD and SPECIFIED_INITIAL_INTERBED_STATE are not specified then delay bed head values specified in the PACKAGEDATA block are relative to simulated values if the first stress period is steady-state or initial GWF heads if the first stress period is transient.
- **effective_stress_lag** (*boolean*) -
 - `effective_stress_lag` (*boolean*) keyword to indicate the effective stress from the previous time step will be used to calculate specific storage values. This option can 1) help with convergence in models with thin cells and water table elevations close to land surface; 2) is identical to the approach used in the SUBWT package for MODFLOW-2005; and 3) is only used if the effective-stress formulation is being used. By default, current effective stress values are used to calculate specific storage values.
- **strainib_filerecord** (*[interbedstrain_filename]*) -
 - `interbedstrain_filename` (*string*) name of the comma-separated-values output file to write final interbed strain information.
- **straincg_filerecord** (*[coarsestrain_filename]*) -
 - `coarsestrain_filename` (*string*) name of the comma-separated-values output file to write final coarse-grained material strain information.
- **compaction_filerecord** (*[compaction_filename]*) -
 - `compaction_filename` (*string*) name of the binary output file to write compaction information.
- **compaction_elastic_filerecord** (*[elastic_compaction_filename]*) -
 - `elastic_compaction_filename` (*string*) name of the binary output file to write elastic interbed compaction information.

- **compaction_inelastic_filerecord** (*[inelastic_compaction_filename]*) -
 - *inelastic_compaction_filename* (string) name of the binary output file to write inelastic interbed compaction information.
- **compaction_interbed_filerecord** (*[interbed_compaction_filename]*) -
 - *interbed_compaction_filename* (string) name of the binary output file to write interbed compaction information.
- **compaction_coarse_filerecord** (*[coarse_compaction_filename]*) -
 - *coarse_compaction_filename* (string) name of the binary output file to write elastic coarse-grained material compaction information.
- **zdisplacement_filerecord** (*[zdisplacement_filename]*) -
 - *zdisplacement_filename* (string) name of the binary output file to write z-displacement information.
- **package_convergence_filerecord** (*[package_convergence_filename]*) -
 - *package_convergence_filename* (string) name of the comma spaced values output file to write package convergence information.
- **timeseries** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **ninterbeds** (*integer*) -
 - *ninterbeds* (integer) is the number of CSUB interbed systems. More than 1 CSUB interbed systems can be assigned to a GWF cell; however, only 1 GWF cell can be assigned to a single CSUB interbed system.
- **maxsig0** (*integer*) -
 - *maxsig0* (integer) is the maximum number of cells that can have a specified stress offset. More than 1 stress offset can be assigned to a GWF cell. By default, MAXSIG0 is 0.
- **cg_ske_cr** (*[double]*) -
 - *cg_ske_cr* (double) is the initial elastic coarse-grained material specific storage or recompression index. The recompression index is specified if COMPRESSION_INDICES is specified in the OPTIONS block. Specified or calculated elastic coarse-grained material specific storage values are not adjusted

from initial values if HEAD_BASED is specified in the OPTIONS block.

- **cg_theta** ([double]) -
 - cg_theta (double) is the initial porosity of coarse-grained materials.
- **sgm** ([double]) -
 - sgm (double) is the specific gravity of moist or unsaturated sediments. If not specified, then a default value of 1.7 is assigned.
- **sgs** ([double]) -
 - sgs (double) is the specific gravity of saturated sediments. If not specified, then a default value of 2.0 is assigned.
- **packagedata** ([icsubno, cellid, cdelay, pcs0, thick_frac, rnb, ssv_cc,]) -
 - sse_cr, theta, kv, h0, boundname]**
 - icsubno (integer) integer value that defines the CSUB interbed number associated with the specified PACKAGEDATA data on the line. CSUBNO must be greater than zero and less than or equal to NINTERBEDS. CSUB information must be specified for every CSUB cell or the program will terminate with an error. The program will also terminate with an error if information for a CSUB interbed number is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - cellid ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - cdelay (string) character string that defines the subsidence delay type for the interbed. Possible subsidence package CDELAY strings include: NODELAY-character keyword to indicate that delay will not be simulated in the interbed. DELAY-character keyword to indicate that delay will be simulated in the interbed.
 - pcs0 (double) is the initial offset from the calculated initial effective stress or initial preconsolidation stress in the interbed, in units of height of a column of water. PCS0 is the initial preconsolidation stress if SPECIFIED_INITIAL_INTERBED_STATE or

SPECIFIED_INITIAL_PRECONSOLIDATION_STRESS are specified in the OPTIONS block. If HEAD_BASED is specified in the OPTIONS block, PCS0 is the initial offset from the calculated initial head or initial preconsolidation head in the CSUB interbed and the initial preconsolidation stress is calculated from the calculated initial effective stress or calculated initial geostatic stress, respectively.

- thick_frac (double) is the interbed thickness or cell fraction of the interbed. Interbed thickness is specified as a fraction of the cell thickness if CELL_FRACTION is specified in the OPTIONS block.
 - rnb (double) is the interbed material factor equivalent number of interbeds in the interbed system represented by the interbed. RNB must be greater than or equal to 1 if CDELAY is DELAY. Otherwise, RNB can be any value.
 - ssv_cc (double) is the initial inelastic specific storage or compression index of the interbed. The compression index is specified if COMPRESSION_INDICES is specified in the OPTIONS block. Specified or calculated interbed inelastic specific storage values are not adjusted from initial values if HEAD_BASED is specified in the OPTIONS block.
 - sse_cr (double) is the initial elastic coarse-grained material specific storage or recompression index of the interbed. The recompression index is specified if COMPRESSION_INDICES is specified in the OPTIONS block. Specified or calculated interbed elastic specific storage values are not adjusted from initial values if HEAD_BASED is specified in the OPTIONS block.
 - theta (double) is the initial porosity of the interbed.
 - kv (double) is the vertical hydraulic conductivity of the delay interbed. KV must be greater than 0 if CDELAY is DELAY. Otherwise, KV can be any value.
 - h0 (double) is the initial offset from the head in cell cellid or the initial head in the delay interbed. H0 is the initial head in the delay bed if SPECIFIED_INITIAL_INTERBED_STATE or SPECIFIED_INITIAL_DELAY_HEAD are specified in the OPTIONS block. H0 can be any value if CDELAY is NODELAY.
 - boundname (string) name of the CSUB cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.
- **stress_period_data** ([cellid, sig0]) -
 - cellid ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable,

which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.

- sig0 (double) is the stress offset for the cell. SIG0 is added to the calculated geostatic stress for the cell. SIG0 is specified only if MAXSIG0 is specified to be greater than 0 in the DIMENSIONS block. If the Options block includes a TIMESERIESFILE entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulaktab package must have a mfgwflak package parent_file.

```
cg_ske_cr = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
cg_theta = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
compaction_coarse_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
compaction_elastic_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
compaction_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
compaction_inelastic_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
compaction_interbed_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```

dfn = [['header'], ['block options', 'name boundnames', 'type keyword', 'shape',
'reader urword', 'optional true'], ['block options', 'name print_input', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name save_flows',
'type keyword', 'reader urword', 'optional true'], ['block options', 'name gammaw',
'type double precision', 'reader urword', 'optional true', 'default_value 9806.65'],
['block options', 'name beta', 'type double precision', 'reader urword', 'optional
true', 'default_value 4.6512e-10'], ['block options', 'name head_based', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name
initial_preconsolidation_head', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name ndelaycells', 'type integer', 'reader urword', 'optional
true'], ['block options', 'name compression_indices', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name update_material_properties',
'type keyword', 'reader urword', 'optional true'], ['block options', 'name
cell_fraction', 'type keyword', 'reader urword', 'optional true'], ['block options',
'name specified_initial_interbed_state', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name specified_initial_preconsolidation_stress', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name
specified_initial_delay_head', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name effective_stress_lag', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name strainib_filerecord', 'type record
strain_csv_interbed fileout interbedstrain_filename', 'shape', 'reader urword',
'tagged true', 'optional true'], ['block options', 'name strain_csv_interbed', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name fileout', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
interbedstrain_filename', 'type string', 'shape', 'in_record true', 'reader urword',
'tagged false', 'optional false'], ['block options', 'name straincg_filerecord',
'type record strain_csv_coarse fileout coarsestrain_filename', 'shape', 'reader
urword', 'tagged true', 'optional true'], ['block options', 'name
strain_csv_coarse', 'type keyword', 'shape', 'in_record true', 'reader urword',
'tagged true', 'optional false'], ['block options', 'name coarsestrain_filename',
'type string', 'shape', 'in_record true', 'reader urword', 'tagged false', 'optional
false'], ['block options', 'name compaction_filerecord', 'type record compaction
fileout compaction_filename', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name compaction', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
compaction_filename', 'type string', 'shape', 'in_record true', 'reader urword',
'tagged false', 'optional false'], ['block options', 'name
compaction_elastic_filerecord', 'type record compaction_elastic fileout
elastic_compaction_filename', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name compaction_elastic', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name elastic_compaction_filename', 'type string', 'shape', 'in_record
true', 'reader urword', 'tagged false', 'optional false'], ['block options', 'name
compaction_inelastic_filerecord', 'type record compaction_inelastic fileout
inelastic_compaction_filename', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name compaction_inelastic', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name inelastic_compaction_filename', 'type string', 'shape', 'in_record
true', 'reader urword', 'tagged false', 'optional false'], ['block options', 'name
compaction_interbed_filerecord', 'type record compaction_interbed fileout
interbed_compaction_filename', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name compaction_interbed', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name interbed_compaction_filename', 'type string', 'shape', 'in_record
true', 'reader urword', 'tagged false', 'optional false'], ['block options', 'name
compaction_coarse_filerecord', 'type record compaction_coarse fileout
coarse_compaction_filename', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name compaction_coarse', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block

```

```

dfn_file_name = 'gwf-csub.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfcsub'

package_convergence_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator
object>

packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

sgm = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

sgs = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

straincg_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

strainib_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

stress_period_data = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

zdisplacement_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

flopy.mf6.modflow.mfgwfdis module

```

class ModflowGwfdis(model, loading_package=False, length_units=None, nogrb=None,
                    xorigin=None, yorigin=None, angrot=None, nlay=1, nrow=2, ncol=2,
                    delr=1.0, delc=1.0, top=1.0, botm=0.0, idomain=None, filename=None,
                    pname=None, **kwargs)

```

Bases: [*MFPackage*](#)

ModflowGwfdis defines a dis package within a gwf6 model.

Parameters

- **model** ([*MFModel*](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **length_units** (*string*) -
 - length_units (*string*) is the length units used for this model. Values can be “FEET”, “METERS”, or “CENTIMETERS”. If not specified, the default is “UNKNOWN”.
- **nogrb** (*boolean*) -
 - nogrb (*boolean*) keyword to deactivate writing of the binary grid file.
- **xorigin** (*double*) -

- **xorigin** (double) x-position of the lower-left corner of the model grid. A default value of zero is assigned if not specified. The value for XORIGIN does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **yorigin** (double) -
 - **yorigin** (double) y-position of the lower-left corner of the model grid. If not specified, then a default value equal to zero is used. The value for YORIGIN does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **angrot** (double) -
 - **angrot** (double) counter-clockwise rotation angle (in degrees) of the lower-left corner of the model grid. If not specified, then a default value of 0.0 is assigned. The value for ANGROT does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **nlay** (integer) -
 - **nlay** (integer) is the number of layers in the model grid.
- **nrow** (integer) -
 - **nrow** (integer) is the number of rows in the model grid.
- **ncol** (integer) -
 - **ncol** (integer) is the number of columns in the model grid.
- **delr** ([double]) -
 - **delr** (double) is the column spacing in the row direction.
- **delc** ([double]) -
 - **delc** (double) is the row spacing in the column direction.
- **top** ([double]) -
 - **top** (double) is the top elevation for each cell in the top model layer.
- **botm** ([double]) -
 - **botm** (double) is the bottom elevation for each cell.
- **idomain** ([integer]) -
 - **idomain** (integer) is an optional array that characterizes the existence status of a cell. If the IDOMAIN array is not specified, then all model cells exist within the solution. If the IDOMAIN value for a cell is 0, the cell does not exist in the simulation. Input and output values will be read and written for the cell, but internal to the program, the cell is excluded from the solution. If the IDOMAIN value for a cell is 1 or greater, the cell exists in the simulation. If the IDOMAIN value for a cell is -1, the cell does not exist in the simulation. Furthermore, the first existing cell above will be

connected to the first existing cell below. This type of cell is referred to as a “vertical pass through” cell.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulaktab package must have a mfgwflak package parent_file.

```
botm = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
delc = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
delr = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
dfn = [['header'], ['block options', 'name length_units', 'type string', 'reader
urword', 'optional true'], ['block options', 'name nogrb', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name xorigin', 'type double
precision', 'reader urword', 'optional true'], ['block options', 'name yorigin',
'type double precision', 'reader urword', 'optional true'], ['block options', 'name
angrot', 'type double precision', 'reader urword', 'optional true'], ['block
dimensions', 'name nlay', 'type integer', 'reader urword', 'optional false',
'default_value 1'], ['block dimensions', 'name nrow', 'type integer', 'reader
urword', 'optional false', 'default_value 2'], ['block dimensions', 'name ncol',
'type integer', 'reader urword', 'optional false', 'default_value 2'], ['block
griddata', 'name delr', 'type double precision', 'shape (ncol)', 'reader readarray',
'default_value 1.0'], ['block griddata', 'name delc', 'type double precision',
'shape (nrow)', 'reader readarray', 'default_value 1.0'], ['block griddata', 'name
top', 'type double precision', 'shape (ncol, nrow)', 'reader readarray',
'default_value 1.0'], ['block griddata', 'name botm', 'type double precision',
'shape (ncol, nrow, nlay)', 'reader readarray', 'layered true', 'default_value 0.'],
['block griddata', 'name idomain', 'type integer', 'shape (ncol, nrow, nlay)',
'reader readarray', 'layered true', 'optional true']]
```

```
dfn_file_name = 'gwf-dis.dfn'
```

```
idomain = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
package_abbr = 'gwfdis'
```

```
top = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfdisu module

```
class ModflowGwfdisu(model, loading_package=False, length_units=None, nogrb=None,
                      xorigin=None, yorigin=None, angrot=None,
                      vertical_offset_tolerance=0.0, nodes=None, nja=None, nvert=None,
                      top=None, bot=None, area=None, idomain=None, iac=None, ja=None,
                      ihc=None, cl12=None, hwva=None, angldegx=None, vertices=None,
                      cell12d=None, filename=None, pname=None, **kwargs)
```

Bases: *MFPackage*

ModflowGwfdisu defines a disu package within a gwf6 model.

Parameters

- **model** (*MFModel*) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **length_units** (*string*) -
 - length_units (*string*) is the length units used for this model. Values can be "FEET", "METERS", or "CENTIMETERS". If not specified, the default is "UNKNOWN".
- **nogrb** (*boolean*) -
 - nogrb (*boolean*) keyword to deactivate writing of the binary grid file.
- **xorigin** (*double*) -
 - xorigin (*double*) x-position of the origin used for model grid vertices. This value should be provided in a real-world coordinate system. A default value of zero is assigned if not specified. The value for XORIGIN does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **yorigin** (*double*) -
 - yorigin (*double*) y-position of the origin used for model grid vertices. This value should be provided in a real-world coordinate system. If not specified, then a default value equal to zero is used. The value for YORIGIN does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **angrot** (*double*) -
 - angrot (*double*) counter-clockwise rotation angle (in degrees) of the model grid coordinate system relative to a real-world coordinate system. If not specified, then a default value of 0.0 is assigned. The value for ANGROT does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **vertical_offset_tolerance** (*double*) -
 - vertical_offset_tolerance (*double*) checks are performed to ensure that the top of a cell is not higher than the bottom of an overlying cell. This option can be used to specify the tolerance that is used for checking. If top of a cell is above the bottom of an overlying cell by a value less than this tolerance, then the program will not terminate with an error. The default value is zero. This option should generally not be used.
- **nodes** (*integer*) -
 - nodes (*integer*) is the number of cells in the model grid.
- **nja** (*integer*) -

- `nja` (integer) is the sum of the number of connections and `NODES`. When calculating the total number of connections, the connection between cell `n` and cell `m` is considered to be different from the connection between cell `m` and cell `n`. Thus, `NJA` is equal to the total number of connections, including `n` to `m` and `m` to `n`, and the total number of cells.
- **`nvert` ([integer]) -**
 - `nvert` (integer) is the total number of (x, y) vertex pairs used to define the plan-view shape of each cell in the model grid. If `NVERT` is not specified or is specified as zero, then the `VERTICES` and `CELL2D` blocks below are not read. `NVERT` and the accompanying `VERTICES` and `CELL2D` blocks should be specified for most simulations. If the `XT3D` or `SAVE_SPECIFIC_DISCHARGE` options are specified in the `NPF` Package, then this information is required.
- **`top` ([double]) -**
 - `top` (double) is the top elevation for each cell in the model grid.
- **`bot` ([double]) -**
 - `bot` (double) is the bottom elevation for each cell.
- **`area` ([double]) -**
 - `area` (double) is the cell surface area (in plan view).
- **`idomain` ([integer]) -**
 - `idomain` (integer) is an optional array that characterizes the existence status of a cell. If the `IDOMAIN` array is not specified, then all model cells exist within the solution. If the `IDOMAIN` value for a cell is 0, the cell does not exist in the simulation. Input and output values will be read and written for the cell, but internal to the program, the cell is excluded from the solution. If the `IDOMAIN` value for a cell is 1 or greater, the cell exists in the simulation. `IDOMAIN` values of -1 cannot be specified for the `DISU` Package.
- **`iac` ([integer]) -**
 - `iac` (integer) is the number of connections (plus 1) for each cell. The sum of all the entries in `IAC` must be equal to `NJA`.
- **`ja` ([integer]) -**
 - `ja` (integer) is a list of cell number (`n`) followed by its connecting cell numbers (`m`) for each of the `m` cells connected to cell `n`. The number of values to provide for cell `n` is `IAC(n)`. This list is sequentially provided for the first to the last cell. The first value in the list must be cell `n` itself, and the remaining cells must be listed in an increasing order (sorted from lowest number to highest). Note that the cell and its connections are only supplied for the `GWF` cells and their connections to the other `GWF` cells. Also note that the `JA` list input may be divided such that every node and its connectivity list can be on a separate line for ease in readability of

the file. To further ease readability of the file, the node number of the cell whose connectivity is subsequently listed, may be expressed as a negative number, the sign of which is subsequently converted to positive by the code. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.

- **ihc** ([integer]) -
 - **ihc** (integer) is an index array indicating the direction between node *n* and all of its *m* connections. If **IHC** = 0 then cell *n* and cell *m* are connected in the vertical direction. Cell *n* overlies cell *m* if the cell number for *n* is less than *m*; cell *m* overlies cell *n* if the cell number for *m* is less than *n*. If **IHC** = 1 then cell *n* and cell *m* are connected in the horizontal direction. If **IHC** = 2 then cell *n* and cell *m* are connected in the horizontal direction, and the connection is vertically staggered. A vertically staggered connection is one in which a cell is horizontally connected to more than one cell in a horizontal connection.
- **cl12** ([double]) -
 - **cl12** (double) is the array containing connection lengths between the center of cell *n* and the shared face with each adjacent *m* cell.
- **hwva** ([double]) -
 - **hwva** (double) is a symmetric array of size **NJA**. For horizontal connections, entries in **HWVA** are the horizontal width perpendicular to flow. For vertical connections, entries in **HWVA** are the vertical area for flow. Thus, values in the **HWVA** array contain dimensions of both length and area. Entries in the **HWVA** array have a one-to-one correspondence with the connections specified in the **JA** array. Likewise, there is a one-to-one correspondence between entries in the **HWVA** array and entries in the **IHC** array, which specifies the connection type (horizontal or vertical). Entries in the **HWVA** array must be symmetric; the program will terminate with an error if the value for **HWVA** for an *n* to *m* connection does not equal the value for **HWVA** for the corresponding *n* to *m* connection.
- **angldegx** ([double]) -
 - **angldegx** (double) is the angle (in degrees) between the horizontal x-axis and the outward normal to the face between a cell and its connecting cells. The angle varies between zero and 360.0 degrees, where zero degrees points in the positive x-axis direction, and 90 degrees points in the positive y-axis direction. **ANGLDEGX** is only needed if horizontal anisotropy is specified in the NPF Package, if the **XT3D** option is used in the NPF Package, or if the **SAVE_SPECIFIC_DISCHARGE** option is specified in the NPF Package. **ANGLDEGX** does not need to be specified if these conditions are not met. **ANGLDEGX** is of size **NJA**; values specified for vertical connections and for

the diagonal position are not used. Note that ANGLDEGX is read in degrees, which is different from MODFLOW-USG, which reads a similar variable (ANGLEX) in radians.

- **vertices** (*[iv, xv, yv]*) -
 - *iv* (integer) is the vertex number. Records in the VERTICES block must be listed in consecutive order from 1 to NVERT. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *xv* (double) is the x-coordinate for the vertex.
 - *yv* (double) is the y-coordinate for the vertex.
- **cell2d** (*[icell2d, xc, yc, nvert, icvert]*) -
 - *icell2d* (integer) is the cell2d number. Records in the CELL2D block must be listed in consecutive order from 1 to NODES. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *xc* (double) is the x-coordinate for the cell center.
 - *yc* (double) is the y-coordinate for the cell center.
 - *nvert* (integer) is the number of vertices required to define the cell. There may be a different number of vertices for each cell.
 - *icvert* (integer) is an array of integer values containing vertex numbers (in the VERTICES block) used to define the cell. Vertices must be listed in clockwise order. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

`angldegx = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>`

`area = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>`

`bot = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>`

`cell2d = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

`cl12 = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>`

```
dfn = [['header'], ['block options', 'name length_units', 'type string', 'reader
urword', 'optional true'], ['block options', 'name nogrb', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name xorigin', 'type double
precision', 'reader urword', 'optional true'], ['block options', 'name yorigin',
'type double precision', 'reader urword', 'optional true'], ['block options', 'name
angrot', 'type double precision', 'reader urword', 'optional true'], ['block
options', 'name vertical_offset_tolerance', 'type double precision', 'reader
urword', 'optional true', 'default_value 0.0', 'mf6internal voffsettol'], ['block
dimensions', 'name nodes', 'type integer', 'reader urword', 'optional false'],
['block dimensions', 'name nja', 'type integer', 'reader urword', 'optional false'],
['block dimensions', 'name nvert', 'type integer', 'reader urword', 'optional
true'], ['block griddata', 'name top', 'type double precision', 'shape (nodes)',
'reader readarray'], ['block griddata', 'name bot', 'type double precision', 'shape
(nodes)', 'reader readarray'], ['block griddata', 'name area', 'type double
precision', 'shape (nodes)', 'reader readarray'], ['block griddata', 'name idomain',
'type integer', 'shape (nodes)', 'reader readarray', 'layered false', 'optional
true'], ['block connectiondata', 'name iac', 'type integer', 'shape (nodes)',
'reader readarray'], ['block connectiondata', 'name ja', 'type integer', 'shape
(nja)', 'reader readarray', 'numeric_index true', 'jagged_array iac'], ['block
connectiondata', 'name ihc', 'type integer', 'shape (nja)', 'reader readarray',
'jagged_array iac'], ['block connectiondata', 'name cl12', 'type double precision',
'shape (nja)', 'reader readarray', 'jagged_array iac'], ['block connectiondata',
'name hwva', 'type double precision', 'shape (nja)', 'reader readarray',
'jagged_array iac'], ['block connectiondata', 'name angldegx', 'type double
precision', 'optional true', 'shape (nja)', 'reader readarray', 'jagged_array iac'],
['block vertices', 'name vertices', 'type recarray iv xv yv', 'shape (nvert)',
'reader urword', 'optional false'], ['block vertices', 'name iv', 'type integer',
'in_record true', 'tagged false', 'reader urword', 'optional false', 'numeric_index
true'], ['block vertices', 'name xv', 'type double precision', 'in_record true',
'tagged false', 'reader urword', 'optional false'], ['block vertices', 'name yv',
'type double precision', 'in_record true', 'tagged false', 'reader urword',
'optional false'], ['block cell2d', 'name cell2d', 'type recarray icell2d xc yc
ncvert icvert', 'shape (nodes)', 'reader urword', 'optional false'], ['block
cell2d', 'name icell2d', 'type integer', 'in_record true', 'tagged false', 'reader
urword', 'optional false', 'numeric_index true'], ['block cell2d', 'name xc', 'type
double precision', 'in_record true', 'tagged false', 'reader urword', 'optional
false'], ['block cell2d', 'name yc', 'type double precision', 'in_record true',
'tagged false', 'reader urword', 'optional false'], ['block cell2d', 'name ncvert',
'type integer', 'in_record true', 'tagged false', 'reader urword', 'optional
false'], ['block cell2d', 'name icvert', 'type integer', 'shape (ncvert)',
'in_record true', 'tagged false', 'reader urword', 'optional false', 'numeric_index
true']]
```

```
dfn_file_name = 'gwf-disu.dfn'
```

```
hwva = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
iac = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
idomain = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
ihc = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
ja = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```

package_abbr = 'gwfdisu'

top = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

vertices = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

`flopy.mf6.modflow.mfgwfdiscv` module

```

class ModflowGwfdiscv(model, loading_package=False, length_units=None, nogrb=None,
                      xorigin=None, yorigin=None, angrot=None, nlay=None, ncpl=None,
                      nvert=None, top=None, botm=None, idomain=None, vertices=None,
                      cell1d=None, filename=None, pname=None, **kwargs)

```

Bases: `MFPackage`

ModflowGwfdiscv defines a discv package within a gwf6 model.

Parameters

- **model** (`MFModel`) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (`bool`) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **length_units** (`string`) -
 - `length_units` (`string`) is the length units used for this model. Values can be "FEET", "METERS", or "CENTIMETERS". If not specified, the default is "UNKNOWN".
- **nogrb** (`boolean`) -
 - `nogrb` (`boolean`) keyword to deactivate writing of the binary grid file.
- **xorigin** (`double`) -
 - `xorigin` (`double`) x-position of the origin used for model grid vertices. This value should be provided in a real-world coordinate system. A default value of zero is assigned if not specified. The value for XORIGIN does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **yorigin** (`double`) -
 - `yorigin` (`double`) y-position of the origin used for model grid vertices. This value should be provided in a real-world coordinate system. If not specified, then a default value equal to zero is used. The value for YORIGIN does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **angrot** (`double`) -
 - `angrot` (`double`) counter-clockwise rotation angle (in degrees) of the model grid coordinate system relative to a real-world coordinate system. If not specified, then a default value of 0.0 is assigned. The value for ANGROT does not affect the model

simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.

- **nlay** (*integer*) -
 - nlay (*integer*) is the number of layers in the model grid.
- **ncpl** (*integer*) -
 - ncpl (*integer*) is the number of cells per layer. This is a constant value for the grid and it applies to all layers.
- **nvert** (*integer*) -
 - nvert (*integer*) is the total number of (x, y) vertex pairs used to characterize the horizontal configuration of the model grid.
- **top** (*[double]*) -
 - top (*double*) is the top elevation for each cell in the top model layer.
- **botm** (*[double]*) -
 - botm (*double*) is the bottom elevation for each cell.
- **idomain** (*[integer]*) -
 - idomain (*integer*) is an optional array that characterizes the existence status of a cell. If the IDOMAIN array is not specified, then all model cells exist within the solution. If the IDOMAIN value for a cell is 0, the cell does not exist in the simulation. Input and output values will be read and written for the cell, but internal to the program, the cell is excluded from the solution. If the IDOMAIN value for a cell is 1 or greater, the cell exists in the simulation. If the IDOMAIN value for a cell is -1, the cell does not exist in the simulation. Furthermore, the first existing cell above will be connected to the first existing cell below. This type of cell is referred to as a “vertical pass through” cell.
- **vertices** (*[iv, xv, yv]*) -
 - iv (*integer*) is the vertex number. Records in the VERTICES block must be listed in consecutive order from 1 to NVERT. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - xv (*double*) is the x-coordinate for the vertex.
 - yv (*double*) is the y-coordinate for the vertex.
- **cell2d** (*[icell2d, xc, yc, nvert, icvert]*) -
 - icell2d (*integer*) is the CELL2D number. Records in the CELL2D block must be listed in consecutive order from the first to the last. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.

- xc (double) is the x-coordinate for the cell center.
- yc (double) is the y-coordinate for the cell center.
- ncvrt (integer) is the number of vertices required to define the cell. There may be a different number of vertices for each cell.
- icvrt (integer) is an array of integer values containing vertex numbers (in the VERTICES block) used to define the cell. Vertices must be listed in clockwise order. Cells that are connected must share vertices. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mftullaktab package must have a mfgwflak package parent_file.

```
botm = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
cell2d = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
dfn = [['header'], ['block options', 'name length_units', 'type string', 'reader
urword', 'optional true'], ['block options', 'name nogrb', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name xorigin', 'type double
precision', 'reader urword', 'optional true'], ['block options', 'name yorigin',
'type double precision', 'reader urword', 'optional true'], ['block options', 'name
angrot', 'type double precision', 'reader urword', 'optional true'], ['block
dimensions', 'name nlay', 'type integer', 'reader urword', 'optional false'],
['block dimensions', 'name ncpl', 'type integer', 'reader urword', 'optional
false'], ['block dimensions', 'name nvert', 'type integer', 'reader urword',
'optional false'], ['block griddata', 'name top', 'type double precision', 'shape
(ncpl)', 'reader readarray'], ['block griddata', 'name botm', 'type double
precision', 'shape (ncpl, nlay)', 'reader readarray', 'layered true'], ['block
griddata', 'name idomain', 'type integer', 'shape (ncpl, nlay)', 'reader readarray',
'layered true', 'optional true'], ['block vertices', 'name vertices', 'type recarray
iv xv yv', 'shape (nvert)', 'reader urword', 'optional false'], ['block vertices',
'name iv', 'type integer', 'in_record true', 'tagged false', 'reader urword',
'optional false', 'numeric_index true'], ['block vertices', 'name xv', 'type double
precision', 'in_record true', 'tagged false', 'reader urword', 'optional false'],
['block vertices', 'name yv', 'type double precision', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block cell2d', 'name cell2d', 'type
recarray icell2d xc yc ncvrt icvrt', 'shape (ncpl)', 'reader urword', 'optional
false'], ['block cell2d', 'name icell2d', 'type integer', 'in_record true', 'tagged
false', 'reader urword', 'optional false', 'numeric_index true'], ['block cell2d',
'name xc', 'type double precision', 'in_record true', 'tagged false', 'reader
urword', 'optional false'], ['block cell2d', 'name yc', 'type double precision',
'in_record true', 'tagged false', 'reader urword', 'optional false'], ['block
cell2d', 'name ncvrt', 'type integer', 'in_record true', 'tagged false', 'reader
urword', 'optional false'], ['block cell2d', 'name icvrt', 'type integer', 'shape
(ncvrt)', 'in_record true', 'tagged false', 'reader urword', 'optional false',
'numeric_index true']]
```

```
dfn_file_name = 'gwf-disv.dfn'

idomain = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

package_abbr = 'gwfdisv'

top = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

vertices = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfdrn module

```
class ModflowGwfdrn(model, loading_package=False, auxiliary=None, auxmultname=None,
                    auxdepthname=None, boundnames=None, print_input=None,
                    print_flows=None, save_flows=None, timeseries=None,
                    observations=None, mover=None, maxbound=None,
                    stress_period_data=None, filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwfdrn defines a drn package within a gwf6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **auxiliary** (*[string]*) -
 - auxiliary (*string*) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** (*string*) -
 - auxmultname (*string*) name of auxiliary variable to be used as multiplier of drain conductance.
- **auxdepthname** (*string*) -
 - auxdepthname (*string*) name of a variable listed in AUXILIARY that defines the depth at which drainage discharge will be scaled. If a positive value is specified for the AUXDEPTHNAME AUXILIARY variable, then ELEV is the elevation at which the drain starts to discharge and ELEV + DDRN (assuming DDRN is the AUXDEPTHNAME variable) is the elevation when the drain

conductance (COND) scaling factor is 1. If a negative drainage depth value is specified for DDRN, then ELEV + DDRN is the elevation at which the drain starts to discharge and ELEV is the elevation when the conductance (COND) scaling factor is 1. A linear- or cubic-scaling is used to scale the drain conductance (COND) when the Standard or Newton-Raphson Formulation is used, respectively. This discharge scaling option is described in more detail in Chapter 3 of the Supplemental Technical Information.

- **boundnames** (*boolean*) -
 - boundnames (boolean) keyword to indicate that boundary names may be provided with the list of drain cells.
- **print_input** (*boolean*) -
 - print_input (boolean) keyword to indicate that the list of drain information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (boolean) keyword to indicate that the list of drain flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (boolean) keyword to indicate that drain flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **timeseries** (*{varname:data} or timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data} or continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **mover** (*boolean*) -
 - mover (boolean) keyword to indicate that this instance of the Drain Package can be used with the Water Mover (MVR) Package. When the MOVER option is specified, additional memory is allocated within the package to store the available, provided, and received water.
- **maxbound** (*integer*) -

- maxbound (integer) integer value specifying the maximum number of drains cells that will be specified for use during any stress period.
- **stress_period_data** ([*cellid*, *elev*, *cond*, *aux*, *boundname*]) -
 - *cellid* ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *elev* (double) is the elevation of the drain. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - *cond* (double) is the hydraulic conductance of the interface between the aquifer and the drain. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - *aux* (double) represents the values of the auxiliary variables for each drain. The values of auxiliary variables must be present for each drain. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - *boundname* (string) name of the drain cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>


```

dfn = [['header', 'multi-package'], ['block options', 'name auxiliary', 'type
string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name
auxmultname', 'type string', 'shape', 'reader urword', 'optional true'], ['block
options', 'name auxdepthname', 'type string', 'shape', 'reader urword', 'optional
true'], ['block options', 'name boundnames', 'type keyword', 'shape', 'reader
urword', 'optional true'], ['block options', 'name print_input', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name print_flows', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name save_flows',
'type keyword', 'reader urword', 'optional true'], ['block options', 'name
ts_filerecord', 'type record ts6 filein ts6_filename', 'shape', 'reader urword',
'tagged true', 'optional true', 'construct_package ts', 'construct_data timeseries',
'parameter_name timeseries'], ['block options', 'name ts6', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader urword',
'tagged true', 'optional false'], ['block options', 'name ts6_filename', 'type
string', 'preserve_case true', 'in_record true', 'reader urword', 'optional false',
'tagged false'], ['block options', 'name obs_filerecord', 'type record obs6 filein
obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package obs', 'construct_data continuous', 'parameter_name
observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block options', 'name mover', 'type
keyword', 'tagged true', 'reader urword', 'optional true'], ['block dimensions',
'name maxbound', 'type integer', 'reader urword', 'optional false'], ['block
period', 'name iper', 'type integer', 'block_variable True', 'in_record true',
'tagged false', 'shape', 'valid', 'reader urword', 'optional false'], ['block
period', 'name stress_period_data', 'type recarray cellid elev cond aux boundname',
'shape (maxbound)', 'reader urword'], ['block period', 'name cellid', 'type
integer', 'shape (ncelldim)', 'tagged false', 'in_record true', 'reader urword'],
['block period', 'name elev', 'type double precision', 'shape', 'tagged false',
'in_record true', 'reader urword', 'time_series true'], ['block period', 'name
cond', 'type double precision', 'shape', 'tagged false', 'in_record true', 'reader
urword', 'time_series true'], ['block period', 'name aux', 'type double precision',
'in_record true', 'tagged false', 'shape (naux)', 'reader urword', 'optional true',
'time_series true'], ['block period', 'name boundname', 'type string', 'shape',
'tagged false', 'in_record true', 'reader urword', 'optional true']]

```

```
dfn_file_name = 'gwf-drn.dfn'
```

```
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'gwfdn'
```

```
stress_period_data = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfevt module

```
class ModflowGwfevt(model, loading_package=False, fixed_cell=None, auxiliary=None,
                    auxmultname=None, boundnames=None, print_input=None,
                    print_flows=None, save_flows=None, timeseries=None,
                    observations=None, surf_rate_specified=None, maxbound=None,
                    nseg=None, stress_period_data=None, filename=None, pname=None,
                    **kwargs)
```

Bases: [`MFPackage`](#)

ModflowGwfevt defines a evt package within a gwf6 model.

Parameters

- **model** ([`MFModel`](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** ([`bool`](#)) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **fixed_cell** ([`boolean`](#)) -
 - **fixed_cell** ([`boolean`](#)) indicates that evapotranspiration will not be reassigned to a cell underlying the cell specified in the list if the specified cell is inactive.
- **auxiliary** ([`\[string\]`](#)) -
 - **auxiliary** ([`string`](#)) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for `naux`. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** ([`string`](#)) -
 - **auxmultname** ([`string`](#)) name of auxiliary variable to be used as multiplier of evapotranspiration rate.
- **boundnames** ([`boolean`](#)) -
 - **boundnames** ([`boolean`](#)) keyword to indicate that boundary names may be provided with the list of evapotranspiration cells.
- **print_input** ([`boolean`](#)) -
 - **print_input** ([`boolean`](#)) keyword to indicate that the list of evapotranspiration information will be written to the listing file immediately after it is read.
- **print_flows** ([`boolean`](#)) -
 - **print_flows** ([`boolean`](#)) keyword to indicate that the list of evapotranspiration flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT”

is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.

- **save_flows** (*boolean*) -
 - save_flows (*boolean*) keyword to indicate that evapotranspiration flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **timeseries** (*{varname:data} or timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data} or continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **surf_rate_specified** (*boolean*) -
 - surf_rate_specified (*boolean*) indicates that the proportion of the evapotranspiration rate at the ET surface will be specified as PETM0 in list input.
- **maxbound** (*integer*) -
 - maxbound (*integer*) integer value specifying the maximum number of evapotranspiration cells that will be specified for use during any stress period.
- **nseg** (*integer*) -
 - nseg (*integer*) number of ET segments. Default is one. When NSEG is greater than 1, the PXDP and PETM arrays must be of size NSEG - 1 and be listed in order from the uppermost segment down. Values for PXDP must be listed first followed by the values for PETM. PXDP defines the extinction-depth proportion at the bottom of a segment. PETM defines the proportion of the maximum ET flux rate at the bottom of a segment.
- **stress_period_data** (*[cellid, surface, rate, depth, pxdp, petm, petm0, aux,)* -
boundname]
 - cellid ((*integer, ...*)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based

when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.

- surface (double) is the elevation of the ET surface (L). If the Options block includes a TIMESERIESFILE entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- rate (double) is the maximum ET flux rate (LT^{-1}). If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- depth (double) is the ET extinction depth (L). If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- pxdp (double) is the proportion of the ET extinction depth at the bottom of a segment (dimensionless). pxdp is an array of size (nseg - 1). Values in pxdp must be greater than 0.0 and less than 1.0. pxdp values for a cell must increase monotonically. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- petm (double) is the proportion of the maximum ET flux rate at the bottom of a segment (dimensionless). petm is an array of size (nseg - 1). If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- petm0 (double) is the proportion of the maximum ET flux rate that will apply when head is at or above the ET surface (dimensionless). PETM0 is read only when the SURF_RATE_SPECIFIED option is used. If the Options block includes a TIMESERIESFILE entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- aux (double) represents the values of the auxiliary variables for each evapotranspiration. The values of auxiliary variables must be present for each evapotranspiration. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- boundname (string) name of the evapotranspiration cell. BOUNDNAME is an ASCII character variable that can contain

as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfuillaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```
dfn = [['header', 'multi-package'], ['block options', 'name fixed_cell', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name auxmultname', 'type string', 'shape', 'reader urword', 'optional true'], ['block options', 'name boundnames', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block options', 'name print_input', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name print_flows', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name save_flows', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name ts_filerecord', 'type record ts6 filein ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package ts', 'construct_data timeseries', 'parameter_name timeseries'], ['block options', 'name ts6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name ts6_filename', 'type string', 'preserve_case true', 'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block options', 'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package obs', 'construct_data continuous', 'parameter_name observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged false', 'reader urword', 'optional false'], ['block options', 'name surf_rate_specified', 'type keyword', 'reader urword', 'optional true'], ['block dimensions', 'name maxbound', 'type integer', 'reader urword', 'optional false'], ['block dimensions', 'name nseg', 'type integer', 'reader urword', 'optional false'], ['block period', 'name iper', 'type integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader urword', 'optional false'], ['block period', 'name stress_period_data', 'type recarray cellid surface rate depth pxdp petm petm0 aux boundname', 'shape (maxbound)', 'reader urword'], ['block period', 'name cellid', 'type integer', 'shape (ncellid)', 'tagged false', 'in_record true', 'reader urword'], ['block period', 'name surface', 'type double precision', 'shape', 'tagged false', 'in_record true', 'reader urword', 'time_series true'], ['block period', 'name rate', 'type double precision', 'shape', 'tagged false', 'in_record true', 'reader urword', 'time_series true'], ['block period', 'name depth', 'type double precision', 'shape', 'tagged false', 'in_record true', 'reader urword', 'time_series true'], ['block period', 'name pxdp', 'type double precision', 'shape (nseg-1)', 'tagged false', 'in_record true', 'reader urword', 'time_series true'], ['block period', 'name petm', 'type double precision', 'shape (nseg-1)', 'tagged false', 'in_record true', 'reader urword', 'time_series true'], ['block period', 'name petm0', 'type double precision', 'shape', 'tagged false', 'in_record true', 'reader urword', 'optional true', 'time_series true'], ['block period', 'name aux', 'type double precision', 'in_record true', 'tagged false', 'shape (naux)', 'reader urword', 'optional true', 'time_series true'], ['block period', 'name boundname', 'type string', 'shape', 'tagged false', 'in_record true', 'reader urword', 'optional true']]

dfn_file_name = 'gwf-evt.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfEvt'

stress_period_data = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

`ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

flopy.mf6.modflow.mfgwfevta module

```
class ModflowGwfevta(model, loading_package=False, readasarrays=True, fixed_cell=None,
    auxiliary=None, auxmultname=None, print_input=None,
    print_flows=None, save_flows=None, timearrayseries=None,
    observations=None, ievt=None, surface=0.0, rate=0.001, depth=1.0,
    aux=None, filename=None, pname=None, **kwargs)
```

Bases: [`MFPackage`](#)

ModflowGwfevta defines a evta package within a gw6 model.

Parameters

- **model** ([`MFModel`](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** ([`bool`](#)) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **readasarrays** ([`boolean`](#)) -
 - readasarrays ([`boolean`](#)) indicates that array-based input will be used for the Evapotranspiration Package. This keyword must be specified to use array-based input.
- **fixed_cell** ([`boolean`](#)) -
 - fixed_cell ([`boolean`](#)) indicates that evapotranspiration will not be reassigned to a cell underlying the cell specified in the list if the specified cell is inactive.
- **auxiliary** ([`\[string\]`](#)) -
 - auxiliary ([`string`](#)) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** ([`string`](#)) -
 - auxmultname ([`string`](#)) name of auxiliary variable to be used as multiplier of evapotranspiration rate.
- **print_input** ([`boolean`](#)) -
 - print_input ([`boolean`](#)) keyword to indicate that the list of evapotranspiration information will be written to the listing file immediately after it is read.
- **print_flows** ([`boolean`](#)) -

- `print_flows` (boolean) keyword to indicate that the list of evapotranspiration flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **`save_flows`** (*boolean*) -
 - `save_flows` (boolean) keyword to indicate that evapotranspiration flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **`timearrayseries`** (*{varname:data} or tas_array data*) -
 - Contains data for the tas package. Data can be stored in a dictionary containing data for the tas package with variable names as keys and package data as values. Data just for the `timearrayseries` variable is also acceptable. See tas package documentation for more information.
- **`observations`** (*{varname:data} or continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the `observations` variable is also acceptable. See obs package documentation for more information.
- **`ievt`** (*[integer]*) -
 - `ievt` (integer) IEVT is the layer number that defines the layer in each vertical column where evapotranspiration is applied. If IEVT is omitted, evapotranspiration by default is applied to cells in layer 1. If IEVT is specified, it must be specified as the first variable in the PERIOD block or MODFLOW will terminate with an error. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.
- **`surface`** (*[double]*) -
 - `surface` (double) is the elevation of the ET surface (L).
- **`rate`** (*[double]*) -
 - `rate` (double) is the maximum ET flux rate (LT^{-1}).
- **`depth`** (*[double]*) -
 - `depth` (double) is the ET extinction depth (L).
- **`aux(iaux)`** (*[double]*) -
 - `aux(iaux)` (double) is an array of values for auxiliary variable AUX(IAUX), where `iaux` is a value from 1 to NAUX, and AUX(IAUX) must be listed as part of the auxiliary variables. A separate array can be specified for each auxiliary variable. If an array is not specified for an auxiliary variable, then a value of zero is assigned. If the value specified here for the auxiliary

variable is the same as auxmultname, then the evapotranspiration rate will be multiplied by this array.

- `filename` (*String*) - File name for this package.
- `pname` (*String*) - Package name for this package.
- `parent_file` (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutil*). For example, mftullaktab package must have a mfgwflak package `parent_file`.

```
aux = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
depth = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
dfn = [['header', 'multi-package'], ['block options', 'name readasarrays', 'type keyword', 'shape', 'reader urword', 'optional false', 'default_value True'], ['block options', 'name fixed_cell', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name auxmultname', 'type string', 'shape', 'reader urword', 'optional true'], ['block options', 'name print_input', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name print_flows', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name save_flows', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name tas_filerecord', 'type record tas6 filein tas6_filename', 'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package tas', 'construct_data tas_array', 'parameter_name timearrayseries'], ['block options', 'name tas6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name tas6_filename', 'type string', 'preserve_case true', 'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block options', 'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package obs', 'construct_data continuous', 'parameter_name observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged false', 'reader urword', 'optional false'], ['block period', 'name iper', 'type integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader urword', 'optional false'], ['block period', 'name ievt', 'type integer', 'shape (ncol*nrow; ncpl)', 'reader readarray', 'numeric_index true', 'optional true'], ['block period', 'name surface', 'type double precision', 'shape (ncol*nrow; ncpl)', 'reader readarray', 'default_value 0.'], ['block period', 'name rate', 'type double precision', 'shape (ncol*nrow; ncpl)', 'reader readarray', 'default_value 1.e-3'], ['block period', 'name depth', 'type double precision', 'shape (ncol*nrow; ncpl)', 'reader readarray', 'default_value 1.0'], ['block period', 'name aux(iaux)', 'type double precision', 'shape (ncol*nrow; ncpl)', 'reader readarray']]
```

```
dfn_file_name = 'gwf-evta.dfn'
```

```
ievt = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbrev = 'gwfevta'

rate = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

surface = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

tas_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfgghb module

```
class ModflowGwfgghb(model, loading_package=False, auxiliary=None, auxmultname=None,
                      boundnames=None, print_input=None, print_flows=None, save_flows=None,
                      timeseries=None, observations=None, mover=None, maxbound=None,
                      stress_period_data=None, filename=None, pname=None, **kwargs)
```

Bases: [*MFPackage*](#)

ModflowGwfgghb defines a ghb package within a gwf6 model.

Parameters

- **model** (*MFModel*) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **auxiliary** (*[string]*) -
 - auxiliary (*string*) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** (*string*) -
 - auxmultname (*string*) name of auxiliary variable to be used as multiplier of general-head boundary conductance.
- **boundnames** (*boolean*) -
 - boundnames (*boolean*) keyword to indicate that boundary names may be provided with the list of general-head boundary cells.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of general- head boundary information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -

- `print_flows` (boolean) keyword to indicate that the list of general-head boundary flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **`save_flows`** (boolean) -
 - `save_flows` (boolean) keyword to indicate that general-head boundary flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **`timeseries`** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **`observations`** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **`mover`** (boolean) -
 - `mover` (boolean) keyword to indicate that this instance of the General-Head Boundary Package can be used with the Water Mover (MVR) Package. When the MOVER option is specified, additional memory is allocated within the package to store the available, provided, and received water.
- **`maxbound`** (integer) -
 - `maxbound` (integer) integer value specifying the maximum number of general-head boundary cells that will be specified for use during any stress period.
- **`stress_period_data`** (*[cellid, bhead, cond, aux, boundname]*) -
 - `cellid` ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - `bhead` (double) is the boundary head. If the Options block includes a TIMESERIESFILE entry (see the "Time-Variable Input"

section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- `cond` (double) is the hydraulic conductance of the interface between the aquifer cell and the boundary. If the Options block includes a `TIMESERIESFILE` entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- `aux` (double) represents the values of the auxiliary variables for each general-head boundary. The values of auxiliary variables must be present for each general-head boundary. The values must be specified in the order of the auxiliary variables specified in the `OPTIONS` block. If the package supports time series and the Options block includes a `TIMESERIESFILE` entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- `boundname` (string) name of the general-head boundary cell. `BOUNDNAME` is an ASCII character variable that can contain as many as 40 characters. If `BOUNDNAME` contains spaces in it, then the entire name must be enclosed within single quotes.
- `filename` (*String*) - File name for this package.
- `pname` (*String*) - Package name for this package.
- `parent_file` (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (`mfutl*`). For example, `mfutllaktab` package must have a `mfgwflak` package `parent_file`.

`auxiliary` = `<flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

```

dfn = [['header', 'multi-package'], ['block options', 'name auxiliary', 'type
string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name
auxmultname', 'type string', 'shape', 'reader urword', 'optional true'], ['block
options', 'name boundnames', 'type keyword', 'shape', 'reader urword', 'optional
true'], ['block options', 'name print_input', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name print_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name save_flows', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name ts_filerecord', 'type
record ts6 filein ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional
true', 'construct_package ts', 'construct_data timeseries', 'parameter_name
timeseries'], ['block options', 'name ts6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
filein', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name ts6_filename', 'type string',
'preserve_case true', 'in_record true', 'reader urword', 'optional false', 'tagged
false'], ['block options', 'name obs_filerecord', 'type record obs6 filein
obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package obs', 'construct_data continuous', 'parameter_name
observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block options', 'name mover', 'type
keyword', 'tagged true', 'reader urword', 'optional true'], ['block dimensions',
'name maxbound', 'type integer', 'reader urword', 'optional false'], ['block
period', 'name iper', 'type integer', 'block_variable True', 'in_record true',
'tagged false', 'shape', 'valid', 'reader urword', 'optional false'], ['block
period', 'name stress_period_data', 'type recarray cellid bhead cond aux boundname',
'shape (maxbound)', 'reader urword'], ['block period', 'name cellid', 'type
integer', 'shape (ncelldim)', 'tagged false', 'in_record true', 'reader urword'],
['block period', 'name bhead', 'type double precision', 'shape', 'tagged false',
'in_record true', 'reader urword', 'time_series true'], ['block period', 'name
cond', 'type double precision', 'shape', 'tagged false', 'in_record true', 'reader
urword', 'time_series true'], ['block period', 'name aux', 'type double precision',
'in_record true', 'tagged false', 'shape (naux)', 'reader urword', 'optional true',
'time_series true'], ['block period', 'name boundname', 'type string', 'shape',
'tagged false', 'in_record true', 'reader urword', 'optional true']]

```

```
dfn_file_name = 'gwf-ghb.dfn'
```

```
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'gwfghb'
```

```
stress_period_data = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfgnc module

```
class GwfgncPackages(model_or_sim, parent, pkg_type, filerecord, package=None,
                    package_class=None)
```

Bases: [*MFChildPackages*](#)

GwfgncPackages is a container class for the ModflowGwfgnc class.

initialize()

Initializes a new ModflowGwfgnc package removing any sibling child packages attached to the same parent package. See ModflowGwfgnc init documentation for definition of parameters.

append_package()

Adds a new ModflowGwfgnc package to the container. See ModflowGwfgnc init documentation for definition of parameters.

```
append_package(print_input=None, print_flows=None, explicit=None, numgnc=None,
               numalphaj=None, gncdata=None, filename=None, pname=None)
```

```
initialize(print_input=None, print_flows=None, explicit=None, numgnc=None,
           numalphaj=None, gncdata=None, filename=None, pname=None)
```

```
package_abbr = 'gwfgncpackages'
```

```
class ModflowGwfgnc(model, loading_package=False, print_input=None, print_flows=None,
                   explicit=None, numgnc=None, numalphaj=None, gncdata=None,
                   filename=None, pname=None, **kwargs)
```

Bases: [*MFPackage*](#)

ModflowGwfgnc defines a gnc package within a gwf6 model.

Parameters

- **model** ([*MFModel*](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of GNC information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (*boolean*) keyword to indicate that the list of GNC flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **explicit** (*boolean*) -
 - explicit (*boolean*) keyword to indicate that the ghost node correction is applied in an explicit manner on the right-hand side of the matrix. The explicit approach will likely require

additional outer iterations. If the keyword is not specified, then the correction will be applied in an implicit manner on the left-hand side. The implicit approach will likely converge better, but may require additional memory. If the EXPLICIT keyword is not specified, then the BICGSTAB linear acceleration option should be specified within the LINEAR block of the Sparse Matrix Solver.

- **numgnc** (*integer*) -
 - numgnc (*integer*) is the number of GNC entries.
- **numalphaj** (*integer*) -
 - numalphaj (*integer*) is the number of contributing factors.
- **gncdata** ([*cellidn*, *cellidm*, *cellidsj*, *alphasj*]) -
 - *cellidn* ((*integer*, ...)) is the cellid of the cell, *n*, in which the ghost node is located. For a structured grid that uses the DIS input file, CELLIDN is the layer, row, and column numbers of the cell. For a grid that uses the DISV input file, CELLIDN is the layer number and CELL2D number for the two cells. If the model uses the unstructured discretization (DISU) input file, then CELLIDN is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *cellidm* ((*integer*, ...)) is the cellid of the connecting cell, *m*, to which flow occurs from the ghost node. For a structured grid that uses the DIS input file, CELLIDM is the layer, row, and column numbers of the cell. For a grid that uses the DISV input file, CELLIDM is the layer number and CELL2D number for the two cells. If the model uses the unstructured discretization (DISU) input file, then CELLIDM is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *cellidsj* ((*integer*, ...)) is the array of CELLIDS for the contributing *j* cells, which contribute to the interpolated head value at the ghost node. This item contains one CELLID for each of the contributing cells of the ghost node. Note that if the number of actual contributing cells needed by the user is less than NUMALPHAJ for any ghost node, then a dummy CELLID of zero(s) should be inserted with an associated contributing factor of zero. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column numbers of the cell. For a grid that uses the DISV input file, CELLID is the layer number and cell2d number for the two cells. If the model uses the unstructured discretization (DISU) input file, then CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically

subtract one when loading index variables and add one when writing index variables.

- `alphasj` (double) is the contributing factors for each contributing node in `CELLIDSJ`. Note that if the number of actual contributing cells is less than `NUMALPHAJ` for any ghost node, then dummy `CELLIDS` should be inserted with an associated contributing factor of zero. The sum of `ALPHASJ` should be less than one. This is because one minus the sum of `ALPHASJ` is equal to the alpha term (alpha n in equation 4-61 of the GWF Model report) that is multiplied by the head in cell n .
- `filename` (*String*) - File name for this package.
- `pname` (*String*) - Package name for this package.
- `parent_file` (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfullaktab package must have a mfgwflak package `parent_file`.

```
dfn = [['header'], ['block options', 'name print_input', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name print_flows', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name explicit', 'type
keyword', 'tagged true', 'reader urword', 'optional true'], ['block dimensions',
'name numgnc', 'type integer', 'reader urword', 'optional false'], ['block
dimensions', 'name numalphaj', 'type integer', 'reader urword', 'optional false'],
['block gncdata', 'name gncdata', 'type recarray cellidn cellidm cellidsj alphasj',
'shape (maxbound)', 'reader urword'], ['block gncdata', 'name cellidn', 'type
integer', 'shape', 'tagged false', 'in_record true', 'reader urword', 'numeric_index
true'], ['block gncdata', 'name cellidm', 'type integer', 'shape', 'tagged false',
'in_record true', 'reader urword', 'numeric_index true'], ['block gncdata', 'name
cellidsj', 'type integer', 'shape (numalphaj)', 'tagged false', 'in_record true',
'reader urword', 'numeric_index true'], ['block gncdata', 'name alphasj', 'type
double precision', 'shape (numalphaj)', 'tagged false', 'in_record true', 'reader
urword']]
```

```
dfn_file_name = 'gwf-gnc.dfn'
```

```
gncdata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'gwf gnc'
```

flopy.mf6.modflow.mfgwfgwf module

```
class ModflowGwfgwf(simulation, loading_package=False, exgtype='GWF6-GWF6',
                    exgnamea=None, exgnameb=None, auxiliary=None, boundnames=None,
                    print_input=None, print_flows=None, save_flows=None,
                    cell_averaging=None, cvoptions=None, newton=None, xt3d=None,
                    gncdata=None, perioddata=None, observations=None,
                    dev_interfacemodel_on=None, nexg=None, exchangedata=None,
                    filename=None, pname=None, **kwargs)
```

Bases: *MFPackage*

ModflowGwfgwf defines a gwfgwf package.

Parameters

- **simulation** ([MFSimulation](#)) - Simulation that this package is a part of. Package is automatically added to simulation when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **exgtype** (<string>) -
 - is the exchange type (GWF-GWF or GWF-GWT).
- **exgmnamea** (<string>) -
 - is the name of the first model that is part of this exchange.
- **exgmnameb** (<string>) -
 - is the name of the second model that is part of this exchange.
- **auxiliary** ([string]) -
 - auxiliary (string) an array of auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided. Most auxiliary variables will not be used by the GWF-GWF Exchange, but they will be available for use by other parts of the program. If an auxiliary variable with the name "ANGLDEGX" is found, then this information will be used as the angle (provided in degrees) between the connection face normal and the x axis, where a value of zero indicates that a normal vector points directly along the positive x axis. The connection face normal is a normal vector on the cell face shared between the cell in model 1 and the cell in model 2 pointing away from the model 1 cell. Additional information on "ANGLDEGX" is provided in the description of the DISU Package. If an auxiliary variable with the name "CDIST" is found, then this information will be used as the straight-line connection distance, including the vertical component, between the two cell centers. Both ANGLDEGX and CDIST are required if specific discharge is calculated for either of the groundwater models.
- **boundnames** (*boolean*) -
 - boundnames (boolean) keyword to indicate that boundary names may be provided with the list of GWF Exchange cells.
- **print_input** (*boolean*) -
 - print_input (boolean) keyword to indicate that the list of exchange entries will be echoed to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (boolean) keyword to indicate that the list of exchange flow rates will be printed to the listing file for every stress period in which "SAVE BUDGET" is specified in Output Control.
- **save_flows** (*boolean*) -
 - save_flows (boolean) keyword to indicate that cell-by-cell flow terms will be written to the budget file for each model

provided that the Output Control for the models are set up with the “BUDGET SAVE FILE” option.

- **cell_averaging** (*string*) -
 - **cell_averaging** (*string*) is a keyword and text keyword to indicate the method that will be used for calculating the conductance for horizontal cell connections. The text value for **CELL_AVERAGING** can be “HARMONIC”, “LOGARITHMIC”, or “AMT-LMK”, which means “arithmetic-mean thickness and logarithmic-mean hydraulic conductivity”. If the user does not specify a value for **CELL_AVERAGING**, then the harmonic-mean method will be used.
- **cvoptions** (*[dewatered]*) -
 - **dewatered** (*string*) If the **DEWATERED** keyword is specified, then the vertical conductance is calculated using only the saturated thickness and properties of the overlying cell if the head in the underlying cell is below its top.
- **newton** (*boolean*) -
 - **newton** (*boolean*) keyword that activates the Newton-Raphson formulation for groundwater flow between connected, convertible groundwater cells. Cells will not dry when this option is used.
- **xt3d** (*boolean*) -
 - **xt3d** (*boolean*) keyword that activates the XT3D formulation between the cells connected with this GWF-GWF Exchange.
- **gncdata** (*{varname:data} or gncdata data*) -
 - Contains data for the gnc package. Data can be stored in a dictionary containing data for the gnc package with variable names as keys and package data as values. Data just for the **gncdata** variable is also acceptable. See gnc package documentation for more information.
- **perioddata** (*{varname:data} or perioddata data*) -
 - Contains data for the mvr package. Data can be stored in a dictionary containing data for the mvr package with variable names as keys and package data as values. Data just for the **perioddata** variable is also acceptable. See mvr package documentation for more information.
- **observations** (*{varname:data} or continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the **observations** variable is also acceptable. See obs package documentation for more information.
- **dev_interfacemodel_on** (*boolean*) -
 - **dev_interfacemodel_on** (*boolean*) activates the interface model mechanism for calculating the coefficients at (and possibly near) the exchange. This keyword should only be used for development purposes.

- **nexg** (*integer*) -
 - nexg (*integer*) keyword and integer value specifying the number of GWF-GWF exchanges.
- **exchangedata** (*[cellidm1, cellidm2, ihc, cl1, cl2, hwva, aux, boundname]*) -
 - cellidm1 (*(integer, ...)*) is the cellid of the cell in model 1 as specified in the simulation name file. For a structured grid that uses the DIS input file, CELLIDM1 is the layer, row, and column numbers of the cell. For a grid that uses the DISV input file, CELLIDM1 is the layer number and CELL2D number for the two cells. If the model uses the unstructured discretization (DISU) input file, then CELLIDM1 is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - cellidm2 (*(integer, ...)*) is the cellid of the cell in model 2 as specified in the simulation name file. For a structured grid that uses the DIS input file, CELLIDM2 is the layer, row, and column numbers of the cell. For a grid that uses the DISV input file, CELLIDM2 is the layer number and CELL2D number for the two cells. If the model uses the unstructured discretization (DISU) input file, then CELLIDM2 is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - ihc (*integer*) is an integer flag indicating the direction between node *n* and all of its *m* connections. If IHC = 0 then the connection is vertical. If IHC = 1 then the connection is horizontal. If IHC = 2 then the connection is horizontal for a vertically staggered grid.
 - cl1 (*double*) is the distance between the center of cell 1 and the its shared face with cell 2.
 - cl2 (*double*) is the distance between the center of cell 2 and the its shared face with cell 1.
 - hwva (*double*) is the horizontal width of the flow connection between cell 1 and cell 2 if IHC > 0, or it is the area perpendicular to flow of the vertical connection between cell 1 and cell 2 if IHC = 0.
 - aux (*double*) represents the values of the auxiliary variables for each GWFGWF Exchange. The values of auxiliary variables must be present for each exchange. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block.
 - boundname (*string*) name of the GWF Exchange cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** ([MFPackage](#)) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfuillaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

dfn = [['header', 'multi-package'], ['block options', 'name auxiliary', 'type
string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name
boundnames', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block
options', 'name print_input', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name print_flows', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name save_flows', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name cell_averaging', 'type string', 'valid
harmonic logarithmic amt-lmk', 'reader urword', 'optional true'], ['block options',
'name cvoptions', 'type record variablecv dewatered', 'reader urword', 'optional
true'], ['block options', 'name variablecv', 'in_record true', 'type keyword',
'reader urword'], ['block options', 'name dewatered', 'in_record true', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name newton', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name xt3d', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name
gnc_filerecord', 'type record gnc6 filein gnc6_filename', 'shape', 'reader urword',
'tagged true', 'optional true', 'construct_package gnc', 'construct_data gncdata',
'parameter_name gncdata'], ['block options', 'name filein', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name gnc6', 'type keyword', 'shape', 'in_record true', 'reader urword',
'tagged true', 'optional false'], ['block options', 'name gnc6_filename', 'type
string', 'preserve_case true', 'in_record true', 'tagged false', 'reader urword',
'optional false'], ['block options', 'name mvr_filerecord', 'type record mvr6 filein
mvr6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package mvr', 'construct_data perioddata', 'parameter_name perioddata'],
['block options', 'name mvr6', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name mvr6_filename',
'type string', 'preserve_case true', 'in_record true', 'tagged false', 'reader
urword', 'optional false'], ['block options', 'name obs_filerecord', 'type record
obs6 filein obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional
true', 'construct_package obs', 'construct_data continuous', 'parameter_name
observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block options', 'name
dev_interfacemodel_on', 'type keyword', 'reader urword', 'optional true'], ['block
dimensions', 'name nexg', 'type integer', 'reader urword', 'optional false'],
['block exchangedata', 'name exchangedata', 'type recarray cellidm1 cellidm2 ihc cl1
cl2 hwva aux boundname', 'reader urword', 'optional false'], ['block exchangedata',
'name cellidm1', 'type integer', 'in_record true', 'tagged false', 'reader urword',
'optional false', 'numeric_index true'], ['block exchangedata', 'name cellidm2',
'type integer', 'in_record true', 'tagged false', 'reader urword', 'optional false',
'numeric_index true'], ['block exchangedata', 'name ihc', 'type integer', 'in_record
true', 'tagged false', 'reader urword', 'optional false'], ['block exchangedata',
'name cl1', 'type double precision', 'in_record true', 'tagged false', 'reader
urword', 'optional false'], ['block exchangedata', 'name cl2', 'type double
precision', 'in_record true', 'tagged false', 'reader urword', 'optional false'],
['block exchangedata', 'name hwva', 'type double precision', 'in_record true',
'tagged false', 'reader urword', 'optional false'], ['block exchangedata', 'name
aux', 'type double precision', 'in_record true', 'tagged false', 'shape (naux)',
'reader urword', 'optional true'], ['block exchangedata', 'name boundname', 'type
string', 'shape', 'tagged false', 'in_record true', 'reader urword', 'optional
true']]

```

```
dfn_file_name = 'exg-gwfgwf.dfn'
```

```

exchangedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
gnc_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
mvr_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
package_abbr = 'gwfgwf'

```

flopy.mf6.modflow.mfgwfgwt module

```

class ModflowGwfgwt(simulation, loading_package=False, exgtype='GWF6-GWT6',
                    exgmnamea=None, exgmnameb=None, filename=None, pname=None, **kwargs)

```

Bases: [MFPackage](#)

ModflowGwfgwt defines a gwfgwt package.

Parameters

- **simulation** ([MFSimulation](#)) - Simulation that this package is a part of. Package is automatically added to simulation when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **exgtype** (<string>) -
 - is the exchange type (GWF-GWF or GWF-GWT).
- **exgmnamea** (<string>) -
 - is the name of the first model that is part of this exchange.
- **exgmnameb** (<string>) -
 - is the name of the second model that is part of this exchange.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** ([MFPackage](#)) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

```

dfn = [['header']]
dfn_file_name = 'exg-gwfgwt.dfn'
package_abbr = 'gwfgwt'

```

flopy.mf6.modflow.mfgwfhfb module

```
class ModflowGwfhfb(model, loading_package=False, print_input=None, maxhfb=None,
                    stress_period_data=None, filename=None, pname=None, **kwargs)
```

Bases: *MFPackage*

ModflowGwfhfb defines a hfb package within a gwf6 model.

Parameters

- **model** (*MFModel*) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **print_input** (*boolean*) -
 - print_input (boolean) keyword to indicate that the list of horizontal flow barriers will be written to the listing file immediately after it is read.
- **maxhfb** (*integer*) -
 - maxhfb (integer) integer value specifying the maximum number of horizontal flow barriers that will be entered in this input file. The value of MAXHFB is used to allocate memory for the horizontal flow barriers.
- **stress_period_data** (*[cellid1, cellid2, hydchr]*) -
 - cellid1 ((integer, ...)) identifier for the first cell. For a structured grid that uses the DIS input file, CELLID1 is the layer, row, and column numbers of the cell. For a grid that uses the DISV input file, CELLID1 is the layer number and CELL2D number for the two cells. If the model uses the unstructured discretization (DISU) input file, then CELLID1 is the node numbers for the cell. The barrier is located between cells designated as CELLID1 and CELLID2. For models that use the DIS and DISV grid types, the layer number for CELLID1 and CELLID2 must be the same. For all grid types, cells must be horizontally adjacent or the program will terminate with an error. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - cellid2 ((integer, ...)) identifier for the second cell. See CELLID1 for description of how to specify. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - hydchr (double) is the hydraulic characteristic of the horizontal- flow barrier. The hydraulic characteristic is the barrier hydraulic conductivity divided by the width of the horizontal-flow barrier. If the hydraulic characteristic is negative, then the absolute value of HYDCHR acts as a multiplier to the conductance between the two model cells specified as

containing the barrier. For example, if the value for HYDCHR was specified as -1.5, the conductance calculated for the two cells would be multiplied by 1.5.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mftullaktab package must have a mfgwflak package parent_file.

```
dfn = [['header'], ['block options', 'name print_input', 'type keyword', 'reader
urword', 'optional true'], ['block dimensions', 'name maxhfb', 'type integer',
'reader urword', 'optional false'], ['block period', 'name iper', 'type integer',
'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader
urword', 'optional false'], ['block period', 'name stress_period_data', 'type
recarray cellid1 cellid2 hydchr', 'shape (maxhfb)', 'reader urword'], ['block
period', 'name cellid1', 'type integer', 'shape (ncelldim)', 'tagged false',
'in_record true', 'reader urword'], ['block period', 'name cellid2', 'type integer',
'shape (ncelldim)', 'tagged false', 'in_record true', 'reader urword'], ['block
period', 'name hydchr', 'type double precision', 'shape', 'tagged false', 'in_record
true', 'reader urword']]
```

```
dfn_file_name = 'gwf-hfb.dfn'
```

```
package_abbr = 'gwfhfb'
```

```
stress_period_data = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfic module

```
class ModflowGwfic(model, loading_package=False, strt=1.0, filename=None, pname=None,
                    **kwargs)
```

Bases: *MFPackage*

ModflowGwfic defines a ic package within a gwf6 model.

Parameters

- **model** (*MFModel*) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **strt** (*[double]*) -
 - strt (double) is the initial (starting) head-that is, head at the beginning of the GWF Model simulation. STRT must be specified for all simulations, including steady-state simulations. One value is read for every model cell. For simulations in which the first stress period is steady state, the values used for STRT generally do not affect the simulation (exceptions may occur if cells go dry and (or) rewet). The execution time, however, will be less if STRT includes hydraulic heads that are close to the steady-state solution. A head value lower than the cell bottom can be provided if a cell should start as dry.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfuillaktab package must have a mfgwflak package parent_file.

```
dfn = [['header'], ['block griddata', 'name strt', 'type double precision', 'shape
(nodes)', 'reader readarray', 'layered true', 'default_value 1.0']]
```

```
dfn_file_name = 'gwf-ic.dfn'
```

```
package_abbr = 'gwfic'
```

```
strt = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

flopy.mf6.modflow.mfgwflak module

```
class ModflowGwflak(model, loading_package=False, auxiliary=None, boundnames=None,
    print_input=None, print_stage=None, print_flows=None,
    save_flows=None, stage_filerecord=None, budget_filerecord=None,
    budgetcsv_filerecord=None, package_convergence_filerecord=None,
    timeseries=None, observations=None, mover=None, surfdep=None,
    maximum_iterations=None, maximum_stage_change=None,
    time_conversion=None, length_conversion=None, nlakes=None,
    noutlets=None, ntables=None, packagedata=None, connectiondata=None,
    tables=None, outlets=None, perioddata=None, filename=None,
    pname=None, **kwargs)
```

Bases: *MFPackage*

ModflowGwflak defines a lak package within a gwf6 model.

Parameters

- **model** (*MFModel*) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **auxiliary** (*[string]*) -
 - auxiliary (*string*) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **boundnames** (*boolean*) -

- boundnames (boolean) keyword to indicate that boundary names may be provided with the list of lake cells.
- **print_input** (*boolean*) -
 - print_input (boolean) keyword to indicate that the list of lake information will be written to the listing file immediately after it is read.
- **print_stage** (*boolean*) -
 - print_stage (boolean) keyword to indicate that the list of lake stages will be printed to the listing file for every stress period in which "HEAD PRINT" is specified in Output Control. If there is no Output Control option and PRINT_STAGE is specified, then stages are printed for the last time step of each stress period.
- **print_flows** (*boolean*) -
 - print_flows (boolean) keyword to indicate that the list of lake flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (boolean) keyword to indicate that lake flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **stage_filerecord** (*[stagefile]*) -
 - stagefile (string) name of the binary output file to write stage information.
- **budget_filerecord** (*[budgetfile]*) -
 - budgetfile (string) name of the binary output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
 - budgetcsvfile (string) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **package_convergence_filerecord** (*[package_convergence_filename]*) -
 - package_convergence_filename (string) name of the comma spaced values output file to write package convergence information.
- **timeseries** (*{varname:data} or timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.

- **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **mover** (*boolean*) -
 - mover (boolean) keyword to indicate that this instance of the LAK Package can be used with the Water Mover (MVR) Package. When the MOVER option is specified, additional memory is allocated within the package to store the available, provided, and received water.
- **surfdep** (*double*) -
 - surfdep (double) real value that defines the surface depression depth for VERTICAL lake-GWF connections. If specified, SURFDEP must be greater than or equal to zero. If SURFDEP is not specified, a default value of zero is used for all vertical lake-GWF connections.
- **maximum_iterations** (*integer*) -
 - maximum_iterations (integer) integer value that defines the maximum number of Newton-Raphson iterations allowed for a lake. By default, MAXIMUM_ITERATIONS is equal to 100. MAXIMUM_ITERATIONS would only need to be increased from the default value if one or more lakes in a simulation has a large water budget error.
- **maximum_stage_change** (*double*) -
 - maximum_stage_change (double) real value that defines the lake stage closure tolerance. By default, MAXIMUM_STAGE_CHANGE is equal to 1×10^{-5} . The MAXIMUM_STAGE_CHANGE would only need to be increased or decreased from the default value if the water budget error for one or more lakes is too small or too large, respectively.
- **time_conversion** (*double*) -
 - time_conversion (double) real value that is used to convert user- specified Manning's roughness coefficients or gravitational acceleration used to calculate outlet flows from seconds to model time units. TIME_CONVERSION should be set to 1.0, 60.0, 3,600.0, 86,400.0, and 31,557,600.0 when using time units (TIME_UNITS) of seconds, minutes, hours, days, or years in the simulation, respectively. CONVTIME does not need to be specified if no lake outlets are specified or TIME_UNITS are seconds.
- **length_conversion** (*double*) -
 - length_conversion (double) real value that is used to convert outlet user-specified Manning's roughness coefficients or gravitational acceleration used to calculate outlet flows from meters to model length units. LENGTH_CONVERSION

should be set to 3.28081, 1.0, and 100.0 when using length units (LENGTH_UNITS) of feet, meters, or centimeters in the simulation, respectively. LENGTH_CONVERSION does not need to be specified if no lake outlets are specified or LENGTH_UNITS are meters.

- **nlakes** (*integer*) -
 - nlakes (*integer*) value specifying the number of lakes that will be simulated for all stress periods.
- **noutlets** (*integer*) -
 - outlets (*integer*) value specifying the number of outlets that will be simulated for all stress periods. If NOUTLETS is not specified, a default value of zero is used.
- **ntables** (*integer*) -
 - ntables (*integer*) value specifying the number of lakes tables that will be used to define the lake stage, volume relation, and surface area. If NTABLES is not specified, a default value of zero is used.
- **packagedata** (*[lakeno, strt, nlakeconn, aux, boundname]*) -
 - lakeno (*integer*) integer value that defines the lake number associated with the specified PACKAGEDATA data on the line. LAKENO must be greater than zero and less than or equal to NLAKES. Lake information must be specified for every lake or the program will terminate with an error. The program will also terminate with an error if information for a lake is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - strt (*double*) real value that defines the starting stage for the lake.
 - nlakeconn (*integer*) integer value that defines the number of GWF cells connected to this (LAKENO) lake. There can only be one vertical lake connection to each GWF cell. NLAKECONN must be greater than zero.
 - aux (*double*) represents the values of the auxiliary variables for each lake. The values of auxiliary variables must be present for each lake. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the "Time-Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - boundname (*string*) name of the lake cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.

- **connectiondata** ([*lakeno*, *iconn*, *cellid*, *claktype*, *bedleak*, *belev*, *telev*,) -
connlen, **connwidth**]
 - *lakeno* (integer) integer value that defines the lake number associated with the specified CONNECTIONDATA data on the line. LAKENO must be greater than zero and less than or equal to NLAKES. Lake connection information must be specified for every lake connection to the GWF model (NLAKECONN) or the program will terminate with an error. The program will also terminate with an error if connection information for a lake connection to the GWF model is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *iconn* (integer) integer value that defines the GWF connection number for this lake connection entry. ICONN must be greater than zero and less than or equal to NLAKECONN for lake LAKENO. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *cellid* ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *claktype* (string) character string that defines the lake-GWF connection type for the lake connection. Possible lake-GWF connection type strings include: VERTICAL-character keyword to indicate the lake-GWF connection is vertical and connection conductance calculations use the hydraulic conductivity corresponding to the K_{33} tensor component defined for CELLID in the NPF package. HORIZONTAL-character keyword to indicate the lake-GWF connection is horizontal and connection conductance calculations use the hydraulic conductivity corresponding to the K_{11} tensor component defined for CELLID in the NPF package. EMBEDDEDH-character keyword to indicate the lake-GWF connection is embedded in a single cell and connection conductance calculations use the hydraulic conductivity corresponding to the K_{11} tensor component defined for CELLID in the NPF package. EMBEDDEDV-character keyword to indicate the lake-GWF connection is embedded in a single cell and connection conductance calculations use the hydraulic conductivity corresponding to the K_{33} tensor component defined

- for CELLID in the NPF package. Embedded lakes can only be connected to a single cell (NLAKECONN = 1) and there must be a lake table associated with each embedded lake.
- bedleak (string) character string or real value that defines the bed leakance for the lake-GWF connection. BEDLEAK must be greater than or equal to zero or specified to be NONE. If BEDLEAK is specified to be NONE, the lake-GWF connection conductance is solely a function of aquifer properties in the connected GWF cell and lakebed sediments are assumed to be absent.
- belev (double) real value that defines the bottom elevation for a HORIZONTAL lake-GWF connection. Any value can be specified if CLAKTYPE is VERTICAL, EMBEDDEDH, or EMBEDDEDV. If CLAKTYPE is HORIZONTAL and BELEV is not equal to TELEV, BELEV must be greater than or equal to the bottom of the GWF cell CELLID. If BELEV is equal to TELEV, BELEV is reset to the bottom of the GWF cell CELLID.
- telev (double) real value that defines the top elevation for a HORIZONTAL lake-GWF connection. Any value can be specified if CLAKTYPE is VERTICAL, EMBEDDEDH, or EMBEDDEDV. If CLAKTYPE is HORIZONTAL and TELEV is not equal to BELEV, TELEV must be less than or equal to the top of the GWF cell CELLID. If TELEV is equal to BELEV, TELEV is reset to the top of the GWF cell CELLID.
- connlen (double) real value that defines the distance between the connected GWF CELLID node and the lake for a HORIZONTAL, EMBEDDEDH, or EMBEDDEDV lake-GWF connection. CONLENN must be greater than zero for a HORIZONTAL, EMBEDDEDH, or EMBEDDEDV lake-GWF connection. Any value can be specified if CLAKTYPE is VERTICAL.
- connwidth (double) real value that defines the connection face width for a HORIZONTAL lake-GWF connection. CONNWIDTH must be greater than zero for a HORIZONTAL lake-GWF connection. Any value can be specified if CLAKTYPE is VERTICAL, EMBEDDEDH, or EMBEDDEDV.
- **tables** ([lakeno, tab6_filename]) -
 - lakeno (integer) integer value that defines the lake number associated with the specified TABLES data on the line. LAKENO must be greater than zero and less than or equal to NLAKES. The program will terminate with an error if table information for a lake is specified more than once or the number of specified tables is less than NTABLES. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - tab6_filename (string) character string that defines the path and filename for the file containing lake table data for the lake connection. The TAB6_FILENAME file includes the number of entries in the file and the relation between stage, volume,

and surface area for each entry in the file. Lake table files for EMBEDDEDH and EMBEDDEDV lake-GWF connections also include lake-GWF exchange area data for each entry in the file. Instructions for creating the TAB6_FILENAME input file are provided in Lake Table Input File section.

- **outlets** ([*outletno*, *lakein*, *lakeout*, *couttype*, *invert*, *width*, *rough*,) -
slope]
 - *outletno* (integer) integer value that defines the outlet number associated with the specified OUTLETS data on the line. OUTLETNO must be greater than zero and less than or equal to NOUTLETS. Outlet information must be specified for every outlet or the program will terminate with an error. The program will also terminate with an error if information for a outlet is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *lakein* (integer) integer value that defines the lake number that outlet is connected to. LAKEIN must be greater than zero and less than or equal to NLAKES. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *lakeout* (integer) integer value that defines the lake number that outlet discharge from lake outlet OUTLETNO is routed to. LAKEOUT must be greater than or equal to zero and less than or equal to NLAKES. If LAKEOUT is zero, outlet discharge from lake outlet OUTLETNO is discharged to an external boundary. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *couttype* (string) character string that defines the outlet type for the outlet OUTLETNO. Possible COUTTYPE strings include: SPECIFIED- character keyword to indicate the outlet is defined as a specified flow. MANNING-character keyword to indicate the outlet is defined using Manning's equation. WEIR-character keyword to indicate the outlet is defined using a sharp weir equation.
 - *invert* (double) real value that defines the invert elevation for the lake outlet. Any value can be specified if COUTTYPE is SPECIFIED. If the Options block includes a TIMESERIESFILE entry (see the "Time- Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - *width* (double) real value that defines the width of the lake outlet. Any value can be specified if COUTTYPE is SPECIFIED.

If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- rough (double) real value that defines the roughness coefficient for the lake outlet. Any value can be specified if COUTTYPE is not MANNING. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- slope (double) real value that defines the bed slope for the lake outlet. Any value can be specified if COUTTYPE is not MANNING. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- **perioddata** ([number, laksetting]) -
 - number (integer) integer value that defines the lake or outlet number associated with the specified PERIOD data on the line. NUMBER must be greater than zero and less than or equal to NLAKES for a lake number and less than or equal to NOUTLETS for an outlet number. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - laksetting (keystring) line of information that is parsed into a keyword and values. Keyword values that can be used to start the LAKSETTING string include both keywords for lake settings and keywords for outlet settings. Keywords for lake settings include: STATUS, STAGE, RAINFALL, EVAPORATION, RUNOFF, INFLOW, WITHDRAWAL, and AUXILIARY. Keywords for outlet settings include RATE, INVERT, WIDTH, SLOPE, and ROUGH.

status

[[string]]

- * status (string) keyword option to define lake status. STATUS can be ACTIVE, INACTIVE, or CONSTANT. By default, STATUS is ACTIVE.

stage

[[string]]

- * stage (string) real or character value that defines the stage for the lake. The specified STAGE is only applied if the lake is a constant stage lake. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

rainfall

[[string]]

- * rainfall (string) real or character value that defines the rainfall rate (LT^{-1}) for the lake. Value must be greater than or equal to zero. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

evaporation

[[string]]

- * evaporation (string) real or character value that defines the maximum evaporation rate (LT^{-1}) for the lake. Value must be greater than or equal to zero. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

runoff

[[string]]

- * runoff (string) real or character value that defines the runoff rate (L^3T^{-1}) for the lake. Value must be greater than or equal to zero. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

inflow

[[string]]

- * inflow (string) real or character value that defines the volumetric inflow rate (L^3T^{-1}) for the lake. Value must be greater than or equal to zero. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value. By default, inflow rates are zero for each lake.

withdrawal

[[string]]

- * withdrawal (string) real or character value that defines the maximum withdrawal rate (L^3T^{-1}) for the lake. Value must be greater than or equal to zero. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

rate

[[string]]

- * rate (string) real or character value that defines the extraction rate for the lake outflow. A positive value indicates inflow and a negative value indicates outflow

from the lake. RATE only applies to active (IBOUND > 0) lakes. A specified RATE is only applied if COUTTYPE for the OUTLETNO is SPECIFIED. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value. By default, the RATE for each SPECIFIED lake outlet is zero.

invert

[[string]]

* invert (string) real or character value that defines the invert elevation for the lake outlet. A specified INVERT value is only used for active lakes if COUTTYPE for lake outlet OUTLETNO is not SPECIFIED. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

width

[[string]]

* width (string) real or character value that defines the width of the lake outlet. A specified WIDTH value is only used for active lakes if COUTTYPE for lake outlet OUTLETNO is not SPECIFIED. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

slope

[[string]]

* slope (string) real or character value that defines the bed slope for the lake outlet. A specified SLOPE value is only used for active lakes if COUTTYPE for lake outlet OUTLETNO is MANNING. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

rough

[[string]]

* rough (string) real value that defines the roughness coefficient for the lake outlet. Any value can be specified if COUTTYPE is not MANNING. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

auxiliaryrecord

[[auxname, auxval]]

* **auxname** (string) name for the auxiliary variable to be assigned AUXVAL. AUXNAME must match one of the auxiliary variable names defined in the OPTIONS block. If AUXNAME does not match one of the auxiliary variable names defined in the OPTIONS block the data are ignored.

* **auxval** (double) value for the auxiliary variable. If the Options block includes a TIMESERIESFILE entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfullaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

connectiondata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

dfn = [['header', 'multi-package'], ['block options', 'name auxiliary', 'type
string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name
boundnames', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block
options', 'name print_input', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name print_stage', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name print_flows', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name save_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name stage_filerecord', 'type record
stage fileout stagefile', 'shape', 'reader urword', 'tagged true', 'optional true'],
['block options', 'name stage', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name stagefile', 'type
string', 'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged
false', 'optional false'], ['block options', 'name budget_filerecord', 'type record
budget fileout budgetfile', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name budget', 'type keyword', 'shape', 'in_record true',
'reader urword', 'tagged true', 'optional false'], ['block options', 'name fileout',
'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name budgetfile', 'type string', 'preserve_case true',
'shape', 'in_record true', 'reader urword', 'tagged false', 'optional false'],
['block options', 'name budgetcsv_filerecord', 'type record budgetcsv fileout
budgetcsvfile', 'shape', 'reader urword', 'tagged true', 'optional true'], ['block
options', 'name budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name budgetcsvfile',
'type string', 'preserve_case true', 'shape', 'in_record true', 'reader urword',
'tagged false', 'optional false'], ['block options', 'name
package_convergence_filerecord', 'type record package_convergence fileout
package_convergence_filename', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name package_convergence', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name package_convergence_filename', 'type string', 'shape', 'in_record
true', 'reader urword', 'tagged false', 'optional false'], ['block options', 'name
ts_filerecord', 'type record ts6 filein ts6_filename', 'shape', 'reader urword',
'tagged true', 'optional true', 'construct_package ts', 'construct_data timeseries',
'parameter_name timeseries'], ['block options', 'name ts6', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader urword',
'tagged true', 'optional false'], ['block options', 'name ts6_filename', 'type
string', 'preserve_case true', 'in_record true', 'reader urword', 'optional false',
'tagged false'], ['block options', 'name obs_filerecord', 'type record obs6 filein
obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package obs', 'construct_data continuous', 'parameter_name
observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block options', 'name mover', 'type
keyword', 'tagged true', 'reader urword', 'optional true'], ['block options', 'name
surfdep', 'type double precision', 'reader urword', 'optional true'], ['block
options', 'name maximum_iterations', 'type integer', 'reader urword', 'optional
true'], ['block options', 'name maximum_stage_change', 'type double precision',
'reader urword', 'optional true'], ['block options', 'name time_conversion', 'type
double precision', 'reader urword', 'optional true'], ['block options', 'name
length_conversion', 'type double precision', 'reader urword', 'optional true'],
['block dimensions', 'name nlakes', 'type integer', 'reader urword', 'optional
false'], ['block dimensions', 'name noutlets', 'type integer', 'reader urword',
'optional false'], ['block dimensions', 'name ntables', 'type integer', 'reader
urword', 'optional false'], ['block packagedata', 'name packagedata', 'type recarray
lakeno strt nlakeconn aux boundname', 'shape (maxbound)', 'reader urword', 'optional
true'], ['block packagedata', 'name lakeno', 'type integer', 'shape', 'tagged false', 'in_record
true', 'reader urword', 'numeric_index true'], ['block packagedata', 'name strt',
'type double precision', 'shape', 'tagged false', 'in_record true', 'reader

```

```

dfn_file_name = 'gwf-lak.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

outlets = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwflak'

package_convergence_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator
object>

packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

stage_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

tables = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

flopy.mf6.modflow.mfgwfmaw module

```

class ModflowGwfmaw(model, loading_package=False, auxiliary=None, boundnames=None,
    print_input=None, print_head=None, print_flows=None, save_flows=None,
    head_filerecord=None, budget_filerecord=None,
    budgetcsv_filerecord=None, no_well_storage=None,
    flow_correction=None, flowing_wells=None, shutdown_theta=None,
    shutdown_kappa=None, mfrcsv_filerecord=None, timeseries=None,
    observations=None, mover=None, nmawwells=None, packagedata=None,
    connectiondata=None, perioddata=None, filename=None, pname=None,
    **kwargs)

```

Bases: [MFPackage](#)

ModflowGwfmaw defines a maw package within a gw6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **auxiliary** (*[string]*) -
 - auxiliary (string) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if

auxiliary variables are specified on more than one line in the options block.

- **boundnames** (*boolean*) -
 - boundnames (*boolean*) keyword to indicate that boundary names may be provided with the list of multi-aquifer well cells.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of multi- aquifer well information will be written to the listing file immediately after it is read.
- **print_head** (*boolean*) -
 - print_head (*boolean*) keyword to indicate that the list of multi- aquifer well heads will be printed to the listing file for every stress period in which “HEAD PRINT” is specified in Output Control. If there is no Output Control option and PRINT_HEAD is specified, then heads are printed for the last time step of each stress period.
- **print_flows** (*boolean*) -
 - print_flows (*boolean*) keyword to indicate that the list of multi- aquifer well flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (*boolean*) keyword to indicate that multi-aquifer well flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **head_filerecord** (*[headfile]*) -
 - headfile (*string*) name of the binary output file to write head information.
- **budget_filerecord** (*[budgetfile]*) -
 - budgetfile (*string*) name of the binary output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
 - budgetcsvfile (*string*) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **no_well_storage** (*boolean*) -
 - no_well_storage (*boolean*) keyword that deactivates inclusion of well storage contributions to the multi-aquifer well package continuity equation.
- **flow_correction** (*boolean*) -

- `flow_correction` (boolean) keyword that activates flow corrections in cases where the head in a multi-aquifer well is below the bottom of the screen for a connection or the head in a convertible cell connected to a multi-aquifer well is below the cell bottom. When flow corrections are activated, unit head gradients are used to calculate the flow between a multi-aquifer well and a connected GWF cell. By default, flow corrections are not made.
- **`flowing_wells`** (boolean) -
 - `flowing_wells` (boolean) keyword that activates the flowing wells option for the multi-aquifer well package.
- **`shutdown_theta`** (double) -
 - `shutdown_theta` (double) value that defines the weight applied to discharge rate for wells that limit the water level in a discharging well (defined using the `HEAD_LIMIT` keyword in the stress period data). `SHUTDOWN_THETA` is used to control discharge rate oscillations when the flow rate from the aquifer is less than the specified flow rate from the aquifer to the well. Values range between 0.0 and 1.0, and larger values increase the weight (decrease under-relaxation) applied to the well discharge rate. The `HEAD_LIMIT` option has been included to facilitate backward compatibility with previous versions of MODFLOW but use of the `RATE_SCALING` option instead of the `HEAD_LIMIT` option is recommended. By default, `SHUTDOWN_THETA` is 0.7.
- **`shutdown_kappa`** (double) -
 - `shutdown_kappa` (double) value that defines the weight applied to discharge rate for wells that limit the water level in a discharging well (defined using the `HEAD_LIMIT` keyword in the stress period data). `SHUTDOWN_KAPPA` is used to control discharge rate oscillations when the flow rate from the aquifer is less than the specified flow rate from the aquifer to the well. Values range between 0.0 and 1.0, and larger values increase the weight applied to the well discharge rate. The `HEAD_LIMIT` option has been included to facilitate backward compatibility with previous versions of MODFLOW but use of the `RATE_SCALING` option instead of the `HEAD_LIMIT` option is recommended. By default, `SHUTDOWN_KAPPA` is 0.0001.
- **`mfrcsv_filerecord`** ([`mfrcsvfile`]) -
 - `mfrcsvfile` (string) name of the comma-separated value (CSV) output file to write information about multi-aquifer well extraction or injection rates that have been reduced by the program. Entries are only written if the extraction or injection rates are reduced.
- **`timeseries`** ({`varname:data`} or *timeseries data*) -
 - Contains data for the `ts` package. Data can be stored in a dictionary containing data for the `ts` package with variable names as keys and package data as values. Data just for

the timeseries variable is also acceptable. See ts package documentation for more information.

- **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **mover** (*boolean*) -
 - mover (boolean) keyword to indicate that this instance of the MAW Package can be used with the Water Mover (MVR) Package. When the MOVER option is specified, additional memory is allocated within the package to store the available, provided, and received water.
- **nmawwells** (*integer*) -
 - nmawwells (integer) integer value specifying the number of multi- aquifer wells that will be simulated for all stress periods.
- **packagedata** (*[wellno, radius, bottom, strt, condeqn, ngwfnodes, aux,)* -
boundname]
 - wellno (integer) integer value that defines the well number associated with the specified PACKAGEDATA data on the line. WELLNO must be greater than zero and less than or equal to NMAWWELLS. Multi- aquifer well information must be specified for every multi-aquifer well or the program will terminate with an error. The program will also terminate with an error if information for a multi-aquifer well is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - radius (double) radius for the multi-aquifer well. The program will terminate with an error if the radius is less than or equal to zero.
 - bottom (double) bottom elevation of the multi-aquifer well. If CONDEQN is SPECIFIED, THIEM, SKIN, or COMPOSITE, BOTTOM is set to the cell bottom in the lowermost GWF cell connection in cases where the specified well bottom is above the bottom of this GWF cell. If CONDEQN is MEAN, BOTTOM is set to the lowermost GWF cell connection screen bottom in cases where the specified well bottom is above this value. The bottom elevation defines the lowest well head that will be simulated when the NEWTON UNDER_RELAXATION option is specified in the GWF model name file. The bottom elevation is also used to calculate volumetric storage in the well.

- strt (double) starting head for the multi-aquifer well. The program will terminate with an error if the starting head is less than the specified well bottom.
 - condeqn (string) character string that defines the conductance equation that is used to calculate the saturated conductance for the multi-aquifer well. Possible multi-aquifer well CONDEQN strings include: SPECIFIED-character keyword to indicate the multi-aquifer well saturated conductance will be specified. THIEM-character keyword to indicate the multi-aquifer well saturated conductance will be calculated using the Thiem equation, which considers the cell top and bottom, aquifer hydraulic conductivity, and effective cell and well radius. SKIN-character keyword to indicate that the multi-aquifer well saturated conductance will be calculated using the cell top and bottom, aquifer and screen hydraulic conductivity, and well and skin radius. CUMULATIVE-character keyword to indicate that the multi-aquifer well saturated conductance will be calculated using a combination of the Thiem and SKIN equations. MEAN-character keyword to indicate the multi-aquifer well saturated conductance will be calculated using the aquifer and screen top and bottom, aquifer and screen hydraulic conductivity, and well and skin radius. The CUMULATIVE conductance equation is identical to the SKIN LOSSTYPE in the Multi-Node Well (MNW2) package for MODFLOW-2005. The program will terminate with an error condition if CONDEQN is SKIN or CUMULATIVE and the calculated saturated conductance is less than zero; if an error condition occurs, it is suggested that the THEIM or MEAN conductance equations be used for these multi-aquifer wells.
 - ngwfnodes (integer) integer value that defines the number of GWF nodes connected to this (WELLNO) multi-aquifer well. NGWFNODES must be greater than zero.
 - aux (double) represents the values of the auxiliary variables for each multi-aquifer well. The values of auxiliary variables must be present for each multi-aquifer well. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the "Time-Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - boundname (string) name of the multi-aquifer well cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.
- **connectiondata** ([wellno, icon, cellid, scrn_top, scrn_bot, hk_skin,] -

radius_skin]

- wellno (integer) integer value that defines the well number associated with the specified CONNECTIONDATA data on the line. WELLNO must be greater than zero and less than or equal to NMAWWELLS. Multi- aquifer well connection information must be specified for every multi-aquifer well connection to the GWF model (NGWFNODES) or the program will terminate with an error. The program will also terminate with an error if connection information for a multi-aquifer well connection to the GWF model is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
- icon (integer) integer value that defines the GWF connection number for this multi-aquifer well connection entry. ICONN must be greater than zero and less than or equal to NGWFNODES for multi-aquifer well WELLNO. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
- cellid ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. One or more screened intervals can be connected to the same CELLID if CONDEQN for a well is MEAN. The program will terminate with an error if MAW wells using SPECIFIED, THIEB, SKIN, or CUMULATIVE conductance equations have more than one connection to the same CELLID. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
- scrn_top (double) value that defines the top elevation of the screen for the multi-aquifer well connection. If CONDEQN is SPECIFIED, THIEB, SKIN, or COMPOSITE, SCRN_TOP can be any value and is set to the top of the cell. If CONDEQN is MEAN, SCRN_TOP is set to the multi-aquifer well connection cell top if the specified value is greater than the cell top. The program will terminate with an error if the screen top is less than the screen bottom.
- scrn_bot (double) value that defines the bottom elevation of the screen for the multi-aquifer well connection. If CONDEQN is SPECIFIED, THIEB, SKIN, or COMPOSITE, SCRN_BOT can be any value is set to the bottom of the cell. If

CONDEQN is MEAN, SCRNBOT is set to the multi-aquifer well connection cell bottom if the specified value is less than the cell bottom. The program will terminate with an error if the screen bottom is greater than the screen top.

- `hk_skin` (double) value that defines the skin (filter pack) hydraulic conductivity (if CONDEQN for the multi-aquifer well is SKIN, CUMULATIVE, or MEAN) or conductance (if CONDEQN for the multi-aquifer well is SPECIFIED) for each GWF node connected to the multi-aquifer well (NGWFNODES). If CONDEQN is SPECIFIED, HK_SKIN must be greater than or equal to zero. HK_SKIN can be any value if CONDEQN is THIEM. Otherwise, HK_SKIN must be greater than zero. If CONDEQN is SKIN, the contrast between the cell transmissivity (the product of geometric mean horizontal hydraulic conductivity and the cell thickness) and the well transmissivity (the product of HK_SKIN and the screen thicknesses) must be greater than one in node CELLID or the program will terminate with an error condition; if an error condition occurs, it is suggested that the HK_SKIN be reduced to a value less than K11 and K22 in node CELLID or the THEIM or MEAN conductance equations be used for these multi-aquifer wells.
- `radius_skin` (double) real value that defines the skin radius (filter pack radius) for the multi-aquifer well. RADIUS_SKIN can be any value if CONDEQN is SPECIFIED or THIEM. If CONDEQN is SKIN, CUMULATIVE, or MEAN, the program will terminate with an error if RADIUS_SKIN is less than or equal to the RADIUS for the multi-aquifer well.

• **perioddata** (*[wellno, mawsetting]*) -

- `wellno` (integer) integer value that defines the well number associated with the specified PERIOD data on the line. WELLNO must be greater than zero and less than or equal to NMAWWELLS. This argument is an index variable, which means that it should be treated as zero- based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.
- `mawsetting` (keystring) line of information that is parsed into a keyword and values. Keyword values that can be used to start the MAWSETTING string include: STATUS, FLOWING_WELL, RATE, WELL_HEAD, HEAD_LIMIT, SHUT_OFF, RATE_SCALING, and AUXILIARY.

status

[[string]]

- * `status` (string) keyword option to define well status. STATUS can be ACTIVE, INACTIVE, or CONSTANT. By default, STATUS is ACTIVE.

flowing_wellrecord

[[fwelev, fwcond, fwrlen]]

- * `fwelev` (double) elevation used to determine whether

or not the well is flowing.

- * `fwcond` (double) conductance used to calculate the discharge of a free flowing well. Flow occurs when the head in the well is above the well top elevation (`FWELEV`).
- * `fwrlen` (double) length used to reduce the conductance of the flowing well. When the head in the well drops below the well top plus the reduction length, then the conductance is reduced. This reduction length can be used to improve the stability of simulations with flowing wells so that there is not an abrupt change in flowing well rates.

rate

[[double]]

- * `rate` (double) is the volumetric pumping rate for the multi-aquifer well. A positive value indicates recharge and a negative value indicates discharge (pumping). `RATE` only applies to active (`IBOUND > 0`) multi-aquifer wells. If the Options block includes a `TIMESERIESFILE` entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value. By default, the `RATE` for each multi-aquifer well is zero.

well_head

[[double]]

- * `well_head` (double) is the head in the multi-aquifer well. `WELL_HEAD` is only applied to constant head (`STATUS` is `CONSTANT`) and inactive (`STATUS` is `INACTIVE`) multi-aquifer wells. If the Options block includes a `TIMESERIESFILE` entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value. The program will terminate with an error if `WELL_HEAD` is less than the bottom of the well.

head_limit

[[string]]

- * `head_limit` (string) is the limiting water level (head) in the well, which is the minimum of the well `RATE` or the well inflow rate from the aquifer. `HEAD_LIMIT` can be applied to extraction wells (`RATE < 0`) or injection wells (`RATE > 0`). `HEAD_LIMIT` can be deactivated by specifying the text string ‘OFF’. The `HEAD_LIMIT` option is based on the `HEAD_LIMIT` functionality available in the MNW2 (Konikow et al., 2009) package for MODFLOW-2005. The `HEAD_LIMIT` option has been included to facilitate backward compatibility with previous versions of MODFLOW but use of the `RATE_SCALING` option instead of the

HEAD_LIMIT option is recommended. By default, HEAD_LIMIT is 'OFF'.

shutoffrecord

[[minrate, maxrate]]

- * minrate (double) is the minimum rate that a well must exceed to shutoff a well during a stress period. The well will shut down during a time step if the flow rate to the well from the aquifer is less than MINRATE. If a well is shut down during a time step, reactivation of the well cannot occur until the next time step to reduce oscillations. MINRATE must be less than maxrate.
- * maxrate (double) is the maximum rate that a well must exceed to reactivate a well during a stress period. The well will reactivate during a timestep if the well was shutdown during the previous time step and the flow rate to the well from the aquifer exceeds maxrate. Reactivation of the well cannot occur until the next time step if a well is shutdown to reduce oscillations. maxrate must be greater than MINRATE.

rate_scalingrecord

[[pump_elevation, scaling_length]]

- * pump_elevation (double) is the elevation of the multi-aquifer well pump (PUMP_ELEVATION). PUMP_ELEVATION should not be less than the bottom elevation (BOTTOM) of the multi-aquifer well.
- * scaling_length (double) height above the pump elevation (SCALING_LENGTH). If the simulated well head is below this elevation (pump elevation plus the scaling length), then the pumping rate is reduced.

auxiliaryrecord

[[auxname, auxval]]

- * auxname (string) name for the auxiliary variable to be assigned AUXVAL. AUXNAME must match one of the auxiliary variable names defined in the OPTIONS block. If AUXNAME does not match one of the auxiliary variable names defined in the OPTIONS block the data are ignored.
- * auxval (double) value for the auxiliary variable. If the Options block includes a TIMESERIESFILE entry (see the "Time- Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.

- `parent_file` ([MFPackage](#)) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfuillaktab package must have a mfgwflak package `parent_file`.

`auxiliary` = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

`budget_filerecord` = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

`budgetcsv_filerecord` = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

`connectiondata` = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

dfn = [['header', 'multi-package'], ['block options', 'name auxiliary', 'type
string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name
boundnames', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block
options', 'name print_input', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name print_head', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name print_flows', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name save_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name head_filerecord', 'type record
head fileout headfile', 'shape', 'reader urword', 'tagged true', 'optional true'],
['block options', 'name head', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name headfile', 'type
string', 'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged
false', 'optional false'], ['block options', 'name budget_filerecord', 'type record
budget fileout budgetfile', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name budget', 'type keyword', 'shape', 'in_record true',
'reader urword', 'tagged true', 'optional false'], ['block options', 'name fileout',
'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name budgetfile', 'type string', 'preserve_case true',
'shape', 'in_record true', 'reader urword', 'tagged false', 'optional false'],
['block options', 'name budgetcsv_filerecord', 'type record budgetcsv fileout
budgetcsvfile', 'shape', 'reader urword', 'tagged true', 'optional true'], ['block
options', 'name budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name budgetcsvfile',
'type string', 'preserve_case true', 'shape', 'in_record true', 'reader urword',
'tagged false', 'optional false'], ['block options', 'name no_well_storage', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name
flow_correction', 'type keyword', 'reader urword', 'optional true'], ['block
options', 'name flowing_wells', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name shutdown_theta', 'type double precision', 'reader urword',
'optional true'], ['block options', 'name shutdown_kappa', 'type double precision',
'reader urword', 'optional true'], ['block options', 'name mfrcsv_filerecord', 'type
record maw_flow_reduce_csv fileout mfrcsvfile', 'shape', 'reader urword', 'tagged
true', 'optional true'], ['block options', 'name maw_flow_reduce_csv', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name mfrcsvfile', 'type string', 'preserve_case true',
'shape', 'in_record true', 'reader urword', 'tagged false', 'optional false'],
['block options', 'name ts_filerecord', 'type record ts6 filein ts6_filename',
'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package ts',
'construct_data timeseries', 'parameter_name timeseries'], ['block options', 'name
ts6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name filein', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name ts6_filename', 'type string', 'preserve_case true', 'in_record
true', 'reader urword', 'optional false', 'tagged false'], ['block options', 'name
obs_filerecord', 'type record obs6 filein obs6_filename', 'shape', 'reader urword',
'tagged true', 'optional true', 'construct_package obs', 'construct_data
continuous', 'parameter_name observations'], ['block options', 'name obs6', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name obs6_filename', 'type string', 'preserve_case
true', 'in_record true', 'tagged false', 'reader urword', 'optional false'], ['block
options', 'name mover', 'type keyword', 'tagged true', 'reader urword', 'optional
true'], ['block dimensions', 'name nmawwells', 'type integer', 'reader urword',
'optional false'], ['block packagedata', 'name packagedata', 'type recarray wellno
radius bottom strt condeqn ngwfnodes aux boundname', 'shape (nmawwells)', 'reader
urword'], ['block packagedata', 'name wellno', 'type integer', 'shape', 'tagged
false', 'in_record true', 'reader urword', 'numeric_index true'], ['block
packagedata', 'name radius', 'type double precision', 'shape', 'tagged false',
1413
'in_record true', 'reader urword'], ['block packagedata', 'name bottom', 'type
double precision', 'shape', 'tagged false', 'in_record true', 'reader urword'],
['block packagedata', 'name strt', 'type double precision', 'shape', 'tagged false',

```

```
dfn_file_name = 'gwf-maw.dfn'

head_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

mfrcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfmau'

packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfmvr module

```
class GwfmvrPackages(model_or_sim, parent, pkg_type, filerecord, package=None,
                    package_class=None)
```

Bases: [*MFChildPackages*](#)

GwfmvrPackages is a container class for the ModflowGwfmvr class.

initialize()

Initializes a new ModflowGwfmvr package removing any sibling child packages attached to the same parent package. See ModflowGwfmvr init documentation for definition of parameters.

append_package()

Adds a new ModflowGwfmvr package to the container. See ModflowGwfmvr init documentation for definition of parameters.

```
append_package(print_input=None, print_flows=None, modelnames=None,
               budget_filerecord=None, budgetcsv_filerecord=None, maxmvr=None,
               maxpackages=None, packages=None, perioddata=None, filename=None,
               pname=None)
```

```
initialize(print_input=None, print_flows=None, modelnames=None,
           budget_filerecord=None, budgetcsv_filerecord=None, maxmvr=None,
           maxpackages=None, packages=None, perioddata=None, filename=None,
           pname=None)
```

package_abbr = 'gwfmvrapackages'

```
class ModflowGwfmvr(parent_model_or_package, loading_package=False, print_input=None,
                   print_flows=None, modelnames=None, budget_filerecord=None,
                   budgetcsv_filerecord=None, maxmvr=None, maxpackages=None,
                   packages=None, perioddata=None, filename=None, pname=None, **kwargs)
```

Bases: [*MFPackage*](#)

ModflowGwfmvr defines a mvr package within a gw6 model. This package can only be used to move water between packages within a single model. To move water between models use ModflowMvr.

Parameters

- **parent_model_or_package** (*MFModel/MFPackage*) -
Parent_model_or_package that this package is a part of. Package is automatically added to parent_model_or_package when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **print_input** (*boolean*) -
 - print_input (boolean) keyword to indicate that the list of MVR information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (boolean) keyword to indicate that the list of MVR flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **modelnames** (*boolean*) -
 - modelnames (boolean) keyword to indicate that all package names will be preceded by the model name for the package. Model names are required when the Mover Package is used with a GWF-GWF Exchange. The MODELNAME keyword should not be used for a Mover Package that is for a single GWF Model.
- **budget_filerecord** (*[budgetfile]*) -
 - budgetfile (string) name of the output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
 - budgetcsvfile (string) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **maxmvr** (*integer*) -
 - maxmvr (integer) integer value specifying the maximum number of water mover entries that will be specified for any stress period.
- **maxpackages** (*integer*) -
 - maxpackages (integer) integer value specifying the number of unique packages that are included in this water mover input file.
- **packages** (*[mname, pname]*) -
 - mname (string) name of model containing the package. Model names are assigned by the user in the simulation name file.
 - pname (string) is the name of a package that may be included in a subsequent stress period block. The package name is assigned in the name file for the GWF Model. Package names

are optionally provided in the name file. If they are not provided by the user, then packages are assigned a default value, which is the package acronym followed by a hyphen and the package number. For example, the first Drain Package is named DRN-1. The second Drain Package is named DRN-2, and so forth.

- **perioddata** ([*mname1*, *pname1*, *id1*, *mname2*, *pname2*, *id2*, *mvrtype*, *value*]) -
 - *mname1* (string) name of model containing the package, PNAME1.
 - *pname1* (string) is the package name for the provider. The package PNAME1 must be designated to provide water through the MVR Package by specifying the keyword "MOVER" in its OPTIONS block.
 - *id1* (integer) is the identifier for the provider. For the standard boundary packages, the provider identifier is the number of the boundary as it is listed in the package input file. (Note that the order of these boundaries may change by stress period, which must be accounted for in the Mover Package.) So the first well has an identifier of one. The second is two, and so forth. For the advanced packages, the identifier is the reach number (SFR Package), well number (MAW Package), or UZF cell number. For the Lake Package, ID1 is the lake outlet number. Thus, outflows from a single lake can be routed to different streams, for example. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *mname2* (string) name of model containing the package, PNAME2.
 - *pname2* (string) is the package name for the receiver. The package PNAME2 must be designated to receive water from the MVR Package by specifying the keyword "MOVER" in its OPTIONS block.
 - *id2* (integer) is the identifier for the receiver. The receiver identifier is the reach number (SFR Package), Lake number (LAK Package), well number (MAW Package), or UZF cell number. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *mvrtype* (string) is the character string signifying the method for determining how much water will be moved. Supported values are "FACTOR" "EXCESS" "THRESHOLD" and "UPTO". These four options determine how the receiver flow rate, Q_R , is calculated. These options mirror the options defined for the cprior variable in the SFR package, with the term "FACTOR" being functionally equivalent to the "FRACTION" option for cprior.
 - *value* (double) is the value to be used in the equation for

calculating the amount of water to move. For the “FACTOR” option, VALUE is the α factor. For the remaining options, VALUE is the specified flow rate, Q_S .

- `filename` (*String*) - File name for this package.
- `pname` (*String*) - Package name for this package.
- `parent_file` (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulaktab package must have a mfgwflak package parent_file.

`budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

`budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

```
dfn = [['header'], ['block options', 'name print_input', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name print_flows', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name modelnames', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name
budget_filerecord', 'type record budget fileout budgetfile', 'shape', 'reader
urword', 'tagged true', 'optional true'], ['block options', 'name budget', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name fileout', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
budgetfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budgetcsv_filerecord', 'type record budgetcsv fileout budgetcsvfile', 'shape',
'reader urword', 'tagged true', 'optional true'], ['block options', 'name
budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name budgetcsvfile', 'type string',
'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged false',
'optional false'], ['block dimensions', 'name maxmvr', 'type integer', 'reader
urword', 'optional false'], ['block dimensions', 'name maxpackages', 'type integer',
'reader urword', 'optional false'], ['block packages', 'name packages', 'type
recarray mname pname', 'reader urword', 'shape (npackages)', 'optional false'],
['block packages', 'name mname', 'type string', 'reader urword', 'shape', 'tagged
false', 'in_record true', 'optional true'], ['block packages', 'name pname', 'type
string', 'reader urword', 'shape', 'tagged false', 'in_record true', 'optional
false'], ['block period', 'name iper', 'type integer', 'block_variable True',
'in_record true', 'tagged false', 'shape', 'valid', 'reader urword', 'optional
false'], ['block period', 'name perioddata', 'type recarray mname1 pname1 id1 mname2
pname2 id2 mvrtype value', 'shape (maxbound)', 'reader urword'], ['block period',
'name mname1', 'type string', 'reader urword', 'shape', 'tagged false', 'in_record
true', 'optional true'], ['block period', 'name pname1', 'type string', 'shape',
'tagged false', 'in_record true', 'reader urword'], ['block period', 'name id1',
'type integer', 'shape', 'tagged false', 'in_record true', 'reader urword',
'numeric_index true'], ['block period', 'name mname2', 'type string', 'reader
urword', 'shape', 'tagged false', 'in_record true', 'optional true'], ['block
period', 'name pname2', 'type string', 'shape', 'tagged false', 'in_record true',
'reader urword'], ['block period', 'name id2', 'type integer', 'shape', 'tagged
false', 'in_record true', 'reader urword', 'numeric_index true'], ['block period',
'name mvrtype', 'type string', 'shape', 'tagged false', 'in_record true', 'reader
urword'], ['block period', 'name value', 'type double precision', 'shape', 'tagged
false', 'in_record true', 'reader urword']]
```

`dfn_file_name = 'gwf-mvr.dfn'`

```
package_abbrev = 'gwf6mvr'

packages = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

`flopy.mf6.modflow.mfgwfnam` module

```
class ModflowGwfnam(model, loading_package=False, list=None, print_input=None,
                    print_flows=None, save_flows=None, newtonoptions=None, packages=None,
                    filename=None, pname=None, **kwargs)
```

Bases: `MFPackage`

ModflowGwfnam defines a nam package within a gw6 model.

Parameters

- **model** (`MFModel`) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (`bool`) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **list** (`string`) -
 - list (`string`) is name of the listing file to create for this GWF model. If not specified, then the name of the list file will be the basename of the GWF model name file and the '.lst' extension. For example, if the GWF name file is called "my.model.nam" then the list file will be called "my.model.lst".
- **print_input** (`boolean`) -
 - print_input (`boolean`) keyword to indicate that the list of all model stress package information will be written to the listing file immediately after it is read.
- **print_flows** (`boolean`) -
 - print_flows (`boolean`) keyword to indicate that the list of all model package flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (`boolean`) -
 - save_flows (`boolean`) keyword to indicate that all model package flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **newtonoptions** (`[under_relaxation]`) -
 - under_relaxation (`string`) keyword that indicates whether the groundwater head in a cell will be under-relaxed when water levels fall below the bottom of the model below any given cell. By default, Newton-Raphson UNDER_RELAXATION is not applied.

- **packages** (*[ftype, fname, pname]*) -
 - *ftype* (string) is the file type, which must be one of the following character values shown in table in mf6io.pdf. *Ftype* may be entered in any combination of uppercase and lowercase.
 - *fname* (string) is the name of the file containing the package input. The path to the file should be included if the file is not located in the folder where the program was run.
 - *pname* (string) is the user-defined name for the package. PNAME is restricted to 16 characters. No spaces are allowed in PNAME. PNAME character values are read and stored by the program for stress packages only. These names may be useful for labeling purposes when multiple stress packages of the same type are located within a single GWF Model. If PNAME is specified for a stress package, then PNAME will be used in the flow budget table in the listing file; it will also be used for the text entry in the cell-by-cell budget file. PNAME is case insensitive and is stored in all upper case letters.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

```
dfn = [['header'], ['block options', 'name list', 'type string', 'reader urword',
'optional true', 'preserve_case true'], ['block options', 'name print_input', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name print_flows',
'type keyword', 'reader urword', 'optional true'], ['block options', 'name
save_flows', 'type keyword', 'reader urword', 'optional true'], ['block options',
'name newtonoptions', 'type record newton under_relaxation', 'reader urword',
'optional true'], ['block options', 'name newton', 'in_record true', 'type keyword',
'reader urword'], ['block options', 'name under_relaxation', 'in_record true', 'type
keyword', 'reader urword', 'optional true'], ['block packages', 'name packages',
'type recarray ftype fname pname', 'reader urword', 'optional false'], ['block
packages', 'name ftype', 'in_record true', 'type string', 'tagged false', 'reader
urword'], ['block packages', 'name fname', 'in_record true', 'type string',
'preserve_case true', 'tagged false', 'reader urword'], ['block packages', 'name
pname', 'in_record true', 'type string', 'tagged false', 'reader urword', 'optional
true']]
```

```
dfn_file_name = 'gwf-nam.dfn'
```

```
package_abbr = 'gwfnam'
```

```
packages = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

`flopy.mf6.modflow.mfgwfnpf` module

```
class ModflowGwfnpf(model, loading_package=False, save_flows=None, print_flows=None,
    alternative_cell_averaging=None, thickstrt=None, cvoptions=None,
    perched=None, rewet_record=None, xt3doptions=None,
    save_specific_discharge=None, save_saturation=None, k22overk=None,
    k33overk=None, perioddata=None, dev_no_newton=None,
    dev_modflowusg_upstream_weighted_saturation=None,
    dev_modflownwt_upstream_weighting=None,
    dev_minimum_saturated_thickness=None, dev_omega=None, icelltype=0,
    k=1.0, k22=None, k33=None, angle1=None, angle2=None, angle3=None,
    wetdry=None, filename=None, pname=None, **kwargs)
```

Bases: [`MFPackage`](#)

`ModflowGwfnpf` defines a npf package within a gw6 model.

Parameters

- **model** ([`MFModel`](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** ([`bool`](#)) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **save_flows** ([`boolean`](#)) -
 - `save_flows` ([`boolean`](#)) keyword to indicate that budget flow terms will be written to the file specified with “BUDGET SAVE FILE” in Output Control.
- **print_flows** ([`boolean`](#)) -
 - `print_flows` ([`boolean`](#)) keyword to indicate that calculated flows between cells will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period. This option can produce extremely large list files because all cell-by-cell flows are printed. It should only be used with the NPF Package for models that have a small number of cells.
- **alternative_cell_averaging** ([`string`](#)) -
 - `alternative_cell_averaging` ([`string`](#)) is a text keyword to indicate that an alternative method will be used for calculating the conductance for horizontal cell connections. The text value for `ALTERNATIVE_CELL_AVERAGING` can be “LOGARITHMIC”, “AMT-LMK”, or “AMT-HMK”. “AMT-LMK” signifies that the conductance will be calculated using arithmetic-mean thickness and logarithmic-mean hydraulic conductivity. “AMT-HMK” signifies that the conductance will be calculated using arithmetic-mean thickness and harmonic-mean hydraulic conductivity. If the user does not specify a value for `ALTERNATIVE_CELL_AVERAGING`, then the harmonic-mean method will be used. This option cannot be used if the XT3D option is invoked.
- **thickstrt** ([`boolean`](#)) -

- thickstrt (boolean) indicates that cells having a negative ICELLTYPE are confined, and their cell thickness for conductance calculations will be computed as STRT-BOT rather than TOP-BOT. This option should be used with caution as it only affects conductance calculations in the NPF Package.
- **cvoptions** ([*dewatered*]) -
 - dewatered (string) If the DEWATERED keyword is specified, then the vertical conductance is calculated using only the saturated thickness and properties of the overlying cell if the head in the underlying cell is below its top.
- **perched** (*boolean*) -
 - perched (boolean) keyword to indicate that when a cell is overlying a dewatered convertible cell, the head difference used in Darcy's Law is equal to the head in the overlying cell minus the bottom elevation of the overlying cell. If not specified, then the default is to use the head difference between the two cells.
- **rewet_record** ([*wetfct*, *iwetit*, *ihdwet*]) -
 - wetfct (double) is a keyword and factor that is included in the calculation of the head that is initially established at a cell when that cell is converted from dry to wet.
 - iwetit (integer) is a keyword and iteration interval for attempting to wet cells. Wetting is attempted every IWETIT iteration. This applies to outer iterations and not inner iterations. If IWETIT is specified as zero or less, then the value is changed to 1.
 - ihdwet (integer) is a keyword and integer flag that determines which equation is used to define the initial head at cells that become wet. If IHDWET is 0, $h = BOT + WETFCT (hm - BOT)$. If IHDWET is not 0, $h = BOT + WETFCT (THRESH)$.
- **xt3doptions** ([*rhs*]) -
 - rhs (string) If the RHS keyword is also included, then the XT3D additional terms will be added to the right-hand side. If the RHS keyword is excluded, then the XT3D terms will be put into the coefficient matrix.
- **save_specific_discharge** (*boolean*) -
 - save_specific_discharge (boolean) keyword to indicate that x, y, and z components of specific discharge will be calculated at cell centers and written to the budget file, which is specified with "BUDGET SAVE FILE" in Output Control. If this option is activated, then additional information may be required in the discretization packages and the GWF Exchange package (if GWF models are coupled). Specifically, ANGLDEGX must be specified in the CONNECTIONDATA block of the DISU Package; ANGLDEGX must also be specified for the GWF Exchange as an auxiliary variable.
- **save_saturation** (*boolean*) -

- **save_saturation** (boolean) keyword to indicate that cell saturation will be written to the budget file, which is specified with "BUDGET SAVE FILE" in Output Control. Saturation will be saved to the budget file as an auxiliary variable saved with the DATA-SAT text label. Saturation is a cell variable that ranges from zero to one and can be used by post processing programs to determine how much of a cell volume is saturated. If ICELLTYPE is 0, then saturation is always one.
- **k22overk** (boolean) -
 - **k22overk** (boolean) keyword to indicate that specified K22 is a ratio of K22 divided by K. If this option is specified, then the K22 array entered in the NPF Package will be multiplied by K after being read.
- **k33overk** (boolean) -
 - **k33overk** (boolean) keyword to indicate that specified K33 is a ratio of K33 divided by K. If this option is specified, then the K33 array entered in the NPF Package will be multiplied by K after being read.
- **perioddata** ({varname:data} or tvk_perioddata data) -
 - Contains data for the tvk package. Data can be stored in a dictionary containing data for the tvk package with variable names as keys and package data as values. Data just for the perioddata variable is also acceptable. See tvk package documentation for more information.
- **dev_no_newton** (boolean) -
 - **dev_no_newton** (boolean) turn off Newton for unconfined cells
- **dev_modflowusg_upstream_weighted_saturation** (boolean) -
 - **dev_modflowusg_upstream_weighted_saturation** (boolean) use MODFLOW-USG upstream-weighted saturation approach
- **dev_modflownwt_upstream_weighting** (boolean) -
 - **dev_modflownwt_upstream_weighting** (boolean) use MODFLOW-NWT approach for upstream weighting
- **dev_minimum_saturated_thickness** (double) -
 - **dev_minimum_saturated_thickness** (double) set minimum allowed saturated thickness
- **dev_omega** (double) -
 - **dev_omega** (double) set saturation omega value
- **icelltype** ([integer]) -
 - **icelltype** (integer) flag for each cell that specifies how saturated thickness is treated. 0 means saturated thickness is held constant; >0 means saturated thickness varies with computed head when head is below the cell top; <0 means saturated thickness varies with computed head unless the THICKSTRT option is in effect. When THICKSTRT is in effect,

a negative value for ICELLTYPE indicates that the saturated thickness value used in conductance calculations in the NPF Package will be computed as STRT-BOT and held constant. If the THICKSTRT option is not in effect, then negative values provided by the user for ICELLTYPE are automatically reassigned by the program to a value of one.

- **k** ([double]) -
 - k (double) is the hydraulic conductivity. For the common case in which the user would like to specify the horizontal hydraulic conductivity and the vertical hydraulic conductivity, then K should be assigned as the horizontal hydraulic conductivity, K33 should be assigned as the vertical hydraulic conductivity, and K22 and the three rotation angles should not be specified. When more sophisticated anisotropy is required, then K corresponds to the K11 hydraulic conductivity axis. All included cells (IDOMAIN > 0) must have a K value greater than zero.
- **k22** ([double]) -
 - k22 (double) is the hydraulic conductivity of the second ellipsoid axis (or the ratio of K22/K if the K22OVERK option is specified); for an unrotated case this is the hydraulic conductivity in the y direction. If K22 is not included in the GRIDDATA block, then K22 is set equal to K. For a regular MODFLOW grid (DIS Package is used) in which no rotation angles are specified, K22 is the hydraulic conductivity along columns in the y direction. For an unstructured DISU grid, the user must assign principal x and y axes and provide the angle for each cell face relative to the assigned x direction. All included cells (IDOMAIN > 0) must have a K22 value greater than zero.
- **k33** ([double]) -
 - k33 (double) is the hydraulic conductivity of the third ellipsoid axis (or the ratio of K33/K if the K33OVERK option is specified); for an unrotated case, this is the vertical hydraulic conductivity. When anisotropy is applied, K33 corresponds to the K33 tensor component. All included cells (IDOMAIN > 0) must have a K33 value greater than zero.
- **angle1** ([double]) -
 - angle1 (double) is a rotation angle of the hydraulic conductivity tensor in degrees. The angle represents the first of three sequential rotations of the hydraulic conductivity ellipsoid. With the K11, K22, and K33 axes of the ellipsoid initially aligned with the x, y, and z coordinate axes, respectively, ANGLE1 rotates the ellipsoid about its K33 axis (within the x - y plane). A positive value represents counter-clockwise rotation when viewed from any point on the positive K33 axis, looking toward the center of the ellipsoid. A value of zero indicates that the K11 axis lies within the x - z plane. If ANGLE1 is not specified, default values of zero are assigned to ANGLE1, ANGLE2, and

ANGLE3, in which case the K11, K22, and K33 axes are aligned with the x, y, and z axes, respectively.

- **angle2** (*[double]*) -
 - angle2 (double) is a rotation angle of the hydraulic conductivity tensor in degrees. The angle represents the second of three sequential rotations of the hydraulic conductivity ellipsoid. Following the rotation by ANGLE1 described above, ANGLE2 rotates the ellipsoid about its K22 axis (out of the x - y plane). An array can be specified for ANGLE2 only if ANGLE1 is also specified. A positive value of ANGLE2 represents clockwise rotation when viewed from any point on the positive K22 axis, looking toward the center of the ellipsoid. A value of zero indicates that the K11 axis lies within the x - y plane. If ANGLE2 is not specified, default values of zero are assigned to ANGLE2 and ANGLE3; connections that are not user- designated as vertical are assumed to be strictly horizontal (that is, to have no z component to their orientation); and connection lengths are based on horizontal distances.
- **angle3** (*[double]*) -
 - angle3 (double) is a rotation angle of the hydraulic conductivity tensor in degrees. The angle represents the third of three sequential rotations of the hydraulic conductivity ellipsoid. Following the rotations by ANGLE1 and ANGLE2 described above, ANGLE3 rotates the ellipsoid about its K11 axis. An array can be specified for ANGLE3 only if ANGLE1 and ANGLE2 are also specified. An array must be specified for ANGLE3 if ANGLE2 is specified. A positive value of ANGLE3 represents clockwise rotation when viewed from any point on the positive K11 axis, looking toward the center of the ellipsoid. A value of zero indicates that the K22 axis lies within the x - y plane.
- **wetdry** (*[double]*) -
 - wetdry (double) is a combination of the wetting threshold and a flag to indicate which neighboring cells can cause a cell to become wet. If WETDRY < 0, only a cell below a dry cell can cause the cell to become wet. If WETDRY > 0, the cell below a dry cell and horizontally adjacent cells can cause a cell to become wet. If WETDRY is 0, the cell cannot be wetted. The absolute value of WETDRY is the wetting threshold. When the sum of BOT and the absolute value of WETDRY at a dry cell is equalled or exceeded by the head at an adjacent cell, the cell is wetted. WETDRY must be specified if "REWET" is specified in the OPTIONS block. If "REWET" is not specified in the options block, then WETDRY can be entered, and memory will be allocated for it, even though it is not used.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this

```
package. Only needed for utility packages (mfutl*). For example,  
mfutllaktab package must have a mfgwflak package parent_file.  
angle1 = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>  
angle2 = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>  
angle3 = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```

dfn = [['header'], ['block options', 'name save_flows', 'type keyword', 'reader
urword', 'optional true', 'mf6internal ipakcb'], ['block options', 'name
print_flows', 'type keyword', 'reader urword', 'optional true', 'mf6internal
iprflow'], ['block options', 'name alternative_cell_averaging', 'type string',
'valid logarithmic amt-lmk amt-hmk', 'reader urword', 'optional true', 'mf6internal
cellavg'], ['block options', 'name thickstrt', 'type keyword', 'reader urword',
'optional true', 'mf6internal ithickstrt'], ['block options', 'name cvoptions',
'type record variablecv dewatered', 'reader urword', 'optional true'], ['block
options', 'name variablecv', 'in_record true', 'type keyword', 'reader urword',
'mf6internal ivarcv'], ['block options', 'name dewatered', 'in_record true', 'type
keyword', 'reader urword', 'optional true', 'mf6internal idewatcv'], ['block
options', 'name perched', 'type keyword', 'reader urword', 'optional true',
'mf6internal iperched'], ['block options', 'name rewet_record', 'type record rewet
wetfct iwetit ihdwet', 'reader urword', 'optional true'], ['block options', 'name
rewet', 'type keyword', 'in_record true', 'reader urword', 'optional false',
'mf6internal irewet'], ['block options', 'name wetfct', 'type double precision',
'in_record true', 'reader urword', 'optional false'], ['block options', 'name
iwetit', 'type integer', 'in_record true', 'reader urword', 'optional false'],
['block options', 'name ihdwet', 'type integer', 'in_record true', 'reader urword',
'optional false'], ['block options', 'name xt3doptions', 'type record xt3d rhs',
'reader urword', 'optional true'], ['block options', 'name xt3d', 'in_record true',
'type keyword', 'reader urword', 'mf6internal ixt3d'], ['block options', 'name rhs',
'in_record true', 'type keyword', 'reader urword', 'optional true', 'mf6internal
ixt3drhs'], ['block options', 'name save_specific_discharge', 'type keyword',
'reader urword', 'optional true', 'mf6internal isavspdis'], ['block options', 'name
save_saturation', 'type keyword', 'reader urword', 'optional true', 'mf6internal
isavsat'], ['block options', 'name k22overk', 'type keyword', 'reader urword',
'optional true', 'mf6internal ik22overk'], ['block options', 'name k33overk', 'type
keyword', 'reader urword', 'optional true', 'mf6internal ik33overk'], ['block
options', 'name tvk_filerecord', 'type record tvk6 filein tvk6_filename', 'shape',
'reader urword', 'tagged true', 'optional true', 'construct_package tvk',
'construct_data tvk_perioddata', 'parameter_name perioddata'], ['block options',
'name tvk6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name filein', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name tvk6_filename', 'type string', 'preserve_case true', 'in_record
true', 'reader urword', 'optional false', 'tagged false'], ['block options', 'name
dev_no_newton', 'type keyword', 'reader urword', 'optional true', 'mf6internal
inewton'], ['block options', 'name dev_modflowusg_upstream_weighted_saturation',
'type keyword', 'reader urword', 'optional true', 'mf6internal iusgnrhc'], ['block
options', 'name dev_modflownwt_upstream_weighting', 'type keyword', 'reader urword',
'optional true', 'mf6internal inwtupw'], ['block options', 'name
dev_minimum_saturated_thickness', 'type double precision', 'reader urword',
'optional true', 'mf6internal satmin'], ['block options', 'name dev_omega', 'type
double precision', 'reader urword', 'optional true', 'mf6internal satomega'],
['block griddata', 'name icelltype', 'type integer', 'shape (nodes)', 'valid',
'reader readarray', 'layered true', 'optional', 'default_value 0'], ['block
griddata', 'name k', 'type double precision', 'shape (nodes)', 'valid', 'reader
readarray', 'layered true', 'optional', 'default_value 1.0'], ['block griddata',
'name k22', 'type double precision', 'shape (nodes)', 'valid', 'reader readarray',
'layered true', 'optional true'], ['block griddata', 'name k33', 'type double
precision', 'shape (nodes)', 'valid', 'reader readarray', 'layered true', 'optional
true'], ['block griddata', 'name angle1', 'type double precision', 'shape (nodes)',
'valid', 'reader readarray', 'layered true', 'optional true'], ['block griddata',
'name angle2', 'type double precision', 'shape (nodes)', 'valid', 'reader
readarray', 'layered true', 'optional true'], ['block griddata', 'name angle3',
'type double precision', 'shape (nodes)', 'valid', 'reader readarray', 'layered
true', 'optional true'], ['block griddata', 'name wetdry', 'type double precision',
'shape (nodes)', 'valid', 'reader readarray', 'layered true', 'optional true']]

```

```

dfn_file_name = 'gwf-npf.dfn'

icelltype = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

k = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

k22 = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

k33 = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

package_abbr = 'gwfnpf'

rewet_record = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

tvk_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

wetdry = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

```

flopy.mf6.modflow.mfgwfoc module

```

class ModflowGwfoc(model, loading_package=False, budget_filerecord=None,
                    budgetcsv_filerecord=None, head_filerecord=None, headprintrecord=None,
                    saverecord=None, printrecord=None, filename=None, pname=None,
                    **kwargs)

```

Bases: [MFPackage](#)

ModflowGwfoc defines a oc package within a gwf6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **budget_filerecord** (*[budgetfile]*) -
 - budgetfile (string) name of the output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
 - budgetcsvfile (string) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **head_filerecord** (*[headfile]*) -
 - headfile (string) name of the output file to write head information.
- **headprintrecord** (*[columns, width, digits, format]*) -
 - columns (integer) number of columns for writing data.
 - width (integer) width for writing each number.
 - digits (integer) number of digits to use for writing a number.

- format (string) write format can be EXPONENTIAL, FIXED, GENERAL, or SCIENTIFIC.
- **saverecord** ([*rtype*, *ocsetting*]) -
 - *rtype* (string) type of information to save or print. Can be BUDGET or HEAD.
 - *ocsetting* (keysting) specifies the steps for which the data will be saved.
 - all**
 - [[keyword]]
 - * all (keyword) keyword to indicate save for all time steps in period.
 - first**
 - [[keyword]]
 - * first (keyword) keyword to indicate save for first step in period. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.
 - last**
 - [[keyword]]
 - * last (keyword) keyword to indicate save for last step in period. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.
 - frequency**
 - [[integer]]
 - * frequency (integer) save at the specified time step frequency. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.
 - steps**
 - [[integer]]
 - * steps (integer) save for each step specified in STEPS. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.
- **printrecord** ([*rtype*, *ocsetting*]) -
 - *rtype* (string) type of information to save or print. Can be BUDGET or HEAD.
 - *ocsetting* (keysting) specifies the steps for which the data will be saved.
 - all**
 - [[keyword]]
 - * all (keyword) keyword to indicate save for all time steps in period.

first

[[keyword]]

* first (keyword) keyword to indicate save for first step in period. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

last

[[keyword]]

* last (keyword) keyword to indicate save for last step in period. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

frequency

[[integer]]

* frequency (integer) save at the specified time step frequency. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

steps

[[integer]]

* steps (integer) save for each step specified in STEPS. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfullaktab package must have a mfgwflak package parent_file.

budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```
dfn = [['header'], ['block options', 'name budget_filerecord', 'type record budget
fileout budgetfile', 'shape', 'reader urword', 'tagged true', 'optional true'],
['block options', 'name budget', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name fileout', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name budgetfile', 'type string', 'preserve_case true',
'shape', 'in_record true', 'reader urword', 'tagged false', 'optional false'],
['block options', 'name budgetcsv_filerecord', 'type record budgetcsv fileout
budgetcsvfile', 'shape', 'reader urword', 'tagged true', 'optional true'], ['block
options', 'name budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name budgetcsvfile',
'type string', 'preserve_case true', 'shape', 'in_record true', 'reader urword',
'tagged false', 'optional false'], ['block options', 'name head_filerecord', 'type
record head fileout headfile', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name head', 'type keyword', 'shape', 'in_record true',
'reader urword', 'tagged true', 'optional false'], ['block options', 'name
headfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
headprintrecord', 'type record head print_format formatrecord', 'shape', 'reader
urword', 'optional true'], ['block options', 'name print_format', 'type keyword',
'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'],
['block options', 'name formatrecord', 'type record columns width digits format',
'shape', 'in_record true', 'reader urword', 'tagged', 'optional false'], ['block
options', 'name columns', 'type integer', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional'], ['block options', 'name width', 'type integer',
'shape', 'in_record true', 'reader urword', 'tagged true', 'optional'], ['block
options', 'name digits', 'type integer', 'shape', 'in_record true', 'reader urword',
'tagged true', 'optional'], ['block options', 'name format', 'type string', 'shape',
'in_record true', 'reader urword', 'tagged false', 'optional false'], ['block
period', 'name iper', 'type integer', 'block_variable True', 'in_record true',
'tagged false', 'shape', 'valid', 'reader urword', 'optional false'], ['block
period', 'name saverecord', 'type record save rtype ocsetting', 'shape', 'reader
urword', 'tagged false', 'optional true'], ['block period', 'name save', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block period', 'name printrecord', 'type record print rtype ocsetting',
'shape', 'reader urword', 'tagged false', 'optional true'], ['block period', 'name
print', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block period', 'name rtype', 'type string', 'shape', 'in_record
true', 'reader urword', 'tagged false', 'optional false'], ['block period', 'name
ocsetting', 'type keystring all first last frequency steps', 'shape', 'tagged
false', 'in_record true', 'reader urword'], ['block period', 'name all', 'type
keyword', 'shape', 'in_record true', 'reader urword'], ['block period', 'name
first', 'type keyword', 'shape', 'in_record true', 'reader urword'], ['block
period', 'name last', 'type keyword', 'shape', 'in_record true', 'reader urword'],
['block period', 'name frequency', 'type integer', 'shape', 'tagged true',
'in_record true', 'reader urword'], ['block period', 'name steps', 'type integer',
'shape (<nstp)', 'tagged true', 'in_record true', 'reader urword']]

dfn_file_name = 'gwf-oc.dfn'

head_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

headprintrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfoc'
```



```
printrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
saverecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

`flopy.mf6.modflow.mfgwfrch` module

```
class ModflowGwfrch(model, loading_package=False, fixed_cell=None, auxiliary=None,
                    auxmultname=None, boundnames=None, print_input=None,
                    print_flows=None, save_flows=None, timeseries=None,
                    observations=None, maxbound=None, stress_period_data=None,
                    filename=None, pname=None, **kwargs)
```

Bases: `MFPackage`

ModflowGwfrch defines a rch package within a gwf6 model.

Parameters

- **model** (`MFMModel`) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **fixed_cell** (*boolean*) -
 - `fixed_cell` (*boolean*) indicates that recharge will not be reassigned to a cell underlying the cell specified in the list if the specified cell is inactive.
- **auxiliary** (*[string]*) -
 - `auxiliary` (*string*) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for `naux`. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** (*string*) -
 - `auxmultname` (*string*) name of auxiliary variable to be used as multiplier of recharge.
- **boundnames** (*boolean*) -
 - `boundnames` (*boolean*) keyword to indicate that boundary names may be provided with the list of recharge cells.
- **print_input** (*boolean*) -
 - `print_input` (*boolean*) keyword to indicate that the list of recharge information will be written to the listing file immediately after it is read.

- **print_flows** (*boolean*) -
 - print_flows (*boolean*) keyword to indicate that the list of recharge flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (*boolean*) keyword to indicate that recharge flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **timeseries** (*{varname:data} or timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data} or continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **maxbound** (*integer*) -
 - maxbound (*integer*) integer value specifying the maximum number of recharge cells that will be specified for use during any stress period.
- **stress_period_data** (*[cellid, recharge, aux, boundname]*) -
 - cellid (*(integer, ...)*) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - recharge (*double*) is the recharge flux rate (LT^{-1}). This rate is multiplied inside the program by the surface area of the cell to calculate the volumetric recharge rate. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- aux (double) represents the values of the auxiliary variables for each recharge. The values of auxiliary variables must be present for each recharge. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- boundname (string) name of the recharge cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfuillaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```
dfn = [['header', 'multi-package'], ['block options', 'name fixed_cell', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name auxmultname', 'type string', 'shape', 'reader urword', 'optional true'], ['block options', 'name boundnames', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block options', 'name print_input', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name print_flows', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name save_flows', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name ts_filerecord', 'type record ts6 filein ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package ts', 'construct_data timeseries', 'parameter_name timeseries'], ['block options', 'name ts6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name ts6_filename', 'type string', 'preserve_case true', 'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block options', 'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package obs', 'construct_data continuous', 'parameter_name observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged false', 'reader urword', 'optional false'], ['block dimensions', 'name maxbound', 'type integer', 'reader urword', 'optional false'], ['block period', 'name iper', 'type integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader urword', 'optional false'], ['block period', 'name stress_period_data', 'type recarray cellid recharge aux boundname', 'shape (maxbound)', 'reader urword'], ['block period', 'name cellid', 'type integer', 'shape (ncelldim)', 'tagged false', 'in_record true', 'reader urword'], ['block period', 'name recharge', 'type double precision', 'shape', 'tagged false', 'in_record true', 'reader urword', 'time_series true'], ['block period', 'name aux', 'type double precision', 'in_record true', 'tagged false', 'shape (naux)', 'reader urword', 'optional true', 'time_series true'], ['block period', 'name boundname', 'type string', 'shape', 'tagged false', 'in_record true', 'reader urword', 'optional true']]]
```

```
dfn_file_name = 'gwf-rch.dfn'
```

```
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'gwf-rch'
```

```
stress_period_data = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfrcha module

```
class ModflowGwfrcha(model, loading_package=False, readasarrays=True, fixed_cell=None,
                      auxiliary=None, auxmultname=None, print_input=None,
                      print_flows=None, save_flows=None, timearrayseries=None,
                      observations=None, irch=None, recharge=0.001, aux=None,
                      filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwfrcha defines a rcha package within a gw6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** ([bool](#)) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **readasarrays** ([boolean](#)) -
 - readasarrays ([boolean](#)) indicates that array-based input will be used for the Recharge Package. This keyword must be specified to use array-based input.
- **fixed_cell** ([boolean](#)) -
 - fixed_cell ([boolean](#)) indicates that recharge will not be reassigned to a cell underlying the cell specified in the list if the specified cell is inactive.
- **auxiliary** ([\[string\]](#)) -
 - auxiliary ([string](#)) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** ([string](#)) -
 - auxmultname ([string](#)) name of auxiliary variable to be used as multiplier of recharge.
- **print_input** ([boolean](#)) -
 - print_input ([boolean](#)) keyword to indicate that the list of recharge information will be written to the listing file immediately after it is read.
- **print_flows** ([boolean](#)) -
 - print_flows ([boolean](#)) keyword to indicate that the list of recharge flow rates will be printed to the listing file for

- every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (boolean) keyword to indicate that recharge flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
 - **timearrayseries** (*{varname:data}* or *tas_array data*) -
 - Contains data for the tas package. Data can be stored in a dictionary containing data for the tas package with variable names as keys and package data as values. Data just for the timearrayseries variable is also acceptable. See tas package documentation for more information.
 - **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
 - **irch** (*[integer]*) -
 - irch (integer) IRCH is the layer number that defines the layer in each vertical column where recharge is applied. If IRCH is omitted, recharge by default is applied to cells in layer 1. IRCH can only be used if READASARRAYS is specified in the OPTIONS block. If IRCH is specified, it must be specified as the first variable in the PERIOD block or MODFLOW will terminate with an error. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - **recharge** (*[double]*) -
 - recharge (double) is the recharge flux rate (LT^{-1}). This rate is multiplied inside the program by the surface area of the cell to calculate the volumetric recharge rate. The recharge array may be defined by a time-array series (see the “Using Time-Array Series in a Package” section).
 - **aux** (*[double]*) -
 - aux (double) is an array of values for auxiliary variable aux(iaux), where iaux is a value from 1 to naux, and aux(iaux) must be listed as part of the auxiliary variables. A separate array can be specified for each auxiliary variable. If an array is not specified for an auxiliary variable, then a value of zero is assigned. If the value specified here for the auxiliary variable is the same as auxmultname, then the recharge array will be multiplied by this array.

- `filename` (*String*) - File name for this package.
- `pname` (*String*) - Package name for this package.
- `parent_file` (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package `parent_file`.

```
aux = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['header', 'multi-package'], ['block options', 'name readasarrays', 'type keyword', 'shape', 'reader urword', 'optional false', 'default_value True'], ['block options', 'name fixed_cell', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name auxmultname', 'type string', 'shape', 'reader urword', 'optional true'], ['block options', 'name print_input', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name print_flows', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name save_flows', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name tas_filerecord', 'type record tas6 filein tas6_filename', 'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package tas', 'construct_data tas_array', 'parameter_name timearrayseries'], ['block options', 'name tas6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name tas6_filename', 'type string', 'preserve_case true', 'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block options', 'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package obs', 'construct_data continuous', 'parameter_name observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged false', 'reader urword', 'optional false'], ['block period', 'name iper', 'type integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader urword', 'optional false'], ['block period', 'name irch', 'type integer', 'shape (ncol*nrow; ncpl)', 'reader readarray', 'numeric_index true', 'optional true'], ['block period', 'name recharge', 'type double precision', 'shape (ncol*nrow; ncpl)', 'reader readarray', 'default_value 1.e-3'], ['block period', 'name aux', 'type double precision', 'shape (ncol*nrow; ncpl)', 'reader readarray', 'optional true']]

dfn_file_name = 'gwf-rcha.dfn'

irch = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwf-rcha'

recharge = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

tas_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfriv module

```
class ModflowGwfriv(model, loading_package=False, auxiliary=None, auxmultname=None,
                    boundnames=None, print_input=None, print_flows=None, save_flows=None,
                    timeseries=None, observations=None, mover=None, maxbound=None,
                    stress_period_data=None, filename=None, pname=None, **kwargs)
```

Bases: [*MFPackage*](#)

ModflowGwfriv defines a riv package within a gwf6 model.

Parameters

- **model** ([*MFModel*](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **auxiliary** (*[string]*) -
 - auxiliary (*string*) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** (*string*) -
 - auxmultname (*string*) name of auxiliary variable to be used as multiplier of riverbed conductance.
- **boundnames** (*boolean*) -
 - boundnames (*boolean*) keyword to indicate that boundary names may be provided with the list of river cells.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of river information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (*boolean*) keyword to indicate that the list of river flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -

- `save_flows` (boolean) keyword to indicate that river flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **timeseries** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **mover** (*boolean*) -
 - `mover` (boolean) keyword to indicate that this instance of the River Package can be used with the Water Mover (MVR) Package. When the MOVER option is specified, additional memory is allocated within the package to store the available, provided, and received water.
- **maxbound** (*integer*) -
 - `maxbound` (integer) integer value specifying the maximum number of rivers cells that will be specified for use during any stress period.
- **stress_period_data** (*[cellid, stage, cond, rbot, aux, boundname]*) -
 - `cellid` ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - `stage` (double) is the head in the river. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - `cond` (double) is the riverbed hydraulic conductance. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- rbot (double) is the elevation of the bottom of the riverbed. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - aux (double) represents the values of the auxiliary variables for each river. The values of auxiliary variables must be present for each river. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - boundname (string) name of the river cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.
- **filename** (*String*) - File name for this package.
 - **pname** (*String*) - Package name for this package.
 - **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.
- auxiliary** = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

dfn = [['header', 'multi-package'], ['block options', 'name auxiliary', 'type
string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name
auxmultname', 'type string', 'shape', 'reader urword', 'optional true'], ['block
options', 'name boundnames', 'type keyword', 'shape', 'reader urword', 'optional
true'], ['block options', 'name print_input', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name print_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name save_flows', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name ts_filerecord', 'type
record ts6 filein ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional
true', 'construct_package ts', 'construct_data timeseries', 'parameter_name
timeseries'], ['block options', 'name ts6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
filein', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name ts6_filename', 'type string',
'preserve_case true', 'in_record true', 'reader urword', 'optional false', 'tagged
false'], ['block options', 'name obs_filerecord', 'type record obs6 filein
obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package obs', 'construct_data continuous', 'parameter_name
observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block options', 'name mover', 'type
keyword', 'tagged true', 'reader urword', 'optional true'], ['block dimensions',
'name maxbound', 'type integer', 'reader urword', 'optional false'], ['block
period', 'name iper', 'type integer', 'block_variable True', 'in_record true',
'tagged false', 'shape', 'valid', 'reader urword', 'optional false'], ['block
period', 'name stress_period_data', 'type recarray cellid stage cond rbot aux
boundname', 'shape (maxbound)', 'reader urword'], ['block period', 'name cellid',
'type integer', 'shape (ncellid)', 'tagged false', 'in_record true', 'reader
urword'], ['block period', 'name stage', 'type double precision', 'shape', 'tagged
false', 'in_record true', 'reader urword', 'time_series true'], ['block period',
'name cond', 'type double precision', 'shape', 'tagged false', 'in_record true',
'reader urword', 'time_series true'], ['block period', 'name rbot', 'type double
precision', 'shape', 'tagged false', 'in_record true', 'reader urword', 'time_series
true'], ['block period', 'name aux', 'type double precision', 'in_record true',
'tagged false', 'shape (naux)', 'reader urword', 'optional true', 'time_series
true'], ['block period', 'name boundname', 'type string', 'shape', 'tagged false',
'in_record true', 'reader urword', 'optional true']]

```

```
dfn_file_name = 'gwf-riv.dfn'
```

```
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'gwf-riv'
```

```
stress_period_data = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfsfr module

```
class ModflowGwfsfr(model, loading_package=False, auxiliary=None, boundnames=None,
    print_input=None, print_stage=None, print_flows=None,
    save_flows=None, stage_filerecord=None, budget_filerecord=None,
    budgetcsv_filerecord=None, package_convergence_filerecord=None,
    timeseries=None, observations=None, mover=None,
    maximum_picard_iterations=None, maximum_iterations=None,
    maximum_depth_change=None, unit_conversion=None,
    length_conversion=None, time_conversion=None, nreaches=None,
    packagedata=None, crosssections=None, connectiondata=None,
    diversions=None, perioddata=None, filename=None, pname=None,
    **kwargs)
```

Bases: [MFPackage](#)

ModflowGwfsfr defines a sfr package within a gwf6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **auxiliary** (*[string]*) -
 - auxiliary (*string*) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **boundnames** (*boolean*) -
 - boundnames (*boolean*) keyword to indicate that boundary names may be provided with the list of stream reach cells.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of stream reach information will be written to the listing file immediately after it is read.
- **print_stage** (*boolean*) -
 - print_stage (*boolean*) keyword to indicate that the list of stream reach stages will be printed to the listing file for every stress period in which "HEAD PRINT" is specified in Output Control. If there is no Output Control option and PRINT_STAGE is specified, then stages are printed for the last time step of each stress period.

- **print_flows** (*boolean*) -
 - print_flows (*boolean*) keyword to indicate that the list of stream reach flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (*boolean*) keyword to indicate that stream reach flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **stage_filerecord** (*[stagefile]*) -
 - stagefile (*string*) name of the binary output file to write stage information.
- **budget_filerecord** (*[budgetfile]*) -
 - budgetfile (*string*) name of the binary output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
 - budgetcsvfile (*string*) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **package_convergence_filerecord** (*[package_convergence_filename]*) -
 - package_convergence_filename (*string*) name of the comma spaced values output file to write package convergence information.
- **timeseries** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **mover** (*boolean*) -
 - mover (*boolean*) keyword to indicate that this instance of the SFR Package can be used with the Water Mover (MVR) Package. When the MOVER option is specified, additional memory is allocated within the package to store the available, provided, and received water.
- **maximum_picard_iterations** (*integer*) -

- `maximum_picard_iterations` (integer) integer value that defines the maximum number of Streamflow Routing picard iterations allowed when solving for reach stages and flows as part of the GWF formulate step. Picard iterations are used to minimize differences in SFR package results between subsequent GWF picard (non-linear) iterations as a result of non-optimal reach numbering. If reaches are numbered in order, from upstream to downstream, `MAXIMUM_PICARD_ITERATIONS` can be set to 1 to reduce model run time. By default, `MAXIMUM_PICARD_ITERATIONS` is equal to 100.
- **`maximum_iterations` (integer) -**
 - `maximum_iterations` (integer) integer value that defines the maximum number of Streamflow Routing Newton-Raphson iterations allowed for a reach. By default, `MAXIMUM_ITERATIONS` is equal to 100. `MAXIMUM_ITERATIONS` would only need to be increased from the default value if one or more reach in a simulation has a large water budget error.
- **`maximum_depth_change` (double) -**
 - `maximum_depth_change` (double) real value that defines the depth closure tolerance. By default, `MAXIMUM_DEPTH_CHANGE` is equal to 1×10^{-5} . The `MAXIMUM_STAGE_CHANGE` would only need to be increased or decreased from the default value if the water budget error for one or more reach is too small or too large, respectively.
- **`unit_conversion` (double) -**
 - `unit_conversion` (double) real value that is used to convert user- specified Manning's roughness coefficients from seconds per meters:math:^{1/3} to model length and time units. A constant of 1.486 is used for flow units of cubic feet per second, and a constant of 1.0 is used for units of cubic meters per second. The constant must be multiplied by 86,400 when using time units of days in the simulation.
- **`length_conversion` (double) -**
 - `length_conversion` (double) real value that is used to convert user- specified Manning's roughness coefficients from meters to model length units. `LENGTH_CONVERSION` should be set to 3.28081, 1.0, and 100.0 when using length units (`LENGTH_UNITS`) of feet, meters, or centimeters in the simulation, respectively. `LENGTH_CONVERSION` does not need to be specified if `LENGTH_UNITS` are meters.
- **`time_conversion` (double) -**
 - `time_conversion` (double) real value that is used to convert user- specified Manning's roughness coefficients from seconds to model time units. `TIME_CONVERSION` should be set to 1.0, 60.0, 3,600.0, 86,400.0, and 31,557,600.0 when using time units (`TIME_UNITS`) of seconds, minutes, hours, days, or years in the simulation, respectively. `TIME_CONVERSION` does not need to be specified if `TIME_UNITS` are seconds.

- **nreaches** (*integer*) -
 - **nreaches** (*integer*) integer value specifying the number of stream reaches. There must be NREACHES entries in the PACKAGEDATA block.
- **packagedata** (*[rno, cellid, rlen, rwid, rgrd, rtp, rbth, rhk, man, ncon,)* -
 - ustrf, ndv, aux, boundname]**
 - **rno** (*integer*) integer value that defines the reach number associated with the specified PACKAGEDATA data on the line. RNO must be greater than zero and less than or equal to NREACHES. Reach information must be specified for every reach or the program will terminate with an error. The program will also terminate with an error if information for a reach is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - **cellid** (*((integer, ...))*) The keyword 'NONE' must be specified for reaches that are not connected to an underlying GWF cell. The keyword 'NONE' is used for reaches that are in cells that have IDOMAIN values less than one or are in areas not covered by the GWF model grid. Reach-aquifer flow is not calculated if the keyword 'NONE' is specified. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - **rlen** (*double*) real value that defines the reach length. RLEN must be greater than zero.
 - **rwid** (*double*) real value that defines the reach width. RWID must be greater than zero.
 - **rgrd** (*double*) real value that defines the stream gradient (slope) across the reach. RGRD must be greater than zero.
 - **rtp** (*double*) real value that defines the bottom elevation of the reach.
 - **rbth** (*double*) real value that defines the thickness of the reach streambed. RBTH can be any value if CELLID is 'NONE'. Otherwise, RBTH must be greater than zero.
 - **rhk** (*double*) real value that defines the hydraulic conductivity of the reach streambed. RHK can be any positive value if CELLID is 'NONE'. Otherwise, RHK must be greater than zero.
 - **man** (*string*) real or character value that defines the Manning's roughness coefficient for the reach. MAN must be greater than zero. If the Options block includes a TIMESERIESFILE entry (see the "Time- Variable Input"

section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- `ncon` (integer) integer value that defines the number of reaches connected to the reach. If a value of zero is specified for `NCON` an entry for `RNO` is still required in the subsequent `CONNECTIONDATA` block.
 - `ustrf` (double) real value that defines the fraction of upstream flow from each upstream reach that is applied as upstream inflow to the reach. The sum of all `USTRF` values for all reaches connected to the same upstream reach must be equal to one and `USTRF` must be greater than or equal to zero. If the Options block includes a `TIMESERIESFILE` entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - `ndv` (integer) integer value that defines the number of downstream diversions for the reach.
 - `aux` (double) represents the values of the auxiliary variables for each stream reach. The values of auxiliary variables must be present for each stream reach. The values must be specified in the order of the auxiliary variables specified in the `OPTIONS` block. If the package supports time series and the Options block includes a `TIMESERIESFILE` entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - `boundname` (string) name of the stream reach cell. `BOUNDNAME` is an ASCII character variable that can contain as many as 40 characters. If `BOUNDNAME` contains spaces in it, then the entire name must be enclosed within single quotes.
- **`crosssections`** (`[rno, tab6_filename]`) -
 - `rno` (integer) integer value that defines the reach number associated with the specified cross-section table file on the line. `RNO` must be greater than zero and less than or equal to `NREACHES`. The program will also terminate with an error if table information for a reach is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - `tab6_filename` (string) character string that defines the path and filename for the file containing cross-section table data for the reach. The `TAB6_FILENAME` file includes the number of entries in the file and the station elevation data in terms of the fractional width and the reach depth. Instructions for creating the `TAB6_FILENAME` input file are provided in `SFR Reach Cross-Section Table Input File` section.
 - **`connectiondata`** (`[rno, ic]`) -

- `rno` (integer) integer value that defines the reach number associated with the specified CONNECTIONDATA data on the line. RNO must be greater than zero and less than or equal to NREACHES. Reach connection information must be specified for every reach or the program will terminate with an error. The program will also terminate with an error if connection information for a reach is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
- `ic` (double_precision) integer value that defines the reach number of the reach connected to the current reach and whether it is connected to the upstream or downstream end of the reach. Negative IC numbers indicate connected reaches are connected to the downstream end of the current reach. Positive IC numbers indicate connected reaches are connected to the upstream end of the current reach. The absolute value of IC must be greater than zero and less than or equal to NREACHES. IC should not be specified when NCON is zero but must be specified otherwise. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
- **diversions** (`[rno, idv, iconr, cprior]`) -
 - `rno` (integer) integer value that defines the reach number associated with the specified DIVERSIONS data on the line. RNO must be greater than zero and less than or equal to NREACHES. Reach diversion information must be specified for every reach with a NDV value greater than 0 or the program will terminate with an error. The program will also terminate with an error if diversion information for a given reach diversion is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - `idv` (integer) integer value that defines the downstream diversion number for the diversion for reach RNO. IDV must be greater than zero and less than or equal to NDV for reach RNO. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - `iconr` (integer) integer value that defines the downstream reach that will receive the diverted water. IDV must be greater than zero and less than or equal to NREACHES. Furthermore, reach ICONR must be a downstream connection for reach RNO. This argument is an index variable, which means that it should be treated as zero-based when working with

FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.

- `cprior` (string) character string value that defines the the prioritization system for the diversion, such as when insufficient water is available to meet all diversion stipulations, and is used in conjunction with the value of `FLOW` value specified in the `STRESS_PERIOD_DATA` section. Available diversion options include: (1) `CPRIOR` = 'FRACTION', then the amount of the diversion is computed as a fraction of the streamflow leaving reach `RNO` (Q_{DS}); in this case, $0.0 \leq \text{DIVFLOW} \leq 1.0$. (2) `CPRIOR` = 'EXCESS', a diversion is made only if Q_{DS} for reach `RNO` exceeds the value of `DIVFLOW`. If this occurs, then the quantity of water diverted is the excess flow ($Q_{DS} - \text{DIVFLOW}$) and Q_{DS} from reach `RNO` is set equal to `DIVFLOW`. This represents a flood-control type of diversion, as described by Danskin and Hanson (2002). (3) `CPRIOR` = 'THRESHOLD', then if Q_{DS} in reach `RNO` is less than the specified diversion flow `DIVFLOW`, no water is diverted from reach `RNO`. If Q_{DS} in reach `RNO` is greater than or equal to `DIVFLOW`, `DIVFLOW` is diverted and Q_{DS} is set to the remainder ($Q_{DS} - \text{DIVFLOW}$). This approach assumes that once flow in the stream is sufficiently low, diversions from the stream cease, and is the 'priority' algorithm that originally was programmed into the STR1 Package (Prudic, 1989). (4) `CPRIOR` = 'UPTO' - if Q_{DS} in reach `RNO` is greater than or equal to the specified diversion flow `DIVFLOW`, Q_{DS} is reduced by `DIVFLOW`. If Q_{DS} in reach `RNO` is less than `DIVFLOW`, `DIVFLOW` is set to Q_{DS} and there will be no flow available for reaches connected to downstream end of reach `RNO`.
- `perioddata` (`[rno, sfrsetting]`) -
 - `rno` (integer) integer value that defines the reach number associated with the specified `PERIOD` data on the line. `RNO` must be greater than zero and less than or equal to `NREACHES`. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - `sfrsetting` (keystring) line of information that is parsed into a keyword and values. Keyword values that can be used to start the `SFRSETTING` string include: `STATUS`, `MANNING`, `STAGE`, `INFLOW`, `RAINFALL`, `EVAPORATION`, `RUNOFF`, `DIVERSION`, `UPSTREAM_FRACTION`, and `AUXILIARY`.

status

[[string]]

- * `status` (string) keyword option to define stream reach status. `STATUS` can be `ACTIVE`, `INACTIVE`, or `SIMPLE`. The `SIMPLE STATUS` option simulates streamflow using a user-specified stage for a reach or a stage set to the top of the reach (`depth = 0`). In cases where the simulated leakage

calculated using the specified stage exceeds the sum of inflows to the reach, the stage is set to the top of the reach and leakage is set equal to the sum of inflows. Upstream fractions should be changed using the UPSTREAM_FRACTION SFRSETTING if the status for one or more reaches is changed to ACTIVE or INACTIVE. For example, if one of two downstream connections for a reach is inactivated, the upstream fraction for the active and inactive downstream reach should be changed to 1.0 and 0.0, respectively, to ensure that the active reach receives all of the downstream outflow from the upstream reach. By default, STATUS is ACTIVE.

manning

[[string]]

* manning (string) real or character value that defines the Manning's roughness coefficient for the reach. MANNING must be greater than zero. If the Options block includes a TIMESERIESFILE entry (see the "Time-Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

stage

[[string]]

* stage (string) real or character value that defines the stage for the reach. The specified STAGE is only applied if the reach uses the simple routing option. If STAGE is not specified for reaches that use the simple routing option, the specified stage is set to the top of the reach. If the Options block includes a TIMESERIESFILE entry (see the "Time- Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

inflow

[[string]]

* inflow (string) real or character value that defines the volumetric inflow rate for the streamflow routing reach. If the Options block includes a TIMESERIESFILE entry (see the "Time-Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value. By default, inflow rates are zero for each reach.

rainfall

[[string]]

* rainfall (string) real or character value that defines the volumetric rate per unit area of water added by precipitation directly on the streamflow routing reach. If the Options block includes a

TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value. By default, rainfall rates are zero for each reach.

evaporation

[[string]]

* evaporation (string) real or character value that defines the volumetric rate per unit area of water subtracted by evaporation from the streamflow routing reach. A positive evaporation rate should be provided. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value. If the volumetric evaporation rate for a reach exceeds the sources of water to the reach (upstream and specified inflows, rainfall, and runoff but excluding groundwater leakage into the reach) the volumetric evaporation rate is limited to the sources of water to the reach. By default, evaporation rates are zero for each reach.

runoff

[[string]]

* runoff (string) real or character value that defines the volumetric rate of diffuse overland runoff that enters the streamflow routing reach. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value. If the volumetric runoff rate for a reach is negative and exceeds inflows to the reach (upstream and specified inflows, and rainfall but excluding groundwater leakage into the reach) the volumetric runoff rate is limited to inflows to the reach and the volumetric evaporation rate for the reach is set to zero. By default, runoff rates are zero for each reach.

diversionrecord

[[idv, divflow]]

* idv (integer) an integer value specifying which diversion of reach RNO that DIVFLOW is being specified for. Must be less or equal to ndv for the current reach (RNO). This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.

* `divflow` (double) real or character value that defines the volumetric diversion (DIVFLOW) rate for the streamflow routing reach. If the Options block includes a `TIMESERIESFILE` entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

upstream_fraction

[[double]]

* `upstream_fraction` (double) real value that defines the fraction of upstream flow (USTRF) from each upstream reach that is applied as upstream inflow to the reach. The sum of all USTRF values for all reaches connected to the same upstream reach must be equal to one.

cross_sectionrecord

[[tab6_filename]]

* `tab6_filename` (string) character string that defines the path and filename for the file containing cross-section table data for the reach. The `TAB6_FILENAME` file includes the number of entries in the file and the station elevation data in terms of the fractional width and the reach depth. Instructions for creating the `TAB6_FILENAME` input file are provided in SFR Reach Cross-Section Table Input File section.

auxiliaryrecord

[[auxname, auxval]]

* `auxname` (string) name for the auxiliary variable to be assigned `AUXVAL`. `AUXNAME` must match one of the auxiliary variable names defined in the `OPTIONS` block. If `AUXNAME` does not match one of the auxiliary variable names defined in the `OPTIONS` block the data are ignored.

* `auxval` (double) value for the auxiliary variable. If the Options block includes a `TIMESERIESFILE` entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package `parent_file`.

`auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

`budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

`budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

```
connectiondata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>  
crosssections = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```

dfn = [['header', 'multi-package'], ['block options', 'name auxiliary', 'type
string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name
boundnames', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block
options', 'name print_input', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name print_stage', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name print_flows', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name save_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name stage_filerecord', 'type record
stage fileout stagefile', 'shape', 'reader urword', 'tagged true', 'optional true'],
['block options', 'name stage', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name stagefile', 'type
string', 'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged
false', 'optional false'], ['block options', 'name budget_filerecord', 'type record
budget fileout budgetfile', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name budget', 'type keyword', 'shape', 'in_record true',
'reader urword', 'tagged true', 'optional false'], ['block options', 'name fileout',
'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name budgetfile', 'type string', 'preserve_case true',
'shape', 'in_record true', 'reader urword', 'tagged false', 'optional false'],
['block options', 'name budgetcsv_filerecord', 'type record budgetcsv fileout
budgetcsvfile', 'shape', 'reader urword', 'tagged true', 'optional true'], ['block
options', 'name budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name budgetcsvfile',
'type string', 'preserve_case true', 'shape', 'in_record true', 'reader urword',
'tagged false', 'optional false'], ['block options', 'name
package_convergence_filerecord', 'type record package_convergence fileout
package_convergence_filename', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name package_convergence', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name package_convergence_filename', 'type string', 'shape', 'in_record
true', 'reader urword', 'tagged false', 'optional false'], ['block options', 'name
ts_filerecord', 'type record ts6 filein ts6_filename', 'shape', 'reader urword',
'tagged true', 'optional true', 'construct_package ts', 'construct_data timeseries',
'parameter_name timeseries'], ['block options', 'name ts6', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader urword',
'tagged true', 'optional false'], ['block options', 'name ts6_filename', 'type
string', 'preserve_case true', 'in_record true', 'reader urword', 'optional false',
'tagged false'], ['block options', 'name obs_filerecord', 'type record obs6 filein
obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package obs', 'construct_data continuous', 'parameter_name
observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block options', 'name mover', 'type
keyword', 'tagged true', 'reader urword', 'optional true'], ['block options', 'name
maximum_picard_iterations', 'type integer', 'reader urword', 'optional true'],
['block options', 'name maximum_iterations', 'type integer', 'reader urword',
'optional true'], ['block options', 'name maximum_depth_change', 'type double
precision', 'reader urword', 'optional true'], ['block options', 'name
unit_conversion', 'type double precision', 'reader urword', 'optional true',
'deprecated 6.4.2'], ['block options', 'name length_conversion', 'type double
precision', 'reader urword', 'optional true'], ['block options', 'name
time_conversion', 'type double precision', 'reader urword', 'optional true'],
['block dimensions', 'name nreaches', 'type integer', 'reader urword', 'optional
false'], ['block packagedata', 'name packagedata', 'type recarray rno cellid rlen
1453
8.1. MODFLOW 6
with input rtp rbth rhk man ncon ustrf ndv aux boundname', 'shape (maxbound)',
'reader urword'], ['block packagedata', 'name rno', 'type integer', 'shape', 'tagged
false', 'in_record true', 'reader urword', 'numeric_index true'], ['block
packagedata', 'name cellid', 'type integer', 'shape (ncelldim)', 'tagged false',

```

```
dfn_file_name = 'gwf-sfr.dfn'

diversions = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfsfr'

package_convergence_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator
object>

packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

stage_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfsto module

```
class ModflowGwfsto(model, loading_package=False, save_flows=None,
                    storagecoefficient=None, ss_confined_only=None, perioddata=None,
                    iconvert=0, ss=1e-05, sy=0.15, steady_state=None, transient=None,
                    filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwfsto defines a sto package within a gwf6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **save_flows** (*boolean*) -
 - save_flows (*boolean*) keyword to indicate that cell-by-cell flow terms will be written to the file specified with “BUDGET SAVE FILE” in Output Control.
- **storagecoefficient** (*boolean*) -
 - storagecoefficient (*boolean*) keyword to indicate that the SS array is read as storage coefficient rather than specific storage.
- **ss_confined_only** (*boolean*) -
 - ss_confined_only (*boolean*) keyword to indicate that compressible storage is only calculated for a convertible cell (ICONVERT>0) when the cell is under confined conditions (head greater than or equal to the top of the cell). This option has no effect on cells that are marked as being always confined (ICONVERT=0). This option is identical to the approach used to calculate storage changes under confined conditions in MODFLOW-2005.

- **perioddata** (*{varname:data}* or *tvb_perioddata data*) -
 - Contains data for the tvb package. Data can be stored in a dictionary containing data for the tvb package with variable names as keys and package data as values. Data just for the perioddata variable is also acceptable. See tvb package documentation for more information.
- **iconvert** (*[integer]*) -
 - iconvert (integer) is a flag for each cell that specifies whether or not a cell is convertible for the storage calculation. 0 indicates confined storage is used. >0 indicates confined storage is used when head is above cell top and a mixed formulation of unconfined and confined storage is used when head is below cell top.
- **ss** (*[double]*) -
 - ss (double) is specific storage (or the storage coefficient if STORAGECOEFFICIENT is specified as an option). Specific storage values must be greater than or equal to 0. If the CSUB Package is included in the GWF model, specific storage must be zero for every cell.
- **sy** (*[double]*) -
 - sy (double) is specific yield. Specific yield values must be greater than or equal to 0. Specific yield does not have to be specified if there are no convertible cells (ICONVERT=0 in every cell).
- **steady_state** (*boolean*) -
 - steady-state (boolean) keyword to indicate that stress period IPER is steady-state. Steady-state conditions will apply until the TRANSIENT keyword is specified in a subsequent BEGIN PERIOD block. If the CSUB Package is included in the GWF model, only the first and last stress period can be steady-state.
- **transient** (*boolean*) -
 - transient (boolean) keyword to indicate that stress period IPER is transient. Transient conditions will apply until the STEADY-STATE keyword is specified in a subsequent BEGIN PERIOD block.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

```
dfn = [['header'], ['block options', 'name save_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name storagecoefficient', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name
ss_confined_only', 'type keyword', 'reader urword', 'optional true'], ['block
options', 'name tvs_filerecord', 'type record tvs6 filein tvs_filename', 'shape',
'reader urword', 'tagged true', 'optional true', 'construct_package tvs',
'construct_data tvs_perioddata', 'parameter_name perioddata'], ['block options',
'name tvs6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name filein', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name tvs_filename', 'type string', 'preserve_case true', 'in_record
true', 'reader urword', 'optional false', 'tagged false'], ['block griddata', 'name
iconvert', 'type integer', 'shape (nodes)', 'valid', 'reader readarray', 'layered
true', 'optional false', 'default_value 0'], ['block griddata', 'name ss', 'type
double precision', 'shape (nodes)', 'valid', 'reader readarray', 'layered true',
'optional false', 'default_value 1.e-5'], ['block griddata', 'name sy', 'type double
precision', 'shape (nodes)', 'valid', 'reader readarray', 'layered true', 'optional
false', 'default_value 0.15'], ['block period', 'name iper', 'type integer',
'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader
urword', 'optional false'], ['block period', 'name steady-state', 'type keyword',
'shape', 'valid', 'reader urword', 'optional true'], ['block period', 'name
transient', 'type keyword', 'shape', 'valid', 'reader urword', 'optional true']]

dfn_file_name = 'gwf-sto.dfn'

iconvert = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

package_abbr = 'gwfsto'

ss = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

sy = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

tvs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfufz module

```
class ModflowGwfufz(model, loading_package=False, auxiliary=None, auxmultname=None,
                    boundnames=None, print_input=None, print_flows=None, save_flows=None,
                    wc_filerecord=None, budget_filerecord=None,
                    budgetcsv_filerecord=None, package_convergence_filerecord=None,
                    timeseries=None, observations=None, mover=None, simulate_et=None,
                    linear_gwet=None, square_gwet=None, simulate_gwseep=None,
                    unsat_etwc=None, unsat_etae=None, nuzfcells=None, ntrailwaves=7,
                    nwavesets=40, packagedata=None, perioddata=None, filename=None,
                    pname=None, **kwargs)
```

Bases: [*MFPackage*](#)

ModflowGwfufz defines a uzf package within a gwf6 model.

Parameters

- **model** ([*MFModel*](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.

- **auxiliary** (*[string]*) -
 - auxiliary (string) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** (*string*) -
 - auxmultname (string) name of auxiliary variable to be used as multiplier of GWF cell area used by UZF cell.
- **boundnames** (*boolean*) -
 - boundnames (boolean) keyword to indicate that boundary names may be provided with the list of UZF cells.
- **print_input** (*boolean*) -
 - print_input (boolean) keyword to indicate that the list of UZF information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (boolean) keyword to indicate that the list of UZF flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (boolean) keyword to indicate that UZF flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **wc_filerecord** (*[wcfile]*) -
 - wcfile (string) name of the binary output file to write water content information.
- **budget_filerecord** (*[budgetfile]*) -
 - budgetfile (string) name of the binary output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
 - budgetcsvfile (string) name of the comma-separated value (CSV) output file to write budget summary information. A budget

summary record will be written to this file for each time step of the simulation.

- **package_convergence_filerecord** (*[package_convergence_filename]*) -
 - *package_convergence_filename* (string) name of the comma spaced values output file to write package convergence information.
- **timeseries** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **mover** (*boolean*) -
 - *mover* (boolean) keyword to indicate that this instance of the UZF Package can be used with the Water Mover (MVR) Package. When the MOVER option is specified, additional memory is allocated within the package to store the available, provided, and received water.
- **simulate_et** (*boolean*) -
 - *simulate_et* (boolean) keyword specifying that ET in the unsaturated (UZF) and saturated zones (GWF) will be simulated. ET can be simulated in the UZF cell and not the GWF cell by omitting keywords LINEAR_GWET and SQUARE_GWET.
- **linear_gwet** (*boolean*) -
 - *linear_gwet* (boolean) keyword specifying that groundwater ET will be simulated using the original ET formulation of MODFLOW-2005.
- **square_gwet** (*boolean*) -
 - *square_gwet* (boolean) keyword specifying that groundwater ET will be simulated by assuming a constant ET rate for groundwater levels between land surface (TOP) and land surface minus the ET extinction depth (TOP-EXTDP). Groundwater ET is smoothly reduced from the PET rate to zero over a nominal interval at TOP-EXTDP.
- **simulate_gwseep** (*boolean*) -
 - *simulate_gwseep* (boolean) keyword specifying that groundwater discharge (GWSEEP) to land surface will be simulated. Groundwater discharge is nonzero when groundwater head is greater than land surface. This option is no longer recommended; a better approach is to use the Drain Package with discharge scaling as a way to handle seepage to

land surface. The Drain Package with discharge scaling is described in Chapter 3 of the Supplemental Technical Information.

- **unsat_etwc** (*boolean*) -
 - unsat_etwc (*boolean*) keyword specifying that ET in the unsaturated zone will be simulated as a function of the specified PET rate while the water content (THETA) is greater than the ET extinction water content (EXTWC).
- **unsat_etae** (*boolean*) -
 - unsat_etae (*boolean*) keyword specifying that ET in the unsaturated zone will be simulated using a capillary pressure based formulation. Capillary pressure is calculated using the Brooks-Corey retention function.
- **nuzfcells** (*integer*) -
 - nuzfcells (*integer*) is the number of UZF cells. More than one UZF cell can be assigned to a GWF cell; however, only one GWF cell can be assigned to a single UZF cell. If more than one UZF cell is assigned to a GWF cell, then an auxiliary variable should be used to reduce the surface area of the UZF cell with the AUXMULTNAME option.
- **ntrailwaves** (*integer*) -
 - ntrailwaves (*integer*) is the number of trailing waves. A recommended value of 7 can be used for NTRAILWAVES. This value can be increased to lower mass balance error in the unsaturated zone.
- **nwavesets** (*integer*) -
 - nwavesets (*integer*) is the number of wave sets. A recommended value of 40 can be used for NWAVESETS. This value can be increased if more waves are required to resolve variations in water content within the unsaturated zone.
- **packagedata** (*[iuzno, cellid, landflag, ivertcon, surfdep, vks, thtr, thts,)* -
 - thti, eps, boundname]**
 - iuzno (*integer*) integer value that defines the UZF cell number associated with the specified PACKAGEDATA data on the line. IUZNO must be greater than zero and less than or equal to NUZFCELLS. UZF information must be specified for every UZF cell or the program will terminate with an error. The program will also terminate with an error if information for a UZF cell is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - cellid (*(integer, ...)*) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS

input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.

- landflag (integer) integer value set to one for land surface cells indicating that boundary conditions can be applied and data can be specified in the PERIOD block. A value of 0 specifies a non-land surface cell.
- invertcon (integer) integer value set to specify underlying UZF cell that receives water flowing to bottom of cell. If unsaturated zone flow reaches the water table before the cell bottom, then water is added to the GWF cell instead of flowing to the underlying UZF cell. A value of 0 indicates the UZF cell is not connected to an underlying UZF cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
- surfdep (double) is the surface depression depth of the UZF cell.
- vks (double) is the saturated vertical hydraulic conductivity of the UZF cell. This value is used with the Brooks-Corey function and the simulated water content to calculate the partially saturated hydraulic conductivity.
- thtr (double) is the residual (irreducible) water content of the UZF cell. This residual water is not available to plants and will not drain into underlying aquifer cells.
- thts (double) is the saturated water content of the UZF cell. The values for saturated and residual water content should be set in a manner that is consistent with the specific yield value specified in the Storage Package. The saturated water content must be greater than the residual content.
- thti (double) is the initial water content of the UZF cell. The value must be greater than or equal to the residual water content and less than or equal to the saturated water content.
- eps (double) is the exponent used in the Brooks-Corey function. The Brooks-Corey function is used by UZF to calculate hydraulic conductivity under partially saturated conditions as a function of water content and the user-specified saturated hydraulic conductivity.
- boundname (string) name of the UZF cell. BOUNDNAME is an ASCII character variable that can contain as many as 40

characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.

• **perioddata** ([iuzno, finf, pet, extdp, extwc, ha, hroot, rootact, aux]) -

- iuzno (integer) integer value that defines the UZF cell number associated with the specified PERIOD data on the line. This argument is an index variable, which means that it should be treated as zero- based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.
- finf (string) real or character value that defines the applied infiltration rate of the UZF cell (LT^{-1}). If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- pet (string) real or character value that defines the potential evapotranspiration rate of the UZF cell and specified GWF cell. Evapotranspiration is first removed from the unsaturated zone and any remaining potential evapotranspiration is applied to the saturated zone. If IVERTCON is greater than zero then residual potential evapotranspiration not satisfied in the UZF cell is applied to the underlying UZF and GWF cells. PET is always specified, but is only used if SIMULATE_ET is specified in the OPTIONS block. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- extdp (string) real or character value that defines the evapotranspiration extinction depth of the UZF cell. If IVERTCON is greater than zero and EXTDP extends below the GWF cell bottom then remaining potential evapotranspiration is applied to the underlying UZF and GWF cells. EXTDP is always specified, but is only used if SIMULATE_ET is specified in the OPTIONS block. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- extwc (string) real or character value that defines the evapotranspiration extinction water content of the UZF cell. EXTWC is always specified, but is only used if SIMULATE_ET and UNSAT_ETWC are specified in the OPTIONS block. The evapotranspiration rate from the unsaturated zone will be set to zero when the calculated water content is at or less than this value. The value for EXTWC cannot be less than the residual water content, and if it is specified as being less than the residual water content it is set to the residual water content. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- **ha** (string) real or character value that defines the air entry potential (head) of the UZF cell. HA is always specified, but is only used if SIMULATE_ET and UNSAT_ETAE are specified in the OPTIONS block. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - **hroot** (string) real or character value that defines the root potential (head) of the UZF cell. HROOT is always specified, but is only used if SIMULATE_ET and UNSAT_ETAE are specified in the OPTIONS block. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - **rootact** (string) real or character value that defines the root activity function of the UZF cell. ROOTACT is the length of roots in a given volume of soil divided by that volume. Values range from 0 to about 3 cm^{-2} , depending on the plant community and its stage of development. ROOTACT is always specified, but is only used if SIMULATE_ET and UNSAT_ETAE are specified in the OPTIONS block. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - **aux** (double) represents the values of the auxiliary variables for each UZF. The values of auxiliary variables must be present for each UZF. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- **filename** (String) - File name for this package.
 - **pname** (String) - Package name for this package.
 - **parent_file** (MFPackage) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulllaktab package must have a mfgwflak package parent_file.

```
auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```



```

dfn = [['header', 'multi-package'], ['block options', 'name auxiliary', 'type
string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name
auxmultname', 'type string', 'shape', 'reader urword', 'optional true'], ['block
options', 'name boundnames', 'type keyword', 'shape', 'reader urword', 'optional
true'], ['block options', 'name print_input', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name print_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name save_flows', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name wc_filerecord', 'type
record water_content fileout wfile', 'shape', 'reader urword', 'tagged true',
'optional true'], ['block options', 'name water_content', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name wfile', 'type string', 'preserve_case true', 'shape', 'in_record
true', 'reader urword', 'tagged false', 'optional false'], ['block options', 'name
budget_filerecord', 'type record budget fileout budgetfile', 'shape', 'reader
urword', 'tagged true', 'optional true'], ['block options', 'name budget', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name fileout', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
budgetfile', 'preserve_case true', 'type string', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budgetcsv_filerecord', 'type record budgetcsv fileout budgetcsvfile', 'shape',
'reader urword', 'tagged true', 'optional true'], ['block options', 'name
budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name budgetcsvfile', 'type string',
'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged false',
'optional false'], ['block options', 'name package_convergence_filerecord', 'type
record package_convergence fileout package_convergence_filename', 'shape', 'reader
urword', 'tagged true', 'optional true'], ['block options', 'name
package_convergence', 'type keyword', 'shape', 'in_record true', 'reader urword',
'tagged true', 'optional false'], ['block options', 'name
package_convergence_filename', 'type string', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name ts_filerecord',
'type record ts6 filein ts6_filename', 'shape', 'reader urword', 'tagged true',
'optional true', 'construct_package ts', 'construct_data timeseries',
'parameter_name timeseries'], ['block options', 'name ts6', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader urword',
'tagged true', 'optional false'], ['block options', 'name ts6_filename', 'type
string', 'preserve_case true', 'in_record true', 'reader urword', 'optional false',
'tagged false'], ['block options', 'name obs_filerecord', 'type record obs6 filein
obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package obs', 'construct_data continuous', 'parameter_name
observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block options', 'name mover', 'type
keyword', 'tagged true', 'reader urword', 'optional true'], ['block options', 'name
simulate_et', 'type keyword', 'tagged true', 'reader urword', 'optional true'],
['block options', 'name linear_gwet', 'type keyword', 'tagged true', 'reader
urword', 'optional true'], ['block options', 'name square_gwet', 'type keyword',
'tagged true', 'reader urword', 'optional true'], ['block options', 'name
simulate_gwseep', 'type keyword', 'tagged true', 'reader urword', 'optional true'],
['block options', 'name unsat_etwc', 'type keyword', 'tagged true', 'reader urword',
'optional true'], ['block options', 'name unsat_etae', 'type keyword', 'tagged
true', 'reader urword', 'optional true'], ['block dimensions', 'name nuzfcells',
'type integer', 'reader urword', 'optional false'], ['block dimensions', 'name
nuzfcells', 'type integer', 'reader urword', 'optional false', 'default_value 1463'],
['block dimensions', 'name nwavesets', 'type integer', 'reader urword', 'optional
false', 'default_value 40'], ['block packagedata', 'name packagedata', 'type
recarray iuzno cellid landflag invertcon surfdep vks thtr thts thti eps boundname',

```

```
dfn_file_name = 'gwf-uzf.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfufz'

package_convergence_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator
object>

packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

wc_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwfvsc module

```
class ModflowGwfvsc(model, loading_package=False, viscref=1.0,
                    temperature_species_name=None, thermal_formulation=None,
                    thermal_a2=10.0, thermal_a3=248.37, thermal_a4=133.15,
                    viscosity_filerecord=None, nviscspecies=None, packagedata=None,
                    filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwfvsc defines a vsc package within a gwf6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **viscref** (*double*) -
 - viscref (double) fluid reference viscosity used in the equation of state. This value is set to 1.0 if not specified as an option.
- **temperature_species_name** (*string*) -
 - temperature_species_name (string) string used to identify the auxspeciesname in PACKAGEDATA that corresponds to the temperature species. There can be only one occurrence of this temperature species name in the PACKAGEDATA block or the program will terminate with an error. This value has no effect if viscosity does not depend on temperature.
- **thermal_formulation** (*string*) -
 - thermal_formulation (string) may be used for specifying which viscosity formulation to use for the temperature species. Can be either LINEAR or NONLINEAR. The LINEAR viscosity formulation is the default.
- **thermal_a2** (*double*) -

- `thermal_a2` (double) is an empirical parameter specified by the user for calculating viscosity using a nonlinear formulation. If A2 is not specified, a default value of 10.0 is assigned (Voss, 1984).
- **`thermal_a3`** (double) -
 - `thermal_a3` (double) is an empirical parameter specified by the user for calculating viscosity using a nonlinear formulation. If A3 is not specified, a default value of 248.37 is assigned (Voss, 1984).
- **`thermal_a4`** (double) -
 - `thermal_a4` (double) is an empirical parameter specified by the user for calculating viscosity using a nonlinear formulation. If A4 is not specified, a default value of 133.15 is assigned (Voss, 1984).
- **`viscosity_filerecord`** (*[viscosityfile]*) -
 - `viscosityfile` (string) name of the binary output file to write viscosity information. The viscosity file has the same format as the head file. Viscosity values will be written to the viscosity file whenever heads are written to the binary head file. The settings for controlling head output are contained in the Output Control option.
- **`nviscspecies`** (*integer*) -
 - `nviscspecies` (integer) number of species used in the viscosity equation of state. If either concentrations or temperature (or both) are used to update viscosity then then `nrhospecies` needs to be at least one.
- **`packagedata`** (*[iviscspec, dviscdc, cviscref, modelname, auxspeciesname]*) -
 - `iviscspec` (integer) integer value that defines the species number associated with the specified PACKAGEDATA data entered on each line. IVISCSPECIES must be greater than zero and less than or equal to NVISCSPECIES. Information must be specified for each of the NVISCSPECIES species or the program will terminate with an error. The program will also terminate with an error if information for a species is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - `dviscdc` (double) real value that defines the slope of the line defining the linear relationship between viscosity and temperature or between viscosity and concentration, depending on the type of species entered on each line. If the value of AUXSPECIESNAME entered on a line corresponds to TEMPERATURE_SPECIES_NAME (in the OPTIONS block), this value will be used when VISCOSITY_FUNC is equal to LINEAR (the default) in the OPTIONS block. When VISCOSITY_FUNC is set

to NONLINEAR, a value for DVISCDC must be specified though it is not used.

- `cvisceref` (double) real value that defines the reference temperature or reference concentration value used for this species in the viscosity equation of state. If `AUXSPECIESNAME` entered on a line corresponds to `TEMPERATURE_SPECIES_NAME` (in the `OPTIONS` block), then `CVISCREF` refers to a reference temperature, otherwise it refers to a reference concentration.
 - `modelname` (string) name of a GWT model used to simulate a species that will be used in the viscosity equation of state. This name will have no effect if the simulation does not include a GWT model that corresponds to this GWF model.
 - `auxspeciesname` (string) name of an auxiliary variable in a GWF stress package that will be used for this species to calculate the viscosity values. If a viscosity value is needed by the Viscosity Package then it will use the temperature or concentration values associated with this `AUXSPECIESNAME` in the viscosity equation of state. For advanced stress packages (LAK, SFR, MAW, and UZF) that have an associated advanced transport package (LKT, SFT, MWT, and UZT), the `FLOW_PACKAGE_AUXILIARY_NAME` option in the advanced transport package can be used to transfer simulated temperature or concentration(s) into the flow package auxiliary variable. In this manner, the Viscosity Package can calculate viscosity values for lakes, streams, multi-aquifer wells, and unsaturated zone flow cells using simulated concentrations.
- **`filename`** (*String*) - File name for this package.
 - **`pname`** (*String*) - Package name for this package.
 - **`parent_file`** ([MFPackage](#)) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulllaktab package must have a mfgwflak package `parent_file`.

```

dfn = [['header'], ['block options', 'name visceref', 'type double precision',
'reader urword', 'optional true', 'default_value 1.0'], ['block options', 'name
temperature_species_name', 'type string', 'shape', 'reader urword', 'optional
true'], ['block options', 'name thermal_formulation', 'type string', 'shape',
'reader urword', 'optional true', 'valid linear nonlinear'], ['block options', 'name
thermal_a2', 'type double precision', 'reader urword', 'optional true',
'default_value 10.'], ['block options', 'name thermal_a3', 'type double precision',
'reader urword', 'optional true', 'default_value 248.37'], ['block options', 'name
thermal_a4', 'type double precision', 'reader urword', 'optional true',
'default_value 133.15'], ['block options', 'name viscosity_filerecord', 'type record
viscosity fileout viscosityfile', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name viscosity', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
fileout', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name viscosityfile', 'type string',
'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged false',
'optional false'], ['block dimensions', 'name nviscspecies', 'type integer', 'reader
urword', 'optional false'], ['block packagedata', 'name packagedata', 'type recarray
iviscspec dviscdc cviscref modelname auxspeciesname', 'shape (nrhospecies)', 'reader
urword'], ['block packagedata', 'name iviscspec', 'type integer', 'shape', 'tagged
false', 'in_record true', 'reader urword', 'numeric_index true'], ['block
packagedata', 'name dviscdc', 'type double precision', 'shape', 'tagged false',
'in_record true', 'reader urword'], ['block packagedata', 'name cviscref', 'type
double precision', 'shape', 'tagged false', 'in_record true', 'reader urword'],
['block packagedata', 'name modelname', 'type string', 'in_record true', 'tagged
false', 'shape', 'reader urword'], ['block packagedata', 'name auxspeciesname',
'type string', 'in_record true', 'tagged false', 'shape', 'reader urword']]

dfn_file_name = 'gwf-vsc.dfn'

package_abbr = 'gwfvc'

packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

viscosity_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

flopy.mf6.modflow.mfgwfwel module

```

class ModflowGwfwel(model, loading_package=False, auxiliary=None, auxmultname=None,
    boundnames=None, print_input=None, print_flows=None, save_flows=None,
    auto_flow_reduce=None, afrcsv_filerecord=None, timeseries=None,
    observations=None, mover=None, maxbound=None,
    stress_period_data=None, filename=None, pname=None, **kwargs)

```

Bases: [MFPackage](#)

ModflowGwfwel defines a wel package within a gw6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **auxiliary** (*[string]*) -

- auxiliary (string) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for `naux`. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** (string) -
 - auxmultname (string) name of auxiliary variable to be used as multiplier of well flow rate.
- **boundnames** (boolean) -
 - boundnames (boolean) keyword to indicate that boundary names may be provided with the list of well cells.
- **print_input** (boolean) -
 - print_input (boolean) keyword to indicate that the list of well information will be written to the listing file immediately after it is read.
- **print_flows** (boolean) -
 - print_flows (boolean) keyword to indicate that the list of well flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (boolean) -
 - save_flows (boolean) keyword to indicate that well flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **auto_flow_reduce** (double) -
 - auto_flow_reduce (double) keyword and real value that defines the fraction of the cell thickness used as an interval for smoothly adjusting negative pumping rates to 0 in cells with head values less than or equal to the bottom of the cell. Negative pumping rates are adjusted to 0 or a smaller negative value when the head in the cell is equal to or less than the calculated interval above the cell bottom. `AUTO_FLOW_REDUCE` is set to 0.1 if the specified value is less than or equal to zero. By default, negative pumping rates are not reduced during a simulation.
- **afrcsv_filerecord** ([afrcsvfile]) -

- afrcsvfile (string) name of the comma-separated value (CSV) output file to write information about well extraction rates that have been reduced by the program. Entries are only written if the extraction rates are reduced.
- **timeseries** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **mover** (*boolean*) -
 - mover (boolean) keyword to indicate that this instance of the Well Package can be used with the Water Mover (MVR) Package. When the MOVER option is specified, additional memory is allocated within the package to store the available, provided, and received water.
- **maxbound** (*integer*) -
 - maxbound (integer) integer value specifying the maximum number of wells cells that will be specified for use during any stress period.
- **stress_period_data** (*[cellid, q, aux, boundname]*) -
 - cellid ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - q (double) is the volumetric well rate. A positive value indicates recharge (injection) and a negative value indicates discharge (extraction). If the Options block includes a TIMESERIESFILE entry (see the "Time-Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - aux (double) represents the values of the auxiliary variables for each well. The values of auxiliary variables must be present for each well. The values must be specified in the order of the auxiliary variables specified in the OPTIONS

block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- **boundname** (string) name of the well cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

afrcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>


```

dfn = [['header', 'multi-package'], ['block options', 'name auxiliary', 'type
string', 'shape (naux)', 'reader urword', 'optional true'], ['block options', 'name
auxmultname', 'type string', 'shape', 'reader urword', 'optional true'], ['block
options', 'name boundnames', 'type keyword', 'shape', 'reader urword', 'optional
true'], ['block options', 'name print_input', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name print_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name save_flows', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name auto_flow_reduce', 'type
double precision', 'reader urword', 'optional true'], ['block options', 'name
afrcsv_filerecord', 'type record auto_flow_reduce_csv fileout afrcsvfile', 'shape',
'reader urword', 'tagged true', 'optional true'], ['block options', 'name
auto_flow_reduce_csv', 'type keyword', 'shape', 'in_record true', 'reader urword',
'tagged true', 'optional false'], ['block options', 'name fileout', 'type keyword',
'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'],
['block options', 'name afrcsvfile', 'type string', 'preserve_case true', 'shape',
'in_record true', 'reader urword', 'tagged false', 'optional false'], ['block
options', 'name ts_filerecord', 'type record ts6 filein ts6_filename', 'shape',
'reader urword', 'tagged true', 'optional true', 'construct_package ts',
'construct_data timeseries', 'parameter_name timeseries'], ['block options', 'name
ts6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name filein', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
options', 'name ts6_filename', 'type string', 'preserve_case true', 'in_record
true', 'reader urword', 'optional false', 'tagged false'], ['block options', 'name
obs_filerecord', 'type record obs6 filein obs6_filename', 'shape', 'reader urword',
'tagged true', 'optional true', 'construct_package obs', 'construct_data
continuous', 'parameter_name observations'], ['block options', 'name obs6', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name obs6_filename', 'type string', 'preserve_case
true', 'in_record true', 'tagged false', 'reader urword', 'optional false'], ['block
options', 'name mover', 'type keyword', 'tagged true', 'reader urword', 'optional
true'], ['block dimensions', 'name maxbound', 'type integer', 'reader urword',
'optional false'], ['block period', 'name iper', 'type integer', 'block_variable
True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader urword',
'optional false'], ['block period', 'name stress_period_data', 'type recarray cellid
q aux boundname', 'shape (maxbound)', 'reader urword'], ['block period', 'name
cellid', 'type integer', 'shape (ncellldim)', 'tagged false', 'in_record true',
'reader urword'], ['block period', 'name q', 'type double precision', 'shape',
'tagged false', 'in_record true', 'reader urword', 'time_series true'], ['block
period', 'name aux', 'type double precision', 'in_record true', 'tagged false',
'shape (naux)', 'reader urword', 'optional true', 'time_series true'], ['block
period', 'name boundname', 'type string', 'shape', 'tagged false', 'in_record true',
'reader urword', 'optional true']]

```

```
dfn_file_name = 'gwf-wel.dfn'
```

```
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'gwfwel'
```

```
stress_period_data = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

8.1.6 MODFLOW 6 Groundwater Transport Model Packages

MODFLOW 6 groundwater transport models support a number of required and optional packages. Once a MODFLOW 6 groundwater transport model object (`mfgwt.ModflowGwt`) has been constructed various packages associated with the groundwater transport model can be constructed.

Contents:

`flopy.mf6.modflow.mfgwtadv` module

```
class ModflowGwtadv(model, loading_package=False, scheme=None, filename=None, pname=None,
                    **kwargs)
```

Bases: `MFPackage`

`ModflowGwtadv` defines a `adv` package within a `gwt6` model.

Parameters

- **model** (`MFModel`) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (`bool`) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **scheme** (`string`) -
 - `scheme` (`string`) scheme used to solve the advection term. Can be upstream, central, or TVD. If not specified, upstream weighting is the default weighting scheme.
- **filename** (`String`) - File name for this package.
- **pname** (`String`) - Package name for this package.
- **parent_file** (`MFPackage`) - Parent package file that references this package. Only needed for utility packages (`mfutl*`). For example, `mfutllaktab` package must have a `mfgwflak` package `parent_file`.

```
dfn = [['header'], ['block options', 'name scheme', 'type string', 'valid central  
upstream tvd', 'reader urword', 'optional true']]
```

```
dfn_file_name = 'gwt-adv.dfn'
```

```
package_abbr = 'gwtadv'
```

`flopy.mf6.modflow.mfgwtapi` module

```
class ModflowGwtapi(model, loading_package=False, boundnames=None, print_input=None,
                    print_flows=None, save_flows=None, observations=None, mover=None,
                    maxbound=None, filename=None, pname=None, **kwargs)
```

Bases: `MFPackage`

`ModflowGwtapi` defines a `api` package within a `gwt6` model.

Parameters

- **model** (`MFModel`) - Model that this package is a part of. Package is automatically added to model when it is initialized.

- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **boundnames** (*boolean*) -
 - boundnames (*boolean*) keyword to indicate that boundary names may be provided with the list of api boundary cells.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of api boundary information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (*boolean*) keyword to indicate that the list of api boundary flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (*boolean*) keyword to indicate that api boundary flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **mover** (*boolean*) -
 - mover (*boolean*) keyword to indicate that this instance of the api boundary Package can be used with the Water Mover (MVR) Package. When the MOVER option is specified, additional memory is allocated within the package to store the available, provided, and received water.
- **maxbound** (*integer*) -
 - maxbound (*integer*) integer value specifying the maximum number of api boundary cells that will be specified for use during any stress period.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

```
dfn = [['header'], ['block options', 'name boundnames', 'type keyword', 'shape',
'reader urword', 'optional true'], ['block options', 'name print_input', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name print_flows',
'type keyword', 'reader urword', 'optional true'], ['block options', 'name
save_flows', 'type keyword', 'reader urword', 'optional true'], ['block options',
'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape', 'reader
urword', 'tagged true', 'optional true', 'construct_package obs', 'construct_data
continuous', 'parameter_name observations'], ['block options', 'name obs6', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name filein', 'type keyword', 'shape', 'in_record true',
'reader urword', 'tagged true', 'optional false'], ['block options', 'name
obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block options', 'name mover', 'type
keyword', 'tagged true', 'reader urword', 'optional true'], ['block dimensions',
'name maxbound', 'type integer', 'reader urword', 'optional false']]

dfn_file_name = 'gwt-api.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwtapi'
```

flopy.mf6.modflow.mfgwtcnc module

```
class ModflowGwtcnc(model, loading_package=False, auxiliary=None, auxmultname=None,
                    boundnames=None, print_input=None, print_flows=None, save_flows=None,
                    timeseries=None, observations=None, maxbound=None,
                    stress_period_data=None, filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwtcnc defines a cnc package within a gwt6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **auxiliary** (*[string]*) -
 - auxiliary (string) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** (*string*) -

- auxmultname (string) name of auxiliary variable to be used as multiplier of concentration value.
- **boundnames** (*boolean*) -
 - boundnames (boolean) keyword to indicate that boundary names may be provided with the list of constant concentration cells.
- **print_input** (*boolean*) -
 - print_input (boolean) keyword to indicate that the list of constant concentration information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (boolean) keyword to indicate that the list of constant concentration flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (boolean) keyword to indicate that constant concentration flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **timeseries** (*{varname:data} or timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data} or continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **maxbound** (*integer*) -
 - maxbound (integer) integer value specifying the maximum number of constant concentrations cells that will be specified for use during any stress period.
- **stress_period_data** (*[cellid, conc, aux, boundname]*) -
 - cellid ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based

when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.

- `conc` (double) is the constant concentration value. If the Options block includes a `TIMESERIESFILE` entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - `aux` (double) represents the values of the auxiliary variables for each constant concentration. The values of auxiliary variables must be present for each constant concentration. The values must be specified in the order of the auxiliary variables specified in the `OPTIONS` block. If the package supports time series and the Options block includes a `TIMESERIESFILE` entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - `boundname` (string) name of the constant concentration cell. `BOUNDNAME` is an ASCII character variable that can contain as many as 40 characters. If `BOUNDNAME` contains spaces in it, then the entire name must be enclosed within single quotes.
- `filename` (*String*) - File name for this package.
 - `pname` (*String*) - Package name for this package.
 - `parent_file` (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (`mfutl*`). For example, `mfutllaktab` package must have a `mfgwflak` package `parent_file`.

`auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

```
dfn = [['header'], ['block options', 'name auxiliary', 'type string', 'shape
(naux)', 'reader urword', 'optional true'], ['block options', 'name auxmultname',
'type string', 'shape', 'reader urword', 'optional true'], ['block options', 'name
boundnames', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block
options', 'name print_input', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name print_flows', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name save_flows', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name ts_filerecord', 'type record ts6 filein
ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package ts', 'construct_data timeseries', 'parameter_name timeseries'],
['block options', 'name ts6', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name ts6_filename', 'type string', 'preserve_case true',
'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block
options', 'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape',
'reader urword', 'tagged true', 'optional true', 'construct_package obs',
'construct_data continuous', 'parameter_name observations'], ['block options', 'name
obs6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name obs6_filename', 'type string',
'preserve_case true', 'in_record true', 'tagged false', 'reader urword', 'optional
false'], ['block dimensions', 'name maxbound', 'type integer', 'reader urword',
'optional false'], ['block period', 'name iper', 'type integer', 'block_variable
True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader urword',
'optional false'], ['block period', 'name stress_period_data', 'type recarray cellid
conc aux boundname', 'shape (maxbound)', 'reader urword'], ['block period', 'name
cellid', 'type integer', 'shape (ncelldim)', 'tagged false', 'in_record true',
'reader urword'], ['block period', 'name conc', 'type double precision', 'shape',
'tagged false', 'in_record true', 'reader urword', 'time_series true'], ['block
period', 'name aux', 'type double precision', 'in_record true', 'tagged false',
'shape (naux)', 'reader urword', 'optional true', 'time_series true'], ['block
period', 'name boundname', 'type string', 'shape', 'tagged false', 'in_record true',
'reader urword', 'optional true']]
```

```
dfn_file_name = 'gwt-cnc.dfn'
```

```
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'gwtcnc'
```

```
stress_period_data = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwt dis module

```
class ModflowGwt dis(model, loading_package=False, length_units=None, nogrb=None,
    xorigin=None, yorigin=None, angrot=None, nlay=1, nrow=2, ncol=2,
    delr=1.0, delc=1.0, top=1.0, botm=0.0, idomain=None, filename=None,
    pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwt dis defines a dis package within a gwt6 model.

Parameters

- **model** (*MFModel*) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **length_units** (*string*) -
 - **length_units** (*string*) is the length units used for this model. Values can be "FEET", "METERS", or "CENTIMETERS". If not specified, the default is "UNKNOWN".
- **nogrb** (*boolean*) -
 - **nogrb** (*boolean*) keyword to deactivate writing of the binary grid file.
- **xorigin** (*double*) -
 - **xorigin** (*double*) x-position of the lower-left corner of the model grid. A default value of zero is assigned if not specified. The value for XORIGIN does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **yorigin** (*double*) -
 - **yorigin** (*double*) y-position of the lower-left corner of the model grid. If not specified, then a default value equal to zero is used. The value for YORIGIN does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **angrot** (*double*) -
 - **angrot** (*double*) counter-clockwise rotation angle (in degrees) of the lower-left corner of the model grid. If not specified, then a default value of 0.0 is assigned. The value for ANGROT does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **nlay** (*integer*) -
 - **nlay** (*integer*) is the number of layers in the model grid.
- **nrow** (*integer*) -
 - **nrow** (*integer*) is the number of rows in the model grid.
- **ncol** (*integer*) -
 - **ncol** (*integer*) is the number of columns in the model grid.
- **delr** (*[double]*) -
 - **delr** (*double*) is the column spacing in the row direction.
- **delc** (*[double]*) -
 - **delc** (*double*) is the row spacing in the column direction.
- **top** (*[double]*) -

- top (double) is the top elevation for each cell in the top model layer.
- **botm** ([double]) -
 - botm (double) is the bottom elevation for each cell.
- **idomain** ([integer]) -
 - idomain (integer) is an optional array that characterizes the existence status of a cell. If the IDOMAIN array is not specified, then all model cells exist within the solution. If the IDOMAIN value for a cell is 0, the cell does not exist in the simulation. Input and output values will be read and written for the cell, but internal to the program, the cell is excluded from the solution. If the IDOMAIN value for a cell is 1, the cell exists in the simulation. If the IDOMAIN value for a cell is -1, the cell does not exist in the simulation. Furthermore, the first existing cell above will be connected to the first existing cell below. This type of cell is referred to as a “vertical pass through” cell.
- **filename** (String) - File name for this package.
- **pname** (String) - Package name for this package.
- **parent_file** (MFPackage) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulaktab package must have a mfgwflak package parent_file.

```

botm = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

delc = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

delr = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

dfn = [['header'], ['block options', 'name length_units', 'type string', 'reader
urword', 'optional true'], ['block options', 'name nogrb', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name xorigin', 'type double
precision', 'reader urword', 'optional true'], ['block options', 'name yorigin',
'type double precision', 'reader urword', 'optional true'], ['block options', 'name
angrot', 'type double precision', 'reader urword', 'optional true'], ['block
dimensions', 'name nlay', 'type integer', 'reader urword', 'optional false',
'default_value 1'], ['block dimensions', 'name nrow', 'type integer', 'reader
urword', 'optional false', 'default_value 2'], ['block dimensions', 'name ncol',
'type integer', 'reader urword', 'optional false', 'default_value 2'], ['block
griddata', 'name delr', 'type double precision', 'shape (ncol)', 'reader readarray',
'default_value 1.0'], ['block griddata', 'name delc', 'type double precision',
'shape (nrow)', 'reader readarray', 'default_value 1.0'], ['block griddata', 'name
top', 'type double precision', 'shape (ncol, nrow)', 'reader readarray',
'default_value 1.0'], ['block griddata', 'name botm', 'type double precision',
'shape (ncol, nrow, nlay)', 'reader readarray', 'layered true', 'default_value 0.'],
['block griddata', 'name idomain', 'type integer', 'shape (ncol, nrow, nlay)',
'reader readarray', 'layered true', 'optional true']]

dfn_file_name = 'gwt-dis.dfn'

idomain = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

```

```
package_abbrev = 'gwtdis'
```

```
top = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

`flopy.mf6.modflow.mfgwtdisu` module

```
class ModflowGwtdisu(model, loading_package=False, length_units=None, nogrb=None,
                      xorigin=None, yorigin=None, angrot=None,
                      vertical_offset_tolerance=0.0, nodes=None, nja=None, nvert=None,
                      top=None, bot=None, area=None, idomain=None, iac=None, ja=None,
                      ihc=None, cl12=None, hwva=None, angldegx=None, vertices=None,
                      cell12d=None, filename=None, pname=None, **kwargs)
```

Bases: [`MFPackage`](#)

`ModflowGwtdisu` defines a `disu` package within a `gwtdis` model.

Parameters

- **model** ([`MFModel`](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** ([`bool`](#)) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **length_units** ([`string`](#)) -
 - `length_units` ([`string`](#)) is the length units used for this model. Values can be “FEET”, “METERS”, or “CENTIMETERS”. If not specified, the default is “UNKNOWN”.
- **nogrb** ([`boolean`](#)) -
 - `nogrb` ([`boolean`](#)) keyword to deactivate writing of the binary grid file.
- **xorigin** ([`double`](#)) -
 - `xorigin` ([`double`](#)) x-position of the origin used for model grid vertices. This value should be provided in a real-world coordinate system. A default value of zero is assigned if not specified. The value for `XORIGIN` does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **yorigin** ([`double`](#)) -
 - `yorigin` ([`double`](#)) y-position of the origin used for model grid vertices. This value should be provided in a real-world coordinate system. If not specified, then a default value equal to zero is used. The value for `YORIGIN` does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **angrot** ([`double`](#)) -
 - `angrot` ([`double`](#)) counter-clockwise rotation angle (in degrees) of the model grid coordinate system relative to a real-world coordinate system. If not specified, then a default value of 0.0 is assigned. The value for `ANGROT` does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.

- **vertical_offset_tolerance** (*double*) -
 - vertical_offset_tolerance (*double*) checks are performed to ensure that the top of a cell is not higher than the bottom of an overlying cell. This option can be used to specify the tolerance that is used for checking. If top of a cell is above the bottom of an overlying cell by a value less than this tolerance, then the program will not terminate with an error. The default value is zero. This option should generally not be used.
- **nodes** (*integer*) -
 - nodes (*integer*) is the number of cells in the model grid.
- **nja** (*integer*) -
 - nja (*integer*) is the sum of the number of connections and NODES. When calculating the total number of connections, the connection between cell n and cell m is considered to be different from the connection between cell m and cell n. Thus, NJA is equal to the total number of connections, including n to m and m to n, and the total number of cells.
- **nvert** (*integer*) -
 - nvert (*integer*) is the total number of (x, y) vertex pairs used to define the plan-view shape of each cell in the model grid. If NVERT is not specified or is specified as zero, then the VERTICES and CELL2D blocks below are not read. NVERT and the accompanying VERTICES and CELL2D blocks should be specified for most simulations. If the XT3D or SAVE_SPECIFIC_DISCHARGE options are specified in the NPF Package, then this information is required.
- **top** (*[double]*) -
 - top (*double*) is the top elevation for each cell in the model grid.
- **bot** (*[double]*) -
 - bot (*double*) is the bottom elevation for each cell.
- **area** (*[double]*) -
 - area (*double*) is the cell surface area (in plan view).
- **idomain** (*[integer]*) -
 - idomain (*integer*) is an optional array that characterizes the existence status of a cell. If the IDOMAIN array is not specified, then all model cells exist within the solution. If the IDOMAIN value for a cell is 0, the cell does not exist in the simulation. Input and output values will be read and written for the cell, but internal to the program, the cell is excluded from the solution. If the IDOMAIN value for a cell is 1 or greater, the cell exists in the simulation. IDOMAIN values of -1 cannot be specified for the DISU Package.
- **iac** (*[integer]*) -

- iac (integer) is the number of connections (plus 1) for each cell. The sum of all the entries in IAC must be equal to NJA.
- **ja** ([integer]) -
 - ja (integer) is a list of cell number (n) followed by its connecting cell numbers (m) for each of the m cells connected to cell n. The number of values to provide for cell n is IAC(n). This list is sequentially provided for the first to the last cell. The first value in the list must be cell n itself, and the remaining cells must be listed in an increasing order (sorted from lowest number to highest). Note that the cell and its connections are only supplied for the GWT cells and their connections to the other GWT cells. Also note that the JA list input may be divided such that every node and its connectivity list can be on a separate line for ease in readability of the file. To further ease readability of the file, the node number of the cell whose connectivity is subsequently listed, may be expressed as a negative number, the sign of which is subsequently converted to positive by the code. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
- **ihc** ([integer]) -
 - ihc (integer) is an index array indicating the direction between node n and all of its m connections. If IHC = 0 then cell n and cell m are connected in the vertical direction. Cell n overlies cell m if the cell number for n is less than m; cell m overlies cell n if the cell number for m is less than n. If IHC = 1 then cell n and cell m are connected in the horizontal direction. If IHC = 2 then cell n and cell m are connected in the horizontal direction, and the connection is vertically staggered. A vertically staggered connection is one in which a cell is horizontally connected to more than one cell in a horizontal connection.
- **cl12** ([double]) -
 - cl12 (double) is the array containing connection lengths between the center of cell n and the shared face with each adjacent m cell.
- **hwva** ([double]) -
 - hwva (double) is a symmetric array of size NJA. For horizontal connections, entries in HWVA are the horizontal width perpendicular to flow. For vertical connections, entries in HWVA are the vertical area for flow. Thus, values in the HWVA array contain dimensions of both length and area. Entries in the HWVA array have a one-to-one correspondence with the connections specified in the JA array. Likewise, there is a one-to-one correspondence between entries in the HWVA array and entries in the IHC array, which specifies the connection type (horizontal or vertical). Entries in the HWVA array must be symmetric; the program will terminate with an error if the

value for HWVA for an n to m connection does not equal the value for HWVA for the corresponding n to m connection.

- **angldegx** ([double]) -
 - **angldegx** (double) is the angle (in degrees) between the horizontal x-axis and the outward normal to the face between a cell and its connecting cells. The angle varies between zero and 360.0 degrees, where zero degrees points in the positive x-axis direction, and 90 degrees points in the positive y-axis direction. ANGLDEGX is only needed if horizontal anisotropy is specified in the NPF Package, if the XT3D option is used in the NPF Package, or if the SAVE_SPECIFIC_DISCHARGE option is specified in the NPF Package. ANGLDEGX does not need to be specified if these conditions are not met. ANGLDEGX is of size NJA; values specified for vertical connections and for the diagonal position are not used. Note that ANGLDEGX is read in degrees, which is different from MODFLOW-USG, which reads a similar variable (ANGLEX) in radians.
- **vertices** ([iv, xv, yv]) -
 - **iv** (integer) is the vertex number. Records in the VERTICES block must be listed in consecutive order from 1 to NVERT. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.
 - **xv** (double) is the x-coordinate for the vertex.
 - **yv** (double) is the y-coordinate for the vertex.
- **cell2d** ([icell2d, xc, yc, nvert, icvert]) -
 - **icell2d** (integer) is the cell2d number. Records in the CELL2D block must be listed in consecutive order from 1 to NODES. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.
 - **xc** (double) is the x-coordinate for the cell center.
 - **yc** (double) is the y-coordinate for the cell center.
 - **nvert** (integer) is the number of vertices required to define the cell. There may be a different number of vertices for each cell.
 - **icvert** (integer) is an array of integer values containing vertex numbers (in the VERTICES block) used to define the cell. Vertices must be listed in clockwise order. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.
- **filename** (String) - File name for this package.
- **pname** (String) - Package name for this package.

- `parent_file` (`MFPackage`) - Parent package file that references this package. Only needed for utility packages (`mfutil*`). For example, `mfutilaktab` package must have a `mfgwflak` package `parent_file`.

```
angldegx = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

area = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

bot = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

cell2d = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

cl12 = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

dfn = [['header'], ['block options', 'name length_units', 'type string', 'reader
urword', 'optional true'], ['block options', 'name nogrb', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name xorigin', 'type double
precision', 'reader urword', 'optional true'], ['block options', 'name yorigin',
'type double precision', 'reader urword', 'optional true'], ['block options', 'name
angrot', 'type double precision', 'reader urword', 'optional true'], ['block
options', 'name vertical_offset_tolerance', 'type double precision', 'reader
urword', 'optional true', 'default_value 0.0', 'mf6internal voffsettol'], ['block
dimensions', 'name nodes', 'type integer', 'reader urword', 'optional false'],
['block dimensions', 'name nja', 'type integer', 'reader urword', 'optional false'],
['block dimensions', 'name nvert', 'type integer', 'reader urword', 'optional
true'], ['block griddata', 'name top', 'type double precision', 'shape (nodes)',
'reader readarray'], ['block griddata', 'name bot', 'type double precision', 'shape
(nodes)', 'reader readarray'], ['block griddata', 'name area', 'type double
precision', 'shape (nodes)', 'reader readarray'], ['block griddata', 'name idomain',
'type integer', 'shape (nodes)', 'reader readarray', 'layered false', 'optional
true'], ['block connectiondata', 'name iac', 'type integer', 'shape (nodes)',
'reader readarray'], ['block connectiondata', 'name ja', 'type integer', 'shape
(nja)', 'reader readarray', 'numeric_index true', 'jagged_array iac'], ['block
connectiondata', 'name ihc', 'type integer', 'shape (nja)', 'reader readarray',
'jagged_array iac'], ['block connectiondata', 'name cl12', 'type double precision',
'shape (nja)', 'reader readarray', 'jagged_array iac'], ['block connectiondata',
'name hwva', 'type double precision', 'shape (nja)', 'reader readarray',
'jagged_array iac'], ['block connectiondata', 'name angldegx', 'type double
precision', 'optional true', 'shape (nja)', 'reader readarray', 'jagged_array iac'],
['block vertices', 'name vertices', 'type recarray iv xv yv', 'shape (nvert)',
'reader urword', 'optional false'], ['block vertices', 'name iv', 'type integer',
'in_record true', 'tagged false', 'reader urword', 'optional false', 'numeric_index
true'], ['block vertices', 'name xv', 'type double precision', 'in_record true',
'tagged false', 'reader urword', 'optional false'], ['block vertices', 'name yv',
'type double precision', 'in_record true', 'tagged false', 'reader urword',
'optional false'], ['block cell2d', 'name cell2d', 'type recarray icell2d xc yc
ncvert icvert', 'shape (nodes)', 'reader urword', 'optional false'], ['block
cell2d', 'name icell2d', 'type integer', 'in_record true', 'tagged false', 'reader
urword', 'optional false', 'numeric_index true'], ['block cell2d', 'name xc', 'type
double precision', 'in_record true', 'tagged false', 'reader urword', 'optional
false'], ['block cell2d', 'name yc', 'type double precision', 'in_record true',
'tagged false', 'reader urword', 'optional false'], ['block cell2d', 'name ncvert',
'type integer', 'in_record true', 'tagged false', 'reader urword', 'optional
false'], ['block cell2d', 'name icvert', 'type integer', 'shape (ncvert)',
'in_record true', 'tagged false', 'reader urword', 'optional false', 'numeric_index
true']]
```

```

dfn_file_name = 'gwt-disu.dfn'

hwva = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
iac = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
idomain = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
ihc = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
ja = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
package_abbr = 'gwtdisu'

top = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
vertices = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

flopy.mf6.modflow.mfgwtdisv module

```

class ModflowGwtdisv(model, loading_package=False, length_units=None, nogrb=None,
                      xorigin=None, yorigin=None, angrot=None, nlay=None, ncpl=None,
                      nvert=None, top=None, botm=None, idomain=None, vertices=None,
                      cell2d=None, filename=None, pname=None, **kwargs)

```

Bases: [MFPackage](#)

ModflowGwtdisv defines a disv package within a gwt6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **length_units** (*string*) -
 - length_units (*string*) is the length units used for this model. Values can be “FEET”, “METERS”, or “CENTIMETERS”. If not specified, the default is “UNKNOWN”.
- **nogrb** (*boolean*) -
 - nogrb (*boolean*) keyword to deactivate writing of the binary grid file.
- **xorigin** (*double*) -
 - xorigin (*double*) x-position of the origin used for model grid vertices. This value should be provided in a real-world coordinate system. A default value of zero is assigned if not specified. The value for XORIGIN does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **yorigin** (*double*) -
 - yorigin (*double*) y-position of the origin used for model grid vertices. This value should be provided in a real-world coordinate system. If not specified, then a default value

equal to zero is used. The value for YORIGIN does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.

- **angrot** (*double*) -
 - angrout (*double*) counter-clockwise rotation angle (in degrees) of the model grid coordinate system relative to a real-world coordinate system. If not specified, then a default value of 0.0 is assigned. The value for ANGROT does not affect the model simulation, but it is written to the binary grid file so that postprocessors can locate the grid in space.
- **nlay** (*integer*) -
 - nlay (*integer*) is the number of layers in the model grid.
- **ncpl** (*integer*) -
 - ncpl (*integer*) is the number of cells per layer. This is a constant value for the grid and it applies to all layers.
- **nvert** (*integer*) -
 - nvert (*integer*) is the total number of (x, y) vertex pairs used to characterize the horizontal configuration of the model grid.
- **top** (*[double]*) -
 - top (*double*) is the top elevation for each cell in the top model layer.
- **botm** (*[double]*) -
 - botm (*double*) is the bottom elevation for each cell.
- **idomain** (*[integer]*) -
 - idomain (*integer*) is an optional array that characterizes the existence status of a cell. If the IDOMAIN array is not specified, then all model cells exist within the solution. If the IDOMAIN value for a cell is 0, the cell does not exist in the simulation. Input and output values will be read and written for the cell, but internal to the program, the cell is excluded from the solution. If the IDOMAIN value for a cell is 1, the cell exists in the simulation. If the IDOMAIN value for a cell is -1, the cell does not exist in the simulation. Furthermore, the first existing cell above will be connected to the first existing cell below. This type of cell is referred to as a “vertical pass through” cell.
- **vertices** (*[iv, xv, yv]*) -
 - iv (*integer*) is the vertex number. Records in the VERTICES block must be listed in consecutive order from 1 to NVERT. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - xv (*double*) is the x-coordinate for the vertex.

- yv (double) is the y-coordinate for the vertex.
- **cell2d** ([icell2d, xc, yc, nvert, icvert]) -
 - icell2d (integer) is the CELL2D number. Records in the CELL2D block must be listed in consecutive order from the first to the last. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - xc (double) is the x-coordinate for the cell center.
 - yc (double) is the y-coordinate for the cell center.
 - nvert (integer) is the number of vertices required to define the cell. There may be a different number of vertices for each cell.
 - icvert (integer) is an array of integer values containing vertex numbers (in the VERTICES block) used to define the cell. Vertices must be listed in clockwise order. Cells that are connected must share vertices. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
- **filename** (String) - File name for this package.
- **pname** (String) - Package name for this package.
- **parent_file** (MFPackage) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfullaktab package must have a mfgwflak package parent_file.

botm = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

cell2d = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```
dfn = [['header'], ['block options', 'name length_units', 'type string', 'reader
urword', 'optional true'], ['block options', 'name nogrb', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name xorigin', 'type double
precision', 'reader urword', 'optional true'], ['block options', 'name yorigin',
'type double precision', 'reader urword', 'optional true'], ['block options', 'name
angrot', 'type double precision', 'reader urword', 'optional true'], ['block
dimensions', 'name nlay', 'type integer', 'reader urword', 'optional false'],
['block dimensions', 'name ncpl', 'type integer', 'reader urword', 'optional
false'], ['block dimensions', 'name nvert', 'type integer', 'reader urword',
'optional false'], ['block griddata', 'name top', 'type double precision', 'shape
(ncpl)', 'reader readarray'], ['block griddata', 'name botm', 'type double
precision', 'shape (ncpl, nlay)', 'reader readarray', 'layered true'], ['block
griddata', 'name idomain', 'type integer', 'shape (ncpl, nlay)', 'reader readarray',
'layered true', 'optional true'], ['block vertices', 'name vertices', 'type recarray
iv xv yv', 'shape (nvert)', 'reader urword', 'optional false'], ['block vertices',
'name iv', 'type integer', 'in_record true', 'tagged false', 'reader urword',
'optional false', 'numeric_index true'], ['block vertices', 'name xv', 'type double
precision', 'in_record true', 'tagged false', 'reader urword', 'optional false'],
['block vertices', 'name yv', 'type double precision', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block cell2d', 'name cell2d', 'type
recarray icell2d xc yc nvert icvert', 'shape (ncpl)', 'reader urword', 'optional
false'], ['block cell2d', 'name icell2d', 'type integer', 'in_record true', 'tagged
false', 'reader urword', 'optional false', 'numeric_index true'], ['block cell2d',
'name xc', 'type double precision', 'in_record true', 'tagged false', 'reader
urword', 'optional false'], ['block cell2d', 'name yc', 'type double precision',
'in_record true', 'tagged false', 'reader urword', 'optional false'], ['block
cell2d', 'name nvert', 'type integer', 'in_record true', 'tagged false', 'reader
urword', 'optional false'], ['block cell2d', 'name icvert', 'type integer', 'shape
(ncvert)', 'in_record true', 'tagged false', 'reader urword', 'optional false',
'numeric_index true']]

dfn_file_name = 'gwt-disv.dfn'

idomain = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

package_abbr = 'gwtdisv'

top = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

vertices = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mf6gwtdsp module

```
class ModflowGwtdsp(model, loading_package=False, xt3d_off=None, xt3d_rhs=None,
                    diffc=None, alh=None, alv=None, ath1=None, ath2=None, atv=None,
                    filename=None, pname=None, **kwargs)
```

Bases: [*MFPackage*](#)

ModflowGwtdsp defines a dsp package within a gwt6 model.

Parameters

- **model** ([*MFModel*](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.

- **xt3d_off** (*boolean*) -
 - xt3d_off (*boolean*) deactivate the xt3d method and use the faster and less accurate approximation. This option may provide a fast and accurate solution under some circumstances, such as when flow aligns with the model grid, there is no mechanical dispersion, or when the longitudinal and transverse dispersivities are equal. This option may also be used to assess the computational demand of the XT3D approach by noting the run time differences with and without this option on.
- **xt3d_rhs** (*boolean*) -
 - xt3d_rhs (*boolean*) add xt3d terms to right-hand side, when possible. This option uses less memory, but may require more iterations.
- **diffc** (*[double]*) -
 - diffc (*double*) effective molecular diffusion coefficient.
- **alh** (*[double]*) -
 - alh (*double*) longitudinal dispersivity in horizontal direction. If flow is strictly horizontal, then this is the longitudinal dispersivity that will be used. If flow is not strictly horizontal or strictly vertical, then the longitudinal dispersivity is a function of both ALH and ALV. If mechanical dispersion is represented (by specifying any dispersivity values) then this array is required.
- **alv** (*[double]*) -
 - alv (*double*) longitudinal dispersivity in vertical direction. If flow is strictly vertical, then this is the longitudinal dispersivity value that will be used. If flow is not strictly horizontal or strictly vertical, then the longitudinal dispersivity is a function of both ALH and ALV. If this value is not specified and mechanical dispersion is represented, then this array is set equal to ALH.
- **ath1** (*[double]*) -
 - ath1 (*double*) transverse dispersivity in horizontal direction. This is the transverse dispersivity value for the second ellipsoid axis. If flow is strictly horizontal and directed in the x direction (along a row for a regular grid), then this value controls spreading in the y direction. If mechanical dispersion is represented (by specifying any dispersivity values) then this array is required.
- **ath2** (*[double]*) -
 - ath2 (*double*) transverse dispersivity in horizontal direction. This is the transverse dispersivity value for the third ellipsoid axis. If flow is strictly horizontal and directed in the x direction (along a row for a regular grid), then this value controls spreading in the z direction. If this value is not specified and mechanical dispersion is represented, then this array is set equal to ATH1.

- **atv** (*[double]*) -
 - atv (double) transverse dispersivity when flow is in vertical direction. If flow is strictly vertical and directed in the z direction, then this value controls spreading in the x and y directions. If this value is not specified and mechanical dispersion is represented, then this array is set equal to ATH2.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutilaktab package must have a mfgwflak package parent_file.

```

alh = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

alv = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

ath1 = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

ath2 = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

atv = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

dfn = [['header'], ['block options', 'name xt3d_off', 'type keyword', 'shape',
'reader urword', 'optional true'], ['block options', 'name xt3d_rhs', 'type
keyword', 'shape', 'reader urword', 'optional true'], ['block griddata', 'name
diffc', 'type double precision', 'shape (nodes)', 'reader readarray', 'layered
true', 'optional true'], ['block griddata', 'name alh', 'type double precision',
'shape (nodes)', 'reader readarray', 'layered true', 'optional true'], ['block
griddata', 'name alv', 'type double precision', 'shape (nodes)', 'reader readarray',
'layered true', 'optional true'], ['block griddata', 'name ath1', 'type double
precision', 'shape (nodes)', 'reader readarray', 'layered true', 'optional true'],
['block griddata', 'name ath2', 'type double precision', 'shape (nodes)', 'reader
readarray', 'layered true', 'optional true'], ['block griddata', 'name atv', 'type
double precision', 'shape (nodes)', 'reader readarray', 'layered true', 'optional
true']]

dfn_file_name = 'gwt-dsp.dfn'

diffc = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

package_abbr = 'gwt dsp'

```

flopy.mf6.modflow.mfgwtfmi module

```

class ModflowGwtfmi(model, loading_package=False, save_flows=None,
                    flow_imbalance_correction=None, packagedata=None, filename=None,
                    pname=None, **kwargs)

```

Bases: *MFPackage*

ModflowGwtfmi defines a fmi package within a gwt6 model.

Parameters

- **model** (*MFModel*) - Model that this package is a part of. Package is automatically added to model when it is initialized.

- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **save_flows** (*boolean*) -
 - `save_flows` (*boolean*) keyword to indicate that FMI flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **flow_imbalance_correction** (*boolean*) -
 - `flow_imbalance_correction` (*boolean*) correct for an imbalance in flows by assuming that any residual flow error comes in or leaves at the concentration of the cell. When this option is activated, the GWT Model budget written to the listing file will contain two additional entries: FLOW-ERROR and FLOW-CORRECTION. These two entries will be equal but opposite in sign. The FLOW-CORRECTION term is a mass flow that is added to offset the error caused by an imprecise flow balance. If these terms are not relatively small, the flow model should be rerun with stricter convergence tolerances.
- **packagedata** (*[flowtype, fname]*) -
 - `flowtype` (*string*) is the word GWFBUDGET, GWFHEAD, GWFMOVER or the name of an advanced GWF stress package. If GWFBUDGET is specified, then the corresponding file must be a budget file from a previous GWF Model run. If an advanced GWF stress package name appears then the corresponding file must be the budget file saved by a LAK, SFR, MAW or UZF Package.
 - `fname` (*string*) is the name of the file containing flows. The path to the file should be included if the file is not located in the folder where the program was run.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (*mfutl**). For example, *mfutllaktab* package must have a *mfgwflak* package `parent_file`.

```
dfn = [['header'], ['block options', 'name save_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name flow_imbalance_correction', 'type
keyword', 'reader urword', 'optional true'], ['block packagedata', 'name
packagedata', 'type recarray flowtype filein fname', 'reader urword', 'optional
false'], ['block packagedata', 'name flowtype', 'in_record true', 'type string',
'tagged false', 'reader urword'], ['block packagedata', 'name filein', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block packagedata', 'name fname', 'in_record true', 'type string',
'preserve_case true', 'tagged false', 'reader urword']]
```

```
dfn_file_name = 'gwt-fmi.dfn'
```

```
package_abbr = 'gwtfmi'
```

```
packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwtgwt module

```
class ModflowGwtgwt(simulation, loading_package=False, exgtype='GWT6-GWT6',
                    exgmnamea=None, exgmnameb=None, gwfname1=None,
                    gwfname2=None, auxiliary=None, boundnames=None,
                    print_input=None, print_flows=None, save_flows=None, adv_scheme=None,
                    dsp_xt3d_off=None, dsp_xt3d_rhs=None, filein=None, perioddata=None,
                    observations=None, dev_interfacemodel_on=None, nexg=None,
                    exchangedata=None, filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwtgwt defines a gwtgwt package.

Parameters

- **simulation** ([MFSimulation](#)) - Simulation that this package is a part of. Package is automatically added to simulation when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **exgtype** (<string>) -
 - is the exchange type (GWF-GWF or GWF-GWT).
- **exgmnamea** (<string>) -
 - is the name of the first model that is part of this exchange.
- **exgmnameb** (<string>) -
 - is the name of the second model that is part of this exchange.
- **gwfname1** (*string*) -
 - gwfname1 (*string*) keyword to specify name of first corresponding GWF Model. In the simulation name file, the GWT6-GWT6 entry contains names for GWT Models (exgmnamea and exgmnameb). The GWT Model with the name exgmnamea must correspond to the GWF Model with the name gwfname1.
- **gwfname2** (*string*) -
 - gwfname2 (*string*) keyword to specify name of second corresponding GWF Model. In the simulation name file, the GWT6-GWT6 entry contains names for GWT Models (exgmnamea and exgmnameb). The GWT Model with the name exgmnameb must correspond to the GWF Model with the name gwfname2.
- **auxiliary** (*[string]*) -
 - auxiliary (*string*) an array of auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided. Most auxiliary variables will not be used by the GWF-GWF Exchange, but they will be available for use by other parts of the program. If an auxiliary variable with the name "ANGLDEGX" is found, then this information will be used as the angle (provided in degrees) between the connection face normal and the x axis, where a value of zero indicates that a normal vector points directly along the positive x axis. The connection face normal is a normal vector on the cell face

shared between the cell in model 1 and the cell in model 2 pointing away from the model 1 cell. Additional information on "ANGLDEGX" is provided in the description of the DISU Package. If an auxiliary variable with the name "CDIST" is found, then this information will be used as the straight-line connection distance, including the vertical component, between the two cell centers. Both ANGLDEGX and CDIST are required if specific discharge is calculated for either of the groundwater models.

- **boundnames** (*boolean*) -
 - boundnames (boolean) keyword to indicate that boundary names may be provided with the list of GWT Exchange cells.
- **print_input** (*boolean*) -
 - print_input (boolean) keyword to indicate that the list of exchange entries will be echoed to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
 - print_flows (boolean) keyword to indicate that the list of exchange flow rates will be printed to the listing file for every stress period in which "SAVE BUDGET" is specified in Output Control.
- **save_flows** (*boolean*) -
 - save_flows (boolean) keyword to indicate that cell-by-cell flow terms will be written to the budget file for each model provided that the Output Control for the models are set up with the "BUDGET SAVE FILE" option.
- **adv_scheme** (*string*) -
 - adv_scheme (string) scheme used to solve the advection term. Can be upstream, central, or TVD. If not specified, upstream weighting is the default weighting scheme.
- **dsp_xt3d_off** (*boolean*) -
 - dsp_xt3d_off (boolean) deactivate the xt3d method for the dispersive flux and use the faster and less accurate approximation for this exchange.
- **dsp_xt3d_rhs** (*boolean*) -
 - dsp_xt3d_rhs (boolean) add xt3d dispersion terms to right-hand side, when possible, for this exchange.
- **filein** (*boolean*) -
 - filein (boolean) keyword to specify that an input filename is expected next.
- **perioddata** (*{varname:data} or perioddata data*) -
 - Contains data for the mvt package. Data can be stored in a dictionary containing data for the mvt package with variable names as keys and package data as values. Data just for

the `perioddata` variable is also acceptable. See `mvt` package documentation for more information.

- **observations** (*{varname:data} or continuous data*) -
 - Contains data for the `obs` package. Data can be stored in a dictionary containing data for the `obs` package with variable names as keys and package data as values. Data just for the `observations` variable is also acceptable. See `obs` package documentation for more information.
- **dev_interfacemodel_on** (*boolean*) -
 - `dev_interfacemodel_on` (*boolean*) activates the interface model mechanism for calculating the coefficients at (and possibly near) the exchange. This keyword should only be used for development purposes.
- **nexg** (*integer*) -
 - `nexg` (*integer*) keyword and integer value specifying the number of GWT-GWT exchanges.
- **exchangedata** (*[cellidm1, cellidm2, ihc, cl1, cl2, hwva, aux, boundname]*) -
 - `cellidm1` (*(integer, ...)*) is the cellid of the cell in model 1 as specified in the simulation name file. For a structured grid that uses the `DIS` input file, `CELLIDM1` is the layer, row, and column numbers of the cell. For a grid that uses the `DISV` input file, `CELLIDM1` is the layer number and `CELL2D` number for the two cells. If the model uses the unstructured discretization (`DISU`) input file, then `CELLIDM1` is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - `cellidm2` (*(integer, ...)*) is the cellid of the cell in model 2 as specified in the simulation name file. For a structured grid that uses the `DIS` input file, `CELLIDM2` is the layer, row, and column numbers of the cell. For a grid that uses the `DISV` input file, `CELLIDM2` is the layer number and `CELL2D` number for the two cells. If the model uses the unstructured discretization (`DISU`) input file, then `CELLIDM2` is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - `ihc` (*integer*) is an integer flag indicating the direction between node `n` and all of its `m` connections. If `IHC = 0` then the connection is vertical. If `IHC = 1` then the connection is horizontal. If `IHC = 2` then the connection is horizontal for a vertically staggered grid.

- `cl1` (double) is the distance between the center of cell 1 and the its shared face with cell 2.
 - `cl2` (double) is the distance between the center of cell 2 and the its shared face with cell 1.
 - `hwva` (double) is the horizontal width of the flow connection between cell 1 and cell 2 if `IHC > 0`, or it is the area perpendicular to flow of the vertical connection between cell 1 and cell 2 if `IHC = 0`.
 - `aux` (double) represents the values of the auxiliary variables for each GWGWT Exchange. The values of auxiliary variables must be present for each exchange. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block.
 - `boundname` (string) name of the GWT Exchange cell. `BOUNDNAME` is an ASCII character variable that can contain as many as 40 characters. If `BOUNDNAME` contains spaces in it, then the entire name must be enclosed within single quotes.
- **`filename`** (*String*) - File name for this package.
 - **`pname`** (*String*) - Package name for this package.
 - **`parent_file`** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (`mfutl*`). For example, `mfutllaktab` package must have a `mfgwflak` package `parent_file`.
- `auxiliary`** = `<flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

```

dfn = [['header', 'multi-package'], ['block options', 'name gwfmodelname1', 'type
string', 'reader urword', 'optional false'], ['block options', 'name gwfmodelname2',
'type string', 'reader urword', 'optional false'], ['block options', 'name
auxiliary', 'type string', 'shape (naux)', 'reader urword', 'optional true'],
['block options', 'name boundnames', 'type keyword', 'shape', 'reader urword',
'optional true'], ['block options', 'name print_input', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name print_flows', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name save_flows', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name adv_scheme',
'type string', 'valid upstream central tvd', 'reader urword', 'optional true'],
['block options', 'name dsp_xt3d_off', 'type keyword', 'shape', 'reader urword',
'optional true'], ['block options', 'name dsp_xt3d_rhs', 'type keyword', 'shape',
'reader urword', 'optional true'], ['block options', 'name filein', 'type keyword',
'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'],
['block options', 'name mvt_filerecord', 'type record mvt6 filein mvt6_filename',
'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package mvt',
'construct_data perioddata', 'parameter_name perioddata'], ['block options', 'name
mvt6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name mvt6_filename', 'type string',
'preserve_case true', 'in_record true', 'tagged false', 'reader urword', 'optional
false'], ['block options', 'name obs_filerecord', 'type record obs6 filein
obs6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package obs', 'construct_data continuous', 'parameter_name
observations'], ['block options', 'name obs6', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
obs6_filename', 'type string', 'preserve_case true', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block options', 'name
dev_interfacemodel_on', 'type keyword', 'reader urword', 'optional true'], ['block
dimensions', 'name nexg', 'type integer', 'reader urword', 'optional false'],
['block exchangedata', 'name exchangedata', 'type recarray cellidm1 cellidm2 ihc cl1
cl2 hwva aux boundname', 'reader urword', 'optional false'], ['block exchangedata',
'name cellidm1', 'type integer', 'in_record true', 'tagged false', 'reader urword',
'optional false', 'numeric_index true'], ['block exchangedata', 'name cellidm2',
'type integer', 'in_record true', 'tagged false', 'reader urword', 'optional false',
'numeric_index true'], ['block exchangedata', 'name ihc', 'type integer', 'in_record
true', 'tagged false', 'reader urword', 'optional false'], ['block exchangedata',
'name cl1', 'type double precision', 'in_record true', 'tagged false', 'reader
urword', 'optional false'], ['block exchangedata', 'name cl2', 'type double
precision', 'in_record true', 'tagged false', 'reader urword', 'optional false'],
['block exchangedata', 'name hwva', 'type double precision', 'in_record true',
'tagged false', 'reader urword', 'optional false'], ['block exchangedata', 'name
aux', 'type double precision', 'in_record true', 'tagged false', 'shape (naux)',
'reader urword', 'optional true'], ['block exchangedata', 'name boundname', 'type
string', 'shape', 'tagged false', 'in_record true', 'reader urword', 'optional
true']]

```

```
dfn_file_name = 'exg-gwtgwt.dfn'
```

```
exchangedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
mvt_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'gwtgwt'
```

flopy.mf6.modflow.mfgwtic module

```
class ModflowGwtic(model, loading_package=False, strt=0.0, filename=None, pname=None,
                    **kwargs)
```

Bases: [MFPackage](#)

ModflowGwtic defines a ic package within a gwt6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **strt** (*[double]*) -
 - strt (double) is the initial (starting) concentration-that is, concentration at the beginning of the GWT Model simulation. STRT must be specified for all GWT Model simulations. One value is read for every model cell.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** ([MFPackage](#)) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfuflaktab package must have a mfgwflak package parent_file.

```
dfn = [['header'], ['block griddata', 'name strt', 'type double precision', 'shape
(nodes)', 'reader readarray', 'layered true', 'default_value 0.0']]
```

```
dfn_file_name = 'gwt-ic.dfn'
```

```
package_abbr = 'gwtic'
```

```
strt = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

flopy.mf6.modflow.mfgwtist module

```
class ModflowGwtist(model, loading_package=False, save_flows=None,
                    budget_filerecord=None, budgetcsv_filerecord=None, sorption=None,
                    first_order_decay=None, zero_order_decay=None, cim_filerecord=None,
                    cimprintrecord=None, porosity=None, volfrac=None, zetaim=None,
                    cim=None, decay=None, decay_sorbed=None, bulk_density=None,
                    distcoef=None, filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwtist defines a ist package within a gwt6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **save_flows** (*boolean*) -

- `save_flows` (boolean) keyword to indicate that IST flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **`budget_filerecord`** (*[budgetfile]*) -
 - `budgetfile` (string) name of the binary output file to write budget information.
- **`budgetcsv_filerecord`** (*[budgetcsvfile]*) -
 - `budgetcsvfile` (string) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **`sorption`** (*boolean*) -
 - `sorption` (boolean) is a text keyword to indicate that sorption will be activated. Use of this keyword requires that `BULK_DENSITY` and `DISTCOEF` are specified in the `GRIDDATA` block. The linear sorption isotherm is the only isotherm presently supported in the IST Package.
- **`first_order_decay`** (*boolean*) -
 - `first_order_decay` (boolean) is a text keyword to indicate that first- order decay will occur. Use of this keyword requires that `DECAY` and `DECAY_SORBED` (if sorption is active) are specified in the `GRIDDATA` block.
- **`zero_order_decay`** (*boolean*) -
 - `zero_order_decay` (boolean) is a text keyword to indicate that zero- order decay will occur. Use of this keyword requires that `DECAY` and `DECAY_SORBED` (if sorption is active) are specified in the `GRIDDATA` block.
- **`cim_filerecord`** (*[cimfile]*) -
 - `cimfile` (string) name of the output file to write immobile concentrations. This file is a binary file that has the same format and structure as a binary head and concentration file. The value for the text variable written to the file is `CIM`. Immobile domain concentrations will be written to this file at the same interval as mobile domain concentrations are saved, as specified in the GWT Model Output Control file.
- **`cimprintrecord`** (*[columns, width, digits, format]*) -
 - `columns` (integer) number of columns for writing data.
 - `width` (integer) width for writing each number.
 - `digits` (integer) number of digits to use for writing a number.
 - `format` (string) write format can be `EXPONENTIAL`, `FIXED`, `GENERAL`, or `SCIENTIFIC`.
- **`porosity`** (*[double]*) -
 - `porosity` (double) porosity of the immobile domain specified as the immobile domain pore volume per immobile domain volume.

- **volfrac** ([double]) -
 - volfrac (double) fraction of the cell volume that consists of this immobile domain. The sum of all immobile domain volume fractions must be less than one.
- **zetaim** ([double]) -
 - zetaim (double) mass transfer rate coefficient between the mobile and immobile domains, in dimensions of per time.
- **cim** ([double]) -
 - cim (double) initial concentration of the immobile domain in mass per length cubed. If CIM is not specified, then it is assumed to be zero.
- **decay** ([double]) -
 - decay (double) is the rate coefficient for first or zero-order decay for the aqueous phase of the immobile domain. A negative value indicates solute production. The dimensions of decay for first-order decay is one over time. The dimensions of decay for zero-order decay is mass per length cubed per time. Decay will have no effect on simulation results unless either first- or zero-order decay is specified in the options block.
- **decay_sorbed** ([double]) -
 - decay_sorbed (double) is the rate coefficient for first or zero-order decay for the sorbed phase of the immobile domain. A negative value indicates solute production. The dimensions of decay_sorbed for first-order decay is one over time. The dimensions of decay_sorbed for zero-order decay is mass of solute per mass of aquifer per time. If decay_sorbed is not specified and both decay and sorption are active, then the program will terminate with an error. decay_sorbed will have no effect on simulation results unless the SORPTION keyword and either first- or zero-order decay are specified in the options block.
- **bulk_density** ([double]) -
 - bulk_density (double) is the bulk density of this immobile domain in mass per length cubed. Bulk density is defined as the immobile domain solid mass per volume of the immobile domain. bulk_density is not required unless the SORPTION keyword is specified in the options block. If the SORPTION keyword is not specified in the options block, bulk_density will have no effect on simulation results.
- **distcoef** ([double]) -
 - distcoef (double) is the distribution coefficient for the equilibrium-controlled linear sorption isotherm in dimensions of length cubed per mass. distcoef is not required unless the SORPTION keyword is specified in the options block. If the SORPTION keyword is not specified in the options block, distcoef will have no effect on simulation results.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

`budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

`budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

`bulk_density = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>`

`cim = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>`

`cim_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

`cimprintrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

`decay = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>`

`decay_sorbed = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>`

```

dfn = [['header'], ['block options', 'name save_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name budget_filerecord', 'type record
budget fileout budgetfile', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name budget', 'type keyword', 'shape', 'in_record true',
'reader urword', 'tagged true', 'optional false'], ['block options', 'name fileout',
'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name budgetfile', 'type string', 'preserve_case true',
'shape', 'in_record true', 'reader urword', 'tagged false', 'optional false'],
['block options', 'name budgetcsv_filerecord', 'type record budgetcsv fileout
budgetcsvfile', 'shape', 'reader urword', 'tagged true', 'optional true'], ['block
options', 'name budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name budgetcsvfile',
'type string', 'preserve_case true', 'shape', 'in_record true', 'reader urword',
'tagged false', 'optional false'], ['block options', 'name sorption', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name
first_order_decay', 'type keyword', 'reader urword', 'optional true'], ['block
options', 'name zero_order_decay', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name cim_filerecord', 'type record cim fileout cimfile',
'shape', 'reader urword', 'tagged true', 'optional true'], ['block options', 'name
cim', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name cimfile', 'type string', 'preserve_case
true', 'shape', 'in_record true', 'reader urword', 'tagged false', 'optional
false'], ['block options', 'name cimprintrecord', 'type record cim print_format
formatrecord', 'shape', 'reader urword', 'optional true'], ['block options', 'name
print_format', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name formatrecord', 'type record
columns width digits format', 'shape', 'in_record true', 'reader urword', 'tagged',
'optional false'], ['block options', 'name columns', 'type integer', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional'], ['block options',
'name width', 'type integer', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional'], ['block options', 'name digits', 'type integer', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional'], ['block options',
'name format', 'type string', 'shape', 'in_record true', 'reader urword', 'tagged
false', 'optional false'], ['block griddata', 'name porosity', 'type double
precision', 'shape (nodes)', 'reader readarray', 'layered true'], ['block griddata',
'name volfrac', 'type double precision', 'shape (nodes)', 'reader readarray',
'layered true'], ['block griddata', 'name zetaim', 'type double precision', 'shape
(nodes)', 'reader readarray', 'layered true'], ['block griddata', 'name cim', 'type
double precision', 'shape (nodes)', 'reader readarray', 'optional true', 'layered
true'], ['block griddata', 'name decay', 'type double precision', 'shape (nodes)',
'reader readarray', 'layered true', 'optional true'], ['block griddata', 'name
decay_sorbed', 'type double precision', 'shape (nodes)', 'reader readarray',
'optional true', 'layered true'], ['block griddata', 'name bulk_density', 'type
double precision', 'shape (nodes)', 'reader readarray', 'optional true', 'layered
true'], ['block griddata', 'name distcoef', 'type double precision', 'shape
(nodes)', 'reader readarray', 'optional true', 'layered true']]

dfn_file_name = 'gwt-ist.dfn'

distcoef = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

package_abbr = 'gwtist'

porosity = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

```

`volfrac = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>`

`zetaim = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>`

`flopy.mf6.modflow.mfgwtlkt` module

```
class ModflowGwtlkt(model, loading_package=False, flow_package_name=None, auxiliary=None,
                    flow_package_auxiliary_name=None, boundnames=None, print_input=None,
                    print_concentration=None, print_flows=None, save_flows=None,
                    concentration_filerecord=None, budget_filerecord=None,
                    budgetcsv_filerecord=None, timeseries=None, observations=None,
                    packagedata=None, lakeperioddata=None, filename=None, pname=None,
                    **kwargs)
```

Bases: `MFPackage`

ModflowGwtlkt defines a lkt package within a gwt6 model.

Parameters

- **model** (`MFModel`) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (`bool`) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **flow_package_name** (`string`) -
 - `flow_package_name` (`string`) keyword to specify the name of the corresponding flow package. If not specified, then the corresponding flow package must have the same name as this advanced transport package (the name associated with this package in the GWT name file).
- **auxiliary** (`[string]`) -
 - `auxiliary` (`string`) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for `naux`. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **flow_package_auxiliary_name** (`string`) -
 - `flow_package_auxiliary_name` (`string`) keyword to specify the name of an auxiliary variable in the corresponding flow package. If specified, then the simulated concentrations from this advanced transport package will be copied into the auxiliary variable specified with this name. Note that the flow package must have an auxiliary variable with this name or the program will terminate with an error. If the flows

for this advanced transport package are read from a file, then this option will have no effect.

- **boundnames** (*boolean*) -
 - boundnames (*boolean*) keyword to indicate that boundary names may be provided with the list of lake cells.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of lake information will be written to the listing file immediately after it is read.
- **print_concentration** (*boolean*) -
 - print_concentration (*boolean*) keyword to indicate that the list of lake concentration will be printed to the listing file for every stress period in which "CONCENTRATION PRINT" is specified in Output Control. If there is no Output Control option and PRINT_CONCENTRATION is specified, then concentration are printed for the last time step of each stress period.
- **print_flows** (*boolean*) -
 - print_flows (*boolean*) keyword to indicate that the list of lake flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (*boolean*) keyword to indicate that lake flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **concentration_filerecord** (*[concfile]*) -
 - concfile (*string*) name of the binary output file to write concentration information.
- **budget_filerecord** (*[budgetfile]*) -
 - budgetfile (*string*) name of the binary output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
 - budgetcsvfile (*string*) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **timeseries** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.

- **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **packagedata** (*[lakeno, strt, aux, boundname]*) -
 - *lakeno* (integer) integer value that defines the lake number associated with the specified PACKAGEDATA data on the line. LAKENO must be greater than zero and less than or equal to NLAKES. Lake information must be specified for every lake or the program will terminate with an error. The program will also terminate with an error if information for a lake is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *strt* (double) real value that defines the starting concentration for the lake.
 - *aux* (double) represents the values of the auxiliary variables for each lake. The values of auxiliary variables must be present for each lake. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - *boundname* (string) name of the lake cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.
- **lakeperioddata** (*[lakeno, laksetting]*) -
 - *lakeno* (integer) integer value that defines the lake number associated with the specified PERIOD data on the line. LAKENO must be greater than zero and less than or equal to NLAKES. This argument is an index variable, which means that it should be treated as zero- based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *laksetting* (keystring) line of information that is parsed into a keyword and values. Keyword values that can be used to start the LAKSETTING string include: STATUS, CONCENTRATION, RAINFALL, EVAPORATION, RUNOFF, and AUXILIARY. These settings are used to assign the concentration of associated with the corresponding flow terms. Concentrations cannot be specified for all flow terms. For example, the Lake Package supports a “WITHDRAWAL” flow term. If this withdrawal term is active, then water will be withdrawn from the lake at the calculated

concentration of the lake.

status

[[string]]

- * status (string) keyword option to define lake status. STATUS can be ACTIVE, INACTIVE, or CONSTANT. By default, STATUS is ACTIVE, which means that concentration will be calculated for the lake. If a lake is inactive, then there will be no solute mass fluxes into or out of the lake and the inactive value will be written for the lake concentration. If a lake is constant, then the concentration for the lake will be fixed at the user specified value.

concentration

[[string]]

- * concentration (string) real or character value that defines the concentration for the lake. The specified CONCENTRATION is only applied if the lake is a constant concentration lake. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

rainfall

[[string]]

- * rainfall (string) real or character value that defines the rainfall solute concentration (ML^{-3}) for the lake. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

evaporation

[[string]]

- * evaporation (string) real or character value that defines the concentration of evaporated water (ML^{-3}) for the lake. If this concentration value is larger than the simulated concentration in the lake, then the evaporated water will be removed at the same concentration as the lake. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

runoff

[[string]]

- * runoff (string) real or character value that defines the concentration of runoff (ML^{-3}) for the lake. Value must be greater than or equal to zero. If the Options block includes a TIMESERIESFILE entry

(see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

ext_inflow

[[string]]

* ext-inflow (string) real or character value that defines the concentration of external inflow (ML^{-3}) for the lake. Value must be greater than or equal to zero. If the Options block includes a TIMESERIESFILE entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

auxiliaryrecord

[[auxname, auxval]]

* auxname (string) name for the auxiliary variable to be assigned AUXVAL. AUXNAME must match one of the auxiliary variable names defined in the OPTIONS block. If AUXNAME does not match one of the auxiliary variable names defined in the OPTIONS block the data are ignored.

* auxval (double) value for the auxiliary variable. If the Options block includes a TIMESERIESFILE entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfuillaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

concentration_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

dfn = [['header', 'multi-package'], ['block options', 'name flow_package_name',
'type string', 'shape', 'reader urword', 'optional true'], ['block options', 'name
auxiliary', 'type string', 'shape (naux)', 'reader urword', 'optional true'],
['block options', 'name flow_package_auxiliary_name', 'type string', 'shape',
'reader urword', 'optional true'], ['block options', 'name boundnames', 'type
keyword', 'shape', 'reader urword', 'optional true'], ['block options', 'name
print_input', 'type keyword', 'reader urword', 'optional true'], ['block options',
'name print_concentration', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name print_flows', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name save_flows', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name concentration_filerecord', 'type record
concentration fileout concfile', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name concentration', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
concfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budget_filerecord', 'type record budget fileout budgetfile', 'shape', 'reader
urword', 'tagged true', 'optional true'], ['block options', 'name budget', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name fileout', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
budgetfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budgetcsv_filerecord', 'type record budgetcsv fileout budgetcsvfile', 'shape',
'reader urword', 'tagged true', 'optional true'], ['block options', 'name
budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name budgetcsvfile', 'type string',
'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged false',
'optional false'], ['block options', 'name ts_filerecord', 'type record ts6 filein
ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package ts', 'construct_data timeseries', 'parameter_name timeseries'],
['block options', 'name ts6', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name ts6_filename', 'type string', 'preserve_case true',
'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block
options', 'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape',
'reader urword', 'tagged true', 'optional true', 'construct_package obs',
'construct_data continuous', 'parameter_name observations'], ['block options', 'name
obs6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name obs6_filename', 'type string',
'preserve_case true', 'in_record true', 'tagged false', 'reader urword', 'optional
false'], ['block packagedata', 'name packagedata', 'type recarray lakeno strt aux
boundname', 'shape (maxbound)', 'reader urword'], ['block packagedata', 'name
lakeno', 'type integer', 'shape', 'tagged false', 'in_record true', 'reader urword',
'numeric_index true'], ['block packagedata', 'name strt', 'type double precision',
'shape', 'tagged false', 'in_record true', 'reader urword'], ['block packagedata',
'name aux', 'type double precision', 'in_record true', 'tagged false', 'shape
(naux)', 'reader urword', 'time_series true', 'optional true'], ['block
packagedata', 'name boundname', 'type string', 'shape', 'tagged false', 'in_record
true', 'reader urword', 'optional true'], ['block period', 'name iper', 'type
integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid',
'reader urword', 'optional false'], ['block period', 'name lakeperioddata', 'type
recarray lakeno laksetting', 'shape', 'reader urword'], ['block period', 'name
lakeno', 'type integer', 'shape', 'tagged false', 'in_record true', 'reader urword',
'numeric_index true'], ['block period', 'name laksetting', 'type keystack status
concentration rainfall evaporation runoff ext-inflow auxiliaryrecord', 'shape', 1507
'tagged false', 'in_record true', 'reader urword'], ['block period', 'name status',
'type string', 'shape', 'tagged true', 'in_record true', 'reader urword'], ['block
period', 'name concentration', 'type string', 'shape', 'tagged true', 'in_record

```

```
dfn_file_name = 'gwt-lkt.dfn'

lakeperioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwtlkt'

packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfmgwtmst module

```
class ModflowGwtmst(model, loading_package=False, save_flows=None,
                    first_order_decay=None, zero_order_decay=None, sorption=None,
                    porosity=None, decay=None, decay_sorbed=None, bulk_density=None,
                    distcoef=None, sp2=None, filename=None, pname=None, **kwargs)
```

Bases: [*MFPackage*](#)

ModflowGwtmst defines a mst package within a gwt6 model.

Parameters

- **model** ([*MFModel*](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **save_flows** (*boolean*) -
 - save_flows (*boolean*) keyword to indicate that MST flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **first_order_decay** (*boolean*) -
 - first_order_decay (*boolean*) is a text keyword to indicate that first- order decay will occur. Use of this keyword requires that DECAY and DECAY_SORBED (if sorption is active) are specified in the GRIDDATA block.
- **zero_order_decay** (*boolean*) -
 - zero_order_decay (*boolean*) is a text keyword to indicate that zero- order decay will occur. Use of this keyword requires that DECAY and DECAY_SORBED (if sorption is active) are specified in the GRIDDATA block.
- **sorption** (*string*) -
 - sorption (*string*) is a text keyword to indicate that sorption will be activated. Valid sorption options include LINEAR, FREUNDLICH, and LANGMUIR. Use of this keyword requires that BULK_DENSITY and DISTCOEF are specified in the GRIDDATA block. If sorption is specified as FREUNDLICH or LANGMUIR then SP2 is also required in the GRIDDATA block.
- **porosity** (*[double]*) -

- porosity (double) is the mobile domain porosity, defined as the mobile domain pore volume per mobile domain volume. Additional information on porosity within the context of mobile and immobile domain transport simulations is included in the MODFLOW 6 Supplemental Technical Information document.
 - **decay** ([double]) -
 - decay (double) is the rate coefficient for first or zero-order decay for the aqueous phase of the mobile domain. A negative value indicates solute production. The dimensions of decay for first-order decay is one over time. The dimensions of decay for zero-order decay is mass per length cubed per time. decay will have no effect on simulation results unless either first- or zero-order decay is specified in the options block.
 - **decay_sorbed** ([double]) -
 - decay_sorbed (double) is the rate coefficient for first or zero-order decay for the sorbed phase of the mobile domain. A negative value indicates solute production. The dimensions of decay_sorbed for first-order decay is one over time. The dimensions of decay_sorbed for zero-order decay is mass of solute per mass of aquifer per time. If decay_sorbed is not specified and both decay and sorption are active, then the program will terminate with an error. decay_sorbed will have no effect on simulation results unless the SORPTION keyword and either first- or zero-order decay are specified in the options block.
 - **bulk_density** ([double]) -
 - bulk_density (double) is the bulk density of the aquifer in mass per length cubed. bulk_density is not required unless the SORPTION keyword is specified. Bulk density is defined as the mobile domain solid mass per mobile domain volume. Additional information on bulk density is included in the MODFLOW 6 Supplemental Technical Information document.
 - **distcoef** ([double]) -
 - distcoef (double) is the distribution coefficient for the equilibrium-controlled linear sorption isotherm in dimensions of length cubed per mass. distcoef is not required unless the SORPTION keyword is specified.
 - **sp2** ([double]) -
 - sp2 (double) is the exponent for the Freundlich isotherm and the sorption capacity for the Langmuir isotherm.
 - **filename** (String) - File name for this package.
 - **pname** (String) - Package name for this package.
 - **parent_file** (MFPackage) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulllaktab package must have a mfgwflak package parent_file.
- bulk_density** = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

```
decay = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

decay_sorbed = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

dfn = [['header'], ['block options', 'name save_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name first_order_decay', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name
zero_order_decay', 'type keyword', 'reader urword', 'optional true'], ['block
options', 'name sorption', 'type string', 'valid linear freundlich langmuir',
'reader urword', 'optional true'], ['block griddata', 'name porosity', 'type double
precision', 'shape (nodes)', 'reader readarray', 'layered true'], ['block griddata',
'name decay', 'type double precision', 'shape (nodes)', 'reader readarray', 'layered
true', 'optional true'], ['block griddata', 'name decay_sorbed', 'type double
precision', 'shape (nodes)', 'reader readarray', 'optional true', 'layered true'],
['block griddata', 'name bulk_density', 'type double precision', 'shape (nodes)',
'reader readarray', 'optional true', 'layered true'], ['block griddata', 'name
distcoef', 'type double precision', 'shape (nodes)', 'reader readarray', 'layered
true', 'optional true'], ['block griddata', 'name sp2', 'type double precision',
'shape (nodes)', 'reader readarray', 'layered true', 'optional true']]

dfn_file_name = 'gwt-mst.dfn'

distcoef = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

package_abbr = 'gwtmst'

porosity = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

sp2 = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

flopy.mf6.modflow.mfmgwtmvt module

```
class GwtmvtPackages(model_or_sim, parent, pkg_type, filerecord, package=None,
                    package_class=None)
```

Bases: [*MFChildPackages*](#)

GwtmvtPackages is a container class for the ModflowGwtmvt class.

initialize()

Initializes a new ModflowGwtmvt package removing any sibling child packages attached to the same parent package. See ModflowGwtmvt init documentation for definition of parameters.

append_package()

Adds a new ModflowGwtmvt package to the container. See ModflowGwtmvt init documentation for definition of parameters.

```
append_package(print_input=None, print_flows=None, save_flows=None,
               budget_filerecord=None, budgetcsv_filerecord=None, filename=None,
               pname=None)
```

```
initialize(print_input=None, print_flows=None, save_flows=None,
           budget_filerecord=None, budgetcsv_filerecord=None, filename=None,
           pname=None)
```



```
package_abbr = 'gwtmvtpackages'
```

```
class ModflowGwtmvt(parent_model_or_package, loading_package=False, print_input=None,
                    print_flows=None, save_flows=None, budget_filerecord=None,
                    budgetcsv_filerecord=None, filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwtmvt defines a mvt package within a gwt6 model.

Parameters

- **parent_model_or_package** (*MFModel/MFPackage*) -
Parent_model_or_package that this package is a part of. Package is automatically added to parent_model_or_package when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **print_input** (*boolean*) -
- print_input (boolean) keyword to indicate that the list of mover information will be written to the listing file immediately after it is read.
- **print_flows** (*boolean*) -
- print_flows (boolean) keyword to indicate that the list of lake flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
- save_flows (boolean) keyword to indicate that lake flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **budget_filerecord** (*[budgetfile]*) -
- budgetfile (string) name of the binary output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
- budgetcsvfile (string) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulaktab package must have a mfgwflak package parent_file.

```
budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
dfn = [['header'], ['block options', 'name print_input', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name print_flows', 'type keyword',
'reader urword', 'optional true'], ['block options', 'name save_flows', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name
budget_filerecord', 'type record budget fileout budgetfile', 'shape', 'reader
urword', 'tagged true', 'optional true'], ['block options', 'name budget', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name fileout', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
budgetfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budgetcsv_filerecord', 'type record budgetcsv fileout budgetcsvfile', 'shape',
'reader urword', 'tagged true', 'optional true'], ['block options', 'name
budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name budgetcsvfile', 'type string',
'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged false',
'optional false']]

dfn_file_name = 'gwt-mvt.dfn'

package_abbr = 'gwtmvt'
```

flopy.mf6.modflow.mfgwtmwt module

```
class ModflowGwtmwt(model, loading_package=False, flow_package_name=None, auxiliary=None,
    flow_package_auxiliary_name=None, boundnames=None, print_input=None,
    print_concentration=None, print_flows=None, save_flows=None,
    concentration_filerecord=None, budget_filerecord=None,
    budgetcsv_filerecord=None, timeseries=None, observations=None,
    packagedata=None, mwtperioddata=None, filename=None, pname=None,
    **kwargs)
```

Bases: [*MFPackage*](#)

ModflowGwtmwt defines a mwt package within a gwt6 model.

Parameters

- **model** ([*MFModel*](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** ([*bool*](#)) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **flow_package_name** ([*string*](#)) -
 - **flow_package_name** ([*string*](#)) keyword to specify the name of the corresponding flow package. If not specified, then the corresponding flow package must have the same name as this advanced transport package (the name associated with this package in the GWT name file).
- **auxiliary** ([*\[string\]*](#)) -
 - **auxiliary** ([*string*](#)) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined

here. The number of auxiliary variables detected on this line determines the value for `naux`. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.

- **flow_package_auxiliary_name** (*string*) -
 - `flow_package_auxiliary_name` (*string*) keyword to specify the name of an auxiliary variable in the corresponding flow package. If specified, then the simulated concentrations from this advanced transport package will be copied into the auxiliary variable specified with this name. Note that the flow package must have an auxiliary variable with this name or the program will terminate with an error. If the flows for this advanced transport package are read from a file, then this option will have no effect.
- **boundnames** (*boolean*) -
 - `boundnames` (*boolean*) keyword to indicate that boundary names may be provided with the list of well cells.
- **print_input** (*boolean*) -
 - `print_input` (*boolean*) keyword to indicate that the list of well information will be written to the listing file immediately after it is read.
- **print_concentration** (*boolean*) -
 - `print_concentration` (*boolean*) keyword to indicate that the list of well concentration will be printed to the listing file for every stress period in which "CONCENTRATION PRINT" is specified in Output Control. If there is no Output Control option and `PRINT_CONCENTRATION` is specified, then concentration are printed for the last time step of each stress period.
- **print_flows** (*boolean*) -
 - `print_flows` (*boolean*) keyword to indicate that the list of well flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - `save_flows` (*boolean*) keyword to indicate that well flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **concentration_filerecord** (*[concfile]*) -
 - `concfile` (*string*) name of the binary output file to write concentration information.

- **budget_filerecord** (*[budgetfile]*) -
 - budgetfile (string) name of the binary output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
 - budgetcsvfile (string) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **timeseries** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **packagedata** (*[mawno, strt, aux, boundname]*) -
 - mawno (integer) integer value that defines the well number associated with the specified PACKAGEDATA data on the line. MAWNO must be greater than zero and less than or equal to NMAWWELLS. Well information must be specified for every well or the program will terminate with an error. The program will also terminate with an error if information for a well is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - strt (double) real value that defines the starting concentration for the well.
 - aux (double) represents the values of the auxiliary variables for each well. The values of auxiliary variables must be present for each well. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - boundname (string) name of the well cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.
- **mwtperioddata** (*[mawno, mwtsetting]*) -

- `mawno` (integer) integer value that defines the well number associated with the specified PERIOD data on the line. `MAWNO` must be greater than zero and less than or equal to `NMAWWELLS`. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.
- `mwtsetting` (keystring) line of information that is parsed into a keyword and values. Keyword values that can be used to start the MWTSETTING string include: `STATUS`, `CONCENTRATION`, `RAINFALL`, `EVAPORATION`, `RUNOFF`, and `AUXILIARY`. These settings are used to assign the concentration associated with the corresponding flow terms. Concentrations cannot be specified for all flow terms. For example, the Multi-Aquifer Well Package supports a "WITHDRAWAL" flow term. If this withdrawal term is active, then water will be withdrawn from the well at the calculated concentration of the well.

status

[[string]]

- * `status` (string) keyword option to define well status. `STATUS` can be `ACTIVE`, `INACTIVE`, or `CONSTANT`. By default, `STATUS` is `ACTIVE`, which means that concentration will be calculated for the well. If a well is inactive, then there will be no solute mass fluxes into or out of the well and the inactive value will be written for the well concentration. If a well is constant, then the concentration for the well will be fixed at the user specified value.

concentration

[[string]]

- * `concentration` (string) real or character value that defines the concentration for the well. The specified `CONCENTRATION` is only applied if the well is a constant concentration well. If the Options block includes a `TIMESERIESFILE` entry (see the "Time-Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

rate

[[string]]

- * `rate` (string) real or character value that defines the injection solute concentration (ML^{-3}) for the well. If the Options block includes a `TIMESERIESFILE` entry (see the "Time-Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

auxiliaryrecord

[[auxname, auxval]]

- * `auxname` (string) name for the auxiliary variable to be assigned `AUXVAL`. `AUXNAME` must match one of the auxiliary variable names defined in the `OPTIONS` block. If `AUXNAME` does not match one of the auxiliary variable names defined in the `OPTIONS` block the data are ignored.
- * `auxval` (double) value for the auxiliary variable. If the Options block includes a `TIMESERIESFILE` entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- `filename` (*String*) - File name for this package.
- `pname` (*String*) - Package name for this package.
- `parent_file` (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (`mfutl*`). For example, `mfutllaktab` package must have a `mfgwflak` package `parent_file`.

`auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

`budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

`budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

`concentration_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

```

dfn = [['header', 'multi-package'], ['block options', 'name flow_package_name',
'type string', 'shape', 'reader urword', 'optional true'], ['block options', 'name
auxiliary', 'type string', 'shape (naux)', 'reader urword', 'optional true'],
['block options', 'name flow_package_auxiliary_name', 'type string', 'shape',
'reader urword', 'optional true'], ['block options', 'name boundnames', 'type
keyword', 'shape', 'reader urword', 'optional true'], ['block options', 'name
print_input', 'type keyword', 'reader urword', 'optional true'], ['block options',
'name print_concentration', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name print_flows', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name save_flows', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name concentration_filerecord', 'type record
concentration fileout concfile', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name concentration', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
concfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budget_filerecord', 'type record budget fileout budgetfile', 'shape', 'reader
urword', 'tagged true', 'optional true'], ['block options', 'name budget', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name fileout', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
budgetfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budgetcsv_filerecord', 'type record budgetcsv fileout budgetcsvfile', 'shape',
'reader urword', 'tagged true', 'optional true'], ['block options', 'name
budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name budgetcsvfile', 'type string',
'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged false',
'optional false'], ['block options', 'name ts_filerecord', 'type record ts6 filein
ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package ts', 'construct_data timeseries', 'parameter_name timeseries'],
['block options', 'name ts6', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name ts6_filename', 'type string', 'preserve_case true',
'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block
options', 'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape',
'reader urword', 'tagged true', 'optional true', 'construct_package obs',
'construct_data continuous', 'parameter_name observations'], ['block options', 'name
obs6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name obs6_filename', 'type string',
'preserve_case true', 'in_record true', 'tagged false', 'reader urword', 'optional
false'], ['block packagedata', 'name packagedata', 'type recarray mawno strt aux
boundname', 'shape (maxbound)', 'reader urword'], ['block packagedata', 'name
mawno', 'type integer', 'shape', 'tagged false', 'in_record true', 'reader urword',
'numeric_index true'], ['block packagedata', 'name strt', 'type double precision',
'shape', 'tagged false', 'in_record true', 'reader urword'], ['block packagedata',
'name aux', 'type double precision', 'in_record true', 'tagged false', 'shape
(naux)', 'reader urword', 'time_series true', 'optional true'], ['block
packagedata', 'name boundname', 'type string', 'shape', 'tagged false', 'in_record
true', 'reader urword', 'optional true'], ['block period', 'name iper', 'type
integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid',
'reader urword', 'optional false'], ['block period', 'name mwtperioddata', 'type
recarray mawno mwtsetting', 'shape', 'reader urword'], ['block period', 'name
mawno', 'type integer', 'shape', 'tagged false', 'in_record true', 'reader urword',
'numeric_index true'], ['block period', 'name mwtsetting', 'type keystack status
concentration rate auxiliaryrecord', 'shape', 'tagged false', 'in_record true', 1517
'reader urword'], ['block period', 'name status', 'type string', 'shape', 'tagged
true', 'in_record true', 'reader urword'], ['block period', 'name concentration',
'type string', 'shape', 'tagged true', 'in_record true', 'time_series true', 'reader

```

```
dfn_file_name = 'gwt-mwt.dfn'
mwtperioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
package_abbr = 'gwtmwt'
packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfmgwt_{nam} module

```
class ModflowGwtnam(model, loading_package=False, list=None, print_input=None,
                    print_flows=None, save_flows=None, packages=None, filename=None,
                    pname=None, **kwargs)
```

Bases: [*MFPackage*](#)

ModflowGwt_{nam} defines a *nam* package within a gwt6 model.

Parameters

- **model** ([*MFModel*](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** ([*bool*](#)) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **list** ([*string*](#)) -
 - **list** ([*string*](#)) is name of the listing file to create for this GWT model. If not specified, then the name of the list file will be the basename of the GWT model name file and the '.lst' extension. For example, if the GWT name file is called "my.model.nam" then the list file will be called "my.model.lst".
- **print_input** ([*boolean*](#)) -
 - **print_input** ([*boolean*](#)) keyword to indicate that the list of all model stress package information will be written to the listing file immediately after it is read.
- **print_flows** ([*boolean*](#)) -
 - **print_flows** ([*boolean*](#)) keyword to indicate that the list of all model package flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** ([*boolean*](#)) -
 - **save_flows** ([*boolean*](#)) keyword to indicate that all model package flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **packages** ([*\[ftype, fname, pname\]*](#)) -

- ftype (string) is the file type, which must be one of the following character values shown in table in mf6io.pdf. Ftype may be entered in any combination of uppercase and lowercase.
- fname (string) is the name of the file containing the package input. The path to the file should be included if the file is not located in the folder where the program was run.
- pname (string) is the user-defined name for the package. PNAME is restricted to 16 characters. No spaces are allowed in PNAME. PNAME character values are read and stored by the program for stress packages only. These names may be useful for labeling purposes when multiple stress packages of the same type are located within a single GWT Model. If PNAME is specified for a stress package, then PNAME will be used in the flow budget table in the listing file; it will also be used for the text entry in the cell-by-cell budget file. PNAME is case insensitive and is stored in all upper case letters.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

```
dfn = [['header'], ['block options', 'name list', 'type string', 'reader urword',
'optional true', 'preserve_case true'], ['block options', 'name print_input', 'type
keyword', 'reader urword', 'optional true'], ['block options', 'name print_flows',
'type keyword', 'reader urword', 'optional true'], ['block options', 'name
save_flows', 'type keyword', 'reader urword', 'optional true'], ['block packages',
'name packages', 'type recarray ftype fname pname', 'reader urword', 'optional
false'], ['block packages', 'name ftype', 'in_record true', 'type string', 'tagged
false', 'reader urword'], ['block packages', 'name fname', 'in_record true', 'type
string', 'preserve_case true', 'tagged false', 'reader urword'], ['block packages',
'name pname', 'in_record true', 'type string', 'tagged false', 'reader urword',
'optional true']]
```

```
dfn_file_name = 'gwt-nam.dfn'
```

```
package_abbr = 'gwtnam'
```

```
packages = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwtoc module

```
class ModflowGwtoc(model, loading_package=False, budget_filerecord=None,
                    budgetcsv_filerecord=None, concentration_filerecord=None,
                    concentrationprintrecord=None, saverecord=None, printrecord=None,
                    filename=None, pname=None, **kwargs)
```

Bases: *MFPackage*

ModflowGwtoc defines a oc package within a gwt6 model.

Parameters

- **model** (*MFModel*) - Model that this package is a part of. Package is automatically added to model when it is initialized.

- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **budget_filerecord** (*[budgetfile]*) -
 - *budgetfile* (string) name of the output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
 - *budgetcsvfile* (string) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **concentration_filerecord** (*[concentrationfile]*) -
 - *concentrationfile* (string) name of the output file to write conc information.
- **concentrationprintrecord** (*[columns, width, digits, format]*) -
 - *columns* (integer) number of columns for writing data.
 - *width* (integer) width for writing each number.
 - *digits* (integer) number of digits to use for writing a number.
 - *format* (string) write format can be EXPONENTIAL, FIXED, GENERAL, or SCIENTIFIC.
- **saverecord** (*[rtype, ocsetting]*) -
 - *rtype* (string) type of information to save or print. Can be BUDGET or CONCENTRATION.
 - *ocsetting* (keystring) specifies the steps for which the data will be saved.

all

[[keyword]]

- * *all* (keyword) keyword to indicate save for all time steps in period.

first

[[keyword]]

- * *first* (keyword) keyword to indicate save for first step in period. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

last

[[keyword]]

- * *last* (keyword) keyword to indicate save for last step in period. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

frequency

[[integer]]

* frequency (integer) save at the specified time step frequency. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

steps

[[integer]]

* steps (integer) save for each step specified in STEPS. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

• **printrecord** ([*rtype*, *ocsetting*]) -

- *rtype* (string) type of information to save or print. Can be BUDGET or CONCENTRATION.

- *ocsetting* (keysting) specifies the steps for which the data will be saved.

all

[[keyword]]

* all (keyword) keyword to indicate save for all time steps in period.

first

[[keyword]]

* first (keyword) keyword to indicate save for first step in period. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

last

[[keyword]]

* last (keyword) keyword to indicate save for last step in period. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

frequency

[[integer]]

* frequency (integer) save at the specified time step frequency. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

steps

[[integer]]

* steps (integer) save for each step specified in STEPS. This keyword may be used in conjunction with other keywords to print or save results for multiple time steps.

• **filename** (*String*) - File name for this package.

• **pname** (*String*) - Package name for this package.

- `parent_file` ([MFPackage](#)) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfuillaktab package must have a mfgwflak package `parent_file`.

```
budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
concentration_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
concentrationprintrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```

dfn = [['header'], ['block options', 'name budget_filerecord', 'type record budget
fileout budgetfile', 'shape', 'reader urword', 'tagged true', 'optional true'],
['block options', 'name budget', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name fileout', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name budgetfile', 'type string', 'preserve_case true',
'shape', 'in_record true', 'reader urword', 'tagged false', 'optional false'],
['block options', 'name budgetcsv_filerecord', 'type record budgetcsv fileout
budgetcsvfile', 'shape', 'reader urword', 'tagged true', 'optional true'], ['block
options', 'name budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name budgetcsvfile',
'type string', 'preserve_case true', 'shape', 'in_record true', 'reader urword',
'tagged false', 'optional false'], ['block options', 'name
concentration_filerecord', 'type record concentration fileout concentrationfile',
'shape', 'reader urword', 'tagged true', 'optional true'], ['block options', 'name
concentration', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name concentrationfile', 'type string',
'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged false',
'optional false'], ['block options', 'name concentrationprintrecord', 'type record
concentration print_format formatrecord', 'shape', 'reader urword', 'optional
true'], ['block options', 'name print_format', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
formatrecord', 'type record columns width digits format', 'shape', 'in_record true',
'reader urword', 'tagged', 'optional false'], ['block options', 'name columns',
'type integer', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional'], ['block options', 'name width', 'type integer', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional'], ['block options', 'name digits',
'type integer', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional'], ['block options', 'name format', 'type string', 'shape', 'in_record
true', 'reader urword', 'tagged false', 'optional false'], ['block period', 'name
iper', 'type integer', 'block_variable True', 'in_record true', 'tagged false',
'shape', 'valid', 'reader urword', 'optional false'], ['block period', 'name
saverecord', 'type record save rtype ocsetting', 'shape', 'reader urword', 'tagged
false', 'optional true'], ['block period', 'name save', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
period', 'name printrecord', 'type record print rtype ocsetting', 'shape', 'reader
urword', 'tagged false', 'optional true'], ['block period', 'name print', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block period', 'name rtype', 'type string', 'shape', 'in_record true',
'reader urword', 'tagged false', 'optional false'], ['block period', 'name
ocsetting', 'type keystring all first last frequency steps', 'shape', 'tagged
false', 'in_record true', 'reader urword'], ['block period', 'name all', 'type
keyword', 'shape', 'in_record true', 'reader urword'], ['block period', 'name
first', 'type keyword', 'shape', 'in_record true', 'reader urword'], ['block
period', 'name last', 'type keyword', 'shape', 'in_record true', 'reader urword'],
['block period', 'name frequency', 'type integer', 'shape', 'tagged true',
'in_record true', 'reader urword'], ['block period', 'name steps', 'type integer',
'shape (<nstp)', 'tagged true', 'in_record true', 'reader urword']]

dfn_file_name = 'gwt-oc.dfn'

package_abbr = 'gwtoc'

printrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

```
saverecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwtsft module

```
class ModflowGwtsft(model, loading_package=False, flow_package_name=None, auxiliary=None,
                    flow_package_auxiliary_name=None, boundnames=None, print_input=None,
                    print_concentration=None, print_flows=None, save_flows=None,
                    concentration_filerecord=None, budget_filerecord=None,
                    budgetcsv_filerecord=None, timeseries=None, observations=None,
                    packagedata=None, reachperioddata=None, filename=None, pname=None,
                    **kwargs)
```

Bases: [*MFPackage*](#)

ModflowGwtsft defines a sft package within a gwt6 model.

Parameters

- **model** ([*MFModel*](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **flow_package_name** (*string*) -
 - **flow_package_name** (*string*) keyword to specify the name of the corresponding flow package. If not specified, then the corresponding flow package must have the same name as this advanced transport package (the name associated with this package in the GWT name file).
- **auxiliary** (*[string]*) -
 - **auxiliary** (*string*) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for `naux`. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **flow_package_auxiliary_name** (*string*) -
 - **flow_package_auxiliary_name** (*string*) keyword to specify the name of an auxiliary variable in the corresponding flow package. If specified, then the simulated concentrations from this advanced transport package will be copied into the auxiliary variable specified with this name. Note that the flow package must have an auxiliary variable with this name or the program will terminate with an error. If the flows for this advanced transport package are read from a file, then this option will have no effect.

- **boundnames** (*boolean*) -
 - boundnames (*boolean*) keyword to indicate that boundary names may be provided with the list of reach cells.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of reach information will be written to the listing file immediately after it is read.
- **print_concentration** (*boolean*) -
 - print_concentration (*boolean*) keyword to indicate that the list of reach concentration will be printed to the listing file for every stress period in which "CONCENTRATION PRINT" is specified in Output Control. If there is no Output Control option and PRINT_CONCENTRATION is specified, then concentration are printed for the last time step of each stress period.
- **print_flows** (*boolean*) -
 - print_flows (*boolean*) keyword to indicate that the list of reach flow rates will be printed to the listing file for every stress period time step in which "BUDGET PRINT" is specified in Output Control. If there is no Output Control option and "PRINT_FLOWS" is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (*boolean*) keyword to indicate that reach flow terms will be written to the file specified with "BUDGET FILEOUT" in Output Control.
- **concentration_filerecord** (*[concfile]*) -
 - concfile (*string*) name of the binary output file to write concentration information.
- **budget_filerecord** (*[budgetfile]*) -
 - budgetfile (*string*) name of the binary output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
 - budgetcsvfile (*string*) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **timeseries** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data}* or *continuous data*) -

- Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **packagedata** (*[rno, strt, aux, boundname]*) -
 - rno (integer) integer value that defines the reach number associated with the specified PACKAGEDATA data on the line. RNO must be greater than zero and less than or equal to NREACHES. Reach information must be specified for every reach or the program will terminate with an error. The program will also terminate with an error if information for a reach is specified more than once. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - strt (double) real value that defines the starting concentration for the reach.
 - aux (double) represents the values of the auxiliary variables for each reach. The values of auxiliary variables must be present for each reach. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - boundname (string) name of the reach cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.
- **reachperioddata** (*[rno, reachsetting]*) -
 - rno (integer) integer value that defines the reach number associated with the specified PERIOD data on the line. RNO must be greater than zero and less than or equal to NREACHES. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - reachsetting (keystring) line of information that is parsed into a keyword and values. Keyword values that can be used to start the REACHSETTING string include: STATUS, CONCENTRATION, RAINFALL, EVAPORATION, RUNOFF, and AUXILIARY. These settings are used to assign the concentration of associated with the corresponding flow terms. Concentrations cannot be specified for all flow terms. For example, the Streamflow Package supports a “DIVERSION” flow term. Diversion water will be routed using the calculated concentration of the reach.

status

[[string]]

* status (string) keyword option to define reach status. STATUS can be ACTIVE, INACTIVE, or CONSTANT. By default, STATUS is ACTIVE, which means that concentration will be calculated for the reach. If a reach is inactive, then there will be no solute mass fluxes into or out of the reach and the inactive value will be written for the reach concentration. If a reach is constant, then the concentration for the reach will be fixed at the user specified value.

concentration

[[string]]

* concentration (string) real or character value that defines the concentration for the reach. The specified CONCENTRATION is only applied if the reach is a constant concentration reach. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

rainfall

[[string]]

* rainfall (string) real or character value that defines the rainfall solute concentration (ML^{-3}) for the reach. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

evaporation

[[string]]

* evaporation (string) real or character value that defines the concentration of evaporated water (ML^{-3}) for the reach. If this concentration value is larger than the simulated concentration in the reach, then the evaporated water will be removed at the same concentration as the reach. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

runoff

[[string]]

* runoff (string) real or character value that defines the concentration of runoff (ML^{-3}) for the reach. Value must be greater than or equal to zero. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values

can be obtained from a time series by entering the time-series name in place of a numeric value.

inflow

[[string]]

* inflow (string) real or character value that defines the concentration of inflow (ML^{-3}) for the reach. Value must be greater than or equal to zero. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

auxiliaryrecord

[[auxname, auxval]]

* auxname (string) name for the auxiliary variable to be assigned AUXVAL. AUXNAME must match one of the auxiliary variable names defined in the OPTIONS block. If AUXNAME does not match one of the auxiliary variable names defined in the OPTIONS block the data are ignored.

* auxval (double) value for the auxiliary variable. If the Options block includes a TIMESERIESFILE entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfuillaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

concentration_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```

dfn = [['header', 'multi-package'], ['block options', 'name flow_package_name',
'type string', 'shape', 'reader urword', 'optional true'], ['block options', 'name
auxiliary', 'type string', 'shape (naux)', 'reader urword', 'optional true'],
['block options', 'name flow_package_auxiliary_name', 'type string', 'shape',
'reader urword', 'optional true'], ['block options', 'name boundnames', 'type
keyword', 'shape', 'reader urword', 'optional true'], ['block options', 'name
print_input', 'type keyword', 'reader urword', 'optional true'], ['block options',
'name print_concentration', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name print_flows', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name save_flows', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name concentration_filerecord', 'type record
concentration fileout concfile', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name concentration', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
concfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budget_filerecord', 'type record budget fileout budgetfile', 'shape', 'reader
urword', 'tagged true', 'optional true'], ['block options', 'name budget', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name fileout', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
budgetfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budgetcsv_filerecord', 'type record budgetcsv fileout budgetcsvfile', 'shape',
'reader urword', 'tagged true', 'optional true'], ['block options', 'name
budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name budgetcsvfile', 'type string',
'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged false',
'optional false'], ['block options', 'name ts_filerecord', 'type record ts6 filein
ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package ts', 'construct_data timeseries', 'parameter_name timeseries'],
['block options', 'name ts6', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name ts6_filename', 'type string', 'preserve_case true',
'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block
options', 'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape',
'reader urword', 'tagged true', 'optional true', 'construct_package obs',
'construct_data continuous', 'parameter_name observations'], ['block options', 'name
obs6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name obs6_filename', 'type string',
'preserve_case true', 'in_record true', 'tagged false', 'reader urword', 'optional
false'], ['block packagedata', 'name packagedata', 'type recarray rno strt aux
boundname', 'shape (maxbound)', 'reader urword'], ['block packagedata', 'name rno',
'type integer', 'shape', 'tagged false', 'in_record true', 'reader urword',
'numeric_index true'], ['block packagedata', 'name strt', 'type double precision',
'shape', 'tagged false', 'in_record true', 'reader urword'], ['block packagedata',
'name aux', 'type double precision', 'in_record true', 'tagged false', 'shape
(naux)', 'reader urword', 'time_series true', 'optional true'], ['block
packagedata', 'name boundname', 'type string', 'shape', 'tagged false', 'in_record
true', 'reader urword', 'optional true'], ['block period', 'name iper', 'type
integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid',
'reader urword', 'optional false'], ['block period', 'name reachperioddata', 'type
recarray rno reachsetting', 'shape', 'reader urword'], ['block period', 'name rno',
'type integer', 'shape', 'tagged false', 'in_record true', 'reader urword',
'numeric_index true'], ['block period', 'name reachsetting', 'type keystring status
concentration rainfall evaporation runoff inflow auxiliaryrecord', 'shape', 'tagged
false', 'in_record true', 'reader urword'], ['block period', 'name status', 'type
string', 'shape', 'tagged true', 'in_record true', 'reader urword'], ['block
period', 'name concentration', 'type string', 'shape', 'tagged true', 'in_record

```

```
dfn_file_name = 'gwt-sft.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwtsft'

packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

reachperioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwtsrc module

```
class ModflowGwtsrc(model, loading_package=False, auxiliary=None, auxmultname=None,
                    boundnames=None, print_input=None, print_flows=None, save_flows=None,
                    timeseries=None, observations=None, maxbound=None,
                    stress_period_data=None, filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwtsrc defines a src package within a gwt6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **auxiliary** (*[string]*) -
 - auxiliary (*string*) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for naux. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **auxmultname** (*string*) -
 - auxmultname (*string*) name of auxiliary variable to be used as multiplier of mass loading rate.
- **boundnames** (*boolean*) -
 - boundnames (*boolean*) keyword to indicate that boundary names may be provided with the list of mass source cells.
- **print_input** (*boolean*) -

- `print_input` (boolean) keyword to indicate that the list of mass source information will be written to the listing file immediately after it is read.
- **`print_flows`** (boolean) -
 - `print_flows` (boolean) keyword to indicate that the list of mass source flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **`save_flows`** (boolean) -
 - `save_flows` (boolean) keyword to indicate that mass source flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **`timeseries`** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **`observations`** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **`maxbound`** (integer) -
 - `maxbound` (integer) integer value specifying the maximum number of sources cells that will be specified for use during any stress period.
- **`stress_period_data`** (*[cellid, smassrate, aux, boundname]*) -
 - `cellid` ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - `smassrate` (double) is the mass source loading rate. A positive value indicates addition of solute mass and a negative value indicates removal of solute mass. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from

a time series by entering the time-series name in place of a numeric value.

- `aux` (double) represents the values of the auxiliary variables for each mass source. The values of auxiliary variables must be present for each mass source. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
- `boundname` (string) name of the mass source cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.

- `filename` (*String*) - File name for this package.
- `pname` (*String*) - Package name for this package.
- `parent_file` (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

`auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>`

```
dfn = [['header'], ['block options', 'name auxiliary', 'type string', 'shape
(naux)', 'reader urword', 'optional true'], ['block options', 'name auxmultname',
'type string', 'shape', 'reader urword', 'optional true'], ['block options', 'name
boundnames', 'type keyword', 'shape', 'reader urword', 'optional true'], ['block
options', 'name print_input', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name print_flows', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name save_flows', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name ts_filerecord', 'type record ts6 filein
ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package ts', 'construct_data timeseries', 'parameter_name timeseries'],
['block options', 'name ts6', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name ts6_filename', 'type string', 'preserve_case true',
'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block
options', 'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape',
'reader urword', 'tagged true', 'optional true', 'construct_package obs',
'construct_data continuous', 'parameter_name observations'], ['block options', 'name
obs6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name obs6_filename', 'type string',
'preserve_case true', 'in_record true', 'tagged false', 'reader urword', 'optional
false'], ['block dimensions', 'name maxbound', 'type integer', 'reader urword',
'optional false'], ['block period', 'name iper', 'type integer', 'block_variable
True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader urword',
'optional false'], ['block period', 'name stress_period_data', 'type recarray cellid
smassrate aux boundname', 'shape (maxbound)', 'reader urword'], ['block period',
'name cellid', 'type integer', 'shape (ncelldim)', 'tagged false', 'in_record true',
'reader urword'], ['block period', 'name smassrate', 'type double precision',
'shape', 'tagged false', 'in_record true', 'reader urword', 'time_series true'],
['block period', 'name aux', 'type double precision', 'in_record true', 'tagged
false', 'shape (naux)', 'reader urword', 'optional true', 'time_series true'],
['block period', 'name boundname', 'type string', 'shape', 'tagged false',
'in_record true', 'reader urword', 'optional true']]
```

```
dfn_file_name = 'gwt-src.dfn'
```

```
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'gwtsrc'
```

```
stress_period_data = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwtssm module

```
class ModflowGwtssm(model, loading_package=False, print_flows=None, save_flows=None,
                    sources=None, fileinput=None, filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowGwtssm defines a ssm package within a gwt6 model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.

- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **print_flows** (*boolean*) -
 - **print_flows** (*boolean*) keyword to indicate that the list of SSM flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - **save_flows** (*boolean*) keyword to indicate that SSM flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **sources** (*[pname, srctype, auxname]*) -
 - **pname** (*string*) name of the flow package for which an auxiliary variable contains a source concentration. If this flow package is represented using an advanced transport package (SFT, LKT, MWT, or UZT), then the advanced transport package will override SSM terms specified here.
 - **srctype** (*string*) keyword indicating how concentration will be assigned for sources and sinks. Keyword must be specified as either AUX or AUXMIXED. For both options the user must provide an auxiliary variable in the corresponding flow package. The auxiliary variable must have the same name as the AUXNAME value that follows. If the AUX keyword is specified, then the auxiliary variable specified by the user will be assigned as the concentration value for groundwater sources (flows with a positive sign). For negative flow rates (sinks), groundwater will be withdrawn from the cell at the simulated concentration of the cell. The AUXMIXED option provides an alternative method for how to determine the concentration of sinks. If the cell concentration is larger than the user-specified auxiliary concentration, then the concentration of groundwater withdrawn from the cell will be assigned as the user-specified concentration. Alternatively, if the user-specified auxiliary concentration is larger than the cell concentration, then groundwater will be withdrawn at the cell concentration. Thus, the AUXMIXED option is designed to work with the Evapotranspiration (EVT) and Recharge (RCH) Packages where water may be withdrawn at a concentration that is less than the cell concentration.
 - **auxname** (*string*) name of the auxiliary variable in the package PNAME. This auxiliary variable must exist and be specified by the user in that package. The values in this auxiliary variable will be used to set the concentration associated with the flows for that boundary package.
- **fileinput** (*[pname, spc6_filename]*) -
 - **pname** (*string*) name of the flow package for which an SPC6 input file contains a source concentration. If this flow

package is represented using an advanced transport package (SFT, LKT, MWT, or UZT), then the advanced transport package will override SSM terms specified here.

- `spc6_filename` (string) character string that defines the path and filename for the file containing source and sink input data for the flow package. The `SPC6_FILENAME` file is a flexible input file that allows concentrations to be specified by stress period and with time series. Instructions for creating the `SPC6_FILENAME` input file are provided in the next section on file input for boundary concentrations.

- `filename` (String) - File name for this package.
- `pname` (String) - Package name for this package.
- `parent_file` (MFPackage) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, `mfutllaktab` package must have a `mfgwflak` package `parent_file`.

```
dfn = [['header'], ['block options', 'name print_flows', 'type keyword', 'reader
urword', 'optional true'], ['block options', 'name save_flows', 'type keyword',
'reader urword', 'optional true'], ['block sources', 'name sources', 'type recarray
pname srctype auxname', 'reader urword', 'optional false'], ['block sources', 'name
pname', 'in_record true', 'type string', 'tagged false', 'reader urword'], ['block
sources', 'name srctype', 'in_record true', 'type string', 'tagged false', 'optional
false', 'reader urword'], ['block sources', 'name auxname', 'in_record true', 'type
string', 'tagged false', 'reader urword', 'optional false'], ['block fileinput',
'name fileinput', 'type recarray pname spc6 filein spc6_filename mixed', 'reader
urword'], ['block fileinput', 'name pname', 'in_record true', 'type string', 'tagged
false', 'reader urword'], ['block fileinput', 'name spc6', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block
fileinput', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block fileinput', 'name spc6_filename',
'type string', 'preserve_case true', 'in_record true', 'reader urword', 'optional
false', 'tagged false'], ['block fileinput', 'name mixed', 'type keyword', 'shape',
'in_record true', 'reader urword', 'tagged true', 'optional true']]
```

```
dfn_file_name = 'gwt-ssm.dfn'
```

```
fileinput = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
package_abbr = 'gwtssm'
```

```
sources = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfgwtuzt module

```
class ModflowGwtuzt(model, loading_package=False, flow_package_name=None, auxiliary=None,
    flow_package_auxiliary_name=None, boundnames=None, print_input=None,
    print_concentration=None, print_flows=None, save_flows=None,
    concentration_filerecord=None, budget_filerecord=None,
    budgetcsv_filerecord=None, timeseries=None, observations=None,
    packagedata=None, uztperioddata=None, filename=None, pname=None,
    **kwargs)
```

Bases: [MFPackage](#)

ModflowGwtuzt defines a uzt package within a gwt6 model.

Parameters

- **model** (*MFMModel*) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **flow_package_name** (*string*) -
 - **flow_package_name** (*string*) keyword to specify the name of the corresponding flow package. If not specified, then the corresponding flow package must have the same name as this advanced transport package (the name associated with this package in the GWT name file).
- **auxiliary** (*[string]*) -
 - **auxiliary** (*string*) defines an array of one or more auxiliary variable names. There is no limit on the number of auxiliary variables that can be provided on this line; however, lists of information provided in subsequent blocks must have a column of data for each auxiliary variable name defined here. The number of auxiliary variables detected on this line determines the value for *naux*. Comments cannot be provided anywhere on this line as they will be interpreted as auxiliary variable names. Auxiliary variables may not be used by the package, but they will be available for use by other parts of the program. The program will terminate with an error if auxiliary variables are specified on more than one line in the options block.
- **flow_package_auxiliary_name** (*string*) -
 - **flow_package_auxiliary_name** (*string*) keyword to specify the name of an auxiliary variable in the corresponding flow package. If specified, then the simulated concentrations from this advanced transport package will be copied into the auxiliary variable specified with this name. Note that the flow package must have an auxiliary variable with this name or the program will terminate with an error. If the flows for this advanced transport package are read from a file, then this option will have no effect.
- **boundnames** (*boolean*) -
 - **boundnames** (*boolean*) keyword to indicate that boundary names may be provided with the list of unsaturated zone flow cells.
- **print_input** (*boolean*) -
 - **print_input** (*boolean*) keyword to indicate that the list of unsaturated zone flow information will be written to the listing file immediately after it is read.
- **print_concentration** (*boolean*) -
 - **print_concentration** (*boolean*) keyword to indicate that the list of UZF cell concentration will be printed to the listing file for every stress period in which "CONCENTRATION PRINT"

is specified in Output Control. If there is no Output Control option and PRINT_CONCENTRATION is specified, then concentration are printed for the last time step of each stress period.

- **print_flows** (*boolean*) -
 - print_flows (*boolean*) keyword to indicate that the list of unsaturated zone flow rates will be printed to the listing file for every stress period time step in which “BUDGET PRINT” is specified in Output Control. If there is no Output Control option and “PRINT_FLOWS” is specified, then flow rates are printed for the last time step of each stress period.
- **save_flows** (*boolean*) -
 - save_flows (*boolean*) keyword to indicate that unsaturated zone flow terms will be written to the file specified with “BUDGET FILEOUT” in Output Control.
- **concentration_filerecord** (*[concfile]*) -
 - concfile (*string*) name of the binary output file to write concentration information.
- **budget_filerecord** (*[budgetfile]*) -
 - budgetfile (*string*) name of the binary output file to write budget information.
- **budgetcsv_filerecord** (*[budgetcsvfile]*) -
 - budgetcsvfile (*string*) name of the comma-separated value (CSV) output file to write budget summary information. A budget summary record will be written to this file for each time step of the simulation.
- **timeseries** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **observations** (*{varname:data}* or *continuous data*) -
 - Contains data for the obs package. Data can be stored in a dictionary containing data for the obs package with variable names as keys and package data as values. Data just for the observations variable is also acceptable. See obs package documentation for more information.
- **packagedata** (*[uzfno, strt, aux, boundname]*) -
 - uzfno (*integer*) integer value that defines the UZF cell number associated with the specified PACKAGEDATA data on the line. UZFNO must be greater than zero and less than or equal to NUZFCELLS. Unsaturated zone flow information must be specified for every UZF cell or the program will terminate with an error. The program will also terminate with an error if information for a UZF cell is specified more than once. This

- argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.
- strt (double) real value that defines the starting concentration for the unsaturated zone flow cell.
 - aux (double) represents the values of the auxiliary variables for each unsaturated zone flow. The values of auxiliary variables must be present for each unsaturated zone flow. The values must be specified in the order of the auxiliary variables specified in the OPTIONS block. If the package supports time series and the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.
 - boundname (string) name of the unsaturated zone flow cell. BOUNDNAME is an ASCII character variable that can contain as many as 40 characters. If BOUNDNAME contains spaces in it, then the entire name must be enclosed within single quotes.
- **uztperioddata** ([uzfno, uztsetting]) -
 - uzfno (integer) integer value that defines the UZF cell number associated with the specified PERIOD data on the line. UZFNO must be greater than zero and less than or equal to NUZFCELLS. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. FloPy will automatically subtract one when loading index variables and add one when writing index variables.
 - uztsetting (keystring) line of information that is parsed into a keyword and values. Keyword values that can be used to start the UZTSETTING string include: STATUS, CONCENTRATION, INFILTRATION, UZET, and AUXILIARY. These settings are used to assign the concentration of associated with the corresponding flow terms. Concentrations cannot be specified for all flow terms.
- status**
[[string]]
- * status (string) keyword option to define UZF cell status. STATUS can be ACTIVE, INACTIVE, or CONSTANT. By default, STATUS is ACTIVE, which means that concentration will be calculated for the UZF cell. If a UZF cell is inactive, then there will be no solute mass fluxes into or out of the UZF cell and the inactive value will be written for the UZF cell concentration. If a UZF cell is constant, then the concentration for the UZF cell will be fixed at the user specified value.
- concentration**
[[string]]

* concentration (string) real or character value that defines the concentration for the unsaturated zone flow cell. The specified CONCENTRATION is only applied if the unsaturated zone flow cell is a constant concentration cell. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

infiltration

[[string]]

* infiltration (string) real or character value that defines the infiltration solute concentration (ML^{-3}) for the UZF cell. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

uzet

[[string]]

* uzet (string) real or character value that defines the concentration of unsaturated zone evapotranspiration water (ML^{-3}) for the UZF cell. If this concentration value is larger than the simulated concentration in the UZF cell, then the unsaturated zone ET water will be removed at the same concentration as the UZF cell. If the Options block includes a TIMESERIESFILE entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

auxiliaryrecord

[[auxname, auxval]]

* auxname (string) name for the auxiliary variable to be assigned AUXVAL. AUXNAME must match one of the auxiliary variable names defined in the OPTIONS block. If AUXNAME does not match one of the auxiliary variable names defined in the OPTIONS block the data are ignored.

* auxval (double) value for the auxiliary variable. If the Options block includes a TIMESERIESFILE entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

```
auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
budgetcsv_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
concentration_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```

dfn = [['header', 'multi-package'], ['block options', 'name flow_package_name',
'type string', 'shape', 'reader urword', 'optional true'], ['block options', 'name
auxiliary', 'type string', 'shape (naux)', 'reader urword', 'optional true'],
['block options', 'name flow_package_auxiliary_name', 'type string', 'shape',
'reader urword', 'optional true'], ['block options', 'name boundnames', 'type
keyword', 'shape', 'reader urword', 'optional true'], ['block options', 'name
print_input', 'type keyword', 'reader urword', 'optional true'], ['block options',
'name print_concentration', 'type keyword', 'reader urword', 'optional true'],
['block options', 'name print_flows', 'type keyword', 'reader urword', 'optional
true'], ['block options', 'name save_flows', 'type keyword', 'reader urword',
'optional true'], ['block options', 'name concentration_filerecord', 'type record
concentration fileout concfile', 'shape', 'reader urword', 'tagged true', 'optional
true'], ['block options', 'name concentration', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
concfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budget_filerecord', 'type record budget fileout budgetfile', 'shape', 'reader
urword', 'tagged true', 'optional true'], ['block options', 'name budget', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name fileout', 'type keyword', 'shape', 'in_record
true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name
budgetfile', 'type string', 'preserve_case true', 'shape', 'in_record true', 'reader
urword', 'tagged false', 'optional false'], ['block options', 'name
budgetcsv_filerecord', 'type record budgetcsv fileout budgetcsvfile', 'shape',
'reader urword', 'tagged true', 'optional true'], ['block options', 'name
budgetcsv', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block options', 'name budgetcsvfile', 'type string',
'preserve_case true', 'shape', 'in_record true', 'reader urword', 'tagged false',
'optional false'], ['block options', 'name ts_filerecord', 'type record ts6 filein
ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional true',
'construct_package ts', 'construct_data timeseries', 'parameter_name timeseries'],
['block options', 'name ts6', 'type keyword', 'shape', 'in_record true', 'reader
urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name ts6_filename', 'type string', 'preserve_case true',
'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block
options', 'name obs_filerecord', 'type record obs6 filein obs6_filename', 'shape',
'reader urword', 'tagged true', 'optional true', 'construct_package obs',
'construct_data continuous', 'parameter_name observations'], ['block options', 'name
obs6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true',
'optional false'], ['block options', 'name obs6_filename', 'type string',
'preserve_case true', 'in_record true', 'tagged false', 'reader urword', 'optional
false'], ['block packagedata', 'name packagedata', 'type recarray uzfn0 strt aux
boundname', 'shape (maxbound)', 'reader urword'], ['block packagedata', 'name
uzfn0', 'type integer', 'shape', 'tagged false', 'in_record true', 'reader urword',
'numeric_index true'], ['block packagedata', 'name strt', 'type double precision',
'shape', 'tagged false', 'in_record true', 'reader urword'], ['block packagedata',
'name aux', 'type double precision', 'in_record true', 'tagged false', 'shape
(naux)', 'reader urword', 'time_series true', 'optional true'], ['block
packagedata', 'name boundname', 'type string', 'shape', 'tagged false', 'in_record
true', 'reader urword', 'optional true'], ['block period', 'name iper', 'type
integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid',
'reader urword', 'optional false'], ['block period', 'name uztperioddata', 'type
recarray uzfn0 uztsetting', 'shape', 'reader urword'], ['block period', 'name
uzfn0', 'type integer', 'shape', 'tagged false', 'in_record true', 'reader urword',
'numeric_index true'], ['block period', 'name uztsetting', 'type keystack status
concentration infiltration uzet auxiliaryrecord', 'shape', 'tagged false',
'in_record true', 'reader urword'], ['block period', 'name status', 'type string',
'shape', 'tagged true', 'in_record true', 'reader urword'], ['block period', 'name
concentration', 'type string', 'shape', 'tagged true', 'in_record true',

```

```
dfn_file_name = 'gwt-uzt.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwtuzt'

packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

uztperioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

8.1.7 MODFLOW 6 Utility Packages

MODFLOW 6 has several utility packages that can be associated with other packages. This includes the obs package, which can be used to output model results specific to its parent package, and the time series and time array series packages, which can be used to provide time series input for other packages.

Contents:

flopy.mf6.modflow.mfutlats module

```
class ModflowUtlats(parent_package, loading_package=False, maxats=1, perioddata=None,
                    filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowUtlats defines a ats package within a utl model.

Parameters

- **parent_package** ([MFPackage](#)) - Parent_package that this package is a part of. Package is automatically added to parent_package when it is initialized.
- **loading_package** ([bool](#)) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **maxats** ([integer](#)) -
 - maxats ([integer](#)) is the number of records in the subsequent perioddata block that will be used for adaptive time stepping.
- **perioddata** ([\[iperats, dt0, dtmin, dtmax, dtadj, dtfailadj\]](#)) -
 - iperats ([integer](#)) is the period number to designate for adaptive time stepping. The remaining ATS values on this line will apply to period iperats. iperats must be greater than zero. A warning is printed if iperats is greater than nper. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - dt0 ([double](#)) is the initial time step length for period iperats. If dt0 is zero, then the final step from the previous stress period will be used as the initial time step.

- The program will terminate with an error message if `dt0` is negative.
- `dtmin` (double) is the minimum time step length for this period. This value must be greater than zero and less than `dtmax`. `dtmin` must be a small value in order to ensure that simulation times end at the end of stress periods and the end of the simulation. A small value, such as 1.e-5, is recommended.
 - `dtmax` (double) is the maximum time step length for this period. This value must be greater than `dtmin`.
 - `dtadj` (double) is the time step multiplier factor for this period. If the number of outer solver iterations are less than the product of the maximum number of outer iterations (`OUTER_MAXIMUM`) and `ATS_OUTER_MAXIMUM_FRACTION` (an optional variable in the IMS input file with a default value of 1/3), then the time step length is multiplied by `dtadj`. If the number of outer solver iterations are greater than the product of the maximum number of outer iterations and `ATS_OUTER_MAXIMUM_FRACTION`, then the time step length is divided by `dtadj`. `dtadj` must be zero, one, or greater than one. If `dtadj` is zero or one, then it has no effect on the simulation. A value between 2.0 and 5.0 can be used as an initial estimate.
 - `dtfailadj` (double) is the divisor of the time step length when a time step fails to converge. If there is solver failure, then the time step will be tried again with a shorter time step length calculated as the previous time step length divided by `dtfailadj`. `dtfailadj` must be zero, one, or greater than one. If `dtfailadj` is zero or one, then time steps will not be retried with shorter lengths. In this case, the program will terminate with an error, or it will continue if the `CONTINUE` option is set in the simulation name file. Initial tests with this variable should be set to 5.0 or larger to determine if convergence can be achieved.
- **filename** (*String*) - File name for this package.
 - **pname** (*String*) - Package name for this package.
 - **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (*mfutl**). For example, *mfutllaktab* package must have a *mfgwflak* package *parent_file*.

```
dfn = [['header'], ['block dimensions', 'name maxats', 'type integer', 'reader
urword', 'optional false', 'default_value 1'], ['block perioddata', 'name
perioddata', 'type recarray iperats dt0 dtmin dtmax dtadj dtfailadj', 'reader
urword', 'optional false'], ['block perioddata', 'name iperats', 'type integer',
'in_record true', 'tagged false', 'reader urword', 'optional false', 'numeric_index
true'], ['block perioddata', 'name dt0', 'type double precision', 'in_record true',
'tagged false', 'reader urword', 'optional false'], ['block perioddata', 'name
dtmin', 'type double precision', 'in_record true', 'tagged false', 'reader urword',
'optional false'], ['block perioddata', 'name dtmax', 'type double precision',
'in_record true', 'tagged false', 'reader urword', 'optional false'], ['block
perioddata', 'name dtadj', 'type double precision', 'in_record true', 'tagged
false', 'reader urword', 'optional false'], ['block perioddata', 'name dtfailadj',
'type double precision', 'in_record true', 'tagged false', 'reader urword',
'optional false']]
```

```
dfn_file_name = 'utl-ats.dfn'
```

```
package_abbr = 'utlats'
```

```
perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
class UtlatsPackages(model_or_sim, parent, pkg_type, filerecord, package=None,
                    package_class=None)
```

Bases: [MFChildPackages](#)

UtlatsPackages is a container class for the ModflowUtlats class.

initialize()

Initializes a new ModflowUtlats package removing any sibling child packages attached to the same parent package. See ModflowUtlats init documentation for definition of parameters.

append_package()

Adds a new ModflowUtlats package to the container. See ModflowUtlats init documentation for definition of parameters.

```
append_package(maxats=1, perioddata=None, filename=None, pname=None)
```

```
initialize(maxats=1, perioddata=None, filename=None, pname=None)
```

```
package_abbr = 'utlatspackages'
```

flopy.mf6.modflow.mfutilaktab module

```
class ModflowUtlaktab(model, loading_package=False, nrow=None, ncol=None, table=None,
                    filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowUtlaktab defines a laktab package within a utl model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.

- **nrow** (*integer*) -
 - nrow (*integer*) integer value specifying the number of rows in the lake table. There must be NROW rows of data in the TABLE block.
- **ncol** (*integer*) -
 - ncol (*integer*) integer value specifying the number of columns in the lake table. There must be NCOL columns of data in the TABLE block. For lakes with HORIZONTAL and/or VERTICAL CTYPE connections, NCOL must be equal to 3. For lakes with EMBEDDEDH or EMBEDDEDV CTYPE connections, NCOL must be equal to 4.
- **table** (*[stage, volume, sarea, barea]*) -
 - stage (*double*) real value that defines the stage corresponding to the remaining data on the line.
 - volume (*double*) real value that defines the lake volume corresponding to the stage specified on the line.
 - sarea (*double*) real value that defines the lake surface area corresponding to the stage specified on the line.
 - barea (*double*) real value that defines the lake-GWF exchange area corresponding to the stage specified on the line. BAREA is only specified if the CLAKTYPE for the lake is EMBEDDEDH or EMBEDDEDV.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

```
dfn = [['header', 'multi-package'], ['block dimensions', 'name nrow', 'type
integer', 'reader urword', 'optional false'], ['block dimensions', 'name ncol',
'type integer', 'reader urword', 'optional false'], ['block table', 'name table',
'type recarray stage volume sarea barea', 'shape (nrow)', 'reader urword'], ['block
table', 'name stage', 'type double precision', 'shape', 'tagged false', 'in_record
true', 'reader urword'], ['block table', 'name volume', 'type double precision',
'shape', 'tagged false', 'in_record true', 'reader urword'], ['block table', 'name
sarea', 'type double precision', 'shape', 'tagged false', 'in_record true', 'reader
urword'], ['block table', 'name barea', 'type double precision', 'shape', 'tagged
false', 'in_record true', 'reader urword', 'optional true']]
```

```
dfn_file_name = 'utl-laktab.dfn'
```

```
package_abbr = 'utllaktab'
```

```
table = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfutlobs module

```
class ModflowUtlobs(parent_model_or_package, loading_package=False, digits=None,
                    print_input=None, continuous=None, filename=None, pname=None,
                    **kwargs)
```

Bases: [*MFPackage*](#)

ModflowUtlobs defines a obs package within a utl model.

Parameters

- **parent_model_or_package** (*MFModel/MFPackage*) -
Parent_model_or_package that this package is a part of. Package is automatically added to parent_model_or_package when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **digits** (*integer*) -
 - digits (*integer*) Keyword and an integer digits specifier used for conversion of simulated values to text on output. If not specified, the default is the maximum number of digits stored in the program (as written with the G0 Fortran specifier). When simulated values are written to a comma-separated value text file specified in a CONTINUOUS block below, the digits specifier controls the number of significant digits with which simulated values are written to the output file. The digits specifier has no effect on the number of significant digits with which the simulation time is written for continuous observations. If DIGITS is specified as zero, then observations are written with the default setting, which is the maximum number of digits.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of observation information will be written to the listing file immediately after it is read.
- **continuous** (*[obsname, obstype, id, id2]*) -
 - obsname (*string*) string of 1 to 40 nonblank characters used to identify the observation. The identifier need not be unique; however, identification and post-processing of observations in the output files are facilitated if each observation is given a unique name.
 - obstype (*string*) a string of characters used to identify the observation type.
 - id (*string*) Text identifying cell where observation is located. For packages other than NPF, if boundary names are defined in the corresponding package input file, ID can be a boundary name. Otherwise ID is a cellid. If the model discretization is type DIS, cellid is three integers (layer, row, column). If the discretization is DISV, cellid is two integers (layer, cell number). If the discretization is DISU, cellid is one integer (node number). This argument

is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.

- `id2` (string) Text identifying cell adjacent to cell identified by ID. The form of ID2 is as described for ID. ID2 is used for intercell- flow observations of a GWF model, for three observation types of the LAK Package, for two observation types of the MAW Package, and one observation type of the UZF Package. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.

- `filename` (String) - File name for this package.
- `pname` (String) - Package name for this package.
- `parent_file` (MFPackage) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulaktab package must have a mfgwflak package `parent_file`.

`continuous` = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```
dfn = [['header', 'multi-package'], ['block options', 'name digits', 'type integer',
'shape', 'reader urword', 'optional true'], ['block options', 'name print_input',
'type keyword', 'reader urword', 'optional true'], ['block continuous', 'name
output', 'type record fileout obs_output_file_name binary', 'shape', 'block_variable
true', 'in_record false', 'reader urword', 'optional false'], ['block continuous',
'name fileout', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged
true', 'optional false'], ['block continuous', 'name obs_output_file_name', 'type
string', 'preserve_case true', 'in_record true', 'shape', 'tagged false', 'reader
urword'], ['block continuous', 'name binary', 'type keyword', 'in_record true',
'shape', 'reader urword', 'optional true'], ['block continuous', 'name continuous',
'type recarray obsname obstype id id2', 'shape', 'reader urword', 'optional false'],
['block continuous', 'name obsname', 'type string', 'shape', 'tagged false',
'in_record true', 'reader urword'], ['block continuous', 'name obstype', 'type
string', 'shape', 'tagged false', 'in_record true', 'reader urword'], ['block
continuous', 'name id', 'type string', 'shape', 'tagged false', 'in_record true',
'reader urword', 'numeric_index true'], ['block continuous', 'name id2', 'type
string', 'shape', 'tagged false', 'in_record true', 'reader urword', 'optional
true', 'numeric_index true']]
```

```
dfn_file_name = 'utl-obs.dfn'
```

```
package_abbr = 'utlobs'
```

```
class UtlobsPackages(model_or_sim, parent, pkg_type, filerecord, package=None,
package_class=None)
```

Bases: [MFChildPackages](#)

UtlobsPackages is a container class for the ModflowUtlobs class.

`initialize()`

Initializes a new ModflowUtlobs package removing any sibling child packages attached to the same parent package. See ModflowUtlobs init documentation for definition of parameters.

```
initialize(digits=None, print_input=None, continuous=None, filename=None,
           pname=None)
```

```
package_abbr = 'utlobspackages'
```

flopy.mf6.modflow.mfutlsfrtab module

```
class ModflowUtlsfrtab(model, loading_package=False, nrow=None, ncol=None, table=None,
                       filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowUtlsfrtab defines a sfrtab package within a utl model.

Parameters

- **model** ([MFModel](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **nrow** (*integer*) -
 - nrow (*integer*) integer value specifying the number of rows in the reach cross-section table. There must be NROW rows of data in the TABLE block.
- **ncol** (*integer*) -
 - ncol (*integer*) integer value specifying the number of columns in the reach cross-section table. There must be NCOL columns of data in the TABLE block. NCOL must be equal to 2 if MANFRACTION is not specified or 3 otherwise.
- **table** (*[xfraction, height, manfraction]*) -
 - xfraction (*double*) real value that defines the station (x) data for the cross-section as a fraction of the width (RWID) of the reach. XFRACTION must be greater than or equal to zero but can be greater than one. XFRACTION values can be used to decrease or increase the width of a reach from the specified reach width (RWID).
 - height (*double*) real value that is the height relative to the top of the lowest elevation of the streambed (RTP) and corresponding to the station data on the same line. HEIGHT must be greater than or equal to zero and at least one cross-section height must be equal to zero.
 - manfraction (*double*) real value that defines the Manning's roughness coefficient data for the cross-section as a fraction of the Manning's roughness coefficient for the reach (MAN) and corresponding to the station data on the same line. MANFRACTION must be greater than zero. MANFRACTION is applied from the XFRACTION value on the same line to the XFRACTION value on the next line. Although a MANFRACTION value is specified on the last line, any value greater than zero can be applied to MANFRACTION(NROW). MANFRACTION is only specified if NCOL is 3. If MANFRACTION is not specified, the Manning's

roughness coefficient for the reach (MAN) is applied to the entire cross- section.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulaktab package must have a mfgwflak package parent_file.

```
dfn = [['header', 'multi-package'], ['block dimensions', 'name nrow', 'type
integer', 'reader urword', 'optional false'], ['block dimensions', 'name ncol',
'type integer', 'reader urword', 'optional false'], ['block table', 'name table',
'type recarray xfraction height manfraction', 'shape (nrow)', 'reader urword'],
['block table', 'name xfraction', 'type double precision', 'shape', 'tagged false',
'in_record true', 'reader urword'], ['block table', 'name height', 'type double
precision', 'shape', 'tagged false', 'in_record true', 'reader urword'], ['block
table', 'name manfraction', 'type double precision', 'shape', 'tagged false',
'in_record true', 'reader urword', 'optional true']]
```

```
dfn_file_name = 'utl-sfrtab.dfn'
```

```
package_abbr = 'utlsfrtab'
```

```
table = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfutlspc module

```
class ModflowUtlspc(model, loading_package=False, print_input=None, timeseries=None,
                    maxbound=None, perioddata=None, filename=None, pname=None, **kwargs)
```

Bases: *MFPackage*

ModflowUtlspc defines a spc package within a utl model.

Parameters

- **model** (*MFModel*) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **print_input** (*boolean*) -
 - print_input (boolean) keyword to indicate that the list of spc information will be written to the listing file immediately after it is read.
- **timeseries** (*{varname:data} or timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **maxbound** (*integer*) -
 - maxbound (integer) integer value specifying the maximum number of spc cells that will be specified for use during any stress period.

- **perioddata** ([*bndno*, *spcsetting*]) -
 - *bndno* (integer) integer value that defines the boundary package feature number associated with the specified PERIOD data on the line. BNDNO must be greater than zero and less than or equal to MAXBOUND. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *spcsetting* (keystring) line of information that is parsed into a keyword and values. Keyword values that can be used to start the MAWSETTING string include: CONCENTRATION.

concentration
[[double]]

* concentration (double) is the boundary concentration. If the Options block includes a TIMESERIESFILE entry (see the “Time- Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value. By default, the CONCENTRATION for each boundary feature is zero.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfulllaktab package must have a mfgwflak package parent_file.

```
dfn = [['header', 'multi-package'], ['block options', 'name print_input', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name ts_filerecord', 'type record ts6 filein ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package ts', 'construct_data timeseries', 'parameter_name timeseries'], ['block options', 'name ts6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name ts6_filename', 'type string', 'preserve_case true', 'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block dimensions', 'name maxbound', 'type integer', 'reader urword', 'optional false'], ['block period', 'name iper', 'type integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader urword', 'optional false'], ['block period', 'name perioddata', 'type recarray bndno spcsetting', 'shape', 'reader urword'], ['block period', 'name bndno', 'type integer', 'shape', 'tagged false', 'in_record true', 'reader urword', 'numeric_index true'], ['block period', 'name spcsetting', 'type keystring concentration', 'shape', 'tagged false', 'in_record true', 'reader urword'], ['block period', 'name concentration', 'type double precision', 'shape', 'tagged true', 'in_record true', 'reader urword', 'time_series true']]
```

```
dfn_file_name = 'utl-spc.dfn'
```

```
package_abbr = 'utlspc'
```



```
perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfutlspca module

```
class ModflowUtlspca(model, loading_package=False, readasarrays=True, print_input=None,
                      timearrayseries=None, concentration=0.0, filename=None, pname=None,
                      **kwargs)
```

Bases: [*MFPackage*](#)

ModflowUtlspca defines a spca package within a utl model.

Parameters

- **model** ([*MFModel*](#)) - Model that this package is a part of. Package is automatically added to model when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **readasarrays** (*boolean*) -
 - readasarrays (*boolean*) indicates that array-based input will be used for the SPC Package. This keyword must be specified to use array- based input.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that the list of spc information will be written to the listing file immediately after it is read.
- **timearrayseries** (*{varname:data} or tas_array data*) -
 - Contains data for the tas package. Data can be stored in a dictionary containing data for the tas package with variable names as keys and package data as values. Data just for the timearrayseries variable is also acceptable. See tas package documentation for more information.
- **concentration** (*[double]*) -
 - concentration (*double*) is the concentration of the associated Recharge or Evapotranspiration stress package. The concentration array may be defined by a time-array series (see the “Using Time- Array Series in a Package” section).
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** ([*MFPackage*](#)) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

```
concentration = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
```

```
dfn = [['header', 'multi-package'], ['block options', 'name readasarrays', 'type keyword', 'shape', 'reader urword', 'optional false', 'default_value True'], ['block options', 'name print_input', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name tas_filerecord', 'type record tas6 filein tas6_filename', 'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package tas', 'construct_data tas_array', 'parameter_name timearrayseries'], ['block options', 'name tas6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name tas6_filename', 'type string', 'preserve_case true', 'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block period', 'name iper', 'type integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader urword', 'optional false'], ['block period', 'name concentration', 'type double precision', 'shape (ncol*nrow; ncpl)', 'reader readarray', 'default_value 0.']]

dfn_file_name = 'utl-spca.dfn'

package_abbrev = 'utlspca'

tas_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

flopy.mf6.modflow.mfutils module

```
class ModflowUtlas(parent_package, loading_package=False, time_series_namerecord=None, interpolation_methodrecord=None, sfacrecord=None, tas_array=None, filename=None, pname=None, **kwargs)
```

Bases: [MFPackage](#)

ModflowUtlas defines a tas package within a utl model.

Parameters

- **parent_package** ([MFPackage](#)) - Parent_package that this package is a part of. Package is automatically added to parent_package when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **time_series_namerecord** (*[time_series_name]*) -
 - time_series_name (string) Name by which a package references a particular time-array series. The name must be unique among all time- array series used in a package.
- **interpolation_methodrecord** (*[interpolation_method]*) -
 - interpolation_method (string) Interpolation method, which is either STEPWISE or LINEAR.
- **sfacrecord** (*[sfacval]*) -
 - sfacval (double) Scale factor, which will multiply all array values in time series. SFAC is an optional attribute; if omitted, SFAC = 1.0.
- **tas_array** (*[double]*) -

- `tas_array` (double) An array of numeric, floating-point values, or a constant value, readable by the U2DREL array-reading utility.
- `filename` (String) - File name for this package.
- `pname` (String) - Package name for this package.
- `parent_file` (MFPackage) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mfutllaktab package must have a mfgwflak package `parent_file`.

```
dfn = [['header', 'multi-package'], ['block attributes', 'name
time_series_namerecord', 'type record name time_series_name', 'shape', 'reader
urword', 'tagged false', 'optional false'], ['block attributes', 'name name', 'type
keyword', 'shape', 'reader urword', 'optional false'], ['block attributes', 'name
time_series_name', 'type string', 'shape anyld', 'tagged false', 'reader urword',
'optional false'], ['block attributes', 'name interpolation_methodrecord', 'type
record method interpolation_method', 'shape', 'reader urword', 'tagged false',
'optional true'], ['block attributes', 'name method', 'type keyword', 'shape',
'reader urword', 'optional false'], ['block attributes', 'name
interpolation_method', 'type string', 'valid stepwise linear linearend', 'shape',
'tagged false', 'reader urword', 'optional false'], ['block attributes', 'name
sfacrecord', 'type record sfac sfacval', 'shape', 'reader urword', 'tagged true',
'optional true'], ['block attributes', 'name sfac', 'type keyword', 'shape', 'reader
urword', 'optional false'], ['block attributes', 'name sfacval', 'type double
precision', 'shape time_series_name', 'tagged false', 'reader urword', 'optional
false'], ['block time', 'name time_from_model_start', 'type double precision',
'block_variable True', 'in_record true', 'shape', 'tagged false', 'valid', 'reader
urword', 'optional false'], ['block time', 'name tas_array', 'type double
precision', 'tagged false', 'just_data true', 'shape (unknown)', 'reader readarray',
'optional false', 'repeating true']]

dfn_file_name = 'utl-tas.dfn'

interpolation_methodrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator
object>

package_abbr = 'utltas'

sfacrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

tas_array = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

time_series_namerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

class UtltasPackages(model_or_sim, parent, pkg_type, filerecord, package=None,
package_class=None)

Bases: MFChildPackages

UtltasPackages is a container class for the ModflowUtltas class.

initialize()

Initializes a new ModflowUtltas package removing any sibling child packages
attached to the same parent package. See ModflowUtltas init documentation for
definition of parameters.
```

`append_package()`

Adds a new ModflowUtlts package to the container. See ModflowUtlts init documentation for definition of parameters.

```
append_package(time_series_namerecord=None, interpolation_methodrecord=None,
               sfacrecord=None, tas_array=None, filename=None, pname=None)
```

```
initialize(time_series_namerecord=None, interpolation_methodrecord=None,
            sfacrecord=None, tas_array=None, filename=None, pname=None)
```

```
package_abbr = 'utltaspackages'
```

`flopy.mf6.modflow.mfutilts` module

```
class ModflowUtlts(parent_package, loading_package=False, time_series_namerecord=None,
                    interpolation_methodrecord=None,
                    interpolation_methodrecord_single=None, sfacrecord=None,
                    sfacrecord_single=None, timeseries=None, filename=None, pname=None,
                    **kwargs)
```

Bases: `MFPackage`

ModflowUtlts defines a ts package within a utl model.

Parameters

- **parent_package** (`MFPackage`) - Parent_package that this package is a part of. Package is automatically added to parent_package when it is initialized.
- **loading_package** (`bool`) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **time_series_namerecord** (`[time_series_names]`) -
 - `time_series_names` (string) Name by which a package references a particular time-array series. The name must be unique among all time- array series used in a package.
- **interpolation_methodrecord** (`[interpolation_method]`) -
 - `interpolation_method` (string) Interpolation method, which is either STEPWISE or LINEAR.
- **interpolation_methodrecord_single** (`[interpolation_method_single]`) -
 - `interpolation_method_single` (string) Interpolation method, which is either STEPWISE or LINEAR.
- **sfacrecord** (`[sfacval]`) -
 - `sfacval` (double) Scale factor, which will multiply all array values in time series. SFAC is an optional attribute; if omitted, SFAC = 1.0.
- **sfacrecord_single** (`[sfacval]`) -
 - `sfacval` (double) Scale factor, which will multiply all array values in time series. SFAC is an optional attribute; if omitted, SFAC = 1.0.

- **timeseries** ([*ts_time*, *ts_array*]) -
 - *ts_time* (double) A numeric time relative to the start of the simulation, in the time unit used in the simulation. Times must be strictly increasing.
 - *ts_array* (double) A 2-D array of numeric, floating-point values, or a constant value, readable by the U2DREL array-reading utility.
- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutil*). For example, mftllaktab package must have a mfgwflak package parent_file.

```
dfn = [['header', 'multi-package'], ['block attributes', 'name
time_series_namerecord', 'type record names time_series_names', 'shape', 'reader
urword', 'tagged false', 'optional false'], ['block attributes', 'name names',
'other_names name', 'type keyword', 'shape', 'reader urword', 'optional false'],
['block attributes', 'name time_series_names', 'type string', 'shape anyld', 'tagged
false', 'reader urword', 'optional false'], ['block attributes', 'name
interpolation_methodrecord', 'type record methods interpolation_method', 'shape',
'reader urword', 'tagged false', 'optional true'], ['block attributes', 'name
methods', 'type keyword', 'shape', 'reader urword', 'optional false'], ['block
attributes', 'name interpolation_method', 'type string', 'valid stepwise linear
linearend', 'shape time_series_names', 'tagged false', 'reader urword', 'optional
false'], ['block attributes', 'name interpolation_methodrecord_single', 'type record
method interpolation_method_single', 'shape', 'reader urword', 'tagged false',
'optional true'], ['block attributes', 'name method', 'type keyword', 'shape',
'reader urword', 'optional false'], ['block attributes', 'name
interpolation_method_single', 'type string', 'valid stepwise linear linearend',
'shape', 'tagged false', 'reader urword', 'optional false'], ['block attributes',
'name sfacrecord', 'type record sfacs sfacval', 'shape', 'reader urword', 'tagged
true', 'optional true'], ['block attributes', 'name sfacs', 'type keyword', 'shape',
'reader urword', 'optional false'], ['block attributes', 'name sfacval', 'type
double precision', 'shape <time_series_name', 'tagged false', 'reader urword',
'optional false'], ['block attributes', 'name sfacrecord_single', 'type record sfac
sfacval', 'shape', 'reader urword', 'tagged true', 'optional true'], ['block
attributes', 'name sfac', 'type keyword', 'shape', 'tagged false', 'reader urword',
'optional false'], ['block timeseries', 'name timeseries', 'type recarray ts_time
ts_array', 'shape', 'reader urword', 'tagged true', 'optional false'], ['block
timeseries', 'name ts_time', 'type double precision', 'shape', 'tagged false',
'reader urword', 'optional false', 'repeating false'], ['block timeseries', 'name
ts_array', 'type double precision', 'shape time_series_names', 'tagged false',
'reader urword', 'optional false']]
```

```
dfn_file_name = 'utl-ts.dfn'
```

```
interpolation_methodrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator
object>
```

```
interpolation_methodrecord_single = <flopy.mf6.data.mfdatautil.ListTemplateGenerator
object>
```

```
package_abbr = 'utlts'

sfacrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

sfacrecord_single = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

time_series_namerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

timeseries = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

class UtltsPackages(model_or_sim, parent, pkg_type, filerecord, package=None,
                    package_class=None)

Bases: MFChildPackages

UtltsPackages is a container class for the ModflowUtlts class.

initialize()
    Initializes a new ModflowUtlts package removing any sibling child packages
    attached to the same parent package. See ModflowUtlts init documentation for
    definition of parameters.

append_package()
    Adds a new ModflowUtlts package to the container. See ModflowUtlts init
    documentation for definition of parameters.

append_package(time_series_namerecord=None, interpolation_methodrecord=None,
               interpolation_methodrecord_single=None, sfacrecord=None,
               sfacrecord_single=None, timeseries=None, filename=None, pname=None)

initialize(time_series_namerecord=None, interpolation_methodrecord=None,
           interpolation_methodrecord_single=None, sfacrecord=None,
           sfacrecord_single=None, timeseries=None, filename=None, pname=None)

package_abbr = 'utltspackages'
```

flopy.mf6.modflow.mfutlvtk module

```
class ModflowUtlvtk(parent_package, loading_package=False, print_input=None,
                   timeseries=None, perioddata=None, filename=None, pname=None,
                   **kwargs)
```

Bases: [MFPackage](#)

ModflowUtlvtk defines a tvk package within a utl model.

Parameters

- **parent_package** ([MFPackage](#)) - Parent_package that this package is a part of. Package is automatically added to parent_package when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **print_input** (*boolean*) -
 - print_input (*boolean*) keyword to indicate that information for each change to the hydraulic conductivity in a cell will be written to the model listing file.
- **timeseries** (*{varname:data}* or *timeseries data*) -

- Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **perioddata** ([*cellid*, *tvksetting*]) -
 - *cellid* ((integer, ...)) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.
 - *tvksetting* (keystring) line of information that is parsed into a property name keyword and values. Property name keywords that can be used to start the TVKSETTING string include: K, K22, and K33.

k

[[double]]

* *k* (double) is the new value to be assigned as the cell's hydraulic conductivity from the start of the specified stress period, as per K in the NPF package. If the OPTIONS block includes a TS6 entry (see the "Time-Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

k22

[[double]]

* *k22* (double) is the new value to be assigned as the cell's hydraulic conductivity of the second ellipsoid axis (or the ratio of K22/K if the K22OVERK NPF package option is specified) from the start of the specified stress period, as per K22 in the NPF package. For an unrotated case this is the hydraulic conductivity in the y direction. If the OPTIONS block includes a TS6 entry (see the "Time-Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

k33

[[double]]

* *k33* (double) is the new value to be assigned as the cell's hydraulic conductivity of the third ellipsoid axis (or the ratio of K33/K if the K33OVERK NPF package option is specified) from the start of

the specified stress period, as per K33 in the NPF package. For an unrotated case, this is the vertical hydraulic conductivity. If the OPTIONS block includes a TS6 entry (see the “Time-Variable Input” section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- **filename** (*String*) - File name for this package.
- **pname** (*String*) - Package name for this package.
- **parent_file** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (mfutl*). For example, mftlaktab package must have a mfgwflak package parent_file.

```
dfn = [['header'], ['block options', 'name print_input', 'type keyword', 'reader urword', 'optional true'], ['block options', 'name ts_filerecord', 'type record ts6 filein ts6_filename', 'shape', 'reader urword', 'tagged true', 'optional true', 'construct_package ts', 'construct_data timeseries', 'parameter_name timeseries'], ['block options', 'name ts6', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name filein', 'type keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional false'], ['block options', 'name ts6_filename', 'type string', 'preserve_case true', 'in_record true', 'reader urword', 'optional false', 'tagged false'], ['block period', 'name iper', 'type integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid', 'reader urword', 'optional false'], ['block period', 'name perioddata', 'type recarray cellid tvksetting', 'shape', 'reader urword'], ['block period', 'name cellid', 'type integer', 'shape (ncelldim)', 'tagged false', 'in_record true', 'reader urword'], ['block period', 'name tvksetting', 'type keystring k k22 k33', 'shape', 'tagged false', 'in_record true', 'reader urword'], ['block period', 'name k', 'type double precision', 'shape', 'tagged true', 'in_record true', 'reader urword', 'time_series true'], ['block period', 'name k22', 'type double precision', 'shape', 'tagged true', 'in_record true', 'reader urword', 'time_series true'], ['block period', 'name k33', 'type double precision', 'shape', 'tagged true', 'in_record true', 'reader urword', 'time_series true']]
```

```
dfn_file_name = 'utl-tvk.dfn'
```

```
package_abbr = 'utltvk'
```

```
perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
```

```
class UtltvkPackages(model_or_sim, parent, pkg_type, filerecord, package=None, package_class=None)
```

Bases: [MFChildPackages](#)

UtltvkPackages is a container class for the ModflowUtltvk class.

initialize()

Initializes a new ModflowUtltvk package removing any sibling child packages attached to the same parent package. See ModflowUtltvk init documentation for definition of parameters.

append_package()

Adds a new ModflowUtltkv package to the container. See ModflowUtltkv init documentation for definition of parameters.

```
append_package(print_input=None, timeseries=None, perioddata=None, filename=None,
                pname=None)
```

```
initialize(print_input=None, timeseries=None, perioddata=None, filename=None,
            pname=None)
```

```
package_abbr = 'utltvkpackages'
```

flopy.mf6.modflow.mfutltsv module

```
class ModflowUtltsv(parent_package, loading_package=False,
                    disable_storage_change_integration=None, print_input=None,
                    timeseries=None, perioddata=None, filename=None, pname=None,
                    **kwargs)
```

Bases: [MFPackage](#)

ModflowUtltsv defines a tvs package within a utl model.

Parameters

- **parent_package** ([MFPackage](#)) - Parent_package that this package is a part of. Package is automatically added to parent_package when it is initialized.
- **loading_package** (*bool*) - Do not set this parameter. It is intended for debugging and internal processing purposes only.
- **disable_storage_change_integration** (*boolean*) -
 - `disable_storage_change_integration` (*boolean*) keyword that deactivates inclusion of storage derivative terms in the STO package matrix formulation. In the absence of this keyword (the default), the groundwater storage formulation will be modified to correctly adjust heads based on transient variations in stored water volumes arising from changes to SS and SY properties.
- **print_input** (*boolean*) -
 - `print_input` (*boolean*) keyword to indicate that information for each change to a storage property in a cell will be written to the model listing file.
- **timeseries** (*{varname:data}* or *timeseries data*) -
 - Contains data for the ts package. Data can be stored in a dictionary containing data for the ts package with variable names as keys and package data as values. Data just for the timeseries variable is also acceptable. See ts package documentation for more information.
- **perioddata** (*[cellid, tvssetting]*) -
 - `cellid` (*(integer, ...)*) is the cell identifier, and depends on the type of grid that is used for the simulation. For a structured grid that uses the DIS input file, CELLID is the

layer, row, and column. For a grid that uses the DISV input file, CELLID is the layer and CELL2D number. If the model uses the unstructured discretization (DISU) input file, CELLID is the node number for the cell. This argument is an index variable, which means that it should be treated as zero-based when working with FloPy and Python. Flopy will automatically subtract one when loading index variables and add one when writing index variables.

- `tvsetting (keystring)` line of information that is parsed into a property name keyword and values. Property name keywords that can be used to start the TVSETTING string include: `SS` and `SY`.

`ss`

`[[double]]`

* `ss (double)` is the new value to be assigned as the cell's specific storage (or storage coefficient if the `STORAGECOEFFICIENT` STO package option is specified) from the start of the specified stress period, as per `SS` in the STO package. Specific storage values must be greater than or equal to 0. If the `OPTIONS` block includes a `TS6` entry (see the "Time-Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

`sy`

`[[double]]`

* `sy (double)` is the new value to be assigned as the cell's specific yield from the start of the specified stress period, as per `SY` in the STO package. Specific yield values must be greater than or equal to 0. If the `OPTIONS` block includes a `TS6` entry (see the "Time-Variable Input" section), values can be obtained from a time series by entering the time-series name in place of a numeric value.

- **`filename`** (*String*) - File name for this package.
- **`pname`** (*String*) - Package name for this package.
- **`parent_file`** (*MFPackage*) - Parent package file that references this package. Only needed for utility packages (`mfutl*`). For example, `mfutllaktab` package must have a `mfgwflak` package `parent_file`.

```

dfn = [['header'], ['block options', 'name disable_storage_change_integration',
'type keyword', 'reader urword', 'optional true'], ['block options', 'name
print_input', 'type keyword', 'reader urword', 'optional true'], ['block options',
'name ts_filerecord', 'type record ts6 filein ts6_filename', 'shape', 'reader
urword', 'tagged true', 'optional true', 'construct_package ts', 'construct_data
timeseries', 'parameter_name timeseries'], ['block options', 'name ts6', 'type
keyword', 'shape', 'in_record true', 'reader urword', 'tagged true', 'optional
false'], ['block options', 'name filein', 'type keyword', 'shape', 'in_record true',
'reader urword', 'tagged true', 'optional false'], ['block options', 'name
ts6_filename', 'type string', 'preserve_case true', 'in_record true', 'reader
urword', 'optional false', 'tagged false'], ['block period', 'name iper', 'type
integer', 'block_variable True', 'in_record true', 'tagged false', 'shape', 'valid',
'reader urword', 'optional false'], ['block period', 'name perioddata', 'type
recarray cellid tvssetting', 'shape', 'reader urword'], ['block period', 'name
cellid', 'type integer', 'shape (ncellldim)', 'tagged false', 'in_record true',
'reader urword'], ['block period', 'name tvssetting', 'type keystring ss sy',
'shape', 'tagged false', 'in_record true', 'reader urword'], ['block period', 'name
ss', 'type double precision', 'shape', 'tagged true', 'in_record true', 'reader
urword', 'time_series true'], ['block period', 'name sy', 'type double precision',
'shape', 'tagged true', 'in_record true', 'reader urword', 'time_series true']]

dfn_file_name = 'utl-tvs.dfn'

package_abbr = 'utltvs'

perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

ts_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

class UtltvsPackages(model_or_sim, parent, pkg_type, filerecord, package=None,
package_class=None)

Bases: MFChildPackages

UtltvsPackages is a container class for the ModflowUtltvs class.

initialize()
    Initializes a new ModflowUtltvs package removing any sibling child packages
    attached to the same parent package. See ModflowUtltvs init documentation for
    definition of parameters.

append_package()
    Adds a new ModflowUtltvs package to the container. See ModflowUtltvs init
    documentation for definition of parameters.

append_package(disable_storage_change_integration=None, print_input=None,
timeseries=None, perioddata=None, filename=None, pname=None)

initialize(disable_storage_change_integration=None, print_input=None,
timeseries=None, perioddata=None, filename=None, pname=None)

package_abbr = 'utltvspackages'

```

8.1.8 MODFLOW 6 Utility Functions

MODFLOW 6 has several utility functions that can be used to process MODFLOW 6 results.

Contents:

`flopy.mf6.utils.binaryfile_utils` module

class `MFOutput(mfdict, path, key)`

Bases: `object`

Wrapper class for Binary Arrays. This class enables directly getting slices from the binary output. It is intended to be called from the `__getitem__` method of the `SimulationDict()` class. Implemented to conserve memory.

Parameters

- **path** (binary file path location) -
- **mfdict** (`SimulationDict()` *object*) -
- **key** (dict key ex. ('flow15','CBC','FLOW RIGHT FACE')) -

Returns

- *Xarray of [n,n,n,n] dimension*
- *Usage*
- *--*
- `>>> val = MFOutput(mfdict, path, key)`
- `>>> return val.data`
- *User interaction*
- *-----*
- `>>> data[(‘flow15’,‘CBC’,‘FLOW RIGHT FACE’)][(,0,1,:)]`
- *or*
- `>>> data[(‘flow15’,‘CBC’,‘FLOW RIGHT FACE’)]`

class `MFOutputRequester(mfdict, path, key)`

Bases: `object`

`MFOutputRequest` class is a helper function to enable the user to query binary data from the `SimulationDict()` object on the fly without actually storing it in the `SimulationDict()` object.

Parameters:

mfdict: `dict`

local instance of the `SimulationDict()` object

path:

pointer to the `MFSimulationPath` object

key: `tuple`

user requested data key

Methods:

MFOutputRequester.querybinarydata
 returns: Xarray object

Examples:

```
>>> data = MFOutputRequester(mfdict, path, key)
>>> data.querybinarydata
```

```
static getkeys(mfdict, path, print_keys=True)
```

flopy.mf6.utils.binarygrid_util module

Module to read MODFLOW 6 binary grid files (*.grb) that define the model grid binary output files. The module contains the MfGrdFile class that can be accessed by the user.

class MfGrdFile(filename, precision='double', verbose=False)

Bases: *FlopyBinaryData*

The MfGrdFile class.

Parameters

- **filename** (*str*) - Name of the MODFLOW 6 binary grid file
- **precision** (*string*) - 'single' or 'double'. Default is 'double'.
- **verbose** (*bool*) - Write information to standard output. Default is False.

Notes

The MfGrdFile class provides simple ways to retrieve data from binary MODFLOW 6 binary grid files (.grb). The binary grid file contains data that can be used for post processing MODFLOW 6 model results. For example, the ia and ja arrays for a model grid.

Examples

```
>>> import flopy
>>> gobj = flopy.utils.MfGrdFile('test.dis.grb')
```

property angrot

Model grid rotation angle. None if not defined in the MODFLOW 6 grid file.

Returns

angrot

Return type

float

property bot

Bottom of the model cells.

Returns

bot

Return type

ndarray of floats

property cell2d

cell2d data for a DISV grid. None for DIS and DISU grids.

Returns

cell2d

Return type

list of lists

property cellcenters

Cell centers (x,y).

Returns

cellcenters

Return type

ndarray of floats

property delc

Cell size in the column direction (x-direction). None if not defined in the MODFLOW 6 grid file.

Returns

delc

Return type

ndarray of floats

property delr

Cell size in the row direction (y-direction). None if not defined in the MODFLOW 6 grid file.

Returns

delr

Return type

ndarray of floats

property grid_type

Grid type defined in the MODFLOW 6 grid file.

Returns

grid_type

Return type

str

property ia

index array that defines indexes for .ja. Each ia value is the starting position of data for a cell. [ia[n]:ia[n+1]] would give you all data for a cell. ia[n] is also the location of data for the diagonal position. See .ja property documentation for an example of getting a cell's number and connected cells

Returns**ia****Return type**

ndarray of ints

property iavert

index array that defines indexes for `.javert`. Each `ia` value is the starting position of data for a cell. `[iavert[n]:iavert[n+1]]` would give you all data for a cell. See `.javert` property documentation for an example of getting cell number and it's vertex numbers. Alternatively, the `.iverts` property can be used to get this information

Returns**iavert****Return type**

ndarray of ints or None for structured grids

property idomain

IDOMAIN for the model grid. None if not defined in the MODFLOW 6 grid file.

Returns**idomain****Return type**

ndarray of ints

property iverts

Vertex numbers comprising each cell for every cell in model grid.

Returns**iverts****Return type**

list of lists of ints

property ja

Flat jagged connection array for a model. `.ja` for a cell includes the cell number and the cell number for all connected cells. Indexes for cells are stored in the `.ia` variable.

Returns**ja****Return type**

ndarray of ints

Examples

```
>>> from flopy.mf6.utils import MfGrdFile
>>> grb = MfGrdFile("my_model.dis.grb")
>>> ia = grb.ia
>>> ja = grb.ja
>>> # get connections for node 0
>>> ja_node0 = ja[ia[0]:ia[1]]
>>> node = ja_node0[0]
>>> connections = ja_node0[1:]
```

property javert

Flat jagged array of vertex numbers that comprise all of the cells

Returns

javerts

Return type

ndarray of ints or None for structured grids

Examples

```
>>> from flopy.mf6.utils import MfGrdFile
>>> grb = MfGrdFile("my_model.dis.grb")
>>> iavert = self.iavert
>>> javert = self.javert
>>> # get vertex numbers for node 0
>>> vertnums = javert[iavert[0]:iavert[1]]
```

property modelgrid

Model grid object.

Returns

modelgrid

Return type

StructuredGrid, VertexGrid, UnstructuredGrid

property ncells

Number of cells.

Returns

ncells

Return type

int

property ncol

Number of columns. None for DISV and DISU grids.

Returns

ncol

Return type

int

property ncpl

Number of cells per layer. None for DISU grids.

Returns

ncpl

Return type

int

property nja

Number of non-zero entries JA vector array.

Returns

nja

Return type

`int`

property nlay

Number of layers. None for DISU grids.

Returns

`nlay`

Return type

`int`

property nodes

Number of nodes.

Returns

`nodes`

Return type

`int`

property nrow

Number of rows. None for DISV and DISU grids.

Returns

`nrow`

Return type

`int`

property shape

Shape of the model grid (tuple).

Returns

`shape`

Return type

`tuple`

property top

Top of the model cells in the upper model layer for DIS and DISV grids. Top of the model cells for DISU grids.

Returns

`top`

Return type

`ndarray of floats`

property verts

x,y location of each vertex that defines the model grid.

Returns

`verts`

Return type

`ndarray of floats`

property xorigin

x-origin of the model grid. None if not defined in the MODFLOW 6 grid file.

Returns
xorigin

Return type
`float`

property yorigin

y-origin of the model grid. None if not defined in the MODFLOW 6 grid file.

Returns
yorigin

Return type
`float`

flopy.mf6.utils.mfobservation module

class MFObservation(*mfdict*, *path*, *key*)

Bases: `object`

Wrapper class to request the MFObservation object: Class is called by the MFSimulation.SimulationDict() class and is not called by the user

Inputs:

mfdict: (dict) the sim.simulation_dict.mfdict object for the flopy project path:
(object) the path object detailing model names and paths *key*: (tuple, strings) user supplied dictionary key to request observation utility data

Returns: --- self.data: (xarray) array of observations

class MFObservationRequester(*mfdict*, *path*, *key*, ****kwargs**)

Bases: `object`

Wrapper class for MFObservation.Observations. Class checks which observation data is available, and creates a dictionary key to access the set of observation data from the SimulationDict()

static getkeys(*mfdict*, *path*)

class Observations(*fi*)

Bases: `object`

Simple class to extract and view Observation files for Uzf models (possibly all obs/hobs)?

Input:

fi = (string) name of the observation binary output file

Methods:**get_data(): (np.array) returns array of observation data**

text = (str) specific modflow record name contained in Obs.out file
 idx = (int), (slice(start, stop)) integer or slice of data to be returned.
 corresponds to kstp*kper - 1 totim = (float) model time value to return data from

list_records(): prints a list of all valid record names contained within the

Obs.out file get_times(): (list) returns list of time values contained in Obs.out

get_nrecords(): (int) returns number of records get_ntimes(): (int) returns number

of times get_nobs(): (int) returns total number of observations (ntimes * nrecords)

get_data(key=None, idx=None, totim=None)

Method to request and return array of data from an Observation output file

Parameters

- **key** ((str) dictionary key for a specific observation contained within) - the observation file (optional)
- **idx** ((int) time index (optional)) -
- **totim** ((float) simulation time (optional)) -

Returns

data

Return type

(list) observation file data in list

get_dataframe(keys=None, idx=None, totim=None, start_datetime=None, timeunit='D')

Creates a pandas dataframe object from the observation data, useful backend if the user does not like the x-array format!

Parameters

- **keys** ((string) sting of dictionary/observation keys separated by comma.) - (optional)
- **idx** ((int) time index location (optional)) -
- **totim** ((float) simulation time (optional)) -
- **start_datetime** ((string) format is 'dd/mm/yyyy' or) - 'dd/mm/yyyy hh:mm:ss' (optional)
- **timeunit** ((string) specifies the time unit associated with totim when) - setting a datetime

Return type

pd.DataFrame

get_nobs()**get_nrecords()****get_ntimes()**

`get_obs_data(key=None, idx=None, totim=None)`

Method to request observation output data as an x-array :param key: within the observation file (optional) :type key: (string) dictionary key for a specific observation contained :param idx: :type idx: (int) time index (optional) :param totim: :type totim: (float) simulation time (optional)

Returns

xarray.DataArray

Return type

(NxN) dimensions are totim, header == keys*

`get_times()`

`list_records()`

`try_float(data)`

flopy.mf6.utils.output_util module

class MF6Output(obj)

Bases: `object`

A class that uses meta programming to get output

Parameters

obj (*PackageInterface object*) -

property csv_names

Method to get csv file names

Return type

`list`

methods()

Method that returns a list of available method calls

Return type

`list`

property obs_names

Method to get obs file names

Return type

`list`

flopy.mf6.utils.postprocessing module

`get_residuals(flowja, grb_file=None, ia=None, ja=None, shape=None, verbose=False)`

Get the residual from the MODFLOW 6 flowja flows. The residual is stored in the diagonal position of the flowja vector.

Parameters

- **flowja** (*ndarray*) - flowja array for a structured MODFLOW 6 model
- **grbfile** (*str*) - MODFLOW 6 binary grid file path
- **ia** (*list or ndarray*) - CRS row pointers. Only required if grb_file is not provided.

- **ja** (*list* or *ndarray*) - CRS column pointers. Only required if *grb_file* is not provided.
- **shape** (*tuple*) - shape of returned residual. A flat array is returned if shape is None and *grbfile* is None.
- **verbose** (*bool*) - Write information to standard output

Returns

residual - Residual for each cell

Return type

ndarray

get_structured_faceflows(*flowja*, *grb_file*=None, *ia*=None, *ja*=None, *nlay*=None, *nrow*=None, *ncol*=None, *verbose*=False)

Get the face flows for the flow right face, flow front face, and flow lower face from the MODFLOW 6 *flowja* flows. This method can be useful for building face flow arrays for MT3DMS, MT3D-USGS, and RT3D. This method only works for a structured MODFLOW 6 model.

Parameters

- **flowja** (*ndarray*) - *flowja* array for a structured MODFLOW 6 model
- **grbfile** (*str*) - MODFLOW 6 binary grid file path
- **ia** (*list* or *ndarray*) - CRS row pointers. Only required if *grb_file* is not provided.
- **ja** (*list* or *ndarray*) - CRS column pointers. Only required if *grb_file* is not provided.
- **nlay** (*int*) - number of layers in the grid. Only required if *grb_file* is not provided.
- **nrow** (*int*) - number of rows in the grid. Only required if *grb_file* is not provided.
- **ncol** (*int*) - number of columns in the grid. Only required if *grb_file* is not provided.
- **verbose** (*bool*) - Write information to standard output

Returns

- **frf** (*ndarray*) - right face flows
- **fff** (*ndarray*) - front face flows
- **flf** (*ndarray*) - lower face flows

flopy.mf6.utils.reference module

Module spatial referencing for flopy model objects

class SpatialReference(*delr*=1.0, *delc*=1.0, *xvdict*=None, *yvdict*=None, *lenuni*=1, *nlay*=1, *xul*=None, *yul*=None, *xadj*=0.0, *yadj*=0.0, *rotation*=0.0, *proj4_str*=None, *distype*='structured')

Bases: *object*

A dynamic inheritance class that locates a gridded model in space

delr
[numpy ndarray] the model discretization delr vector

delc
[numpy ndarray] the model discretization delc vector

lenuni
[int] the length units flag from the discretization package

xul
[float] the x coordinate of the upper left corner of the grid

yul
[float] the y coordinate of the upper left corner of the grid

rotation
[float] the counter-clockwise rotation (in degrees) of the grid

proj4_str: str
a PROJ4 string that identifies the grid in space. warning: case sensitive!

xadj
[float] vertex grid: x vertex adjustment factor

yadj
[float] vertex grid: y vertex adjustment factor

xvdict: dict
dictionary of x-vertices by cellnum ex. {0: (0,1,1,0)}

yvdict: dict
dictionary of y-vertices by cellnum ex. {0: (1,1,0,0)}

distype: str
model grid discretization type

```
class StructuredSpatialReference(delr=1.0, delc=1.0, lenuni=1, nlay=1, xul=None,
                                yul=None, rotation=0.0, proj4_str=None, **kwargs)
```

Bases: `object`

a simple class to locate the model grid in x-y space

Parameters

- **delr** (*numpy ndarray*) - the model discretization delr vector
- **delc** (*numpy ndarray*) - the model discretization delc vector
- **lenuni** (*int*) - the length units flag from the discretization package
- **xul** (*float*) - the x coordinate of the upper left corner of the grid
- **yul** (*float*) - the y coordinate of the upper left corner of the grid
- **rotation** (*float*) - the counter-clockwise rotation (in degrees) of the grid
- **proj4_str** (*str*) - a PROJ4 string that identifies the grid in space. warning: case sensitive!

xedge
 array of column edges
 Type
 ndarray

yedge
 array of row edges
 Type
 ndarray

xgrid
 numpy meshgrid of xedges
 Type
 ndarray

ygrid
 numpy meshgrid of yedges
 Type
 ndarray

xcenter
 array of column centers
 Type
 ndarray

ycenter
 array of row centers
 Type
 ndarray

xcentergrid
 numpy meshgrid of column centers
 Type
 ndarray

ycentergrid
 numpy meshgrid of row centers
 Type
 ndarray

Notes

xul and yul can be explicitly (re)set after SpatialReference instantiation, but only before any of the other attributes and methods are accessed

property attribute_dict

classmethod from_gridspec(*gridspec_file*, *lenuni=0*)

classmethod from_namfile_header(*namefile*)

get_extent()

Get the extent of the rotated and offset grid

Return (xmin, xmax, ymin, ymax)

get_grid_lines()

get the grid lines as a list

get_vertices(i, j)**get_xcenter_array()**

Return a numpy one-dimensional float array that has the cell center x coordinate for every column in the grid in model space - not offset or rotated.

get_xedge_array()

Return a numpy one-dimensional float array that has the cell edge x coordinates for every column in the grid in model space - not offset or rotated. Array is of size (ncol + 1)

get_ycenter_array()

Return a numpy one-dimensional float array that has the cell center x coordinate for every row in the grid in model space - not offset of rotated.

get_yedge_array()

Return a numpy one-dimensional float array that has the cell edge y coordinates for every row in the grid in model space - not offset or rotated. Array is of size (nrow + 1)

interpolate(a, xi, method='nearest')

Use the griddata method to interpolate values from an array onto the points defined in xi. For any values outside of the grid, use 'nearest' to find a value for them.

Parameters

- **a** (*numpy.ndarray*) - array to interpolate from. It must be of size nrow, ncol
- **xi** (*numpy.ndarray*) - array containing x and y point coordinates of size (npts, 2). xi also works with broadcasting so that if a is a 2d array, then xi can be passed in as (xgrid, ygrid).
- **method** ({'linear', 'nearest', 'cubic'}) - method to use for interpolation (default is 'nearest')

Returns

b - array of size (npts)

Return type

numpy.ndarray

property ncol

property nrow

reset(kwargs)**


```

static rotate(x, y, theta, xorigin=0.0, yorigin=0.0)
    Given x and y array-like values calculate the rotation about an arbitrary
    origin and then return the rotated coordinates.  theta is in degrees.

set_spatialreference(xul=None, yul=None, rotation=0.0)
    set spatial reference - can be called from model instance

write_gridSpec(filename)
    write a PEST-style grid specification file

property xcenter
property xcentergrid
property xedge
property xgrid
property ycenter
property ycentergrid
property yedge
property ygrid

```

```

class VertexSpatialReference(xvdict=None, yvdict=None, nlay=1, xadj=0, yadj=0,
                             rotation=0.0, lenuni=1.0, proj4_str=None, **kwargs)

```

Bases: `object`

a simple class to locate the model grid in x-y space

Parameters

- **xvdict** (*dictionary*) - dictionary of x-vertices {1: (0,1,1,0)}
- **yvdict** (*dictionary*) - dictionary of y-vertices {1: (1,0,1,0)}
- **lenuni** (*int*) - the length units flag from the discretization package
- **xadj** (*float*) - the x coordinate of the upper left corner of the grid
- **yadj** (*float*) - the y coordinate of the upper left corner of the grid
- **rotation** (*float*) - the counter-clockwise rotation (in degrees) of the grid
- **proj4_str** (*str*) - a PROJ4 string that identifies the grid in space. warning: case sensitive!

xedge

array of column edges

Type

ndarray

yedge

array of row edges

Type

ndarray

xgrid
numpy meshgrid of xedges
Type
ndarray

ygrid
numpy meshgrid of yedges
Type
ndarray

xcenter
array of column centers
Type
ndarray

ycenter
array of row centers
Type
ndarray

xcentergrid
numpy meshgrid of column centers
Type
ndarray

ycentergrid
numpy meshgrid of row centers
Type
ndarray

Notes

xadj and yuadj can be explicitly (re)set after SpatialReference instantiation, but only before any of the other attributes and methods are accessed

classmethod `from_namfile_header(namefile)`

get_extent()

Get the extent of the rotated and offset grid

Return (xmin, xmax, ymin, ymax)

property `ncpl`

static `rotate(x, y, theta, xorigin=0.0, yorigin=0.0)`

Given x and y array-like values calculate the rotation about an arbitrary origin and then return the rotated coordinates. theta is in degrees.

set_spatialreference(xadj=0.0, yadj=0.0, rotation=0.0)

set spatial reference - can be called from model instance xadj, yadj should be named xadj, yadj since they represent an adjustment factor

property `xarr`

```

property xcenter_array
property xdict
property ydict
property yarr
property ycenter_array
property ydict

```

`flopy.mf6.utils.lakpak_utils` module

`get_lak_connections(modelgrid, lake_map, idomain=None, bedleak=None)`

Function to create lake package connection data from a zero-based integer array of lake numbers. If the shape of lake number array is equal to (nrow, ncol) or (ncpl) then the lakes are on top of the model and are vertically connected to cells at the top of the model. Otherwise the lakes are embedded in the grid.

TODO: implement embedded lakes for VertexGrid

TODO: add support for UnstructuredGrid

Parameters

- **modelgrid** (`StructuredGrid`, `VertexGrid`) - model grid
- **lake_map** (`MaskedArray`, `ndarray`, `list`, `tuple`) - location and zero-based lake number for lakes in the model domain. If lake_map is of size (nrow, ncol) or (ncpl) lakes are located on top of the model and vertically connected to cells in model layer 1. If lake_map is of size (nlay, nrow, ncol) or (nlay, ncpl) lakes are embedded in the model domain and horizontal and vertical lake connections are defined.
- **idomain** (`int` or `ndarray`) - location of inactive cells, which are defined with a zero value. If a ndarray is passed it must be of size (nlay, nrow, ncol) or (nlay, ncpl).
- **bedleak** (`ndarray`, `list`, `tuple`, `float`) - bed leakance for lakes in the model domain. If bedleak is a float the same bed leakance is applied to each lake connection in the model. If bedleak is of size (nrow, ncol) or (ncpl) then all lake connections for the cellid are given the same bed leakance value. If bedleak is None, lake conductance is only a function of aquifer properties for all lakes. Can also pass mixed values as list or ndarray of size (nrow, ncol) or (ncpl) with floats and 'none' strings.

Returns

- **idomain** (`ndarray`) - idomain adjusted to inactivate cells with lakes
- **connection_dict** (`dict`) - dictionary with the zero-based lake number keys and number of connections in a lake values
- **connectiondata** (`list of lists`) - connectiondata block for the lake package

8.1.9 MODFLOW 6 Data

FloPy for MODFLOW 6 data objects (`MFDataArray`, `MFDataList`, `MFDataScalar`) are automatically constructed by FloPy when you construct a package. These data objects provide an interface for getting MODFLOW 6 data in different formats and setting MODFLOW 6 data.

Contents:

`flopy.mf6.data.mfdataarray` module

class `MFArray`(*sim_data*, *model_or_sim*, *structure*, *data=None*, *enable=True*, *path=None*, *dimensions=None*, *block=None*)

Bases: `MFMultiDimVar`

Provides an interface for the user to access and update MODFLOW array data. `MFArray` objects are not designed to be directly constructed by the end user. When a FloPy for MODFLOW 6 package object is constructed, the appropriate `MFArray` objects are automatically built.

Parameters

- **sim_data** (`MFSimulationData`) - data contained in the simulation
- **structure** (`MFDataStructure`) - describes the structure of the data
- **data** (*list* or *ndarray*) - actual data
- **enable** (*bool*) - enable/disable the array
- **path** (*tuple*) - path in the data dictionary to this `MFArray`
- **dimensions** (`MFDataDimensions`) - dimension information related to the model, package, and array

property data

Returns array data. Calls `get_data` with default parameters.

property data_type

Type of data (`DataType`) stored in the array

property dtype

Type of data (`numpy.dtype`) stored in the array

get_data(*layer=None*, *apply_mult=False*, ***kwargs*)

Returns the data associated with layer “layer”. If “layer” is None, returns all data.

Parameters

layer (*int*) -

Returns

data - Array data in an `ndarray`

Return type

`ndarray`

get_file_entry(*layer=None*, *ext_file_action=ExtFileAction.copy_relative_paths*)

Returns a string containing the data in layer “layer” formatted for a MODFLOW 6 file. For unlayered data do not pass in “layer”.

Parameters

- **layer** (*int*) - The layer to return file entry for.
- **ext_file_action** (*ExtFileAction*) - How to handle external paths.

Returns
file entry

Return type
str

get_record(*layer=None*)

Returns the data record associated with layer “layer”. If “layer” is None, returns all data.

Parameters
layer (*int*) -

Returns
data_record - Dictionary containing data record for specified layer or dictionary containing layer numbers (keys) and data record for that layer (values).

Return type
dict

has_data(*layer=None*)

Returns whether layer “layer_num” has any data associated with it.

Parameters
layer_num (*int*) - Layer number to check for data. For unlayered data do not pass anything in

Returns
has data - Returns if there is data.

Return type
bool

load(*first_line, file_handle, block_header, pre_data_comments=None, external_file_info=None*)

Loads data from first_line (the first line of data) and open file file_handle which is pointing to the second line of data. Returns a tuple with the first item indicating whether all data was read and the second item being the last line of text read from the file. This method is for internal flopy use and is not intended for the end user.

Parameters

- **first_line** (*str*) - A string containing the first line of data in this array.
- **file_handle** (*file descriptor*) - A file handle for the data file which points to the second line of data for this array
- **block_header** (*MFBBlockHeader*) - Block header object that contains block header information for the block containing this data
- **pre_data_comments** (*MFComment*) - Comments immediately prior to the data

- **external_file_info** (*list*) - Contains information about storing files externally

Returns

- **more data** (*bool*,)
- **next data line** (*str*)

make_layered()

Changes the data to be stored by layer instead of as a single array.

new_simulation(sim_data)

Initialize MFArrary object for a new simulation

Parameters

sim_data (*MFSimulationData*) - Data dictionary containing simulation data.

plot(filename_base=None, file_extension=None, mflay=None, fignum=None, title=None, **kwargs)

Plot 3-D model input data

Parameters

- **filename_base** (*str*) - Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)
- **file_extension** (*str*) - Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')
- **mflay** (*int*) - MODFLOW zero-based layer number to return. If None, then all layers will be included. (default is None)
- ****kwargs** (*dict*) -

axes

[list of matplotlib.pyplot.axis] List of matplotlib.pyplot.axis that will be used to plot data for each layer. If axes=None axes will be generated. (default is None)

pcolor

[bool] Boolean used to determine if matplotlib.pyplot.pcolormesh plot will be plotted. (default is True)

colorbar

[bool] Boolean used to determine if a color bar will be added to the matplotlib.pyplot.pcolormesh. Only used if pcolor=True. (default is False)

inactive

[bool] Boolean used to determine if a black overlay in inactive cells in a layer will be displayed. (default is True)

contour

[bool] Boolean used to determine if

matplotlib.pyplot.contour plot will be plotted. (default is False)

clabel

[bool] Boolean used to determine if matplotlib.pyplot.clabel will be plotted. Only used if contour=True. (default is False)

grid

[bool] Boolean used to determine if the model grid will be plotted on the figure. (default is False)

masked_values

[list] List of unique values to be excluded from the plot.

Returns

out - Empty list is returned if filename_base is not None. Otherwise a list of matplotlib.pyplot.axis is returned.

Return type

list

property plottable

If the array is plottable

set_data(data, multiplier=None, layer=None)

Sets the contents of the data at layer *layer* to *data* with multiplier *multiplier*. For unlayered data do not pass in *layer*. Data can have the following formats: 1) ndarray - numpy ndarray containing all of the data 2) [data] - python list containing all of the data 3) val - a single constant value to be used for all of the data

Parameters

- **data** (ndarray/list) - An ndarray or nested lists containing the data to set.
- **multiplier** (float) - Multiplier to apply to data
- **layer** (int) - Data layer that is being set

set_layered_data(layered_data)

Sets whether this MFArrray supports layered data

Parameters

layered_data (bool) - Whether data is layered or not.

set_record(data_record)

Sets the contents of the data and metadata to the contents of 'data_record'. For unlayered data do not pass in *layer*. For unlayered data 'data_record' is a dictionary with either of the following key/value pairs: 1) {'filename':filename, 'factor':fct, 'iprn':print, 'data':data} - dictionary defining external file information 2) {'data':data, 'factor':fct, 'iprn':print}- dictionary defining internal information. Data that is layered can also be set by defining For layered data data_record must be a dictionary of dictionaries with zero-based layer numbers for the outer dictionary keys and the inner dictionary as described above:

```
{0: {'data':data_layer_0, 'factor':fct_layer_0, 'iprn':prn_layer_0},
 1: {'data':data_layer_1, 'factor':fct_layer_1, 'iprn':prn_layer_1}}, 2:
{'data':data_layer_2, 'factor':fct_layer_2, 'iprn':prn_layer_2}}}
```

Parameters

data_record (*dict*) - An dictionary of data record information or a dictionary of layers (keys) with a dictionary of data record information for each layer (values).

store_as_external_file(*external_file_path*, *layer=None*, *binary=False*,
replace_existing_external=True, *check_data=True*)

Stores data from layer *layer* to an external file at *external_file_path*. For unlayered data do not pass in *layer*. If *layer* is not specified all layers will be stored with each layer as a separate file. If *replace_existing_external* is set to *False*, this method will not do anything if the data is already in an external file.

Parameters

- **external_file_path** (*str*) - Path to external file
- **layer** (*int*) - Which layer to store in external file, *None* value stores all layers.
- **binary** (*bool*) - Store data in a binary file
- **replace_existing_external** (*bool*) - Whether to replace an existing external file.
- **check_data** (*bool*) - Verify data prior to storing

store_internal(*layer=None*, *check_data=True*)

Stores data from layer *layer* internally. For unlayered data do not pass in *layer*. If *layer* is not specified all layers will be stored internally

Parameters

- **layer** (*int*) - Which layer to store in external file, *None* value stores all layers.
- **check_data** (*bool*) - Verify data prior to storing

supports_layered()

Returns whether this MFArrary supports layered data

Returns

layered data supported - Whether or not this data object supports layered data

Return type

bool

class MFTransientArray(*sim_data*, *model_or_sim*, *structure*, *enable=True*, *path=None*,
dimensions=None, *block=None*)

Bases: *MFArrary*, *MFTransient*

Provides an interface for the user to access and update MODFLOW transient array data. *MFTransientArray* objects are not designed to be directly constructed by the end user. When a FloPy for MODFLOW 6 package object is constructed, the appropriate *MFArrary* objects are automatically built.

Parameters

- **sim_data** (*MFSimulationData*) - data contained in the simulation
- **structure** (*MFDataStructure*) - describes the structure of the data

- **data** (*list* or *ndarray*) - actual data
- **enable** (*bool*) - enable/disable the array
- **path** (*tuple*) - path in the data dictionary to this MFArray
- **dimensions** (*MFDataDimensions*) - dimension information related to the model, package, and array

Examples

add_transient_key(*transient_key*)

Adds a new transient time allowing data for that time to be stored and retrieved using the key *transient_key*. This method is intended for internal library usage only.

Parameters

transient_key (*int*) - Zero-based stress period

property data_type

Type of data (*DataType*) stored in the array

get_data(*key=None*, *apply_mult=True*, ***kwargs*)

Returns the data associated with stress period key *key*. If *layer* is *None*, returns all data for time *key*.

Parameters

- **key** (*int*) - Zero-based stress period of data to return
- **apply_mult** (*bool*) - Whether to apply multiplier to data prior to returning it

get_file_entry(*key=0*, *ext_file_action=ExtFileAction.copy_relative_paths*)

Returns a string containing the data in stress period “key”.

Parameters

- **key** (*int*) - The stress period to return file entry for.
- **ext_file_action** (*ExtFileAction*) - How to handle external paths.

Returns

file entry

Return type

str

get_record(*key=None*)

Returns the data record (data and metadata) associated with stress period key *key*. If *layer* is *None*, returns all data for time *key*.

Parameters

key (*int*) - Zero-based stress period of data to return

has_data(*layer=None*)

Returns whether layer “*layer_num*” has any data associated with it.

Parameters

layer_num (*int*) - Layer number to check for data. For unlayered data do not pass anything in

Returns

has data - Returns if there is data.

Return type

`bool`

`load(first_line, file_handle, block_header, pre_data_comments=None, external_file_info=None)`

Loads data from `first_line` (the first line of data) and open file handle which is pointing to the second line of data. Returns a tuple with the first item indicating whether all data was read and the second item being the last line of text read from the file. This method is for internal flopy use and is not intended to be called by the end user.

Parameters

- **first_line** (`str`) - A string containing the first line of data in this array.
- **file_handle** (*file descriptor*) - A file handle for the data file which points to the second line of data for this array
- **block_header** (`MFBBlockHeader`) - Block header object that contains block header information for the block containing this data
- **pre_data_comments** (`MFComment`) - Comments immediately prior to the data
- **external_file_info** (`list`) - Contains information about storing files externally

Returns

- **more data** (`bool`,)
- **next data line** (`str`)

`plot(kper=None, filename_base=None, file_extension=None, mflay=None, fignum=None, **kwargs)`

Plot transient array model input data

Parameters

- **transient2d** (*flopy.utils.util_array.Transient2D object*) -
- **filename_base** (`str`) - Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if `file_name_base` is not `None`. (default is `None`)
- **file_extension** (`str`) - Valid matplotlib.pyplot file extension for `savefig()`. Only used if `filename_base` is not `None`. (default is `'png'`)
- ****kwargs** (`dict`) -
 - axes**
[`list of matplotlib.pyplot.axis`] List of `matplotlib.pyplot.axis` that will be used to plot data for each layer. If `axes=None` axes will be generated. (default is `None`)

pcolor

[bool] Boolean used to determine if matplotlib.pyplot.pcolormesh plot will be plotted. (default is True)

colorbar

[bool] Boolean used to determine if a color bar will be added to the matplotlib.pyplot.pcolormesh. Only used if pcolor=True. (default is False)

inactive

[bool] Boolean used to determine if a black overlay in inactive cells in a layer will be displayed. (default is True)

contour

[bool] Boolean used to determine if matplotlib.pyplot.contour plot will be plotted. (default is False)

clabel

[bool] Boolean used to determine if matplotlib.pyplot.clabel will be plotted. Only used if contour=True. (default is False)

grid

[bool] Boolean used to determine if the model grid will be plotted on the figure. (default is False)

masked_values

[list] List of unique values to be excluded from the plot.

kper

[str] MODFLOW zero-based stress period number to return. If kper='all' then data for all stress period will be extracted. (default is zero).

Returns

axes - Empty list is returned if filename_base is not None. Otherwise a list of matplotlib.pyplot.axis is returned.

Return type

`list`

remove_transient_key(transient_key)

Removes a new transient time *transient_key* and any data stored at that time. This method is intended for internal library usage only.

Parameters

transient_key (*int*) - Zero-based stress period

set_data(data, multiplier=None, layer=None, key=None)

Sets the contents of the data at layer *layer* and time *key* to *data* with multiplier *multiplier*. For unlayered data do not pass in *layer*.

Parameters

- **data** (*dict*, *ndarray*, *list*) - Data being set. Data can be a dictionary with keys as zero-based stress periods and values as the data. If data is an ndarray or list of lists, it will

be assigned to the the stress period specified in *key*. If any is set to *None*, that stress period of data will be removed.

- **multiplier** (*int*) - multiplier to apply to data
- **layer** (*int*) - Layer of data being set. Keep default of *None* of data is not layered.
- **key** (*int*) - Zero based stress period to assign data too. Does not apply if *data* is a dictionary.

set_record(*data_record*)

Sets data and metadata at layer *layer* and time *key* to *data_record*. For unlayered data do not pass in *layer*.

Parameters

data_record (*dict*) - Data record being set. Data must be a dictionary with keys as zero-based stress periods and values as a dictionary of data and metadata (factor, iprn, filename, binary, data) for a given stress period. How to define the dictionary of data and metadata is described in the MFDData class's *set_record* method.

store_as_external_file(*external_file_path*, *layer=None*, *binary=False*, *replace_existing_external=True*, *check_data=True*)

Stores data from layer *layer* to an external file at *external_file_path*. For unlayered data do not pass in *layer*. If *layer* is not specified all layers will be stored with each layer as a separate file. If *replace_existing_external* is set to *False*, this method will not do anything if the data is already in an external file.

Parameters

- **external_file_path** (*str*) - Path to external file
- **layer** (*int*) - Which layer to store in external file, *None* value stores all layers.
- **binary** (*bool*) - Store data in a binary file
- **replace_existing_external** (*bool*) - Whether to replace an existing external file.
- **check_data** (*bool*) - Verify data prior to storing

store_internal(*layer=None*, *check_data=True*)

Stores data from layer *layer* internally. For unlayered data do not pass in *layer*. If *layer* is not specified all layers will be stored internally.

Parameters

- **layer** (*int*) - Which layer to store internally file, *None* value stores all layers.
- **check_data** (*bool*) - Verify data prior to storing

flopy.mf6.data.mfdatalist module

```
class MFList(sim_data, model_or_sim, structure, data=None, enable=True, path=None,
            dimensions=None, package=None, block=None)
```

Bases: MFMultiDimVar, DataListInterface

Provides an interface for the user to access and update MODFLOW list data. MFList objects are not designed to be directly constructed by the end user. When a flopy for MODFLOW 6 package object is constructed, the appropriate MFList objects are automatically built.

Parameters

- **sim_data** (MFSimulationData) - data contained in the simulation
- **structure** (MFDataStructure) - describes the structure of the data
- **data** (*list* or *ndarray*) - actual data
- **enable** (*bool*) - enable/disable the array
- **path** (*tuple*) - path in the data dictionary to this MFArray
- **dimensions** (MFDataDimensions) - dimension information related to the model, package, and array

```
append_data(data)
```

Appends “data” to the end of this list. Assumes data is in a format that can be appended directly to a numpy recarray.

Parameters

data (*list(tuple)*) - Data to append.

```
append_list_as_record(record)
```

Appends the list *record* as a single record in this list’s recarray. Assumes “data” has the correct dimensions.

Parameters

record (*list*) - List to be appended as a single record to the data’s existing recarray.

```
property data_type
```

Type of data (DataType) stored in the list

```
property dtype
```

Type of data (numpy.dtype) stored in the list

```
get_data(apply_mult=False, **kwargs)
```

Returns the list’s data.

Parameters

apply_mult (*bool*) - Whether to apply a multiplier.

Returns

data

Return type

recarray

```
get_file_entry(ext_file_action=ExtFileAction.copy_relative_paths)
```

Returns a string containing the data formatted for a MODFLOW 6 file.

Parameters

ext_file_action ([ExtFileAction](#)) - How to handle external paths.

Returns

file entry

Return type

[str](#)

get_record()

Returns the list's data and metadata in a dictionary. Data is in key "data" and metadata in keys "filename" and "binary".

Returns

data_record

Return type

[dict](#)

has_data(key=None)

Returns whether this MFList has any data associated with it.

load(first_line, file_handle, block_header, pre_data_comments=None, external_file_info=None)

Loads data from first_line (the first line of data) and open file file_handle which is pointing to the second line of data. Returns a tuple with the first item indicating whether all data was read and the second item being the last line of text read from the file. This method was only designed for internal FloPy use and is not recommended for end users.

Parameters

- **first_line** ([str](#)) - A string containing the first line of data in this list.
- **file_handle** (*file descriptor*) - A file handle for the data file which points to the second line of data for this list
- **block_header** ([MFBlockHeader](#)) - Block header object that contains block header information for the block containing this data
- **pre_data_comments** ([MFComment](#)) - Comments immediately prior to the data
- **external_file_info** ([list](#)) - Contains information about storing files externally

Returns

- **more data** (*bool*,)
- **next data line** (*str*)

new_simulation(sim_data)

Initialize MFList object for a new simulation.

Parameters

sim_data ([MFSimulationData](#)) - Simulation data object for the simulation containing this data.

property package

Package object that this data belongs to.

`plot(key=None, names=None, filename_base=None, file_extension=None, mflay=None, **kwargs)`

Plot boundary condition (MfList) data

Parameters

- **key** (*str*) - MfList dictionary key. (default is None)
- **names** (*list*) - List of names for figure titles. (default is None)
- **filename_base** (*str*) - Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)
- **file_extension** (*str*) - Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')
- **mflay** (*int*) - MODFLOW zero-based layer number to return. If None, then all all layers will be included. (default is None)
- ****kwargs** (*dict*) -

axes

[list of matplotlib.pyplot.axis] List of matplotlib.pyplot.axis that will be used to plot data for each layer. If axes=None axes will be generated. (default is None)

pcolor

[bool] Boolean used to determine if matplotlib.pyplot.pcolormesh plot will be plotted. (default is True)

colorbar

[bool] Boolean used to determine if a color bar will be added to the matplotlib.pyplot.pcolormesh. Only used if pcolor=True. (default is False)

inactive

[bool] Boolean used to determine if a black overlay in inactive cells in a layer will be displayed. (default is True)

contour

[bool] Boolean used to determine if matplotlib.pyplot.contour plot will be plotted. (default is False)

clabel

[bool] Boolean used to determine if matplotlib.pyplot.clabel will be plotted. Only used if contour=True. (default is False)

grid

[bool] Boolean used to determine if the model grid will be plotted on the figure. (default is False)

masked_values

[list] List of unique values to be excluded from the plot.

Returns

out - Empty list is returned if filename_base is not None. Otherwise a list of matplotlib.pyplot.axis is returned.

Return type

list

property plottable

If this list data is plottable

search_data(search_term, col=None)

Searches the list data at column “col” for “search_term”. If col is None search_data searches the entire list.

Parameters

- **search_term** (*str*) - String to search for
- **col** (*int*) - Column number to search

set_data(data, autofill=False, check_data=True)

Sets the contents of the data to “data”. Data can have the following formats:

- 1) recarray - recarray containing the datalist
- 2) [(line_one), (line_two), ...] - list where each line of the datalist is a tuple within the list

If the data is transient, a dictionary can be used to specify each stress period where the dictionary key is <stress period> - 1 and the dictionary value is the datalist data defined above: {0:ndarray, 1:[(line_one), (line_two), ...], 2:{‘filename’:filename}}

Parameters

- **data** (*ndarray/list/dict*) - Data to set
- **autofill** (*bool*) - Automatically correct data
- **check_data** (*bool*) - Whether to verify the data

set_record(data_record, autofill=False, check_data=True)

Sets the contents of the data and metadata to “data_record”. Data_record is a dictionary with has the following format:

```
{‘filename’:filename, ‘binary’:True/False, ‘data’=data}
```

To store to file include ‘filename’ in the dictionary.

Parameters

- **data_record** (*ndarray/list/dict*) - Data and metadata to set
- **autofill** (*bool*) - Automatically correct data
- **check_data** (*bool*) - Whether to verify the data


```
store_as_external_file(external_file_path, binary=False,
                       replace_existing_external=True, check_data=True)
```

Store all data externally in file `external_file_path`. the binary allows storage in a binary file. If `replace_existing_external` is set to `False`, this method will not do anything if the data is already in an external file.

Parameters

- **external_file_path** (*str*) - Path to external file
- **binary** (*bool*) - Store data in a binary file
- **replace_existing_external** (*bool*) - Whether to replace an existing external file.
- **check_data** (*bool*) - Verify data prior to storing

```
store_internal(check_data=True)
```

Store all data internally.

Parameters

- **check_data** (*bool*) - Verify data prior to storing

```
to_array(kper=0, mask=False)
```

Convert stress period boundary condition (MFDataList) data for a specified stress period to a 3-D numpy array.

Parameters

- **kper** (*int*) - MODFLOW zero-based stress period number to return. (default is zero)
- **mask** (*bool*) - return array with `np.NaN` instead of zero

Returns

out - Dictionary of 3-D numpy arrays containing the stress period data for a selected stress period. The dictionary keys are the MFDataList dtype names for the stress period data.

Return type

dict of `numpy.ndarrays`

```
update_record(record, key_index)
```

Updates a record at index “key_index” with the contents of “record”. If the index does not exist `update_record` appends the contents of “record” to this list’s recarray.

Parameters

- **record** (*list*) - New record to update data with
- **key_index** (*int*) - Stress period key of record to update. Only used in transient data types.

```
class MFMultipleList(sim_data, model_or_sim, structure, enable=True, path=None,
                    dimensions=None, package=None, block=None)
```

Bases: `MFTransientList`

Provides an interface for the user to access and update MODFLOW multiple list data. This is list data that is in the same format as the `MFTransientList`, but is not time based.

Parameters

- **sim_data** (*MFSimulationData*) - data contained in the simulation
- **structure** (*MFDataStructure*) - describes the structure of the data
- **data** (*list* or *ndarray*) - actual data
- **enable** (*bool*) - enable/disable the array
- **path** (*tuple*) - path in the data dictionary to this MFArray
- **dimensions** (*MFDataDimensions*) - dimension information related to the model, package, and array

get_data(key=None, apply_mult=False, **kwargs)

Returns the data for stress period key.

Parameters

- **key** (*int*) - Zero-based stress period to return data from.
- **apply_mult** (*bool*) - Apply multiplier

Returns

data

Return type

ndarray

class MFTransientList(sim_data, model_or_sim, structure, enable=True, path=None, dimensions=None, package=None, block=None)

Bases: *MFList*, *MFTransient*, *DataListInterface*

Provides an interface for the user to access and update MODFLOW transient list data.

Parameters

- **sim_data** (*MFSimulationData*) - data contained in the simulation
- **structure** (*MFDataStructure*) - describes the structure of the data
- **data** (*list* or *ndarray*) - actual data
- **enable** (*bool*) - enable/disable the array
- **path** (*tuple*) - path in the data dictionary to this MFArray
- **dimensions** (*MFDataDimensions*) - dimension information related to the model, package, and array

add_transient_key(transient_key)

Adds a new transient time allowing data for that time to be stored and retrieved using the key *transient_key*. Method is used internally by FloPy and is not intended to the end user.

Parameters

transient_key (*int*) - Zero-based stress period to add

append_list_as_record(record, key=0)

Appends the list data as a single record in this list's recarray at time key. Assumes data has the correct dimensions.

Parameters

- **data** (*list*) - Data to append
- **key** (*int*) - Zero based stress period to append data too.

property data

Returns list data. Calls `get_data` with default parameters.

property data_type

Type of data (`DataType`) stored in the list

property dtype

Type of data (`numpy.dtype`) stored in the list

get_data(*key=None, apply_mult=False, **kwargs*)

Returns the data for stress period *key*.

Parameters

- **key** (*int*) - Zero-based stress period to return data from.
- **apply_mult** (*bool*) - Apply multiplier

Returns

data

Return type

`recarray`

get_file_entry(*key=0, ext_file_action=ExtFileAction.copy_relative_paths*)

Returns a string containing the data at time *key* formatted for a MODFLOW 6 file.

Parameters

- **key** (*int*) - Zero based stress period to return data from.
- **ext_file_action** (`ExtFileAction`) - How to handle external paths.

Returns

file entry

Return type

`str`

get_record(*key=None*)

Returns the data for stress period *key*. If no *key* is specified returns all records in a dictionary with zero-based stress period numbers as keys. See MFList's `get_record` documentation for more information on the format of each record returned.

Parameters

- key** (*int*) - Zero-based stress period to return data from.

Returns

data_record

Return type

`dict`

has_data(*key=None*)

Returns whether this MFList has any data associated with it in key "key".

load(*first_line, file_handle, block_header, pre_data_comments=None, external_file_info=None*)

Loads data from `first_line` (the first line of data) and open file `file_handle` which is pointing to the second line of data. Returns a tuple with the first item indicating whether all data was read and the second item being the last line of text read from the file.

Parameters

- **first_line** (*str*) - A string containing the first line of data in this list.
- **file_handle** (*file descriptor*) - A file handle for the data file which points to the second line of data for this array
- **block_header** (*MFBBlockHeader*) - Block header object that contains block header information for the block containing this data
- **pre_data_comments** (*MFComment*) - Comments immediately prior to the data
- **external_file_info** (*list*) - Contains information about storing files externally

property `masked_4D_arrays`

Returns list data as a masked 4D array.

`masked_4D_arrays_itr()`

Returns list data as an iterator of a masked 4D array.

plot(*key=None, names=None, kper=0, filename_base=None, file_extension=None, mflay=None, **kwargs*)

Plot stress period boundary condition (*MfList*) data for a specified stress period

Parameters

- **key** (*str*) - *MfList* dictionary key. (default is *None*)
- **names** (*list*) - List of names for figure titles. (default is *None*)
- **kper** (*int*) - MODFLOW zero-based stress period number to return. (default is zero)
- **filename_base** (*str*) - Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if *file_name_base* is not *None*. (default is *None*)
- **file_extension** (*str*) - Valid matplotlib.pyplot file extension for *savefig()*. Only used if *filename_base* is not *None*. (default is 'png')
- **mflay** (*int*) - MODFLOW zero-based layer number to return. If *None*, then all all layers will be included. (default is *None*)
- ****kwargs** (*dict*) -

`axes`

[list of matplotlib.pyplot.axis] List of matplotlib.pyplot.axis that will be used to plot data for

each layer. If axes=None axes will be generated. (default is None)

pcolor

[bool] Boolean used to determine if matplotlib.pyplot.pcolormesh plot will be plotted. (default is True)

colorbar

[bool] Boolean used to determine if a color bar will be added to the matplotlib.pyplot.pcolormesh. Only used if pcolor=True. (default is False)

inactive

[bool] Boolean used to determine if a black overlay in inactive cells in a layer will be displayed. (default is True)

contour

[bool] Boolean used to determine if matplotlib.pyplot.contour plot will be plotted. (default is False)

clabel

[bool] Boolean used to determine if matplotlib.pyplot.clabel will be plotted. Only used if contour=True. (default is False)

grid

[bool] Boolean used to determine if the model grid will be plotted on the figure. (default is False)

masked_values

[list] List of unique values to be excluded from the plot.

Returns

out - Empty list is returned if filename_base is not None. Otherwise a list of matplotlib.pyplot.axis is returned.

Return type

`list`

remove_transient_key(transient_key)

Remove transient stress period key. Method is used internally by FloPy and is not intended to the end user.

set_data(data, key=None, autofill=False)

Sets the contents of the data at time *key* to *data*.

Parameters

- **data** (`dict`, `recarray`, `list`) - Data being set. Data can be a dictionary with keys as zero-based stress periods and values as the data. If data is a recarray or list of tuples, it will be assigned to the stress period specified in *key*. If any is set to None, that stress period of data will be removed.
- **key** (`int`) - Zero based stress period to assign data too. Does not apply if *data* is a dictionary.
- **autofill** (`bool`) - Automatically correct data.

set_record(*data_record*, *autofill*=False, *check_data*=True)

Sets the contents of the data based on the contents of 'data_record'.

Parameters

- **data_record** (*dict*) - Data_record being set. Data_record must be a dictionary with keys as zero-based stress periods and values as dictionaries containing the data and metadata. See MFList's set_record documentation for more information on the format of the values.
- **autofill** (*bool*) - Automatically correct data
- **check_data** (*bool*) - Whether to verify the data

store_as_external_file(*external_file_path*, *binary*=False, *replace_existing_external*=True, *check_data*=True)

Store all data externally in file *external_file_path*. the *binary* allows storage in a binary file. If *replace_existing_external* is set to False, this method will not do anything if the data is already in an external file.

Parameters

- **external_file_path** (*str*) - Path to external file
- **binary** (*bool*) - Store data in a binary file
- **replace_existing_external** (*bool*) - Whether to replace an existing external file.
- **check_data** (*bool*) - Verify data prior to storing

store_internal(*check_data*=True)

Store all data internally.

Parameters

- **check_data** (*bool*) - Verify data prior to storing

to_array(*kper*=0, *mask*=False)

Returns list data as an array.

update_record(*record*, *key_index*, *key*=0)

Updates a record at index *key_index* and time *key* with the contents of *record*. If the index does not exist *update_record* appends the contents of *record* to this list's recarray.

Parameters

- **record** (*list*) - Record to append
- **key_index** (*int*) - Index to update
- **key** (*int*) - Zero based stress period to append data too

flopy.mf6.data.mfdatascalar module

```
class MFScalar(sim_data, model_or_sim, structure, data=None, enable=True, path=None,
               dimensions=None)
```

Bases: MFData

Provides an interface for the user to access and update MODFLOW scalar data. MFScalar objects are not designed to be directly constructed by the end user. When a flopy for MODFLOW 6 package object is constructed, the appropriate MFScalar objects are automatically built.

Parameters

- **sim_data** (MFSimulationData) - data contained in the simulation
- **structure** (MFDataStructure) - describes the structure of the data
- **data** (*list* or *ndarray*) - actual data
- **enable** (*bool*) - enable/disable the array
- **path** (*tuple*) - path in the data dictionary to this MFArray
- **dimensions** (MFDataDimensions) - dimension information related to the model, package, and array

```
add_one()
```

Adds one if this is an integer scalar

property data

Returns the scalar data. Calls get_data with default parameters.

property data_type

Type of data (DataType) stored in the scalar

property dtype

The scalar's numpy data type (numpy.dtype).

```
get_data(apply_mult=False, **kwargs)
```

Returns the data associated with this object.

Parameters

- **apply_mult** (*bool*) - Parameter does not apply to scalar data.

Returns

data

Return type

str, int, float, recarray

```
get_file_entry(values_only=False, one_based=False,
               ext_file_action=ExtFileAction.copy_relative_paths)
```

Returns a string containing the data formatted for a MODFLOW 6 file.

Parameters

- **values_only** (*bool*) - Return values only excluding keywords
- **one_based** (*bool*) - Return one-based integer values
- **ext_file_action** (ExtFileAction) - How to handle external paths.

Returns
file entry

Return type
`str`

has_data(key=None)

Returns whether this object has data associated with it.

load(first_line, file_handle, block_header, pre_data_comments=None, external_file_info=None)

Loads data from first_line (the first line of data) and open file file_handle which is pointing to the second line of data. Returns a tuple with the first item indicating whether all data was read and the second item being the last line of text read from the file. This method was only designed for internal FloPy use and is not recommended for end users.

Parameters

- **first_line** (`str`) - A string containing the first line of data.
- **file_handle** (*file descriptor*) - A file handle for the data file which points to the second line of data
- **block_header** (`MFBBlockHeader`) - Block header object that contains block header information for the block containing this data
- **pre_data_comments** (`MFComment`) - Comments immediately prior to the data
- **external_file_info** (`list`) - Contains information about storing files externally

Returns

- **more data** (*bool*,)
- **next data line** (*str*)

plot(filename_base=None, file_extension=None, **kwargs)

Helper method to plot scalar objects

Parameters

- **scalar** - flopy.mf6.data.mfscalar object
- **filename_base** - str Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)
- **file_extension** - str Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')

Returns

list matplotlib.axes object

Return type
axes

property plottable

If the scalar is plottable. Currently all scalars are not plottable.

set_data(data)

Sets the contents of the data to *data*.

Parameters

data (*str/int/float/recarray/list*) - Data to set

class MFScalarTransient(*sim_data, model_or_sim, structure, enable=True, path=None, dimensions=None*)

Bases: [MFScalar](#), [MFTransient](#)

Provides an interface for the user to access and update MODFLOW transient scalar data. Transient scalar data is used internally by FloPy and should not be used directly by the end user.

Parameters

- **sim_data** ([MFSimulationData](#)) - data contained in the simulation
- **structure** ([MFDataStructure](#)) - describes the structure of the data
- **data** (*list* or *ndarray*) - actual data
- **enable** (*bool*) - enable/disable the array
- **path** (*tuple*) - path in the data dictionary to this MFArray
- **dimensions** ([MFDataDimensions](#)) - dimension information related to the model, package, and array

add_one(key=0)

Adds one to the data stored at key *key*. Method is used internally by FloPy and is not intended to the end user.

Parameters

key (*int*) - Zero-based stress period to add

add_transient_key(key)

Adds a new transient time allowing data for that time to be stored and retrieved using the key *key*. Method is used internally by FloPy and is not intended to the end user.

Parameters

key (*int*) - Zero-based stress period to add

property data_type

Type of data (*DataType*) stored in the scalar

get_data(key=0, **kwargs)

Returns the data for stress period *key*.

Parameters

key (*int*) - Zero-based stress period to return data from.

Returns

data

Return type

str/int/float/recarray

get_file_entry(key=None, ext_file_action=ExtFileAction.copy_relative_paths)

Returns a string containing the data at time key formatted for a MODFLOW 6 file.

Parameters

- **key** (*int*) - Zero based stress period to return data from.
- **ext_file_action** (*ExtFileAction*) - How to handle external paths.

Returns

file entry

Return type

str

has_data(key=None)

Returns whether this object has data associated with it.

load(first_line, file_handle, block_header, pre_data_comments=None, external_file_info=None)

Loads data from first_line (the first line of data) and open file file_handle which is pointing to the second line of data. Returns a tuple with the first item indicating whether all data was read and the second item being the last line of text read from the file.

Parameters

- **first_line** (*str*) - A string containing the first line of data in this scalar.
- **file_handle** (*file descriptor*) - A file handle for the data file which points to the second line of data for this array
- **block_header** (*MFBBlockHeader*) - Block header object that contains block header information for the block containing this data
- **pre_data_comments** (*MFComment*) - Comments immediately prior to the data
- **external_file_info** (*list*) - Contains information about storing files externally

plot(filename_base=None, file_extension=None, kper=0, fignum=None, **kwargs)

Plot transient scalar model data

Parameters

- **transientscalar** (*flopy.mf6.data.mfdatascalar.MFScalarTransient object*) -
- **filename_base** (*str*) - Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)
- **file_extension** (*str*) - Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')

- ****kwargs** (*dict*) -

axes

[list of matplotlib.pyplot.axis] List of matplotlib.pyplot.axis that will be used to plot data for each layer. If axes=None axes will be generated. (default is None)

pcolor

[bool] Boolean used to determine if matplotlib.pyplot.pcolormesh plot will be plotted. (default is True)

colorbar

[bool] Boolean used to determine if a color bar will be added to the matplotlib.pyplot.pcolormesh. Only used if pcolor=True. (default is False)

inactive

[bool] Boolean used to determine if a black overlay in inactive cells in a layer will be displayed. (default is True)

contour

[bool] Boolean used to determine if matplotlib.pyplot.contour plot will be plotted. (default is False)

clabel

[bool] Boolean used to determine if matplotlib.pyplot.clabel will be plotted. Only used if contour=True. (default is False)

grid

[bool] Boolean used to determine if the model grid will be plotted on the figure. (default is False)

masked_values

[list] List of unique values to be excluded from the plot.

kper

[str] MODFLOW zero-based stress period number to return. If kper='all' then data for all stress period will be extracted. (default is zero).

Returns

axes - Empty list is returned if filename_base is not None. Otherwise a list of matplotlib.pyplot.axis is returned.

Return type

list

property plottable

If the scalar is plottable

set_data(data, key=None)

Sets the contents of the data at time key to data.

Parameters

- **data** (*str/int/float/recarray/list*) - Data being set. Data can be a dictionary with keys as zero-based stress periods and values as the data. If data is a string, integer, double, recarray, or list of tuples, it will be assigned to the stress period specified in key. If any is set to None, that stress period of data will be removed.
- **key** (*int*) - Zero based stress period to assign data too. Does not apply if data is a dictionary.

8.1.10 Build MODFLOW 6 Classes

MODFLOW 6 FloPy classes can be rebuild from MODFLOW 6 definition files. This will allow creation of MODFLOW 6 FloPy classes for development versions of MODFLOW 6.

Contents:

flopy.mf6.utils.createpackages module

createpackages.py is a utility script that reads in the file definition metadata in the .dfn files and creates the package classes in the modflow folder. Run this script any time changes are made to the .dfn files.

To create a new package that is part of an existing model, first create a new dfn file for the package in the mf6/data/dfn folder. 1) Follow the file naming convention <model abbr>-<package abbr>.dfn. 2) Run this script (createpackages.py), and check in your new dfn file, and

the package class and updated __init__.py that createpackages.py created.

A subpackage is a package referenced by another package (vs being referenced in the name file). The tas, ts, and obs packages are examples of subpackages. There are a few additional steps required when creating a subpackage definition file. First, verify that the parent package's dfn file has a file record for the subpackage to the option block. For example, for the time series package the file record definition starts with:

```
block options name ts_filerecord type record ts6 filein ts6_filename
```

Verify that the same naming convention is followed as the example above, specifically:

```
name <subpackage-abbr>_filerecord record <subpackage-abbr>6 filein
<subpackage-abbr>6_filename
```

Next, create the child package definition file in the mf6/data/dfn folder following the naming convention above.

When your child package is ready for release follow the same procedure as other packages along with these a few additional steps required for subpackages.

At the top of the child dfn file add two lines describing how the parent and child packages are related. The first line determines how the subpackage is linked to the package:

```
# flopy subpackage <parent record> <abbreviation> <child data> <data name>
```

- Parent record is the MF6 record name of the filerecord in parent package that references the child packages file name
- Abbreviation is the short abbreviation of the new subclass

- Child data is the name of the child class data that can be passed in as parameter to the parent class. Passing in this parameter to the parent class automatically creates the child class with the data provided.
- Data name is the parent class parameter name that automatically creates the child class with the data provided.

The example below is the first line from the ts subpackage dfn:

```
# flopy subpackage ts_filerecord ts timeseries timeseries
```

The second line determines the variable name of the subpackage's parent and the type of parent (the parent package's object oriented parent):

```
# flopy parent_name_type <parent package variable name> <parent package type>
```

An example below is the second line in the ts subpackage dfn:

```
# flopy parent_name_type parent_package MFPackage
```

There are three possible types (or combination of them) that can be used for "parent package type", MFPackage, MFModel, and MFSimulation. If a package supports multiple types of parents (for example, it can be either in the model namefile or in a package, like the obs package), include all the types supported, separating each type with a / (MFPackage/MFModel).

To create a new type of model choose a unique three letter model abbreviation ("gwf", "gwt", ...). Create a name file dfn with the naming convention <model abbr>-nam.dfn. The name file must have only an options and packages block (see gwf-nam.dfn as an example). Create a new dfn file for each of the packages in your new model, following the naming convention described above.

When your model is ready for release make sure all the dfn files are in the flopy/mf6/data/dfn folder, run createpackages.py, and check in your new dfn files, the package classes, and updated init.py that createpackages.py created.

```
class PackageLevel(value, names=None, *, module=None, qualname=None, type=None, start=1,
                    boundary=None)
```

```
    Bases: Enum
```

```
    model_level = 1
```

```
    sim_level = 0
```

```
add_var(init_vars, class_vars, init_param_list, package_properties, doc_string,
        data_structure_dict, default_value, name, python_name, description, path,
        data_type, basic_init=False, construct_package=None, construct_data=None,
        parameter_name=None, set_param_list=None)
```

```
build_dfn_string(dfn_list, header, package_abbr, flopy_dict)
```

```
build_doc_string(param_name, param_type, param_desc, indent)
```

```
build_init_string(init_string, init_param_list, whitespace=' ')
```

```
build_model_init_vars(param_list)
```

```
build_model_load(model_type)
```

```
clean_class_string(name)
```

```

create_basic_init(clean_ds_name)
create_init_var(clean_ds_name, data_structure_name, init_val=None)
create_package_init_var(parameter_name, package_abbr, data_name, clean_ds_name)
create_packages()
create_property(clean_ds_name)
format_var_list(base_string, var_list, is_tuple=False)
generator_type(data_type)

```

flopy.mf6.utils.generate_classes module

```

backup_existing_dfn(flopy_dfn_path)
delete_files(files, pth, allow_failure=False, exclude=None)
delete_mf6_classes()
download_dfn(owner, branch, new_dfn_pth)
generate_classes(owner='MODFLOW-USGS', branch='master', dfnpth=None, backup=True)

```

Generate the MODFLOW 6 flopy classes using definition files from the MODFLOW 6 GitHub repository or a set of definition files in a folder provided by the user.

Parameters

- **owner** (*str*) - Owner of the MODFLOW 6 repository to use to update the definition files and generate the MODFLOW 6 classes. Default is MODFLOW-USGS.
- **branch** (*str*) - Branch name of the MODFLOW 6 repository to use to update the definition files and generate the MODFLOW 6 classes. Default is master.
- **dfnpth** (*str*) - Path to a definition file folder that will be used to generate the MODFLOW 6 classes. Default is none, which means that the branch will be used instead. dfnpth will take precedence over branch if dfnpth is specified.
- **backup** (*bool*) - Keep a backup of the definition files in dfn_backup with a date and time stamp from when the definition files were replaced.

```

list_files(pth, exts=['py'])
replace_dfn_files(new_dfn_pth, flopy_dfn_path)

```

8.2 Previous Versions of MODFLOW

8.2.1 MODFLOW Base Classes

Contents:

`flopy.mbase` module

`mbase` module

This module contains the base model class from which all of the other models inherit from.

```
class BaseModel(modelname='modflowtest', namefile_ext='nam', exe_name: str | PathLike =
    'mf2005', model_ws: str | PathLike = '.', structured=True, verbose=False,
    **kwargs)
```

Bases: `ModelInterface`

MODFLOW-based models base class.

Parameters

- **modelname** (`str`, default `"modflowtest"`) - Name of the model, which is also used for model file names.
- **namefile_ext** (`str`, default `"nam"`) - Name file extension, without `"."`
- **exe_name** (`str` or `PathLike`, default `"mf2005"`) - Name or path of the modflow executable. If a name is provided, the executable must be on the system path.
- **model_ws** (`str` or `PathLike`, optional, default `"."`) - Path to the model workspace. Model files will be created in this directory. Default is the current working directory.
- **structured** (`bool`, default `True`) - Specify if model grid is structured (default) or unstructured.
- **verbose** (`bool`, default `False`) - Print additional information to the screen.
- ****kwargs** (`dict`, optional) - Used to define: `xll/yll` for the x- and y-coordinates of the lower-left corner of the grid, `xul/yul` for the x- and y-coordinates of the upper-left corner of the grid (deprecated), `rotation` for the grid rotation (default `0.0`), `crs` for the coordinate reference system, and `start_datetime` for model start date (default `"1-1-1970"`).

```
add_existing_package(filename: str | PathLike, ptype=None, copy_to_model_ws=True)
```

Add an existing package to a model instance.

Parameters

- **filename** (`str` or `PathLike`) - Path of the file to add as a package
- **ptype** (optional) - Model package type (e.g. `"lpf"`, `"wel"`, etc). If `None` then the file extension of the filename arg is used

- **copy_to_model_ws** (*bool*) - Copy the package file into the model workspace.

Return type

None

add_external(*fname*: *str* | *PathLike*, *unit*, *binflag*=False, *output*=False)

Assign an external array so that it will be listed as a DATA or DATA(BINARY) entry in the name file. This will allow an outside file package to refer to it.

Parameters

- **fname** (*str* or *PathLike*) - Path of external array
- **unit** (*int*) - Unit number of external array
- **binflag** (*boolean*, *optional*) - Binary or not, default is False

add_output(*fname*: *str* | *PathLike*, *unit*, *binflag*=False, *package*=None)

Assign an external array so that it will be listed as a DATA or DATA(BINARY) entry in the name file. This will allow an outside file package to refer to it.

Parameters

- **fname** (*str*) - filename of external array
- **unit** (*int*) - unit number of external array
- **binflag** (*boolean*) - binary or not. (default is False)

add_output_file(*unit*, *fname*: *str* | *PathLike* | *None* = None, *extension*='cbc', *binflag*=True, *package*=None)

Add an ascii or binary output file for a package

Parameters

- **unit** (*int*) - Unit number of external array
- **fname** (*str* or *PathLike*, *optional*) - Path of external array, default is None
- **extension** (*str*) - Extension to use for the cell-by-cell file. Only used if *fname* is None, default is cbc
- **binflag** (*bool*) - Whether the output file is a binary file, default is True
- **package** (*str*) - The package the output file is attached to, default is None

add_package(*p*)

Add a package.

Parameters

p (*Package object*) -

add_pop_key_list(*key*)

Add a external file unit number to a list that will be used to remove model output (typically binary) files from *ext_unit_dict*.

Parameters

key (*int*) - file unit number

Examples

change_model_ws(*new_pth*: *str* | *PathLike* | *None* = '.', *reset_external*=False)

Change the model work space.

Parameters

new_pth (*str* or *PathLike*) - Path of the new model workspace. If this path does not exist, it will be created. If no value (*None*) is given, the default is the present working directory.

Returns

val - Can be used to see what packages are in the model, and can then be used with `get_package` to pull out individual packages.

Return type

list of strings

check(*f*: *str* | *PathLike* | *None* = *None*, *verbose*=True, *level*=1)

Check model data for common errors.

Parameters

- **f** (*str* or *PathLike*, *optional*, *default None*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If *f* is *None*, check method does not write results to a summary file. (default is *None*)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If *level*=0, summary checks are performed. If *level*=1, full checks are performed.

Return type

None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.check()
```

property **exename**: *str*

export(*f*: *str* | *PathLike*, ***kwargs*)

Method to export a model to netcdf or shapefile based on the extension of the file name (.shp for shapefile, .nc for netcdf)

Parameters

- **f** (*str* or *PathLike*) - The file path
- **kwargs** (*keyword arguments*) -
modelgrid
 [flopy.discretization.Grid instance] user supplied

modelgrid which can be used for exporting in lieu of the modelgrid associated with the model object

Return type

None or Netcdf object

`get_ext_dict_attr(ext_unit_dict=None, unit=None, filetype=None, pop_key=True)`

`get_name_file_entries()`

Get a string representation of the name file.

`get_output(fname: str | PathLike | None = None, unit=None)`

Get an output file from the model by specifying either the file name or the unit number.

Parameters

- **fname** (*str* or *PathLike*, optional) - Path of output array
- **unit** (*int*, optional) - Unit number of output array

`get_output_attribute(fname: str | PathLike | None = None, unit=None, attr=None)`

Get an attribute of a model output file by specifying either the file name or the unit number.

Parameters

- **fname** (*str* or *PathLike*, optional) - path of output array
- **unit** (*int*, optional) - Unit number of output array

`get_package(name)`

Get a package.

Parameters

name (*str*) - Name of the package, 'RIV', 'LPF', etc. (case-insensitive).

Returns

pp - Package object of type *flopy.pakbase.Package*

Return type

Package object

`has_package(name)`

Check if package name is in package list.

Parameters

name (*str*) - Name of the package, 'DIS', 'BAS6', etc. (case-insensitive).

Returns

True if package name exists, otherwise False if not found.

Return type

bool

property `hdry`

property `hnoflo`

property `laycbd`

property laytyp

load_results()

property model_ws: `str`

property modelgrid

property modeltime

property name

Get model name

Returns

`name` - name of model

Return type

`str`

property namefile: `str`

next_ext_unit()

Function to encapsulate next_ext_unit attribute

next_unit(*i=None*)

property packagelist

plot(*SelPackList=None, **kwargs*)

Plot 2-D, 3-D, transient 2-D, and stress period list (MfList) model input data

Parameters

- **SelPackList** (*bool* or *list*) - List of of packages to plot. If SelPackList=None all packages are plotted. (default is None)
- ****kwargs** (*dict*) -

filename_base

[`str`] Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)

file_extension

[`str`] Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')

mflay

[`int`] MODFLOW zero-based layer number to return. If None, then all all layers will be included. (default is None)

kper

[`int`] MODFLOW zero-based stress period number to return. (default is zero)

key

[`str`] MfList dictionary key. (default is None)

Returns

axes - Empty list is returned if filename_base is not None. Otherwise a list of matplotlib.pyplot.axis are returned.

Return type
`list`

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.plot()
```

remove_external(fname: `str` | `PathLike` | `None` = `None`, unit=`None`)

Remove an external file from the model by specifying either the file name or the unit number.

Parameters

- **fname** (`str` or `PathLike`, optional) - Path of external array
- **unit** (`int`, optional) - Unit number of external array

remove_output(fname: `str` | `PathLike` | `None` = `None`, unit=`None`)

Remove an output file from the model by specifying either the file name or the unit number.

Parameters

- **fname** (`str` or `PathLike`, optional) - Path of output array
- **unit** (`int`, optional) - Unit number of output array

remove_package(pname)

Remove a package from this model

Parameters

pname (`string`) - Name of the package, such as 'RIV', 'BAS6', etc.

run_model(silent=`False`, pause=`False`, report=`False`, normal_msg='normal termination')
→ `Tuple[bool, List[str]]`

This method will run the model using `subprocess.Popen`.

Parameters

- **silent** (`boolean`) - Echo run information to screen (default is `True`).
- **pause** (`boolean`, optional) - Pause upon completion (default is `False`).
- **report** (`boolean`, optional) - Save stdout lines to a list (`buff`) which is returned by the method . (default is `False`).
- **normal_msg** (`str`) - Normal termination message used to determine if the run terminated normally. (default is 'normal termination')

Returns

- **success** (`boolean`)
- **buff** (`list of lines of stdout`)

set_model_units()

Every model needs its own set_model_units method

set_output_attribute(fname: *str* | *PathLike* | *None* = *None*, unit=*None*, attr=*None*)

Set a variable in an output file from the model by specifying either the file name or the unit number and a dictionary with attributes to change.

Parameters

- **fname** (*str* or *PathLike*, optional) - Path of output array
- **unit** (*int*, optional) - Unit number of output array

set_version(version)

to_shapefile(filename: *str* | *PathLike*, package_names=*None*, ***kwargs*)

Wrapper function for writing a shapefile for the model grid. If package_names is not None, then search through the requested packages looking for arrays that can be added to the shapefile as attributes

Parameters

- **filename** (*str* or *PathLike*) - Path of the shapefile to write
- **package_names** (*list* of package names (e.g. ["dis", "lpf"]))
- Packages to export data arrays to shapefile. (default is None)

Return type

None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> m.to_shapefile('model.shp', SelPackList)
```

property verbose

property version

write_input(SelPackList=False, check=False)

Write the input.

Parameters

SelPackList (False or *list* of packages) -

write_name_file()

Every Package needs its own writenamefile function

class FileData

Bases: *object*

add_file(fname, unit, binflag=False, output=False, package=None)

class FileDataEntry(fname, unit, binflag=False, output=False, package=None)

Bases: *object*

class ModelInterface

Bases: `object`

abstract `check(f=None, verbose=True, level=1)`

abstract `property exename`

abstract `export(f, **kwargs)`

abstract `get_package_list(ftype=None)`

Get a list of all the package names.

Parameters

ftype (`str`) - Type of package, 'RIV', 'LPF', etc.

Returns

val - Can be used to see what packages are in the model, and can then be used with `get_package` to pull out individual packages.

Return type

`list` of strings

abstract `property hdry`

abstract `property hnoflo`

abstract `property laycbd`

abstract `property laytyp`

abstract `property model_ws`

abstract `property modelgrid`

abstract `property namefile`

abstract `property packagelist`

abstract `property solver_tols`

`update_modelgrid()`

abstract `property verbose`

abstract `property version`

exception `PackageLoadException(error, location="")`

Bases: `Exception`

FloPy package load exception.

resolve_exe(`exe_name: str | PathLike, forgive: bool = False`) → `str`

Resolves the absolute path of the executable, raising `FileNotFoundError` if the executable cannot be found (set `forgive` to `True` to return `None` and warn instead of raising an error).

Parameters

- **exe_name** (`str` or `PathLike`) - The executable's name or path. If only the name is provided, the executable must be on the system path.

- **forgive** (*bool*) - If True and executable cannot be found, return None and warn rather than raising a FileNotFoundError. Defaults to False.

Returns

str

Return type

absolute path to the executable

```
run_model(exe_name: str | PathLike, namefile: str | None, model_ws: str | PathLike =
    '.', silent=False, pause=False, report=False, processors=None, normal_msg='normal
    termination', use_async=False, cargs=None) → Tuple[bool, List[str]]
```

Run the model using subprocess.Popen, optionally collecting stdout and printing timestamped progress. Model workspace, namefile, executable to use, and several other options may be configured, and additional command line arguments may also be provided.

Parameters

- **exe_name** (*str* or *PathLike*) - Executable name or path. If the executable name is provided, the executable must be on the system path. Alternatively, a full path to the executable may be provided.
- **namefile** (*str*, *optional*) - Name of the name file of model to run. The name may be None to run models that don't require a control file (name file)
- **model_ws** (*str* or *PathLike*, *optional*, *default* '.') - Path to the parent directory of the namefile. (default is the current working directory '.')
- **silent** (*boolean*, *default* True) - Whether to suppress model output. (Default is True)
- **pause** (*boolean*, *optional*, *default* False) - Pause and wait for keystroke upon completion. (Default is False)
- **report** (*boolean*, *optional*, *default* False) - Save stdout lines to a list (buff) returned by the method. (Default is False)
- **processors** (*int*) - Number of processors. Parallel simulations are only supported for MODFLOW 6 simulations. (default is None)
- **normal_msg** (*str* or *list*) - Termination message used to determine if the model terminated normally. More than one message can be provided using a list. (Default is 'normal termination')
- **use_async** (*boolean*) - Asynchronously read model stdout and report with timestamps. Good for models taking a long time to run, not good for models that run quickly.
- **cargs** (*str* or *list*, *optional*, *default* None) - Additional command line arguments to pass to the executable. (Default is None)

Returns

- **success** (*boolean*)
- **buff** (*list* of lines of stdout (empty if report is False))

flopy.pakbase module

pakbase module

This module contains the base package class from which all of the other packages inherit from.

```
class Package(parent, extension='glo', name='GLOBAL', unit_number=1, filenames=None,
              allowDuplicates=False)
```

Bases: [*PackageInterface*](#)

Base package class from which most other packages are derived.

Parameters

- **parent** (*object*) - Parent model object.
- **extension** (*str or list*, default "glo") - File extension, without ".", use list to describe more than one.
- **name** (*str or list*, default "GLOBAL") - Package name, use list to describe more than one.
- **unit_number** (*int or list*, default 1) - Unit number, use list to describe more than one.
- **filenames** (*str or list*, default None) -
- **allowDuplicates** (*bool*, default False) - Allow more than one instance of package in parent.

```
static add_to_dtype(dtype, field_names, field_types)
```

Add one or more fields to a structured array data type

Parameters

- **dtype** (*numpy.dtype*) - Input structured array datatype to add to.
- **field_names** (*str or list*) - One or more field names.
- **field_types** (*numpy.dtype or list*) - One or more data types. If one data type is supplied, it is repeated for each field name.

property data_list

```
export(f, **kwargs)
```

Method to export a package to netcdf or shapefile based on the extension of the file name (.shp for shapefile, .nc for netcdf)

Parameters

- **f** (*str*) - filename
- **kwargs** (*keyword arguments*) -

modelgrid

[flopy.discretization.Grid instance] user supplied modelgrid which can be used for exporting in lieu of the modelgrid associated with the model object

Return type

None or Netcdf object


```
level1_arraylist(idx, v, name, txt)
```

```
static load(f: str | bytes | PathLike, model, pak_type, ext_unit_dict=None,
            **kwargs)
```

Default load method for standard boundary packages.

property name

property package_type

property parent

```
plot(**kwargs)
```

Plot 2-D, 3-D, transient 2-D, and stress period list (MfList) package input data

Parameters

****kwargs** (*dict*) -

filename_base

[str] Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)

file_extension

[str] Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')

mflay

[int] MODFLOW zero-based layer number to return. If None, then all all layers will be included. (default is None)

kper

[int] MODFLOW zero-based stress period number to return. (default is zero)

key

[str] MfList dictionary key. (default is None)

Returns

axes - Empty list is returned if filename_base is not None. Otherwise a list of matplotlib.pyplot.axis are returned.

Return type

list

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.dis.plot()
```

property plottable

to_shapefile(*filename*, ****kwargs**)

Export 2-D, 3-D, and transient 2-D model data to shapefile (polygons). Adds an attribute for each layer in each data array

Parameters

filename (*str*) - Shapefile name to write

Return type

None

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.lpf.to_shapefile('test_hk.shp')
```

webdoc()

Open the web documentation.

write_file(*f=None*, *check=False*)

Every Package needs its own write_file function

class PackageInterface

Bases: `object`

check(*f=None*, *verbose=True*, *level=1*, *checktype=None*)

Check package data for common errors.

Parameters

- **f** (*str* or *file handle*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If *f* is None, check method does not write results to a summary file. (default is None)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.
- **checktype** (*check*) - Checker type to be used. By default class `check` is used from `check.py`.

Return type

None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.dis.check()
```

```
abstract property data_list
abstract export(f, **kwargs)
property has_stress_period_data
abstract property name
abstract property package_type
abstract property parent
abstract property plottable
```

8.2.2 MODFLOW Packages

Contents:

`flopy.modflow.mf` module

mf module. Contains the ModflowGlobal, ModflowList, and Modflow classes.

```
class Modflow(modelname='modflowtest', namefile_ext='nam', version='mf2005', exe_name: str |
    PathLike = 'mf2005', structured=True, listunit=2, model_ws: str | PathLike
    = '.', external_path: str | PathLike | None = None, verbose=False,
    **kwargs)
```

Bases: `BaseModel`

MODFLOW Model Class.

Parameters

- **modelname** (`str`, default `"modflowtest"`) - Name of model. This string will be used to name the MODFLOW input that are created with `write_model`.
- **namefile_ext** (`str`, default `"nam"`) - Extension for the namefile.
- **version** (`str`, default `"mf2005"`) - MODFLOW version. Choose one of: `"mf2k"`, `"mf2005"` (default), `"mfnwt"`, or `"mfusg"`.
- **exe_name** (`str` or `PathLike`, default `"mf2005"`) - The name or path of the executable to use.
- **structured** (`bool`, default `True`) - Specify if model grid is structured (default) or unstructured.
- **listunit** (`int`, default 2) - Unit number for the list file.
- **model_ws** (`str` or `PathLike`, default `"."`) - Model workspace. Directory name to create model data sets. (default is the present working directory).

- **external_path** (*str* or *PathLike*, optional) - Location for external files.
- **verbose** (*bool*, default *False*) - Print additional information to the screen.

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
```

`get_ifrefm()`

`get_nrow_ncol_nlay_nper()`

classmethod `load(f: str, version='mf2005', exe_name: str | PathLike = 'mf2005', verbose=False, model_ws: str | PathLike = '.', load_only=None, forgive=False, check=True)`

Load an existing MODFLOW model.

Parameters

- **f** (*str*) - Path to MODFLOW name file to load.
- **version** (*str*, default "mf2005") - MODFLOW version. Choose one of: "mf2k", "mf2005" (default), or "mfnwt". Note that this can be modified on loading packages unique to different MODFLOW versions.
- **exe_name** (*str* or *PathLike*, default "mf2005") - MODFLOW executable name or path.
- **verbose** (*bool*, default *False*) - Show messages that can be useful for debugging.
- **model_ws** (*str* or *PathLike*, default ".") - Model workspace path. Default is the current directory.
- **load_only** (*list*, *str* or *None*) - List of case insensitive packages to load, e.g. ["bas6", "lpf"]. One package can also be specified, e.g. "rch". Default is *None*, which attempts to load all files. An empty list [] will not load any additional packages than is necessary. At a minimum, "dis" is always loaded.
- **forgive** (*bool*, optional) - Option to raise exceptions on package load failure, which can be useful for debugging. Default *False*.
- **check** (*boolean*, optional) - Check model input for common errors. Default *True*.

Return type

flopy.modflow.mf.Modflow

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow.load('model.nam')
```

`load_results(**kwargs)`

`property modelgrid`

`property modeltime`

`property ncol`

`property ncpl`

`property nlay`

`property nper`

`property nrow`

`property nrow_ncol_nlay_nper`

`set_ifrefm(value=True)`

`set_model_units(iunit0=None)`

Write the model name file.

`property solver_tols`

`write_name_file()`

Write the model name file.

`class ModflowGlobal(model, extension='glo')`

Bases: [Package](#)

ModflowGlobal Package class

`write_file()`

Every Package needs its own write_file function

`class ModflowList(model, extension='list', unitnumber=2)`

Bases: [Package](#)

ModflowList Package class

`write_file()`

Every Package needs its own write_file function

`flopy.modflow.mfaddoutsidefile` module

`class mfaddoutsidefile(model, name, extension, unitnumber)`

Bases: [Package](#)

Add a file for which you have a MODFLOW input file

`write_file()`

Every Package needs its own write_file function

flopy.modflow.mfag module

mfag module which contains the ModflowAg class.

Note that the user can access the ModflowAg class as `flopy.modflow.ModflowAg`.

Additional information for this MODFLOW package can be found at <https://www.sciencedirect.com/science/article/pii/S1364815219305080>>`_.

```
class ModflowAg(model, options=None, time_series=None, well_list=None, irrdiversion=None,
                irrwell=None, supwell=None, extension='ag', unitnumber=None,
                filenames=None, nper=0)
```

Bases: [Package](#)

The ModflowAg class is used to build read, write, and edit data from the MODFLOW-NWT AG package.

Parameters

- **model** (*flopy.modflow.Modflow object*) - model object
- **options** (*flopy.utils.OptionBlock object*) - option block object
- **time_series** (*np.recarray*) - numpy recarray for the time series block
- **well_list** (*np.recarray*) - recarray of the well_list block
- **irrdiversion** (*dict {per: np.recarray}*) - dictionary of the irrdiversion block
- **irrwell** (*dict {per: np.recarray}*) - dictionary of the irrwell block
- **supwell** (*dict {per: np.recarray}*) - dictionary of the supwell block
- **extension** (*str, optional*) - default is .ag
- **unitnumber** (*list, optional*) - fortran unit number for modflow, default 69
- **filenames** (*list, optional*) - file name for ModflowAwu package to write input
- **nper** (*int*) - number of stress periods in the model

Examples

load a ModflowAg file

```
>>> import flopy
>>> ml = flopy.modflow.Modflow('agtest')
>>> ag = flopy.modflow.ModflowAg.load('test.ag', ml, nper=2)
```

```
static get_default_dtype(maxells=0, block='well')
```

Function that gets a default dtype for a block

Parameters

- **maxells** (*int*) - maximum number of irrigation links

- **block** (*str*) - str which indicates data set valid options are "well" "tabfile_well" "timeseries" "irrdiversion" "irrwell" "supwell"

Returns
dtype

Return type
(*list*, *tuple*)

static `get_empty(numrecords, maxells=0, block='well')`

Creates an empty record array corresponding to the block data type it is associated with.

Parameters

- **numrecords** (*int*) - number of records to create recarray with
- **maxells** (*int*, *optional*) - maximum number of irrigation links
- **block** (*str*) - str which indicates data set valid options are "well" , "tabfile_well" , "timeseries" , "irrdiversion_modflow" , "irrdiversion_gsflow" , "irrwell_modflow" , "irrwell_gsflow" , "supwell"

Return type
`np.recarray`

classmethod `load(f, model, nper=0, ext_unit_dict=None)`

Method to load the AG package from file

Parameters

- **f** (*str*) - filename
- **model** (*gsflow.modflow.Modflow object*) - model to attach the ag package to
- **nper** (*int*) - number of stress periods in model
- **ext_unit_dict** (*dict*, *optional*) -

Return type
`ModflowAg object`

property `plottable`

property `segment_list`

Method to get a unique list of segments from irrdiversion

Return type
list

write_file(*check=False*)

Write method for ModflowAg

Parameters

- check** (*bool*) - not implemented

flopy.modflow.mfbas module

mfbas module. Contains the ModflowBas class. Note that the user can access the ModflowBas class as `flopy.modflow.ModflowBas`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowBas(model, ibound=1, strt=1.0, ifrefm=True, ixsec=False, ichflg=False,
                  stoper=None, hnoflo=-999.99, extension='bas', unitnumber=None,
                  filenames=None)
```

Bases: [Package](#)

MODFLOW Basic Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ibound** (*array of ints, optional*) - The ibound array (the default is 1).
- **strt** (*array of floats, optional*) - An array of starting heads (the default is 1.0).
- **ifrefm** (*bool, optional*) - Indication if data should be read using free format (the default is True).
- **ixsec** (*bool, optional*) - Indication of whether model is cross sectional or not (the default is False).
- **ichflg** (*bool, optional*) - Flag indicating that flows between constant head cells should be calculated (the default is False).
- **stoper** (*float*) - percent discrepancy that is compared to the budget percent discrepancy continue when the solver convergence criteria are not met. Execution will unless the budget percent discrepancy is greater than stoper (default is None). MODFLOW-2005 only
- **hnoflo** (*float*) - Head value assigned to inactive cells (default is -999.99).
- **extension** (*str, optional*) - File extension (default is 'bas').
- **unitnumber** (*int, optional*) - FORTRAN unit number for this package (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package name will be set to the string. Default is None.

heading

Text string written to top of package input file.

Type

`str`

options

Can be either or a combination of XSECTION, CHTOCH or FREE.

Type
list of str

ifrefm

Indicates whether or not packages will be written as free format.

Type
bool

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> bas = flopy.modflow.ModflowBas(m)
```

check(*f=None*, *verbose=True*, *level=1*, *checktype=None*)

Check package data for common errors.

Parameters

- **f** (*str* or *file handle*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If f is None, check method does not write results to a summary file. (default is None)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.

Return type
None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.bas6.check()
```

property ifrefm

classmethod **load**(*f*, *model*, *ext_unit_dict=None*, *check=True*, ***kwargs*)

Load an existing package.

Parameters

- **f** (*filename* or *file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.

- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.
- **check** (*boolean*) - Check package data for common errors. (default True)
- **kwargs** (*dictionary*) - Keyword arguments that are passed to load. Possible keyword arguments are *nlay*, *nrow*, and *ncol*. If not provided, then the model must contain a discretization package with correct values for these parameters.

Returns

bas - ModflowBas object (of type `flopy.modflow.ModflowBas`)

Return type

ModflowBas object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> bas = flopy.modflow.ModflowBas.load('test.bas', m, nlay=1, nrow=10,
>>>                                     ncol=10)
```

write_file(*check=True*)

Write the package file.

Parameters

check (*boolean*) - Check package data for common errors. (default True)

Return type

None

flopy.modflow.mfbcf module

```
class ModflowBcf(model, ipakcb=None, intercellt=0, laycon=3, trpy=1.0, hdry=-1e+30,
                 iwdflg=0, wetfct=0.1, iwetit=1, ihdwet=0, tran=1.0, hy=1.0, vcont=1.0,
                 sf1=1e-05, sf2=0.15, wetdry=-0.01, extension='bcf', unitnumber=None,
                 filenames=None, add_package=True)
```

Bases: [Package](#)

MODFLOW Block Centered Flow Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.Modflow`) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If *ipakcb* is non-zero cell-by-cell budget data will be saved. (default is 53)

- **intercellt** (*int*) - Intercell transmissivities, harmonic mean (0), arithmetic mean (1), logarithmic mean (2), combination (3). (default is 0)
- **laycon** (*int*) - Layer type, confined (0), unconfined (1), constant T, variable S (2), variable T, variable S (default is 3)
- **trpy** (*float* or array of floats (*nlay*)) - horizontal anisotropy ratio (default is 1.0)
- **hdry** (*float*) - head assigned when cell is dry - used as indicator (default is -1E+30)
- **iwdflg** (*int*) - flag to indicate if wetting is inactive (0) or not (non zero) (default is 0)
- **wetfct** (*float*) - factor used when cell is converted from dry to wet (default is 0.1)
- **iwetit** (*int*) - iteration interval in wetting/drying algorithm (default is 1)
- **ihdwet** (*int*) - flag to indicate how initial head is computed for cells that become wet (default is 0)
- **tran** (*float* or array of floats (*nlay*, *nrow*, *ncol*), optional) - transmissivity (only read if laycon is 0 or 2) (default is 1.0)
- **hy** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - hydraulic conductivity (only read if laycon is 1 or 3) (default is 1.0)
- **vcont** (*float* or array of floats (*nlay-1*, *nrow*, *ncol*)) - vertical leakance between layers (default is 1.0)
- **sf1** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - specific storage (confined) or storage coefficient (unconfined), read when there is at least one transient stress period. (default is 1e-5)
- **sf2** (*float* or array of floats (*nrow*, *ncol*)) - specific yield, only read when laycon is 2 or 3 and there is at least one transient stress period (default is 0.15)
- **wetdry** (*float*) - a combination of the wetting threshold and a flag to indicate which neighboring cells can cause a cell to become wet (default is -0.01)
- **extension** (*string*) - Filename extension (default is 'bcf')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str* or *list* of *str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output name will be created using the model name and .cbc extension (for example, modflowtest.cbc), if ipakcbc is a number greater than zero. If a single string is passed the package will be set to the string and cbc output name will be created using the model name and .cbc extension, if ipakcbc is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

- **add_package** (*bool*) - Flag to add the initialised package object to the parent model object. Default is True.

Notes

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow()
>>> bcf = flopy.modflow.ModflowBcf(m1)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **nper** (*int*) - The number of stress periods. If nper is None, then nper will be obtained from the model object. (default is None).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function *flopy.utils.mfreadnam.parse_namefile*.

Returns

wel - ModflowBcf object.

Return type

ModflowBcf object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> wel = flopy.modflow.ModflowBcf.load('test.bcf', m)
```

write_file(*f=None*)

Write the package file.

Return type

None

flopy.modflow.mfbct module

```
class ModflowBct(model, itrns=1, ibctcb=0, mcomp=1, ic_ibound_flg=1, itvd=1, iadsorb=0,
                 ict=0, cinact=-999.0, ciclose=1e-06, idisp=1, ixdisp=0, diffnc=0.0,
                 izod=0, ifod=0, icbund=1, porosity=0.1, bulkd=1.0, arad=0.0, dlh=0.0,
                 dlv=0.0, dth=0.0, dtv=0.0, scon=0.0, extension='bct', unitnumber=None)
```

Bases: [Package](#)

Block centered transport package class for MODFLOW-USG

write_file()

Write the package file.

Return type

None

flopy.modflow.mfchd module

mfchd module. Contains the ModflowChd class. Note that the user can access the ModflowChd class as *flopy.modflow.ModflowChd*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowChd(model, stress_period_data=None, dtype=None, options=None,
                 extension='chd', unitnumber=None, filenames=None, **kwargs)
```

Bases: [Package](#)

MODFLOW Constant Head Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.modflow.mf.Modflow](#)) to which this package will be added.
- **stress_period_data** (*list of boundaries, recarrays, or dictionary of*) - boundaries.

Each chd cell is defined through definition of layer (int), row (int), column (int), shead (float), ehead (float) shead is the head at the start of the stress period, and ehead is the head at the end of the stress period. The simplest form is a dictionary with a lists of boundaries for each stress period, where each list of boundaries itself is a list of boundaries. Indices of the dictionary are the numbers of the stress period. This gives the form of:

```
stress_period_data =
{0: [
    [lay, row, col, shead, ehead],
    [lay, row, col, shead, ehead],
    [lay, row, col, shead, ehead]
],
1: [
    [lay, row, col, shead, ehead],
    [lay, row, col, shead, ehead],
    [lay, row, col, shead, ehead]
```

(continues on next page)

(continued from previous page)

```
    ], ...
    kper:
    [
        [lay, row, col, shead, ehead],
        [lay, row, col, shead, ehead],
        [lay, row, col, shead, ehead]
    ]
}
```

Note that if the number of lists is smaller than the number of stress periods, then the last list of chds will apply until the end of the simulation. Full details of all options to specify stress_period_data can be found in the flopy3 boundaries Notebook in the basic subdirectory of the examples directory.

- **extension** (*string*) - Filename extension (default is 'chd')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

mxactc

Maximum number of chds for all stress periods. This is calculated automatically by FloPy based on the information in stress_period_data.

Type

int

Notes

Parameters are supported in Flopy only when reading in existing models. Parameter values are converted to native values in Flopy and the connection to "parameters" is thus nonexistent.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lrcd = {0:[[2, 3, 4, 10., 10.1]]}  #this chd will be applied to all
>>>                                     #stress periods
>>> chd = flopy.modflow.ModflowChd(m, stress_period_data=lrcd)
```

add_record(kper, index, values)

static get_default_dtype(structured=True)

static get_empty(ncells=0, aux_names=None, structured=True)

classmethod load(f, model, nper=None, ext_unit_dict=None, check=True)

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **nper** (*int*) - The number of stress periods. If `nper` is `None`, then `nper` will be obtained from the model object. (default is `None`).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then `f` should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

chd - ModflowChd object.

Return type

ModflowChd object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> wel = flopy.modflow.ModflowChd.load('test.chd', m)
```

write_file()

Write the package file.

Return type

`None`

flopy.modflow.mfde4 module

mfde4 module. Contains the ModflowDe4 class. Note that the user can access the ModflowDe4 class as `flopy.modflow.ModflowDe4`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowDe4(model, itmx=50, mxup=0, mxlow=0, mxbw=0, ifreq=3, mutd4=0, accl=1.0,
                  hclose=1e-05, iprd4=1, extension='de4', unitnumber=None, filenames=None)
```

Bases: `Package`

MODFLOW DE4 - Direct Solver Package

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **itmx** (*int*) - Maximum number of iterations for each time step. Specify `ITMAX = 1` if iteration is not desired. Ideally iteration would not be required for direct solution. However, it is necessary to iterate if the flow equation is nonlinear or if computer precision limitations result in inaccurate calculations as indicated by a large water budget error (default is 50).

- **mxup** (*int*) - Maximum number of equations in the upper part of the equations to be solved. This value impacts the amount of memory used by the DE4 Package. If specified as 0, the program will calculate MXUP as half the number of cells in the model, which is an upper limit (default is 0).
- **mxlow** (*int*) - Maximum number of equations in the lower part of equations to be solved. This value impacts the amount of memory used by the DE4 Package. If specified as 0, the program will calculate MXLOW as half the number of cells in the model, which is an upper limit (default is 0).
- **mxbw** (*int*) - Maximum band width plus 1 of the lower part of the head coefficients matrix. This value impacts the amount of memory used by the DE4 Package. If specified as 0, the program will calculate MXBW as the product of the two smallest grid dimensions plus 1, which is an upper limit (default is 0).
- **ifreq** (*int*) - Flag indicating the frequency at which coefficients in head matrix change. IFREQ = 1 indicates that the flow equations are linear and that coefficients of simulated head for all stress terms are constant for all stress periods. IFREQ = 2 indicates that the flow equations are linear, but coefficients of simulated head for some stress terms may change at the start of each stress period. IFREQ = 3 indicates that a nonlinear flow equation is being solved, which means that some terms in the head coefficients matrix depend on simulated head (default is 3).
- **mutd4** (*int*) - Flag that indicates the quantity of information that is printed when convergence information is printed for a time step. MUTD4 = 0 indicates that the number of iterations in the time step and the maximum head change each iteration are printed. MUTD4 = 1 indicates that only the number of iterations in the time step is printed. MUTD4 = 2 indicates no information is printed (default is 0).
- **accl** (*int*) - Multiplier for the computed head change for each iteration. Normally this value is 1. A value greater than 1 may be useful for improving the rate of convergence when using external iteration to solve nonlinear problems (default is 1).
- **hclose** (*float*) - Head change closure criterion. If iterating (ITMX > 1), iteration stops when the absolute value of head change at every node is less than or equal to HCLOSE. HCLOSE is not used if not iterating, but a value must always be specified (default is 1e-5).
- **iprd4** (*int*) - Time step interval for printing out convergence information when iterating (ITMX > 1). If IPRD4 is 2, convergence information is printed every other time step. A value must always be specified even if not iterating (default is 1).
- **extension** (*string*) - Filename extension (default is 'de4')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created

using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> de4 = flopy.modflow.ModflowDe4(m)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

de4

Return type

ModflowDe4 object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> de4 = flopy.modflow.ModflowDe4.load('test.de4', m)
```

write_file()

Write the package file.

Return type

None

flopy.modflow.mfdis module

mfdis module. Contains the ModflowDis class. Note that the user can access the ModflowDis class as `flopy.modflow.ModflowDis`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowDis(model, nlay=1, nrow=2, ncol=2, nper=1, delr=1.0, delc=1.0, laycbd=0,
                 top=1, botm=0, perlen=1, nstp=1, tsmult=1, steady=True, itmuni=4,
                 lenuni=2, extension='dis', unitnumber=None, filenames=None, xul=None,
                 yul=None, rotation=None, proj4_str=None, crs=None, prjfile=None,
                 start_datetime=None)
```

Bases: [Package](#)

MODFLOW Discretization Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.Modflow`) to which this package will be added.
- **nlay** (*int*) - Number of model layers (the default is 1).
- **nrow** (*int*) - Number of model rows (the default is 2).
- **ncol** (*int*) - Number of model columns (the default is 2).
- **nper** (*int*) - Number of model stress periods (the default is 1).
- **delr** (*float or array of floats (ncol), optional*) - An array of spacings along a row (the default is 1.0).
- **delc** (*float or array of floats (nrow), optional*) - An array of spacings along a column (the default is 0.0).
- **laycbd** (*int or array of ints (nlay), optional*) - An array of flags indicating whether or not a layer has a Quasi-3D confining bed below it. 0 indicates no confining bed, and not zero indicates a confining bed. LAYCBD for the bottom layer must be 0. (the default is 0)
- **top** (*float or array of floats (nrow, ncol), optional*) - An array of the top elevation of layer 1. For the common situation in which the top layer represents a water-table aquifer, it may be reasonable to set Top equal to land-surface elevation (the default is 1.0)
- **botm** (*float or array of floats (nlay, nrow, ncol), optional*) - An array of the bottom elevation for each model cell (the default is 0.)
- **perlen** (*float or array of floats (nper)*) - An array of the stress period lengths.
- **nstp** (*int or array of ints (nper)*) - Number of time steps in each stress period (default is 1).
- **tsmult** (*float or array of floats (nper)*) - Time step multiplier (default is 1.0).
- **steady** (*bool or array of bool (nper)*) - true or False indicating whether or not stress period is steady state (default is True).

- **itmuni** (*int*) - Time units, default is days (4)
- **lenuni** (*int*) - Length units, default is meters (2)
- **extension** (*string*) - Filename extension (default is 'dis')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.
- **xul** (*float*) - x coordinate of upper left corner of the grid, default is None, which means xul will be set to zero.
- **yul** (*float*) - y coordinate of upper-left corner of the grid, default is None, which means yul will be calculated as the sum of the delc array. This default, combined with the xul and rotation defaults will place the lower-left corner of the grid at (0, 0).
- **rotation** (*float*) - counter-clockwise rotation (in degrees) of the grid about the lower-left corner. default is 0.0
- **crs** (*pyproj.CRS*, optional if *prjfile* is specified) - Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg "EPSG:26916") or a WKT string.
- **prjfile** (*str or pathlike*, optional if *crs* is specified) - ESRI-style projection file with well-known text defining the CRS for the model grid (must be projected; geographic CRS are not supported).
- **start_datetime** (*str*) - starting datetime of the simulation. default is '1/1/1970'

heading

Text string written to top of package input file.

Type

`str`

Notes**Examples**

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> dis = flopy.modflow.ModflowDis(m)
```

check(*f=None, verbose=True, level=1, checktype=None*)

Check dis package data for zero and negative thicknesses.

Parameters

- **f** (*str or file handle*) - String defining file name or file handle for summary file of check method output. If a sting is

passed a file handle is created. If `f` is `None`, check method does not write results to a summary file. (default is `None`)

- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If `level=0`, summary checks are performed. If `level=1`, full checks are performed.

Return type

`None`

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.dis.check()
```

checklayerthickness()

Check layer thickness.

get_cell_volumes()

Get an array of cell volumes.

Returns

`vol`

Return type

array of floats (`nlay`, `nrow`, `ncol`)

get_final_totim()

Get the totim at the end of the simulation

Returns

`totim` - maximum simulation totim

Return type

float

get_kstp_kper_toffset(*t=0.0*, *use_cached_totim=False*)

Get the stress period, time step, and time offset from passed time.

Parameters

- **t** (*float*) - totim to return the stress period, time step, and toffset for based on time discretization data. Default is `0`.
- **use_cached_totim** (*bool*) - optional flag to use a cached calculation of totim, vs. dynamically calculating totim. Setting to `True` significantly speeds up looped operations that call this function (default is `False`).

Returns

- **kstp** (*int*) - time step in stress period corresponding to passed totim
- **kper** (*int*) - stress period corresponding to passed totim

- **toffset** (*float*) - time offset of passed totim from the beginning of kper

get_layer(*i*, *j*, *elev*)

Return the layer for an elevation at an *i*, *j* location.

Parameters

- **i** (*row index (zero-based)*) -
- **j** (*column index*) -
- **elev** (*elevation (in same units as model)*) -

Returns

k

Return type

zero-based layer index

get_lrc(*nodes*)

Get zero-based layer, row, column from a list of zero-based MODFLOW node numbers.

Returns

v - and column (*j*) for each node in the input list

Return type

list of tuples containing the layer (*k*), row (*i*),

get_node(*lrc_list*)

Get zero-based node number from a list of zero-based MODFLOW layer, row, column tuples.

Returns

v - and column (*j*) tuple in the input list

Return type

list of MODFLOW nodes for each layer (*k*), row (*i*),

get_node_coordinates()

Get *y*, *x*, and *z* cell centroids in local model coordinates.

Returns

- **y** (*list of cell y-centroids*)
- **x** (*list of cell x-centroids*)
- **z** (*array of floats (nlay, nrow, ncol)*)

get_rc_from_node_coordinates(*x*, *y*, *local=True*)

Get the row and column of a point or sequence of points in model coordinates.

Parameters

- **x** (*float or sequence of floats*) - *x* coordinate(s) of points to find in model grid
- **y** (*float or sequence floats*) - *y* coordinate(s) of points to find in model grid

- **local** (*bool*) - x and y coordinates are in model local coordinates. If false, then x and y are in world coordinates. (default is True)

Returns

- **r** (*row or sequence of rows (zero-based)*)
- **c** (*column or sequence of columns (zero-based)*)

get_totim(*use_cached=False*)

Get the totim at the end of each time step

Parameters

use_cached (*bool*) - method to use cached totim values instead of calculating totim dynamically

Returns

totim - numpy array with simulation totim at the end of each time step

Return type

numpy array

get_totim_from_kper_toffset(*kper=0, toffset=0.0, use_cached_totim=False*)

Get totim from a passed kper and time offset from the beginning of a stress period

Parameters

- **kper** (*int*) - stress period. Default is 0
- **toffset** (*float*) - time offset relative to the beginning of kper
- **use_cached_totim** (*bool*) - optional flag to use a cached calculation of totim, vs. dynamically calculating totim. Setting to True significantly speeds up looped operations that call this function (default is False).

Returns

t - totim to return the stress period, time step, and toffset for based on time discretization data. Default is 0.

Return type

float

getbotm(*k=None*)

Get the bottom array.

Returns

- **botm** (*array of floats (nlay, nrow, ncol), or*)
- **botm** (*array of floats (nrow, ncol) if k is not none*)

gettop()

Get the top array.

Returns

top

Return type

array of floats (nrow, ncol)

classmethod `load(f, model, ext_unit_dict=None, check=True)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsefilenamefile`.
- **check** (*bool*) - Check package data for common errors. (default True)

Returns

dis - ModflowDis object.

Return type

ModflowDis object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> dis = flopy.modflow.ModflowDis.load('test.dis', m)
```

write_file(check=True)

Write the package file.

Parameters

- **check** (*bool*) - Check package data for common errors. (default True)

Return type

None

property zcentroids

get_layer(dis, i, j, elev)

Return the layers for elevations at i, j locations.

Parameters

- **dis** (*flopy.modflow.ModflowDis object*) -
- **i** (*scaler or sequence*) - row index (zero-based)
- **j** (*scaler or sequence*) - column index
- **elev** (*scaler or sequence*) - elevation (in same units as model)

Returns

k - zero-based layer index

Return type

np.ndarray (1-D) or scalar

flopy.modflow.mfdrn module

mfdrn module. Contains the ModflowDrn class. Note that the user can access the ModflowDrn class as `flopy.modflow.ModflowDrn`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowDrn(model, ipakcb=None, stress_period_data=None, dtype=None, extension='drn',
                  unitnumber=None, options=None, filenames=None, **kwargs)
```

Bases: [Package](#)

MODFLOW Drain Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.modflow.mf.Modflow](#)) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is None).
- **stress_period_data** (*list of boundaries, recarrays, or dictionary of*) - boundaries. Each drain cell is defined through definition of layer(int), row(int), column(int), elevation(float), conductance(float). The simplest form is a dictionary with a lists of boundaries for each stress period, where each list of boundaries itself is a list of boundaries. Indices of the dictionary are the numbers of the stress period. This gives the form of:

```
stress_period_data =
{0: [
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
  ],
 1: [
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
  ], ...
kper:
  [
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
  ]
}
```

Note that if no values are specified for a certain stress period, then the list of boundaries for the previous stress period for which values were defined is used. Full details of all options to specify stress_period_data can be found in the `flopy3boundaries` Notebook in the basic subdirectory of the examples directory.

- **dtype** (*dtype definition*) - if data type is different from default

- **options** (*list of strings*) - Package options. (default is None).
- **extension** (*string*) - Filename extension (default is 'drn')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output name will be created using the model name and .cbc extension (for example, modflowtest.cbc), if ipakcbc is a number greater than zero. If a single string is passed the package will be set to the string and cbc output names will be created using the model name and .cbc extension, if ipakcbc is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

Parameters are not supported in FloPy. If "RETURNFLOW" is passed in options, the drain return package (DRT) is activated, which expects a different (longer) dtype for stress_period_data

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow()
>>> lrcec = {0:[2, 3, 4, 10., 100.]} #this drain will be applied to all
>>>                                     #stress periods
>>> drn = flopy.modflow.ModflowDrn(m1, stress_period_data=lrcec)
```

add_record(*kper, index, values*)

static get_default_dtype(*structured=True, is_drt=False*)

static get_empty(*ncells=0, aux_names=None, structured=True, is_drt=False*)

classmethod load(*f, model, nper=None, ext_unit_dict=None, check=True*)

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.
- **check** (*boolean*) - Check package data for common errors. (default True)

Returns

drn - ModflowDrn object.

Return type

ModflowDrn object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> drn = flopy.modflow.ModflowDrn.load('test.drn', m)
```

write_file(*check=True*)

Write the package file.

Parameters

check (*boolean*) - Check package data for common errors. (default True)

Return type

None

flopy.modflow.mfdrt module

mfdrt module. Contains the ModflowDrt class. Note that the user can access the ModflowDrt class as *flopy.modflow.ModflowDrt*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowDrt(model, ipakcb=None, stress_period_data=None, dtype=None, extension='drt',
                  unitnumber=None, options=None, filenames=None, **kwargs)
```

Bases: *Package*

MODFLOW Drain Return Package Class.

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is None).
- **stress_period_data** (*list of boundaries, recarrays, or dictionary of*) - boundaries. Each drain return cell is defined through definition of layer(int), row(int), column(int), elevation(float), conductance(float), layerR (int) , rowR (int), colR (int) and rfprop (float). The simplest form is a dictionary with a lists of boundaries for each stress period, where each list of boundaries itself is a list of boundaries. Indices of the dictionary are the numbers of the stress period. This gives the form of:

```
stress_period_data =
{0: [
    [lay, row, col, stage, cond, layerr, rowr, colr, rfprop],
```

(continues on next page)

(continued from previous page)

```

[lay, row, col, stage, cond, layerr, rowr, colr, rfprop],
[lay, row, col, stage, cond, layerr, rowr, colr, rfprop],
],
1: [
[lay, row, col, stage, cond, layerr, rowr, colr, rfprop],
[lay, row, col, stage, cond, layerr, rowr, colr, rfprop],
[lay, row, col, stage, cond, layerr, rowr, colr, rfprop],
], ...
kper:
[
[lay, row, col, stage, cond, layerr, rowr, colr, rfprop],
[lay, row, col, stage, cond, layerr, rowr, colr, rfprop],
[lay, row, col, stage, cond, layerr, rowr, colr, rfprop],
]
}

```

Note that if no values are specified for a certain stress period, then the list of boundaries for the previous stress period for which values were defined is used. Full details of all options to specify `stress_period_data` can be found in the `flopy3boundaries` Notebook in the basic subdirectory of the examples directory.

- **dtype** (*dtype definition*) - if data type is different from default
- **options** (*list of strings*) - Package options. (default is None).
- **extension** (*string*) - Filename extension (default is 'drt')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If `filenames=None` the package name will be created using the model name and package extension and the cbc output name will be created using the model name and `.cbc` extension (for example, `modflowtest.cbc`), if `ipakcbc` is a number greater than zero. If a single string is passed the package will be set to the string and cbc output names will be created using the model name and `.cbc` extension, if `ipakcbc` is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow()
>>> lrcec = {0:[2, 3, 4, 10., 100., 1, 1, 1, 1.0]} #this drain will be applied to
↳ all
>>> #stress periods
>>> drt = flopy.modflow.ModflowDrt(m1, stress_period_data=lrcec)
```

add_record(kper, index, values)

static get_default_dtype(structured=True)

static get_empty(ncells=0, aux_names=None, structured=True, is_drt=False)

classmethod load(f, model, nper=None, ext_unit_dict=None, check=True)

Load an existing package.

Parameters

- **f** (filename or file handle) - File to load.
- **model** (model object) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ext_unit_dict** (dictionary, optional) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.
- **check** (boolean) - Check package data for common errors. (default True)

Returns

drn - ModflowDrt object.

Return type

ModflowDrt object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> drn = flopy.modflow.ModflowDrt.load('test.drt', m)
```

write_file(check=True)

Write the package file.

Parameters

- **check** (boolean) - Check package data for common errors. (default True)

Return type
None

flopy.modflow.mfevt module

mfgghb module. Contains the ModflowEvt class. Note that the user can access the ModflowEvt class as *flopy.modflow.ModflowEvt*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowEvt(model, nevtop=3, ipakcb=None, surf=0.0, evtr=0.001, exdp=1.0, ievt=1,
                 extension='evt', unitnumber=None, filenames=None, external=True)
```

Bases: [Package](#)

MODFLOW Evapotranspiration Package Class.

Parameters

- **model** (*model object*) - The model object (of type flopy.modflow.mf.ModflowEvt) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0).
- **nevtop** (*int*) - is the recharge option code. 1: ET is calculated only for cells in the top grid layer 2: ET to layer defined in ievt 3: ET to highest active cell (default is 3).
- **surf** (*float or filename or ndarray or dict keyed on kper (zero-based)*) - is the ET surface elevation. (default is 0.0, which is used for all stress periods).
- **evtr** (*float or filename or ndarray or dict keyed on kper (zero-based)*) - is the maximum ET flux (default is 1e-3, which is used for all stress periods).
- **exdp** (*float or filename or ndarray or dict keyed on kper (zero-based)*) - is the ET extinction depth (default is 1.0, which is used for all stress periods).
- **ievt** (*int or filename or ndarray or dict keyed on kper (zero-based)*) - is the layer indicator variable (default is 1, which is used for all stress periods).
- **extension** (*string*) - Filename extension (default is 'evt')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output name will be created using the model name and .cbc extension (for example, modflowtest.cbc), if ipakcbc is a number greater than zero. If a single string is passed the package will be set to the string and cbc output names will be created using the model name and .cbc extension, if ipakcbc is a number greater than zero. To define the names for all package files (input and

output) the length of the list of strings should be 2. Default is None.

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> evt = flopy.modflow.ModflowEvt(m, nevtop=3, evtr=1.2e-4)
```

classmethod `load(f, model, nper=None, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **nper** (*int*) - The number of stress periods. If nper is None, then nper will be obtained from the model object. (default is None).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

Returns

evt - ModflowEvt object.

Return type

ModflowEvt object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> evt = flopy.modflow.mfevt.load('test.evt', m)
```

write_file(f=None)

Write the package file.

Return type

None

flopy.modflow.mffhb module

mffhb module. Contains the ModflowFhb class. Note that the user can access the ModflowFhb class as `flopy.modflow.ModflowFhb`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowFhb(model, nbdtim=1, nflw=0, nhed=0, ifhbss=0, ipakcb=None, nfhb1=0,
                 nfhb2=0, ifhbpt=0, bdtimcnstm=1.0, bdtim=[0.0], cnstm5=1.0,
                 ds5=None, cnstm7=1.0, ds7=None, extension='fhb', unitnumber=None,
                 filenames=None)
```

Bases: [Package](#)

MODFLOW Flow and Head Boundary Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.ModflowFhb`) to which this package will be added.
- **nbdtim** (*int*) - The number of times at which flow and head will be specified for all selected cells. (default is 1)
- **nflw** (*int*) - Number of cells at which flows will be specified. (default is 0)
- **nhed** (*int*) - Number of cells at which heads will be specified. (default is 0)
- **ifhbss** (*int*) - FHB steady-state option flag. If the simulation includes any transient-state stress periods, the flag is read but not used; in this case, specified-flow, specified-head, and auxiliary-variable values will be interpolated for steady-state stress periods in the same way that values are interpolated for transient stress periods. If the simulation includes only steady-state stress periods, the flag controls how flow, head, and auxiliary-variable values will be computed for each steady-state solution. (default is 0)
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is None).
- **nfhb1** (*int*) - Number of auxiliary variables whose values will be computed for each time step for each specified-flow cell. Auxiliary variables are currently not supported. (default is 0)
- **nfhb2** (*int*) - Number of auxiliary variables whose values will be computed for each time step for each specified-head cell. Auxiliary variables are currently not supported. (default is 0)
- **ifhbpt** (*int*) - Flag for printing values of data list. Applies to datasets 4b, 5b, 6b, 7b, and 8b. If ifhbpt > 0, datasets read at the beginning of the simulation will be printed. Otherwise, the datasets will not be printed. (default is 0).
- **bdtimcnstm** (*float*) - A constant multiplier for data list bdtim. (default is 1.0)

- **bdtm** (*float* or *list of floats*) - Simulation time at which values of specified flow and (or) values of specified head will be read. `nbdm` values are required. (default is 0.0)
- **cstm5** (*float*) - A constant multiplier for data list `flwrat`. (default is 1.0)
- **ds5** (*list* or *numpy array* or *recarray*) - Each FHB `flwrat` cell (dataset 5) is defined through definition of `layer(int)`, `row(int)`, `column(int)`, `iaux(int)`, `flwrat[nbdm](float)`. There should be `nflw` entries. (default is None) The simplest form is a list of lists with the FHB flow boundaries. This gives the form of:

```
ds5 =  
[  
    [lay, row, col,iaux, flwrat1, flwrat2, ..., flwrat(nbdm)],  
    [lay, row, col,iaux, flwrat1, flwrat2, ..., flwrat(nbdm)],  
    [lay, row, col,iaux, flwrat1, flwrat2, ..., flwrat(nbdm)],  
    [lay, row, col,iaux, flwrat1, flwrat2, ..., flwrat(nbdm)]  
]
```

- **cstm7** (*float*) - A constant multiplier for data list `sbhedt`. (default is 1.0)
- **ds7** (*list* or *numpy array* or *recarray*) - Each FHB `sbhed` cell (dataset 7) is defined through definition of `layer(int)`, `row(int)`, `column(int)`, `iaux(int)`, `sbhed[nbdm](float)`. There should be `nhd` entries. (default is None) The simplest form is a list of lists with the FHB flow boundaries. This gives the form of:

```
ds7 =  
[  
    [lay, row, col,iaux, sbhed1, sbhed2, ..., sbhed(nbdm)],  
    [lay, row, col,iaux, sbhed1, sbhed2, ..., sbhed(nbdm)],  
    [lay, row, col,iaux, sbhed1, sbhed2, ..., sbhed(nbdm)],  
    [lay, row, col,iaux, sbhed1, sbhed2, ..., sbhed(nbdm)]  
]
```

- **extension** (*string*) - Filename extension (default is 'fhb')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str* or *list of str*) - Filenames to use for the package and the output files. If `filenames=None` the package name will be created using the model name and package extension and the cbc output name will be created using the model name and `.cbc` extension (for example, `modflowtest.cbc`), if `ipakcbc` is a number greater than zero. If a single string is passed the package will be set to the string and cbc output names will be created using the model name and `.cbc` extension, if `ipakcbc` is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> fhb = flopy.modflow.ModflowFhb(m)
```

```
static get_default_dtype(nbdtim=1, structured=True, head=False)
```

```
static get_empty(ncells=0, nbdtim=1, structured=True, head=False)
```

```
classmethod load(f, model, nper=None, ext_unit_dict=None)
```

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **nper** (*int*) - The number of stress periods. If nper is None, then nper will be obtained from the model object. (default is None).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

Returns

fhb - ModflowFhb object.

Return type

ModflowFhb object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> fhb = flopy.modflow.ModflowFhb.load('test.fhb', m)
```

```
write_file()
```

Write the package file.

Return type

None

flopy.modflow.mfflwb module

```
class ModflowFlwb(model, nqfb=0, nqcfb=0, nqtfb=0, iufbobsv=0, tomultfb=1.0,
                  nqobfb=None, nqclfb=None, obsnam=None, irefsp=None, toffset=None,
                  flwobs=None, layer=None, row=None, column=None, factor=None,
                  flowtype=None, extension=None, no_print=False, options=None,
                  filenames=None, unitnumber=None)
```

Bases: [Package](#)

Head-dependent flow boundary Observation package class. Minimal working example that will be refactored in a future version.

Parameters

- **nqfb** (*int*) - Number of cell groups for the head-dependent flow boundary observations
- **nqcfb** (*int*) - Greater than or equal to the total number of cells in all cell groups
- **nqtfb** (*int*) - Total number of head-dependent flow boundary observations for all cell groups
- **iufbobsv** (*int*) - unit number where output is saved
- **tomultfb** (*float*) - Time-offset multiplier for head-dependent flow boundary observations. The product of tomultfb and toffset must produce a time value in units consistent with other model input. tomultfb can be dimensionless or can be used to convert the units of toffset to the time unit used in the simulation.
- **nqobfb** (*int list of length nqfb*) - The number of times at which flows are observed for the group of cells
- **nqclfb** (*int list of length nqfb*) - Is a flag, and the absolute value of nqclfb is the number of cells in the group. If nqclfb is less than zero, factor = 1.0 for all cells in the group.
- **obsnam** (*string list of length nqtfb*) - Observation name
- **irefsp** (*int of length nqtfb*) - The zero-based stress period to which the observation time is referenced. The reference point is the beginning of the specified stress period.
- **toffset** (*float list of length nqtfb*) - Is the time from the beginning of the stress period irefsp to the time of the observation. toffset must be in units such that the product of toffset and tomultfb are consistent with other model input. For steady state observations, specify irefsp as the steady state stress period and toffset less than or equal to perlen of the stress period. If perlen is zero, set toffset to zero. If the observation falls within a time step, linearly interpolation is used between values at the beginning and end of the time step.
- **flwobs** (*float list of length nqtfb*) - Observed flow value from the head-dependent flow boundary into the aquifer (+) or the flow from the aquifer into the boundary (-)
- **layer** (*int list of length(nqfb, nqclfb)*) - The zero-based layer index for the cell included in the cell group.

- **row** (*int list of length(nqfb, nqclfb)*) - The zero-based row index for the cell included in the cell group.
- **column** (*int list of length(nqfb, nqclfb)*) - The zero-based column index of the cell included in the cell group.
- **factor** (*float list of length(nqfb, nqclfb)*) - Is the portion of the simulated gain or loss in the cell that is included in the total gain or loss for this cell group (fn of eq. 5).
- **flowtype** (*string*) - String that corresponds to the head-dependent flow boundary condition type (CHD, GHB, DRN, RIV)
- **extension** (*list of string*) - Filename extension. If extension is None, extension is set to ['chob', 'obc', 'gbob', 'obg', 'drob', 'obd', 'rvob', 'obr'] (default is None).
- **no_print** (*boolean*) - When True or 1, a list of flow observations will not be written to the Listing File (default is False)
- **options** (*list of strings*) - Package options (default is None).
- **unitnumber** (*list of int*) - File unit number. If unitnumber is None, unitnumber is set to [40, 140, 41, 141, 42, 142, 43, 143] (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the flwob output name will be created using the model name and .out extension (for example, modflowtest.out), if iufbobsv is a number greater than zero. If a single string is passed the package will be set to the string and flwob output name will be created using the model name and .out extension, if iufbobsv is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

This represents a minimal working example that will be refactored in a future version.

ftype()

classmethod load(*f, model, ext_unit_dict=None, check=True*)

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

- **check** (*boolean*) - Check package data for common errors. (default True)

Returns

flwob - ModflowFlwob package object.

Return type

ModflowFlwob package object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> hobs = flopy.modflow.ModflowFlwob.load('test.drob', m)
```

write_file()

Write the package file

Return type

None

flopy.modflow.mfgage module

mfgage module. Contains the ModflowGage class. Note that the user can access the ModflowGage class as *flopy.modflow.ModflowGage*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowGage(model, numgage=0, gage_data=None, files=None, extension='gage',
                  unitnumber=None, filenames=None, **kwargs)
```

Bases: *Package*

MODFLOW Gage Package Class.

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **numgage** (*int*) - The total number of gages included in the gage file (default is 0).
- **gage_data** (*list or numpy array*) - data for dataset 2a and 2b in the gage package. If a list is provided then the list includes 2 to 3 entries (LAKE UNIT [OUTTYPE]) for each LAK Package entry and 4 entries (GAGESEG GAGERCH UNIT OUTTYPE) for each SFR Package entry. If a numpy array it passed each gage location must have 4 entries, where LAK Package gages can have any value for the second column. The numpy array can be created using the *get_empty()* method available in ModflowGage. Default is None
- **files** (*list of strings*) - Names of gage output files. A file name must be provided for each gage. If files are not provided and filenames=None then a gage name will be created using the model name and the gage number (for example, modflowtest.gage1.go). Default is None.

- **extension** (*string*) - Filename extension (default is 'gage')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and gage output names will be created using the model name and the gage number (for example, modflowtest.gage1.go). If a single string is passed the package will be set to the string and gage output names will be created using the model name and the gage number. To define the names for all gage files (input and output) the length of the list of strings should be numgage + 1. Default is None.

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> gages = [[-1, -26, 1], [-2, -27, 1]]
>>> files = ['gage1.go', 'gage2.go']
>>> gage = flopy.modflow.ModflowGage(m, numgage=2,
>>>                                     gage_data=gages, files=files)
```

static `get_default_dtype()`

static `get_empty(ncells=0, aux_names=None, structured=True)`

classmethod `load(f, model, nper=None, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **nper** (*int*) - The number of stress periods. If nper is None, then nper will be obtained from the model object. (default is None).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function `flopy.utils.mfreadnam.parse_namefile`.

Returns

str - ModflowStr object.

Return type

ModflowStr object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> gage = flopy.modflow.ModflowGage.load('test.gage', m)
```

write_file()

Write the package file.

Return type

None

flopy.modflow.mfghb module

mfghb module. Contains the ModflowGhb class. Note that the user can access the ModflowGhb class as *flopy.modflow.ModflowGhb*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowGhb(model, ipakcb=None, stress_period_data=None, dtype=None, no_print=False,
                  options=None, extension='ghb', unitnumber=None, filenames=None)
```

Bases: *Package*

MODFLOW General-Head Boundary Package Class.

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0).
- **stress_period_data** (*list of boundaries, recarray of boundaries or,*) - dictionary of boundaries.

Each ghb cell is defined through definition of layer(int), row(int), column(int), stage(float), conductance(float) The simplest form is a dictionary with a lists of boundaries for each stress period, where each list of boundaries itself is a list of boundaries. Indices of the dictionary are the numbers of the stress period. This gives the form of:

```
stress_period_data =
{0: [
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
  ],
 1: [
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
  ], ...
kper:
```

(continues on next page)

(continued from previous page)

```
[
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
    [lay, row, col, stage, cond],
]
```

Note that if no values are specified for a certain stress period, then the list of boundaries for the previous stress period for which values were defined is used. Full details of all options to specify `stress_period_data` can be found in the `flopy3boundaries` Notebook in the `basic` subdirectory of the `examples` directory

- **dtype** (*dtype definition*) - if data type is different from default
- **options** (*list of strings*) - Package options. (default is None).
- **extension** (*string*) - Filename extension (default is 'ghb')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If `filenames=None` the package name will be created using the model name and package extension and the cbc output name will be created using the model name and `.cbc` extension (for example, `modflowtest.cbc`), if `ipakcbc` is a number greater than zero. If a single string is passed the package will be set to the string and cbc output names will be created using the model name and `.cbc` extension, if `ipakcbc` is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow()
>>> lrcsc = {0:[2, 3, 4, 10., 100.]} #this ghb will be applied to all
>>>                                     #stress periods
>>> ghb = flopy.modflow.ModflowGhb(m1, stress_period_data=lrcsc)
```

```
add_record(kper, index, values)
```

```
static get_default_dtype(structured=True)
```

```
static get_empty(ncells=0, aux_names=None, structured=True)
```

```
classmethod load(f, model, nper=None, ext_unit_dict=None, check=True)
```

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.Modflow*) to which this package will be added.
- **nper** (*int*) - The number of stress periods. If nper is None, then nper will be obtained from the model object. (default is None).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.
- **check** (*boolean*) - Check package data for common errors. (default True)

Returns

ghb - ModflowGhb object.

Return type

ModflowGhb object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> ghb = flopy.modflow.ModflowGhb.load('test.ghb', m)
```

write_file(*check=True*)

Write the package file.

Parameters

check (*boolean*) - Check package data for common errors. (default True)

Return type

None

flopy.modflow.mfgmg module

mfgmg module. Contains the ModflowGmg class. Note that the user can access the ModflowGmg class as *flopy.modflow.ModflowGmg*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowGmg(model, mxiter=50, iiter=30, iadamp=0, hclose=1e-05, rclose=1e-05,
    relax=1.0, ioutgmg=0, iunitmhc=None, ism=0, isc=0, damp=1.0, dup=0.75,
    dlow=0.01, chglimit=1.0, extension='gmg', unitnumber=None,
    filenames=None)
```

Bases: *Package*

MODFLOW GMG Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **mxiter** (*int*) - maximum number of outer iterations. (default is 50)
- **iiter** (*int*) - maximum number of inner iterations. (default is 30)
- **iadamp** (*int*) - is a flag that controls adaptive damping. The possible values of iadamp are.

If iadamp = 0, then the value assigned to DAMP is used as a constant damping parameter.

If iadamp = 1, the value of damp is used for the first nonlinear iteration. The damping parameter is adaptively varied on the basis of the head change, using Cooley's method as described in Mehl and Hill (2001), for subsequent iterations.

If iadamp = 2, the relative reduced residual damping method documented in Mehl and Hill (2001) and modified by Banta (2006) is used.

When iadamp is specified as 2 and the value specified for DAMP is less than 0.5, the closure criterion for the inner iterations (drclose) is assigned simply as rclose. When damp is between 0.5 and 1.0, inclusive, or when iadamp is specified as 0 or 1, drclose is calculated according to equation 20 on p. 9 of Wilson and Naff (2004).

- **hclose** (*float*) - is the head change criterion for convergence. (default is 1e-5).
- **rclose** (*float*) - is the residual criterion for convergence. (default is 1e-5)
- **relax** (*float*) - is a relaxation parameter for the ILU preconditioned conjugate gradient method. The relax parameter can be used to improve the spectral condition number of the ILU preconditioned system. The value of relax should be approximately one. However, the relaxation parameter can cause the factorization to break down. If this happens, then the gmrg solver will report an assembly error and a value smaller than one for relax should be tried. This item is read only if isc = 4.
- **ioutgmrg** (*int*) - is a flag that controls the output of the gmrg solver. The possible values of ioutgmrg are.

If ioutgmrg = 0, then only the solver inputs are printed.

If ioutgmrg = 1, then for each linear solve, the number of pcg iterations, the value of the damping parameter, the l2norm of the residual, and the maxnorm of the head change and its location (column, row, layer) are printed. At the end of a time/stress period, the total number of gmrg calls, pcg iterations, and a running total of pcg iterations for all time/stress periods are printed.

If ioutgmrg = 2, then the convergence history of the pcg iteration is printed, showing the l2norm of the residual and the convergence factor for each iteration.

ioutgmng = 3 is the same as ioutgmng = 1 except output is sent to the terminal instead of the modflow list output file.

ioutgmng = 4 is the same as ioutgmng = 2 except output is sent to the terminal instead of the modflow list output file.

(default is 0)

- **iunitmhc** (*int*) - is a flag and a unit number, which controls output of maximum head change values. If iunitmhc = 0, maximum head change values are not written to an output file. If iunitmhc > 0, maximum head change values are written to unit iunitmhc. Unit iunitmhc should be listed in the Name file with 'DATA' as the file type. If iunitmhc < 0 or is not present, iunitmhc defaults to 0. (default is 0)
- **ism** (*int*) - is a flag that controls the type of smoother used in the multigrid preconditioner. If ism = 0, then ilu(0) smoothing is implemented in the multigrid preconditioner; this smoothing requires an additional vector on each multigrid level to store the pivots in the ilu factorization. If ism = 1, then symmetric gaussseidel (sgs) smoothing is implemented in the multigrid preconditioner. No additional storage is required if ism = 1; users may want to use this option if available memory is exceeded or nearly exceeded when using ism = 0. Using sgs smoothing is not as robust as ilu smoothing; additional iterations are likely to be required in reducing the residuals. In extreme cases, the solver may fail to converge as the residuals cannot be reduced sufficiently. (default is 0)
- **isc** (*int*) - is a flag that controls semicoarsening in the multigrid preconditioner. If isc = 0, then the rows, columns and layers are all coarsened. If isc = 1, then the rows and columns are coarsened, but the layers are not. If isc = 2, then the columns and layers are coarsened, but the rows are not. If isc = 3, then the rows and layers are coarsened, but the columns are not. If isc = 4, then there is no coarsening. Typically, the value of isc should be 0 or 1. In the case that there are large vertical variations in the hydraulic conductivities, then a value of 1 should be used. If no coarsening is implemented (isc = 4), then the gmng solver is comparable to the pcg2 ilu(0) solver described in Hill (1990) and uses the least amount of memory. (default is 0)
- **damp** (*float*) - is the value of the damping parameter. For linear problems, a value of 1.0 should be used. For nonlinear problems, a value less than 1.0 but greater than 0.0 may be necessary to achieve convergence. A typical value for nonlinear problems is 0.5. Damping also helps control the convergence criterion of the linear solve to alleviate excessive pcg iterations. (default 1.)
- **dup** (*float*) - is the maximum damping value that should be applied at any iteration when the solver is not oscillating; it is dimensionless. An appropriate value for dup will be problem-dependent. For moderately nonlinear problems, reasonable values for dup would be in the range 0.5 to 1.0. For a highly nonlinear problem, a reasonable value for dup could be as small as

- 0.1. When the solver is oscillating, a damping value as large as $2.0 \times \text{DUP}$ may be applied. (default is 0.75)
- **dlow** (*float*) - is the minimum damping value to be generated by the adaptive-damping procedure; it is dimensionless. An appropriate value for dlow will be problem-dependent and will be smaller than the value specified for dup. For a highly nonlinear problem, an appropriate value for dlow might be as small as 0.001. Note that the value specified for the variable, chglimit, could result in application of a damping value smaller than dlow. (default is 0.01)
 - **chglimit** (*float*) - is the maximum allowed head change at any cell between outer iterations; it has units of length. The effect of chglimit is to determine a damping value that, when applied to all elements of the head-change vector, will produce an absolute maximum head change equal to chglimit. (default is 1.0)
 - **extension** (*list string*) - Filename extension (default is 'gmg')
 - **unitnumber** (*int*) - File unit number (default is None).
 - **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the gmg output name will be created using the model name and .cbc extension (for example, modflowtest.gmg.out), if iunitmhc is a number greater than zero. If a single string is passed the package will be set to the string and gmg output names will be created using the model name and .gmg.out extension, if iunitmhc is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Return type

None

Notes**Examples**

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> gmg = flopy.modflow.ModflowGmg(m)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case

`ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

`gmg`

Return type

ModflowGmg object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> gmg = flopy.modflow.ModflowGmg.load('test.gmg', m)
```

write_file()

Write the package file.

Return type

None

flopy.modflow.mfhfb module

mfhfb module. Contains the ModflowHfb class. Note that the user can access the ModflowHfb class as `flopy.modflow.ModflowHfb`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowHfb(model, nphfb=0, mxfb=0, nhfbnp=0, hfb_data=None, nacthfb=0,
                  no_print=False, options=None, extension='hfb', unitnumber=None,
                  filenames=None)
```

Bases: [Package](#)

MODFLOW HFB6 - Horizontal Flow Barrier Package

Parameters

- **model** (*model object*) - The model object (of type: class:`flopy.modflow.mf.Modflow` or `flopy.mfusg.MfUsq`) to which this package will be added.
- **nphfb** (*int*) - Number of horizontal-flow barrier parameters. Note that for an HFB parameter to have an effect in the simulation, it must be defined and made active using NACTHFB to have an effect in the simulation (default is 0).
- **mxfb** (*int*) - Maximum number of horizontal-flow barrier barriers that will be defined using parameters (default is 0).
- **nhfbnp** (*int*) - Number of horizontal-flow barriers not defined by parameters. This is calculated automatically by FloPy based on the information in `layer_row_column_data` (default is 0).
- **hfb_data** (*list of records*) - In its most general form, this is a list of horizontal-flow barrier records. A barrier is conceptualized as being located on the boundary between two adjacent finite difference cells in the same layer. The innermost

list is the layer, row1, column1, row2, column2, and hydrologic characteristics for a single hfb between the cells. The hydraulic characteristic is the barrier hydraulic conductivity divided by the width of the horizontal-flow barrier. (default is None). For a structured model, this gives the form of:

```
hfb_data = [
    [lay, row1, col1, row2, col2, hydchr],
    [lay, row1, col1, row2, col2, hydchr],
    [lay, row1, col1, row2, col2, hydchr],
    ].
```

Or for unstructured (mfusg) models::

```
hfb_data = [
    [node1, node2, hydchr], [node1, node2, hydchr], [node1,
    node2, hydchr],
    ].
```

- **nacthfb** (*int*) - The number of active horizontal-flow barrier parameters (default is 0).
- **no_print** (*boolean*) - When True or 1, a list of horizontal flow barriers will not be written to the Listing File (default is False)
- **options** (*list of strings*) - Package options (default is None).
- **extension** (*string*) - Filename extension (default is 'hfb').
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Parameters are supported in Flopy only when reading in existing models. Parameter values are converted to native values in Flopy and the connection to "parameters" is thus nonexistent.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> hfb_data = [[0, 10, 4, 10, 5, 0.01],[1, 10, 4, 10, 5, 0.01]]
>>> hfb = flopy.modflow.ModflowHfb(m, hfb_data=hfb_data)
```

```
static get_default_dtype(structured=True)
```

Get the default dtype for hfb data

static `get_empty(ncells=0, aux_names=None, structured=True)`

Get an empty recarray that corresponds to hfb dtype and has been extended to include aux variables and associated aux names.

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type: class:`flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

hfb - ModflowHfb object (of type `flopy.modflow.mfbas.ModflowHfb`)

Return type

ModflowHfb object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> hfb = flopy.modflow.ModflowHfb.load('test.hfb', m)
```

write_file()

Write the package file.

Return type

None

`flopy.modflow.mfhob` module

class `HeadObservation(model, tomulth=1.0, obsname='HOBS', layer=0, row=0, column=0, irefsp=None, roff=0.0, coff=0.0, itt=1, tmax=None, mlay=None, time_series_data=None, names=None)`

Bases: `object`

Create single HeadObservation instance from a time series array. A list of HeadObservation instances are passed to the ModflowHob package.

Parameters

- **tomulth** (*float*) - Time-offset multiplier for head observations. Default is 1.
- **obsname** (*string*) - Observation name. Default is 'HOBS'
- **layer** (*int*) - The zero-based layer index of the cell in which the head observation is located. If layer is less than zero,

hydraulic heads from multiple layers are combined to calculate a simulated value. The number of layers equals the absolute value of layer, or `abs(layer)`. Default is 0.

- **row** (*int*) - The zero-based row index for the observation. Default is 0.
- **column** (*int*) - The zero-based column index of the observation. Default is 0.
- **irefsp** (*int*) - The zero-based stress period to which the observation time is referenced.
- **roff** (*float*) - Fractional offset from center of cell in Y direction (between rows). Default is 0.
- **coff** (*float*) - Fractional offset from center of cell in X direction (between columns). Default is 0.
- **itt** (*int*) - Flag that identifies whether head or head changes are used as observations. `itt = 1` specified for heads and `itt = 2` specified if initial value is head and subsequent changes in head. Only specified if `irefsp` is `< 0`. Default is 1.
- **tmax** (*float*) - Maximum simulation time calculated using `get_final_totim` function of `ModflowDis`. Added to avoid repetitive calls.
- **mlay** (*dictionary of length (abs(irefsp))*) - Key represents zero-based layer numbers for multilayer observations and value represents the fractional value for each layer of multilayer observations. If `mlay` is `None`, a default `mlay` of `{0: 1.}` will be used (default is `None`).
- **time_series_data** (*list or numpy array*) - Two-dimensional list or numpy array containing the simulation time of the observation and the observed head `[[totim, hob]]`. If `time_series_data` default is `None`, a default observation of 0. at `totim 0.` will be created (default is `None`).
- **names** (*list*) - List of specified observation names. If `names` is `None`, observation names will be automatically generated from `obsname` and the order of the timeseries data (default is `None`).

Returns

`obs` - `HeadObservation` object.

Return type

[`HeadObservation`](#)

Examples

```
>>> import flopy
>>> model = flopy.modflow.Modflow()
>>> dis = flopy.modflow.ModflowDis(model, nlay=1, nrow=11, ncol=11, nper=2,
...                               perlen=[1,1])
>>> tsd = [[1.,54.4], [2., 55.2]]
>>> obsdata = flopy.modflow.HeadObservation(model, layer=0, row=5,
...                                         column=5, time_series_data=tsd)
```

```
class ModflowHob(model, iuhobsv=None, hobsdry=0, tomulth=1.0, obs_data=None, hobname=None,
                 extension='hob', no_print=False, options=None, unitnumber=None,
                 filenames=None)
```

Bases: [Package](#)

Head Observation package class

Parameters

- **iuhobsv** (*int*) - unit number where output is saved. If iuhobsv is None, a unit number will be assigned (default is None).
- **hobsdry** (*float*) - Value of the simulated equivalent written to the observation output file when the observation is omitted because a cell is dry (default is 0).
- **tomulth** (*float*) - Time step multiplier for head observations. The product of tomulth and toffset must produce a time value in units consistent with other model input. tomulth can be dimensionless or can be used to convert the units of toffset to the time unit used in the simulation (default is 1).
- **obs_data** (*HeadObservation* or *list of HeadObservation instances*)
- A single HeadObservation instance or a list of HeadObservation instances containing all of the data for each observation. If obs_data is None a default HeadObservation with an observation in layer, row, column (0, 0, 0) and a head value of 0 at totim 0 will be created (default is None).
- **hobname** (*str*) - Name of head observation output file. If iuhobsv is greater than 0, and hobname is None, the model basename with a '.hob.out' extension will be used (default is None).
- **extension** (*string*) - Filename extension (default is hob)
- **no_print** (*boolean*) - When True or 1, a list of head observations will not be written to the Listing File (default is False)
- **options** (*list of strings*) - Package options (default is None).
- **unitnumber** (*int*) - File unit number (default is None)
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames is None the package name will be created using the model name and package extension and the hob output name will be created using the model name and .hob.out extension (for example, modflowtest.hob.out), if iuhobsv is a number greater than zero. If a single string is passed the package will be set to the string and hob output name will be created using the model name and .hob.out extension, if iuhobsv is

a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

See also:

Notes

Examples

```
>>> import flopy
>>> model = flopy.modflow.Modflow()
>>> dis = flopy.modflow.ModflowDis(model, nlay=1, nrow=11, ncol=11, nper=2,
...                               perlen=[1,1])
>>> tsd = [[1.,54.4], [2., 55.2]]
>>> obsdata = flopy.modflow.HeadObservation(model, layer=0, row=5,
...                                         column=5, time_series_data=tsd)
>>> hob = flopy.modflow.ModflowHob(model, iuhobsv=51, hobsdry=-9999.,
...                                obs_data=obsdata)
```

classmethod `load(f, model, ext_unit_dict=None, check=True)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.
- **check** (*boolean*) - Check package data for common errors. (default True)

Returns

hob - ModflowHob package object.

Return type

ModflowHob package object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> hobs = flopy.modflow.ModflowHob.load('test.hob', m)
```

write_file()

Write the package file

Return type

None

flopy.modflow.mfhyd module

mfhyd module. Contains the ModflowHydclass. Note that the user can access the ModflowHyd class as *flopy.modflow.ModflowHyd*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowHyd(model, nhyd=1, ihydun=None, hydnoh=-999.0, obsdata=[[ 'BAS', 'HD', 'I', 0,
                                0.0, 0.0, 'HOBS1']], extension=['hyd', 'hyd.bin'], unitnumber=None,
                    filenames=None)
```

Bases: [Package](#)

MODFLOW HYDMOD (HYD) Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.modflow.mf.Modflow](#)) to which this package will be added.
- **nhyd** (*int*) - the maximum number of observation points. (default is 1).
- **ihydun** (*int*) - A flag that is used to determine if hydmod data should be saved. If ihydun is non-zero hydmod data will be saved. (default is 1).
- **hydnoh** (*float*) - is a user-specified value that is output if a value cannot be computed at a hydrograph location. For example, the cell in which the hydrograph is located may be a no-flow cell. (default is -999.)
- **obsdata** - Each row of obsdata includes data defining pckg (3 character string), arr (2 character string), intyp (1 character string) klay (int), xl (float), yl (float), hydlbl (14 character string) for each observation.

pckg

[str] is a 3-character flag to indicate which package is to be addressed by hydmod for the hydrograph of each observation point.

arr

[str] is a text code indicating which model data value is to be accessed for the hydrograph of each observation point.

intyp

[str] is a 1-character value to indicate how the data from the specified feature are to be accessed; The two options are 'I' for interpolated value or 'C' for cell value (intyp must be 'C' for STR and SFR Package hydrographs).

klay

[int] is the layer sequence number (zero-based) of the array to be addressed by HYDMOD.

xl

[float] is the coordinate of the hydrograph point in model units of length measured parallel to model rows, with the origin at the lower left corner of the model grid.

yl
[float] is the coordinate of the hydrograph point in model units of length measured parallel to model columns, with the origin at the lower left corner of the model grid.

hydlbl
[str] is used to form a label for the hydrograph.

extension

[list string] Filename extension (default is ['hyd', 'hyd.bin'])

unitnumber

[int] File unit number (default is None).

filenames

[str or list of str] Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the hydmod output name will be created using the model name and .hyd.bin extension (for example, modflowtest.hyd.bin). If a single string is passed the package will be set to the string and hydmod output name will be created using the model name and .hyd.bin extension. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> hyd = flopy.modflow.ModflowHyd(m)
```

static `get_default_dtype()`

static `get_empty(ncells=0)`

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

hyd

Return type

ModflowHyd object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> hyd = flopy.modflow.ModflowHyd.load('test.hyd', m)
```

`write_file()`

Write the package file.

Return type

None

`flopy.modflow.mflak module`

mflak module. Contains the `ModflowLak` class. Note that the user can access the `ModflowLak` class as `flopy.modflow.ModflowLak`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowLak(model, nlakes=1, ipakcb=None, theta=-1.0, nssitr=0, sscncr=0.0,
    surfdep=0.0, stages=1.0, stage_range=None, tab_files=None,
    tab_units=None, lakarr=None, bdlknc=None, sill_data=None,
    flux_data=None, extension='lak', unitnumber=None, filenames=None,
    options=None, lwrt=0, **kwargs)
```

Bases: [Package](#)

MODFLOW Lake Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **nlakes** (*int*) - NLAKES Number of separate lakes. Sublakes of multiple-lake systems are considered separate lakes for input purposes. The variable NLAKES is used, with certain internal assumptions and approximations, to dimension arrays for the simulation.
- **ipakcb** (*int*) - (ILKCB in MODFLOW documentation) Whether or not to write cell-by-cell flows (yes if ILKCB > 0, no otherwise). If ILKCB < 0 and "Save Budget" is specified in the Output Control or ICBCFL is not equal to 0, the cell-by-cell flows will be printed in the standard output file. ICBCFL is specified in the input to the Output Control Option of MODFLOW.
- **lwrt** (*int or list of ints (one per SP)*) - lwrt > 0, suppresses printout from the lake package. Default is 0 (to print budget information)
- **theta** (*float*) - Explicit (THETA = 0.0), semi-implicit (0.0 < THETA < 1.0), or implicit (THETA = 1.0) solution for lake stages. SURFDEPTH is read only if THETA is assigned a negative value (the negative value of THETA is then changed to a positive value internally by the code). * A new method of solving for lake stage uses only the time-weighting

factor THETA (Merritt and Konikow, 2000, p. 52) for transient simulations. THETA is automatically set to a value of 1.0 for all steady-state stress periods. For transient stress periods, Explicit (THETA = 0.0), semi-implicit (0.0 < THETA < 1.0), or implicit (THETA = 1.0) solutions can be used to calculate lake stages. The option to specify negative values for THETA is supported to allow specification of additional variables (NSSITR, SSCNCR, SURFDEP) for simulations that only include transient stress periods. If THETA is specified as a negative value, then it is converted to a positive value for calculations of lake stage.

- In MODFLOW-2000 and later, ISS is not part of the input. Instead NSSITR or SSCNCR should be included if one or more stress periods is a steady state stress period as defined in Ss/tr in the Discretization file.
- SSCNCR and NSSITR can be read for a transient only simulation by placing a negative sign immediately in front of THETA. A negative THETA sets a flag which assumes input values for NSSITR and SSCNCR will follow THETA in the format as described by Merritt and Konikow (p. 52). A negative THETA is automatically reset to a positive value after values of NSSITR and SSCNCR are read.
- **nssitr** (*int*) - Maximum number of iterations for Newton's method of solution for equilibrium lake stages in each MODFLOW iteration for steady-state aquifer head solution. Only read if ISS (option flag input to DIS Package of MODFLOW indicating steady-state solution) is not zero or if THETA is specified as a negative value. * NSSITR and SSCNCR may be omitted for transient solutions (ISS = 0). * In MODFLOW-2000 and later, ISS is not part of the input.
 Instead NSSITR or SSCNCR should be included if one or more stress periods is a steady state stress period as defined in Ss/tr in the Discretization file.
- SSCNCR and NSSITR can be read for a transient only simulation by placing a negative sign immediately in front of THETA. A negative THETA sets a flag which assumes input values for NSSITR and SSCNCR will follow THETA in the format as described by Merritt and Konikow (p. 52). A negative THETA is automatically reset to a positive value after values of NSSITR and SSCNCR are read.
- If NSSITR = 0, a value of 100 will be used instead.
- **sscnrcr** (*float*) - Convergence criterion for equilibrium lake stage solution by Newton's method. Only read if ISS is not zero or if THETA is specified as a negative value. See notes above for nssitr.
- **surfdepth** (*float*) - The height of small topological variations (undulations) in lake-bottom elevations that can affect groundwater discharge to lakes. SURFDEPTH decreases the lakebed

conductance for vertical flow across a horizontal lakebed caused both by a groundwater head that is between the lakebed and the lakebed plus SURFDEPTH and a lake stage that is also between the lakebed and the lakebed plus SURFDEPTH. This method provides a smooth transition from a condition of no groundwater discharge to a lake, when groundwater head is below the lakebed, to a condition of increasing groundwater discharge to a lake as groundwater head becomes greater than the elevation of the dry lakebed. The method also allows for the transition of seepage from a lake to groundwater when the lake stage decreases to the lakebed elevation. Values of SURFDEPTH ranging from 0.01 to 0.5 have been used successfully in test simulations. SURFDEP is read only if THETA is specified as a negative value.

- **stages** (*float or list of floats*) - The initial stage of each lake at the beginning of the run.
- **stage_range** (*list of tuples (ssmn, ssmx) of length nlakes*) - Where ssmn and ssmx are the minimum and maximum stages allowed for each lake in steady-state solution. * SSMN and SSMX are not needed for a transient run and must be omitted when the solution is transient.
 - When the first stress period is a steady-state stress period, SSMN is defined in record 3.

For subsequent steady-state stress periods, SSMN is defined in record 9a.

- **lakarr** (*array of integers (nlay, nrow, ncol)*) - LKARR A value is read in for every grid cell. If LKARR(I,J,K) = 0, the grid cell is not a lake volume cell. If LKARR(I,J,K) > 0, its value is the identification number of the lake occupying the grid cell. LKARR(I,J,K) must not exceed the value NLAKES. If it does, or if LKARR(I,J,K) < 0, LKARR(I,J,K) is set to zero. Lake cells cannot be overlain by non-lake cells in a higher layer. Lake cells must be inactive cells (IBOUND = 0) and should not be convertible to active cells (WETDRY = 0).

The Lake package can be used when all or some of the model layers containing the lake are confined. The authors recommend using the Layer-Property Flow Package (LPF) for this case, although the BCF and HUF Packages will work too. However, when using the BCF6 package to define aquifer properties, lake/aquifer conductances in the lateral direction are based solely on the lakebed leakance (and not on the lateral transmissivity of the aquifer layer). As before, when the BCF6 package is used, vertical lake/aquifer conductances are based on lakebed conductance and on the vertical hydraulic conductivity of the aquifer layer underlying the lake when the wet/dry option is implemented, and only on the lakebed leakance when the wet/dry option is not implemented.

- **bdlknc** (*array of floats (nlay, nrow, ncol)*) - BDLKNC A value is read in for every grid cell. The value is the lakebed leakance that will be assigned to lake/aquifer interfaces that occur in the corresponding grid cell. If the wet-dry option flag (IWDFLG)

is not active (cells cannot rewet if they become dry), then the BDLKNC values are assumed to represent the combined leakances of the lakebed material and the aquifer material between the lake and the centers of the underlying grid cells, i. e., the vertical conductance values (CV) will not be used in the computation of conductances across lake/aquifer boundary faces in the vertical direction.

IBOUND and WETDRY should be set to zero for every cell for which LKARR is not equal to zero. IBOUND is defined in the input to the Basic Package of MODFLOW. WETDRY is defined in the input to the BCF or other flow package of MODFLOW if the IWDFLG option is active. When used with the HUF package, the Lake Package has been modified to compute effective lake-aquifer conductance solely on the basis of the user-specified value of lakebed leakance; aquifer hydraulic conductivities are not used in this calculation. An appropriate informational message is now printed after the lakebed conductances are written to the main output file.

- **sill_data** (*dict*) - (dataset 8 in documentation) Dict of lists keyed by stress period. Each list has a tuple of dataset 8a, 8b for every multi-lake system, where dataset 8a is another tuple of

IC

[int] The number of sublakes

ISUB

[list of ints] The identification numbers of the sublakes in the sublake system being described in this record. The center lake number is listed first.

And dataset 8b contains

SILLVT

[sequence of floats] A sequence of sill elevations for each sublakes that determines whether the center lake is connected with a given sublake. Values are entered for each sublake in the order the sublakes are listed in the previous record.

- **flux_data** (*dict*) - (dataset 9 in documentation) Dict of lists keyed by stress period. The list for each stress period is a list of lists, with each list containing the variables PRCPLK EVAPLK RNF WTHDRW [SSMN] [SSMX] from the documentation.

PRCPLK

[float] The rate of precipitation per unit area at the surface of a lake (L/T).

EVAPLK

[float] The rate of evaporation per unit area from the surface of a lake (L/T).

RNF

[float] Overland runoff from an adjacent watershed entering the lake. If $RNF > 0$, it is specified directly as a volumetric rate, or flux (L³ /T). If $RNF < 0$, its absolute value is used as a dimensionless

multiplier applied to the product of the lake precipitation rate per unit area (PRCPLK) and the surface area of the lake at its full stage (occupying all layer 1 lake cells). When RNF is entered as a dimensionless multiplier ($RNF < 0$), it is considered to be the product of two proportionality factors. The first is the ratio of the area of the basin contributing runoff to the surface area of the lake when it is at full stage. The second is the fraction of the current rainfall rate that becomes runoff to the lake. This procedure provides a means for the automated computation of runoff rate from a watershed to a lake as a function of varying rainfall rate. For example, if the basin area is 10 times greater than the surface area of the lake, and 20 percent of the precipitation on the basin becomes overland runoff directly into the lake, then set $RNF = -2.0$.

WTHDRW

[float] The volumetric rate, or flux (L^3/T), of water removal from a lake by means other than rainfall, evaporation, surface outflow, or groundwater seepage. A negative value indicates augmentation. Normally, this would be used to specify the rate of artificial withdrawal from a lake for human water use, or if negative, artificial augmentation of a lake volume for aesthetic or recreational purposes.

SSMN

[float] Minimum stage allowed for each lake in steady-state solution. See notes on `ssmn` and `ssmx` above.

SSMX

[float] SSMX Maximum stage allowed for each lake in steady-state solution.

- **options** (*list of strings*) - Package options. (default is None).
- **extension** (*string*) - Filename extension (default is 'lak')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If `filenames=None` the package name will be created using the model name and package extension and the cbc output name will be created using the model name and .cbc extension (for example, `modflowtest.cbc`), if `ipakcbc` is a number greater than zero. If a single string is passed the package will be set to the string and cbc output names will be created using the model name and .cbc extension, if `ipakcbc` is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lak = {}
>>> lak[0] = [[2, 3, 4, 15.6, 1050., -4]] #this lake boundary will be
>>> #applied to all stress periods
>>> lak = flopy.modflow.ModflowLak(m, nstress_period_data=strd)
```

classmethod `load(f, model, nper=None, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **nper** (*int*) - The number of stress periods. If *nper* is *None*, then *nper* will be obtained from the model object. (default is *None*).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function *flopy.utils.mfreadnam.parse_namefile*.

Returns

str - ModflowLak object.

Return type

ModflowLak object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lak = flopy.modflow.ModflowStr.load('test.lak', m)
```

write_file()

Write the package file.

Return type

None

`flopy.modflow.mflmt` module

`mflmt` module. Contains the `ModflowLmt` class. Note that the user can access the `ModflowLmt` class as `flopy.modflow.ModflowLmt`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowLmt(model, output_file_name='mt3d_link.ftl', output_file_unit=54,
                  output_file_header='extended', output_file_format='unformatted',
                  extension='lmt6', package_flows=[], unitnumber=None, filenames=None)
```

Bases: [Package](#)

MODFLOW Link-MT3DMS Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.modflow.mf.Modflow](#)) to which this package will be added.
- **output_file_name** (*string*) - Filename for output file (default is 'mt3d_link.ftl')
- **unitnumber** (*int*) - File unit number (default is 24).
- **output_file_unit** (*int*) - Output file unit number, pertaining to the file identified by `output_file_name` (default is 54).
- **output_file_header** (*string*) - Header for the output file (default is 'extended')
- **output_file_format** (*{'formatted', 'unformatted'}*) - Format of the output file (default is 'unformatted')
- **package_flows** (*['sfr', 'lak', 'uzf']*) - Specifies which of the advanced package flows should be added to the flow-transport link (FTL) file. The addition of these flags may quickly increase the FTL file size. Thus, the user must specifically request their amendment within the FTL file. Default is not to add these terms to the FTL file by omitting the keyword `package_flows` from the LMT input file. One or multiple strings can be passed as a list to the argument.
- **extension** (*string*) - Filename extension (default is 'lmt6')
- **unitnumber** - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If `filenames=None` the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Parameters are supported in Flopy only when reading in existing models. Parameter values are converted to native values in Flopy and the connection to “parameters” is thus nonexistent.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lmt = flopy.modflow.ModflowLmt(m, output_file_name='mt3d_linkage.ftl')
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

lmt - ModflowLmt object.

Return type

ModflowLmt object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lmt = flopy.modflow.ModflowGhb.load('test.lmt', m)
```

write_file()

Write the package file.

Return type

None

flopy.modflow.mflpf module

mflpf module. Contains the ModflowLpf class. Note that the user can access the ModflowLpf class as `flopy.modflow.ModflowLpf`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowLpf(model, laytyp=0, layavg=0, chani=1.0, layvka=0, laywet=0, ipakcb=None,
                 hdry=-1e+30, iwdflg=0, wetfct=0.1, iwetit=1, ihdwet=0, hk=1.0,
                 hani=1.0, vka=1.0, ss=1e-05, sy=0.15, vkcb=0.0, wetdry=-0.01,
                 storagecoefficient=False, constantcv=False, thickstrt=False,
                 nocvcorrection=False, novfc=False, extension='lpf', unitnumber=None,
                 filenames=None, add_package=True)
```

Bases: [Package](#)

MODFLOW Layer Property Flow Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0)
- **hdry** (*float*) - Is the head that is assigned to cells that are converted to dry during a simulation. Although this value plays no role in the model calculations, it is useful as an indicator when looking at the resulting heads that are output from the model. HDRY is thus similar to HNOFLO in the Basic Package, which is the value assigned to cells that are no-flow cells at the start of a model simulation. (default is -1.e30).
- **laytyp** (*int or array of ints (nlay)*) - Layer type, contains a flag for each layer that specifies the layer type. 0 confined >0 convertible <0 convertible unless the THICKSTRT option is in effect. (default is 0).
- **layavg** (*int or array of ints (nlay)*) - Layer average 0 is harmonic mean 1 is logarithmic mean 2 is arithmetic mean of saturated thickness and logarithmic mean of hydraulic conductivity (default is 0).
- **chani** (*float or array of floats (nlay)*) - contains a value for each layer that is a flag or the horizontal anisotropy. If CHANI is less than or equal to 0, then variable HANI defines horizontal anisotropy. If CHANI is greater than 0, then CHANI is the horizontal anisotropy for the entire layer, and HANI is not read. If any HANI parameters are used, CHANI for all layers must be less than or equal to 0. Use as many records as needed to enter a value of CHANI for each layer. The horizontal anisotropy is the ratio of the hydraulic conductivity along columns (the Y direction) to the hydraulic conductivity along rows (the X direction). (default is 1).
- **layvka** (*int or array of ints (nlay)*) - a flag for each layer that indicates whether variable VKA is vertical hydraulic conductivity or the ratio of horizontal to vertical hydraulic conductivity. 0:

VKA is vertical hydraulic conductivity not 0: VKA is the ratio of horizontal to vertical hydraulic conductivity (default is 0).

- **laywet** (*int* or array of ints (*nlay*)) - contains a flag for each layer that indicates if wetting is active. 0 wetting is inactive not 0 wetting is active (default is 0).
- **wetfct** (*float*) - is a factor that is included in the calculation of the head that is initially established at a cell when it is converted from dry to wet. (default is 0.1).
- **iwetit** (*int*) - is the iteration interval for attempting to wet cells. Wetting is attempted every IWETIT iteration. If using the PCG solver (Hill, 1990), this applies to outer iterations, not inner iterations. If IWETIT less than or equal to 0, it is changed to 1. (default is 1).
- **ihdwet** (*int*) - is a flag that determines which equation is used to define the initial head at cells that become wet. (default is 0)
- **hk** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - is the hydraulic conductivity along rows. HK is multiplied by horizontal anisotropy (see CHANI and HANI) to obtain hydraulic conductivity along columns. (default is 1.0).
- **hani** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - is the ratio of hydraulic conductivity along columns to hydraulic conductivity along rows, where HK of item 10 specifies the hydraulic conductivity along rows. Thus, the hydraulic conductivity along columns is the product of the values in HK and HANI. (default is 1.0).
- **vka** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - is either vertical hydraulic conductivity or the ratio of horizontal to vertical hydraulic conductivity depending on the value of LAYVKA. (default is 1.0).
- **ss** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - is specific storage unless the STORAGECOEFFICIENT option is used. When STORAGECOEFFICIENT is used, Ss is confined storage coefficient. (default is 1.e-5).
- **sy** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - is specific yield. (default is 0.15).
- **vkcb** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - is the vertical hydraulic conductivity of a Quasi-three-dimensional confining bed below a layer. (default is 0.0). Note that if an array is passed for vkcb it must be of size (nlay, nrow, ncol) even though the information for the bottom layer is not needed.
- **wetdry** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - is a combination of the wetting threshold and a flag to indicate which neighboring cells can cause a cell to become wet. (default is -0.01).
- **storagecoefficient** (*boolean*) - indicates that variable Ss and SS parameters are read as storage coefficient rather than specific storage. (default is False).

- **constantcv** (*boolean*) - indicates that vertical conductance for an unconfined cell is computed from the cell thickness rather than the saturated thickness. The CONSTANTCV option automatically invokes the NOCVCORRECTION option. (default is False).
- **thickstrt** (*boolean*) - indicates that layers having a negative LAYTYP are confined, and their cell thickness for conductance calculations will be computed as STRT-BOT rather than TOP-BOT. (default is False).
- **nocvcorrection** (*boolean*) - indicates that vertical conductance is not corrected when the vertical flow correction is applied. (default is False).
- **novfc** (*boolean*) - turns off the vertical flow correction under dewatered conditions. This option turns off the vertical flow calculation described on p. 5-8 of USGS Techniques and Methods Report 6-A16 and the vertical conductance correction described on p. 5-18 of that report. (default is False).
- **extension** (*string*) - Filename extension (default is 'lpf')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output name will be created using the model name and .cbc extension (for example, modflowtest.cbc), if ipakcbc is a number greater than zero. If a single string is passed the package will be set to the string and cbc output name will be created using the model name and .cbc extension, if ipakcbc is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.
- **add_package** (*bool*) - Flag to add the initialised package object to the parent model object. Default is True.

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lpf = flopy.modflow.ModflowLpf(m)
```

classmethod `load(f, model, ext_unit_dict=None, check=True)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.

- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.
- **check** (*boolean*) - Check package data for common errors. (default True)

Returns

lpf - ModflowLpf object.

Return type

ModflowLpf object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lpf = flopy.modflow.ModflowLpf.load('test.lpf', m)
```

write_file(*check=True, f=None*)

Write the package file.

Parameters

check (*boolean*) - Check package data for common errors. (default True)

Return type

None

flopy.modflow.mfmult module

mfmult module. Contains the ModflowMlt class. Note that the user can access the ModflowMlt class as `flopy.modflow.ModflowMlt`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

class ModflowMlt(*model, mult_dict=None, extension='mlt', unitnumber=None, filenames=None*)

Bases: [Package](#)

MODFLOW Mult Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **mult_dict** (*dict*) - Dictionary with mult data for the model. *mult_dict* is typically instantiated using load method.
- **extension** (*string*) - Filename extension (default is 'drn')
- **unitnumber** (*int*) - File unit number (default is 21).

Notes

Parameters are supported in FloPy only when reading in existing models. Parameter values are converted to native values in FloPy and the connection to “parameters” is thus nonexistent.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> mlt = flopy.modflow.ModflowZon(m, mult_dict=mult_dict)
```

classmethod `load(f, model, nrow=None, ncol=None, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **nrow** (*int*) - number of rows. If not specified it will be retrieved from the model object. (default is None).
- **ncol** (*int*) - number of columns. If not specified it will be retrieved from the model object. (default is None).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

zone

Return type

ModflowMult dict

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> mlt = flopy.modflow.ModflowMlt.load('test.mlt', m)
```

static `mult_function(mult_dict, line)`

Construct a multiplier for the ‘FUNCTION’ option

write_file()

Write the package file.

Return type

None

Notes

Not implemented because parameters are only supported on load

`flopy.modflow.mfmnw1` module

```
class ModflowMnw1(model, mxmnw=0, ipakcb=None, iwelpt=0, nomoiter=0, kspref=1,
                  well_bynode_qsum=None, losstype='skin', stress_period_data=None,
                  dtype=None, mnwname=None, extension='mnw1', unitnumber=None,
                  filenames=None)
```

Bases: [`Package`](#)

MODFLOW Multi-Node Well 1 Package Class.

Parameters

- **model** (*model object*) - The model object (of type [`flopy.modflow.mf.Modflow`](#)) to which this package will be added.
- **mxmnw** (*integer*) - maximum number of multi-node wells to be simulated
- **ipakcb** (*integer*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0).
- **iwelpt** (*integer*) - verbosity flag
- **nomoiter** (*integer*) - the number of iterations for which flow in MNW wells is calculated
- **kspref** (*string*) - which set of water levels are to be used as reference values for calculating drawdown
- **losstype** (*string*) - head loss type for each well
- **well_bynode_qsum** (*list of lists or None*) - nested list containing file names, unit numbers, and ALLTIME flag for auxiliary output, e.g. `[[‘test.ByNode’,92,’ALLTIME’]]` if None, these optional external filenames and unit numbers are not written out
- **itmp** (*array*) - number of wells to be simulated for each stress period (shape : (NPER))
- **lay_row_col_qdes_mn_multi** (*list of arrays*) - lay, row, col, qdes, and MN or MULTI flag for all well nodes (length : NPER)
- **mnwname** (*string*) - prefix name of file for outputting time series data from MNW1
- **extension** (*string*) - Filename extension (default is ‘mnw1’)
- **unitnumber** (*int*) - File unit number (default is 33).
- **filenames** (*string or list of strings*) - File name of the package (with extension) or a list with the filename of the package and the cell-by-cell budget file for ipakcb. Default is None.

Notes

Parameters are not supported in FloPy.

The functionality of the ADD flag in data set 4 is not supported. Also not supported are all water-quality parameters (Qwval Iqwgrp), water-level limitations (Hlim, Href, DD), non-linear well losses, and pumping limitations (QCUT, Q-%CUT, Qfrcmn, Qfrcmx, DEFAULT).

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow()
>>> mnw1 = flopy.modflow.ModflowMnw1(m1, ...)
```

static `get_default_dtype(structured=True)`

static `get_empty_stress_period_data(itmp, structured=True, default_value=0)`

classmethod `load(f, model, nper=None, gwt=False, nsol=1, ext_unit_dict=None)`

Default load method for standard boundary packages.

write_file()

Write the package file.

Return type

None

skipcomments(line, f)

`flopy.modflow.mfmnw2` module

exception `ItmpError(itmp, nactivewells)`

Bases: `Exception`

class `Mnw(wellid, nnodes=1, nper=1, losstype='skin', pumploc=0, qlimit=0, ppflag=0, pumpcap=0, rw=1, rskin=2, kskin=10, B=None, C=0, P=2.0, cwc=None, pp=1, k=0, i=0, j=0, ztop=0, zbotm=0, node_data=None, stress_period_data=None, pumplay=0, pumprow=0, pumpcol=0, zpump=None, hlim=None, qcut=None, qfrcmn=None, qfrcmx=None, hlift=None, liftq0=None, liftqmax=None, hwtol=None, liftn=None, qn=None, mnwpackage=None)`

Bases: `object`

Multi-Node Well object class

Parameters

- **wellid** (`str` or `int`) - is the name of the well. This is a unique alphanumeric identification label for each well. The text string is limited to 20 alphanumeric characters. If the name of the well includes spaces, then enclose the name in quotes. Flopy converts wellid string to lower case.
- **nnodes** (`int`) - is the number of cells (nodes) associated with this well. NNODES normally is > 0, but for the case of a vertical borehole, setting NNODES < 0 will allow the user to specify

the elevations of the tops and bottoms of well screens or open intervals (rather than grid layer numbers), and the absolute value of NNODES equals the number of open intervals (or well screens) to be specified in dataset 2d. If this option is used, then the model will compute the layers in which the open intervals occur, the lengths of the open intervals, and the relative vertical position of the open interval within a model layer (for example, see figure 14 and related discussion).

- **losstype** (*str*) - is a character flag to determine the user-specified model for well loss (equation 2). Available options (that is, place one of the following approved words in this field) are: NONE there are no well corrections and the head in the well is

assumed to equal the head in the cell. This option (hWELL = hn) is only valid for a single-node well (NNODES = 1). (This is equivalent to using the original WEL Package of MODFLOW, but specifying the single-node well within the MNW2 Package enables the use of constraints.)

THIEM this option allows for only the cell-to-well correction at the well based on the Thiem (1906) equation; head in the well is determined from equation 2 as (hWELL = hn + AQn), and the model computes A on the basis of the user-specified well radius (Rw) and previously defined values of cell transmissivity and grid spacing. Coefficients B and C in equation 2 are automatically set = 0.0. User must define Rw in dataset 2c or 2d.

SKIN this option allows for formation damage or skin corrections at the well. hWELL = hn + AQn + BQn (from equation 2), where A is determined by the model from the value of Rw, and B is determined by the model from Rskin and Kskin. User must define Rw, Rskin, and Kskin in dataset 2c or 2d.

GENERAL head loss is defined with coefficients A, B, and C and power exponent P (hWELL = hn + AQn + BQn + CQnP). A is determined by the model from the value of Rw. User must define Rw, B, C, and P in dataset 2c or 2d. A value of P = 2.0 is suggested if no other data are available (the model allows $1.0 \leq P \leq 3.5$). Entering a value of C = 0 will result in a "linear" model in which the value of B is entered directly (rather than entering properties of the skin, as with the SKIN option).

SPECIFYcwc the user specifies an effective conductance value (equivalent to the combined effects of the A, B, and C well-loss coefficients expressed in equation 15) between the well and the cell representing the aquifer, CWC. User must define CWC in dataset 2c or 2d. If there are multiple screens within the grid cell or if partial penetration corrections are to be made, then the effective value of CWC for the node may be further adjusted automatically by MNW2.

- **pumploc** (*int*) - is an integer flag pertaining to the location along the borehole of the pump intake (if any). If PUMPLOC = 0,

then either there is no pump or the intake location (or discharge point for an injection well) is assumed to occur above the first active node associated with the multi- node well (that is, the node closest to the land surface or to the wellhead). If PUMPLOC > 0, then the cell in which the intake (or outflow) is located will be specified in dataset 2e as a LAY-ROW-COL grid location. For a vertical well only, specifying PUMPLOC < 0, will enable the option to define the vertical position of the pump intake (or outflow) as an elevation in dataset 2e (for the given spatial grid location [ROW-COL] defined for this well in 2d).

- **qlimit** (*int*) - is an integer flag that indicates whether the water level (head) in the well will be used to constrain the pumping rate. If Qlimit = 0, then there are no constraints for this well. If Qlimit > 0, then pumpage will be limited (constrained) by the water level in the well, and relevant parameters are constant in time and defined below in dataset 2f. If Qlimit < 0, then pumpage will be limited (constrained) by the water level in the well, and relevant parameters can vary with time and are defined for every stress period in dataset 4b.
- **ppflag** (*int*) - is an integer flag that determines whether the calculated head in the well will be corrected for the effect of partial penetration of the well screen in the cell. If PPFLAG = 0, then the head in the well will not be adjusted for the effects of partial penetration. If PPFLAG > 0, then the head in the well will be adjusted for the effects of partial penetration if the section of well containing the well screen is vertical (as indicated by identical row-column locations in the grid). If NNODES < 0 (that is, the open intervals of the well are defined by top and bottom elevations), then the model will automatically calculate the fraction of penetration for each node and the relative vertical position of the well screen. If NNODES > 0, then the fraction of penetration for each node must be defined in dataset 2d (see below) and the well screen will be assumed to be centered vertically within the thickness of the cell (except if the well is located in the uppermost model layer that is under unconfined conditions, in which case the bottom of the well screen will be assumed to be aligned with the bottom boundary of the cell and the assumed length of well screen will be based on the initial head in that cell).
- **pumpcap** (*int*) - is an integer flag and value that determines whether the discharge of a pumping (withdrawal) well ($Q < 0.0$) will be adjusted for changes in the lift (or total dynamic head) with time. If PUMPCAP = 0, then the discharge from the well will not be adjusted on the basis of changes in lift. If PUMPCAP > 0 for a withdrawal well, then the discharge from the well will be adjusted on the basis of the lift, as calculated from the most recent water level in the well. In this case, data describing the head-capacity relation for the pump must be listed in datasets 2g and 2h, and the use of that relation can be switched on or off for each stress period using a flag in dataset 4a. The number of entries (lines) in dataset 2h corresponds to the value of PUMPCAP. If PUMPCAP does not equal 0, it must be set to an integer value of

between 1 and 25, inclusive.

- **rw** (*float*) - radius of the well (losstype == 'THIEM', 'SKIN', or 'GENERAL')
- **rskin** (*float*) - radius to the outer limit of the skin (losstype == 'SKIN')
- **kskin** (*float*) - hydraulic conductivity of the skin
- **B** (*float*) - coefficient of the well-loss eqn. (eqn. 2 in MNW2 documentation) (losstype == 'GENERAL')
- **C** (*float*) - coefficient of the well-loss eqn. (eqn. 2 in MNW2 documentation) (losstype == 'GENERAL')
- **P** (*float*) - coefficient of the well-loss eqn. (eqn. 2 in MNW2 documentation) (losstype == 'GENERAL')
- **cwc** (*float*) - cell-to-well conductance. (losstype == 'SPECIFYcwc')
- **pp** (*float*) - fraction of partial penetration for the cell. Only specify if PFLAG > 0 and NNODES > 0.
- **k** (*int*) - layer index of well (zero-based)
- **i** (*int*) - row index of well (zero-based)
- **j** (*int*) - column index of well (zero-based)
- **ztop** (*float*) - top elevation of open intervals of vertical well.
- **zbotm** (*float*) - bottom elevation of open intervals of vertical well.
- **node_data** (*numpy record array*) - table containing MNW data by node. A blank node_data template can be created via the `ModflowMnw2.get_empty_mnw_data()` static method.

Note: Variables in dataset 2d (e.g. `rw`) can be entered as a single value for the entire well (above), or in `node_data` (or dataset 2d) by node. Variables not in dataset 2d (such as `pumpplay`) can be included in node data for convenience (to allow construction of MNW2 package from a table), but are only written to MNW2 as a single variable. When writing non-dataset 2d variables to MNW2 input, the first value for the well will be used.

Other variables (e.g. `hlim`) can be entered here as constant for all stress periods, or by stress period below in `stress_period_data`. See MNW2 input instructions for more details.

Columns are:

- k**
[int] layer index of well (zero-based)
- i**
[int] row index of well (zero-based)
- j**
[int] column index of well (zero-based)

ztop

[float] top elevation of open intervals of vertical well.

zbotm

[float] bottom elevation of open intervals of vertical well.

```
wellid : str losstype : str pumploc : int qlimit : int
ppflag : int pumpcap : int rw : float rskin : float kskin
: float B : float C : float P : float cwc : float pp :
float pumplay : int pumprow : int pumpcol : int zpump
: float hlim : float qcut : int qfrcmn : float qfrcmx :
float hlift : float liftq0 : float liftqmax : float hwtol :
float liftn : float qn : float
```

- **stress_period_data** (*numpy record array*) - table containing MNW pumping data for all stress periods (dataset 4 in the MNW2 input instructions). A blank stress_period_data template can be created via the `Mnw.get_empty_stress_period_data()` static method. Columns are:

per

[int] stress period

qdes

[float] is the actual (or maximum desired, if constraints are to be applied) volumetric pumping rate (negative for withdrawal or positive for injection) at the well (L3/T). Qdes should be set to 0 for nonpumping wells. If constraints are applied, then the calculated volumetric withdrawal or injection rate may be adjusted to range from 0 to Qdes and is not allowed to switch directions between withdrawal and injection conditions during any stress period. When PUMPCAP > 0, in the first stress period in which Qdes is specified with a negative value, Qdes represents the maximum operating discharge for the pump; in subsequent stress periods, any different negative values of Qdes are ignored, although values are subject to adjustment for CapMult. If Qdes >= 0.0, then pump-capacity adjustments are not applied.

capmult

[int] is a flag and multiplier for implementing head-capacity relations during a given stress period. Only specify if PUMPCAP > 0 for this well. If CapMult <= 0, then head-capacity relations are ignored for this stress period. If CapMult = 1.0, then head-capacity relations defined in datasets 2g and 2h are used. If CapMult equals any other positive value (for example, 0.6 or 1.1), then head-capacity relations are used but adjusted and shifted by multiplying the discharge value indicated by the head-capacity curve by the value of CapMult.

cprime

[float] is the concentration in the injected fluid.

Only specify if Qdes > 0 and GWT process is active.

hlim : float qcut : int qfrcmn : float qfrcmx : float

Note: If auxiliary variables are also being used, additional columns for these must be included.

- **pumplay** (*int*) -
- **pumprow** (*int*) -
- **pumpcol** (*int*) - PUMPLAY, PUMPROW, and PUMPCOL are the layer, row, and column numbers, respectively, of the cell (node) in this multi-node well where the pump intake (or outflow) is located. The location defined in dataset 2e should correspond with one of the nodes listed in 2d for this multi-node well. These variables are only read if PUMPLOC > 0 in 2b.
- **zpump** (*float*) - is the elevation of the pump intake (or discharge pipe location for an injection well). Zpump is read only if PUMPLOC < 0; in this case, the model assumes that the borehole is vertical and will compute the layer of the grid in which the pump intake is located.
- **hlim** (*float*) - is the limiting water level (head) in the well, which is a minimum for discharging wells and a maximum for injection wells. For example, in a discharging well, when hWELL falls below hlim, the flow from the well is constrained.
- **qcut** (*int*) - is an integer flag that indicates how pumping limits Qfrcmn and Qfrcmx will be specified. If pumping limits are to be specified as a rate (L3/T), then set QCUT > 0; if pumping limits are to be specified as a fraction of the specified pumping rate (Qdes), then set QCUT < 0. If there is not a minimum pumping rate below which the pump becomes inactive, then set QCUT = 0.
- **qfrcmn** (*float*) - is the minimum pumping rate or fraction of original pumping rate (a choice that depends on QCUT) that a well must exceed to remain active during a stress period. The absolute value of Qfrcmn must be less than the absolute value of Qfrcmx (defined next). Only specify if QCUT != 0.
- **qfrcmx** (*float*) - is the minimum pumping rate or fraction of original pumping rate that must be exceeded to reactivate a well that had been shut off based on Qfrcmn during a stress period. The absolute value of Qfrcmx must be greater than the absolute value of Qfrcmn. Only specify if QCUT != 0.
- **hlift** (*float*) - is the reference head (or elevation) corresponding to the discharge point for the well. This is typically at or above the land surface, and can be increased to account for additional head loss due to friction in pipes.
- **liftq0** (*float*) - is the value of lift (total dynamic head) that exceeds the capacity of the pump. If the calculated lift equals or exceeds this value, then the pump is shut off and discharge from the well ceases.
- **liftqmax** (*float*) - is the value of lift (total dynamic head) corresponding to the maximum pumping (discharge) rate for the

pump. If the calculated lift is less than or equal to LIFTqmax, then the pump will operate at its design capacity, assumed to equal the user-specified value of Qdes (in dataset 4a). LIFTqmax will be associated with the value of Qdes in the first stress period in which Qdes for the well is less than 0.0.

- **hwtol** (*float*) - is a minimum absolute value of change in the computed water level in the well allowed between successive iterations; if the value of hWELL changes from one iteration to the next by a value smaller than this tolerance, then the value of discharge computed from the head capacity curves will be locked for the remainder of that time step. It is recommended that HWtol be set equal to a value approximately one or two orders of magnitude larger than the value of HCLOSE, but if the solution fails to converge, then this may have to be adjusted.
- **liftn** (*float*) - is a value of lift (total dynamic head) that corresponds to a known value of discharge (Qn) for the given pump, specified as the second value in this line.
- **qn** (*float*) - is the value of discharge corresponding to the height of lift (total dynamic head) specified previously on this line. Sign (positive or negative) is ignored.
- **mnwpackage** (*ModflowMnw2 instance*) - package that mnw is attached to

Return type

None

by_node_variables = ['k', 'i', 'j', 'ztop', 'zbotm', 'rw', 'rskin', 'kskin', 'B', 'C', 'P', 'cwc', 'pp']

check(*f=None, verbose=True, level=1, checktype=None*)

Check mnw object for common errors.

Parameters

- **f** (*str* or *file handle*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If f is None, check method does not write results to a summary file. (default is None)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.

Returns

chk

Return type

flopy.utils.check object

static get_default_spd_dtype(*structured=True*)

Get the default stress period data dtype

Parameters

structured (*bool*) - Boolean that defines if a structured (True) or unstructured (False) dtype will be created (default is True). Not implemented for unstructured.

Returns

dtype

Return type

np.dtype

static get_empty_stress_period_data(*nper=0, aux_names=None, structured=True, default_value=0*)

Get an empty stress_period_data recarray that corresponds to dtype

Parameters

- **nper** (*int*) -
- **aux_names** -
- **structured** -
- **default_value** -

Returns

ra - Recarray

Return type

np.recarray

static get_item2_names(*mnw2obj=None, node_data=None*)

Get names for unknown things...

Parameters

- **mnw2obj** (*Mnw object*) - Mnw object (default is None)
- **node_data** (*unknown*) - Unknown what is in this parameter (default is None)

Returns

- **names** (*list*)
- *List of dtype names.*

static get_nnodes(*node_data*)

Get the number of MNW2 nodes.

Parameters

node_data (*list*) - List of nodes???

Returns

nnodes - Number of MNW2 nodes

Return type

int

make_node_data()

Make the node data array from variables entered individually.

Return type

None

```
static sort_node_data(node_data)
```

```
class ModflowMnw2(model, mnwmax=0, noddot=None, ipakcb=0, mnwprnt=0, aux=[],
                  node_data=None, mnw=None, stress_period_data=None, itmp=[],
                  extension='mnw2', unitnumber=None, filenames=None, gwt=False)
```

Bases: [Package](#)

Multi-Node Well 2 Package Class

Parameters

- **model** (*model object*) - The model object (of type :class:'flopy.modflow.mf.Modflow') to which this package will be added.
- **mnwmax** (*int*) - The absolute value of MNWMAX is the maximum number of multi-node wells (MNW) to be simulated. If MNWMAX is a negative number, NODTOT is read.
- **noddot** (*int*) - Maximum number of nodes. The code automatically estimates the maximum number of nodes (NODTOT) as required for allocation of arrays. However, if a large number of horizontal wells are being simulated, or possibly for other reasons, this default estimate proves to be inadequate, a new input option has been added to allow the user to directly specify a value for NODTOT. If this is a desired option, then it can be implemented by specifying a negative value for "MNWMAX"-the first value listed in Record 1 (Line 1) of the MNW2 input data file. If this is done, then the code will assume that the very next value on that line will be the desired value of "NODTOT". The model will then reset "MNWMAX" to its absolute value. The value of "ipakcb" will become the third value on that line, etc.
- **ipakcb** (*int*) -
is a flag and a unit number:
if ipakcb > 0, then it is the unit number to which MNW cell-by-cell flow terms will be recorded whenever cell-by-cell budget data are written to a file (as determined by the outputcontrol options of MODFLOW). if ipakcb = 0, then MNW cell-by-cell flow terms will not be printed
or recorded.

if ipakcb < 0, then well injection or withdrawal rates and water levels in the well and its multiple cells will be printed in the main MODFLOW listing (output) file whenever cell-by-cell budget data are written to a file (as determined by the output control options of MODFLOW).
- **mnwprnt** (*integer*) - Flag controlling the level of detail of information about multi-node wells to be written to the main MODFLOW listing (output) file. If MNWPRNT = 0, then only basic well information will be printed in the main MODFLOW output file; increasing the value of MNWPRNT yields more information, up to a maximum level of detail corresponding with MNWPRNT = 2. (default is 0)

- **aux** (*list of strings*) - (listed as "OPTION" in MNW2 input instructions) is an optional list of character values in the style of "AUXILIARY abc" or "AUX abc" where "abc" is the name of an auxiliary parameter to be read for each multi-node well as part of dataset 4a. Up to 20 parameters can be specified, each of which must be preceded by "AUXILIARY" or "AUX." These parameters will not be used by the MNW2 Package, but they will be available for use by other packages. (default is None)
- **node_data** (*numpy record array*) - master table describing multi-node wells in package. Same format as node_data tables for each Mnw object. See Mnw class documentation for more information.
- **mnw** (*list or dict of Mnw objects*) - Can be supplied instead of node_data and stress_period_data tables (in which case the tables are constructed from the Mnw objects). Otherwise the a dict of Mnw objects (keyed by wellid) is constructed from the tables.
- **stress_period_data** (*dict of numpy record arrays*) - master dictionary of record arrays (keyed by stress period) containing transient input for multi-node wells. Format is the same as stress period data for individual Mnw objects, except the 'per' column is replaced by 'wellid' (containing wellid for each MNW). See Mnw class documentation for more information.
- **itmp** (*list of ints*) - is an integer value for reusing or reading multi-node well data; it can change each stress period. ITMP must be ≥ 0 for the first stress period of a simulation. if $ITMP > 0$, then ITMP is the total number of active multi-node wells simulated during the stress period, and only wells listed in dataset 4a will be active during the stress period. Characteristics of each well are defined in datasets 2 and 4.

if $ITMP = 0$, then no multi-node wells are active for the stress period and the following dataset is skipped.

if $ITMP < 0$, then the same number of wells and well information will be reused from the previous stress period and dataset 4 is skipped.
- **extension** (*string*) - Filename extension (default is 'mnw2')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output name will be created using the model name and .cbc extension (for example, modflowtest.cbc), if ipakcbc is a number greater than zero. If a single string is passed the package will be set to the string and cbc output names will be created using the model name and .cbc extension, if ipakcbc is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

- **gwt** (*boolean*) - Flag indicating whether GW transport process is active

Notes

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow()
>>> mnw2 = flopy.modflow.ModflowMnw2(m1, ...)
```

check(*f=None, verbose=True, level=1, checktype=None*)

Check mnw2 package data for common errors.

Parameters

- **f** (*str* or *file handle*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If *f* is None, check method does not write results to a summary file. (default is None)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.

Returns

chk

Return type

check object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.mnw2.check()
```

export(*f, **kwargs*)

Export MNW2 data

Parameters

- **f** (*file*) -
- **kwargs** -

Returns

e

Return type

export object

get_allnode_data()

Get a version of the `node_data` array that has all MNW2 nodes listed explicitly. For example, MNWs with open intervals encompassing multiple layers would have a row entry for each layer. `Ztop` and `zbotm` values indicate the top and bottom elevations of the node (these are the same as the layer top and bottom if the node fully penetrates that layer).

Returns

allnode_data - Numpy record array of same form as `node_data`, except each row represents only one node.

Return type

`np.recarray`

static get_default_node_dtype(structured=True)

Get default dtype for node data

Parameters

structured (*bool*) - Boolean indicating if a structured (True) or unstructured (False) model (default is True).

Returns

dtype - node data dtype

Return type

`np.dtype`

static get_default_spd_dtype(structured=True)

Get default dtype for stress period data

Parameters

structured (*bool*) - Boolean indicating if a structured (True) or unstructured (False) model (default is True).

Returns

dtype - node data dtype

Return type

`np.dtype`

static get_empty_node_data(maxnodes=0, aux_names=None, structured=True, default_value=0)

get an empty recarray that corresponds to dtype

Parameters

- **maxnodes** (*int*) - Total number of nodes to be simulated (default is 0)
- **aux_names** (*list*) - List of aux name strings (default is None)
- **structured** (*bool*) - Boolean indicating if a structured (True) or unstructured (False) model (default is True).
- **default_value** (*float*) - Default value for float variables (default is 0).

Returns

r - Recarray of default dtype of shape `maxnode`

Return type

`np.recarray`

```
static get_empty_stress_period_data(itmp=0, aux_names=None, structured=True,
                                   default_value=0)
```

Get an empty stress period data recarray

Parameters

- **itmp** (*int*) - Number of entries in this stress period (default is 0).
- **aux_names** (*list*) - List of aux names (default is None).
- **structured** (*bool*) - Boolean indicating if a structured (True) or unstructured (False) model (default is True).
- **default_value** (*float*) - Default value for float variables (default is 0).

Returns

r - Recarray of default dtype of shape itmp

Return type

np.recarray

```
classmethod load(f, model, nper=None, gwt=False, nsol=1, ext_unit_dict=None)
```

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type flopy.modflow.ModflowMnw2) to which this package will be added.
- **nper** (*int*) - Number of periods
- **gwt** (*bool*) -
- **nsol** (*int*) -
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

```
make_mnw_objects()
```

Make a Mnw object

Return type

None

```
make_node_data(mnwobjs)
```

Make node_data recarray from Mnw objects

Parameters

mnwobjs (*Mnw object*) -

Return type

None

```
make_stress_period_data(mnwobjs)
```

Make stress_period_data recarray from Mnw objects

Parameters

mnwobjs (*Mnw object*) -

Return type

None

```
write_file(filename=None, float_format='{:15.7E}', use_tables=True)
```

Write the package file.

Parameters

- **filename** (*str*) -
- **float_format** -
- **use_tables** -

Return type

None

flopy.modflow.mfmnwi module

```
class ModflowMnwi(model, wellflag=None, qsumflag=None, byndflag=None, mnwobs=1,
                  wellid_unit_qndflag_qhbflag_concflag=None, extension='mnwi',
                  unitnumber=None, filenames=None)
```

Bases: [Package](#)

‘Multi-Node Well Information Package Class’

Parameters

- **model** (*model object*) - The model object (of type [flopy.modflow.mf.Modflow](#)) to which this package will be added.
- **wellflag** (*integer*) - Flag indicating output to be written for each MNW node at the end of each stress period
- **qsumflag** (*integer*) - Flag indicating output to be written for each multi-node well
- **byndflag** (*integer*) - Flag indicating output to be written for each MNW node
- **mnwobs** (*integer*) - Number of multi-node wells for which detailed flow, head, and solute data to be saved
- **wellid_unit_qndflag_qhbflag_concflag** (*list of lists*) - Containing wells and related information to be output (length : [MNWOBS][4or5])
- **extension** (*string*) - Filename extension (default is ‘mnwi’)
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the output names will be created using the model name and output extensions. Default is None.

Notes

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow()
>>> ghb = flopy.modflow.ModflowMnwi(m1, ...)
```

check(*f=None*, *verbose=True*, *level=1*, *checktype=None*)

Check mnwi package data for common errors.

Parameters

- **f** (*str* or *file handle*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If *f* is *None*, check method does not write results to a summary file. (default is *None*)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If *level=0*, summary checks are performed. If *level=1*, full checks are performed.

Return type

None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.mnwi.check()
```

classmethod load(*f*, *model*, *nper=None*, *gwt=False*, *nsol=1*, *ext_unit_dict=None*)

Default load method for standard boundary packages.

write_file()

Write the package file.

Return type

None

flopy.modflow.mfnwt module

mfnwt module. Contains the *ModflowNwt* class. Note that the user can access the *ModflowNwt* class as *flopy.modflow.ModflowNwt*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).


```
class ModflowNwt(model, headtol=0.01, fluxtol=500, maxiterout=100, thickfact=1e-05,
    linmeth=1, iprnwt=0, ibotav=0, options='COMPLEX', Continue=False,
    dbdtheta=0.4, dbdkappa=1e-05, dbdgamma=0.0, momfact=0.1, backflag=1,
    maxbackiter=50, backtol=1.1, backreduce=0.7, maxitinner=50, ilumethod=2,
    levfill=5, stoptol=1e-10, msdr=15, iacl=2, norder=1, level=5, north=7,
    iredsys=0, rrctols=0.0, idroptol=1, epsrn=0.0001, hclosxmd=0.0001,
    mxiterxmd=50, extension='nwt', unitnumber=None, filenames=None)
```

Bases: [Package](#)

MODFLOW Nwt Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.modflow.mf.Modflow](#)) to which this package will be added.
- **headtol** (*float*) - is the maximum head change between outer iterations for solution of the nonlinear problem. (default is 1e-4).
- **fluxtol** (*float*) - is the maximum l2 norm for solution of the nonlinear problem. (default is 500).
- **maxiterout** (*int*) - is the maximum number of iterations to be allowed for solution of the outer (nonlinear) problem. (default is 100).
- **thickfact** (*float*) - is the portion of the cell thickness (length) used for smoothly adjusting storage and conductance coefficients to zero. (default is 1e-5).
- **linmeth** (*int*) - is a flag that determines which matrix solver will be used. A value of 1 indicates GMRES will be used A value of 2 indicates XMD will be used. (default is 1).
- **iprnwt** (*int*) - is a flag that indicates whether additional information about solver convergence will be printed to the main listing file. (default is 0).
- **ibotav** (*int*) - is a flag that indicates whether corrections will be made to groundwater head relative to the cell-bottom altitude if the cell is surrounded by dewatered cells (integer). A value of 1 indicates that a correction will be made and a value of 0 indicates no correction will be made. (default is 0).
- **options** (*string*) - SPECIFIED indicates that the optional solver input values listed for items 1 and 2 will be specified in the NWT input file by the user. SIMPLE indicates that default solver input values will be defined that work well for nearly linear models. This would be used for models that do not include nonlinear stress packages, and models that are either confined or consist of a single unconfined layer that is thick enough to contain the water table within a single layer. MODERATE indicates that default solver input values will be defined that work well for moderately nonlinear models. This would be used for models that include nonlinear stress packages, and models that consist of one or more unconfined layers. The MODERATE option should be used when the SIMPLE option does not result in successful convergence. COMPLEX indicates that default solver input values will be defined that work well for highly nonlinear models. This would be used

for models that include nonlinear stress packages, and models that consist of one or more unconfined layers representing complex geology and sw/gw interaction. The COMPLEX option should be used when the MODERATE option does not result in successful convergence. (default is COMPLEX).

- **Continue** (*bool*) - if the model fails to converge during a time step then it will continue to solve the following time step. (default is False). Note the capital C on this option so that it doesn't conflict with a reserved Python language word.
- **dbdtheta** (*float*) - is a coefficient used to reduce the weight applied to the head change between nonlinear iterations. dbdtheta is used to control oscillations in head. Values range between 0.0 and 1.0, and larger values increase the weight (decrease under-relaxation) applied to the head change. (default is 0.4).
- **dbdkappa** (*float*) - is a coefficient used to increase the weight applied to the head change between nonlinear iterations. dbdkappa is used to control oscillations in head. Values range between 0.0 and 1.0, and larger values increase the weight applied to the head change. (default is 1.e-5).
- **dbdgamma** (*float*) - is a factor (used to weight the head change for the previous and current iteration. Values range between 0.0 and 1.0, and greater values apply more weight to the head change calculated during the current iteration. (default is 0.)
- **momfact** (*float*) - is the momentum coefficient and ranges between 0.0 and 1.0. Greater values apply more weight to the head change for the current iteration. (default is 0.1).
- **backflag** (*int*) - is a flag used to specify whether residual control will be used. A value of 1 indicates that residual control is active and a value of 0 indicates residual control is inactive. (default is 1).
- **maxbackiter** (*int*) - is the maximum number of reductions (backtracks) in the head change between nonlinear iterations (integer). A value between 10 and 50 works well. (default is 50).
- **backtol** (*float*) - is the proportional decrease in the root-mean-squared error of the groundwater-flow equation used to determine if residual control is required at the end of a nonlinear iteration. (default is 1.1).
- **backreduce** (*float*) - is a reduction factor used for residual control that reduces the head change between nonlinear iterations. Values should be between 0.0 and 1.0, where smaller values result in smaller head-change values. (default 0.7).
- **maxitinner** (*int*) - (GMRES) is the maximum number of iterations for the linear solution. (default is 50).
- **ilumethod** (*int*) - (GMRES) is the index for selection of the method for incomplete factorization (ILU) used as a preconditioner. (default is 2).

ilumethod = 1 is ILU with drop tolerance and fill limit. Fill-in terms less than drop tolerance times the diagonal are discarded. The number of fill-in terms in each row of L and U is limited to the fill limit. The fill-limit largest elements are kept in the L and U factors.

ilumethod=2 is ILU(k) order k incomplete LU factorization. Fill-in terms of higher order than k in the factorization are discarded.

- **levfill** (*int*) - (GMRES) is the fill limit for ILUMETHOD = 1 and is the level of fill for ilumethod = 2. Recommended values: 5-10 for method 1, 0-2 for method 2. (default is 5).
- **stoptol** (*float*) - (GMRES) is the tolerance for convergence of the linear solver. This is the residual of the linear equations scaled by the norm of the root mean squared error. Usually 1.e-8 to 1.e-12 works well. (default is 1.e-10).
- **msdr** (*int*) - (GMRES) is the number of iterations between restarts of the GMRES Solver. (default is 15).
- **iacl** (*int*) - (XMD) is a flag for the acceleration method: 0 is conjugate gradient, 1 is ORTHOMIN, 2 is Bi-CGSTAB. (default is 2).
- **norder** (*int*) - (XMD) is a flag for the scheme of ordering the unknowns: 0 is original ordering, 1 is RCM ordering, 2 is Minimum Degree ordering. (default is 1).
- **level** (*int*) - (XMD) is the level of fill for incomplete LU factorization. (default is 5).
- **north** (*int*) - (XMD) is the number of orthogonalization for the ORTHOMIN acceleration scheme. A number between 4 and 10 is appropriate. Small values require less storage but more iterations may be required. This number should equal 2 for the other acceleration methods. (default is 7).
- **iredsys** (*int*) - (XMD) is a flag for reduced system preconditioning (integer): 0-do not apply reduced system preconditioning, 1-apply reduced system preconditioning. (default is 0)
- **rrctols** (*int*) - (XMD) is the residual reduction-convergence criteria. (default is 0.).
- **idroptol** (*int*) - (XMD) is a flag for using drop tolerance in the preconditioning: 0-don't use drop tolerance, 1-use drop tolerance. (default is 1).
- **epsrn** (*float*) - (XMD) is the drop tolerance for preconditioning. (default is 1.e-4).
- **hclosexmd** (*float*) - (XMD) is the head closure criteria for inner (linear) iterations. (default is 1.e-4).
- **mxiterxmd** (*int*) - (XMD) is the maximum number of iterations for the linear solution. (default is 50).
- **extension** (*list string*) - Filename extension (default is 'nwt')
- **unitnumber** (*int*) - File unit number (default is None).

- **filenames** (*str* or *list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> nwt = flopy.modflow.ModflowNwt(m)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

Returns

nwt

Return type

ModflowNwt object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> nwt = flopy.modflow.ModflowPcg.load('test.nwt', m)
```

write_file()

Write the package file.

Return type

None

flopy.modflow.mfoc module

mfoc module. Contains the ModflowOc class. Note that the user can access the ModflowOc class as `flopy.modflow.ModflowOc`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowOc(model, ihedfm=0, iddnfm=0, chedfm=None, cddnfm=None, cboufm=None,
                compact=True, stress_period_data={(0, 0): ['save head']}, extension=['oc',
                'hds', 'ddn', 'cbc', 'ibo'], unitnumber=None, filenames=None, label='LABEL',
                **kwargs)
```

Bases: [Package](#)

MODFLOW Output Control Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.modflow.mf.Modflow](#)) to which this package will be added.
- **ihedfm** (*int*) - is a code for the format in which heads will be printed. (default is 0).
- **iddnfm** (*int*) - is a code for the format in which drawdown will be printed. (default is 0).
- **chedfm** (*string*) - is a character value that specifies the format for saving heads. The format must contain 20 characters or less and must be a valid Fortran format that is enclosed in parentheses. The format must be enclosed in apostrophes if it contains one or more blanks or commas. The optional word LABEL after the format is used to indicate that each layer of output should be preceded with a line that defines the output (simulation time, the layer being output, and so forth). If there is no record specifying CHEDFM, then heads are written to a binary (unformatted) file. Binary files are usually more compact than text files, but they are not generally transportable among different computer operating systems or different Fortran compilers. (default is None)
- **cddnfm** (*string*) - is a character value that specifies the format for saving drawdown. The format must contain 20 characters or less and must be a valid Fortran format that is enclosed in parentheses. The format must be enclosed in apostrophes if it contains one or more blanks or commas. The optional word LABEL after the format is used to indicate that each layer of output should be preceded with a line that defines the output (simulation time, the layer being output, and so forth). If there is no record specifying CDDNFM, then drawdowns are written to a binary (unformatted) file. Binary files are usually more compact than text files, but they are not generally transportable among different computer operating systems or different Fortran compilers. (default is None)
- **cboufm** (*string*) - is a character value that specifies the format for saving ibound. The format must contain 20 characters or less and must be a valid Fortran format that is enclosed in parentheses. The format must be enclosed in apostrophes if

it contains one or more blanks or commas. The optional word LABEL after the format is used to indicate that each layer of output should be preceded with a line that defines the output (simulation time, the layer being output, and so forth). If there is no record specifying CBOUFM, then ibounds are written to a binary (unformatted) file. Binary files are usually more compact than text files, but they are not generally transportable among different computer operating systems or different Fortran compilers. (default is None)

- **stress_period_data** (*dictionary of lists*) - Dictionary key is a tuple with the zero-based period and step (IPEROC, ITSOC) for each print/save option list. If stress_period_data is None, then heads are saved for the last time step of each stress period. (default is None)

The list can have any valid MODFLOW OC print/save option:

```
PRINT HEAD PRINT DRAWDOWN PRINT BUDGET SAVE HEAD SAVE DRAWDOWN
SAVE BUDGET SAVE IBOUND
```

The lists can also include (1) DDREFERENCE in the list to reset drawdown reference to the period and step and (2) a list of layers for PRINT HEAD, SAVE HEAD, PRINT DRAWDOWN, SAVE DRAWDOWN, and SAVE IBOUND.

stress_period_data = {(0,1):['save head']} would save the head for the second timestep in the first stress period.

- **compact** (*boolean*) - Save results in compact budget form. (default is True).
- **extension** (*list of strings*) - (default is ['oc', 'hds', 'ddn', 'cbc', 'ibo']).
- **unitnumber** (*list of ints*) - (default is [14, 51, 52, 53, 0]).
- **filenames** (*str or list of str*) - Filenames to use for the package and the head, drawdown, budget (not used), and ibound output files. If filenames=None the package name will be created using the model name and package extension and the output file names will be created using the model name and extensions. If a single string is passed the package will be set to the string and output names will be created using the model name and head, drawdown, budget, and ibound extensions. To define the names for all package files (input and output) the length of the list of strings should be 5. Default is None.

Notes

The “words” method for specifying output control is the only option available. Also, the “compact” budget should normally be used as it produces files that are typically much smaller. The compact budget form is also a requirement for using the MODPATH particle tracking program.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> spd = {(0, 0): ['print head'],
... (0, 1): [],
... (0, 249): ['print head'],
... (0, 250): [],
... (0, 499): ['print head', 'save ibound'],
... (0, 500): [],
... (0, 749): ['print head', 'ddreference'],
... (0, 750): [],
... (0, 999): ['print head']}
>>> oc = flopy.modflow.ModflowOc(m, stress_period_data=spd, cboufm='(20i5)')
```

check(*f=None*, *verbose=True*, *level=1*, *checktype=None*)

Check package data for common errors.

Parameters

- **f** (*str* or *file handle*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If *f* is None, check method does not write results to a summary file. (default is None)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen.
- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.

Return type

None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.oc.check()
```

get_budgetunit()

Get the budget file unit number(s).

Parameters

None -

Returns

ibud - Unit number or list of cell-by-cell budget output unit numbers. None is returned if ipakcb is less than one for all packages.

Return type

integer or list of integers

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> dis = flopy.modflow.ModflowDis(m)
>>> bas = flopy.modflow.ModflowBas(m)
>>> lpf = flopy.modflow.ModflowLpf(m, ipakcb=100)
>>> wel_data = {0: [[0, 0, 0, -1000.]]}
>>> wel = flopy.modflow.ModflowWel(m, ipakcb=101,
... stress_period_data=wel_data)
>>> spd = {(0, 0): ['save head', 'save budget']}
>>> oc = flopy.modflow.ModflowOc(m, stress_period_data=spd)
>>> oc.get_budgetunit()
[100, 101]
```

static `get_ocoutput_units(f, ext_unit_dict=None)`

Get head and drawdown units from a OC file.

Parameters

- **f** (*filename or file handle*) - File to load.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

- **ihedun** (*integer*) - Unit number of the head file.
- **fhead** (*str*) - File name of the head file. Is only defined if *ext_unit_dict* is passed and the unit number is a valid key. , headfilename, oc : ModflowOc object ModflowOc object.
- **iddn** (*integer*) - Unit number of the drawdown file.
- **fddn** (*str*) - File name of the drawdown file. Is only defined if *ext_unit_dict* is passed and the unit number is a valid key.

Examples

```
>>> import flopy
>>> ihds, hf, iddn, df = flopy.modflow.ModflowOc.get_ocoutput_units('test.oc')
```

classmethod `load(f, model, nper=None, nstp=None, nlay=None, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.

- **nper** (*int*) - The number of stress periods. If nper is None, then nper will be obtained from the model object. (default is None).
- **nstp** (*int or list of ints*) - Integer or list of integers containing the number of time steps in each stress period. If nstp is None, then nstp will be obtained from the DIS or DISU packages attached to the model object. The length of nstp must be equal to nper. (default is None).
- **nlay** (*int*) - The number of model layers. If nlay is None, then nlay will be obtained from the model object. nlay only needs to be specified if an empty model object is passed in and the oc file being loaded is defined using numeric codes. (default is None).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

oc - ModflowOc object.

Return type

ModflowOc object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> oc = flopy.modflow.ModflowOc.load('test.oc', m)
```

reset_budgetunit(budgetunit=None, fname=None)

Reset the cell-by-cell budget unit (ipakcb) for every package that can write cell-by-cell data when SAVE BUDGET is specified in the OC file to the specified budgetunit.

Parameters

- **budgetunit** (*int, optional*) - Unit number for cell-by-cell output data. If budgetunit is None then the next available external unit number is assigned. Default is None
- **fname** (*string, optional*) - Filename to use for cell-by-cell output file. If fname=None the cell-by-cell output file will be created using the model name and a '.cbc' file extension. Default is None.

Return type

None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> dis = flopy.modflow.ModflowDis(m)
>>> bas = flopy.modflow.ModflowBas(m)
>>> lpf = flopy.modflow.ModflowLpf(m, ipakcb=100)
>>> wel_data = {0: [[0, 0, 0, -1000.]]}
>>> wel = flopy.modflow.ModflowWel(m, ipakcb=101,
... stress_period_data=wel_data)
>>> spd = {(0, 0): ['save head', 'save budget']}
>>> oc = flopy.modflow.ModflowOc(m, stress_period_data=spd)
>>> oc.reset_budgetunit(budgetunit=1053, fname='test.cbc')
```

`write_file()`

Write the package file.

Return type

None

`flopy.modflow.mfpar module`

mfpar module. Contains the `ModflowPar` class. Note that the user can access the `ModflowPar` class as `flopy.modflow.ModflowPar`.

`class ModflowPar`

Bases: `object`

Class for loading mult, zone, pval, and parameter data for MODFLOW packages that use array data (LPF, UPW, RCH, EVT). Class also includes methods to create data arrays using mult, zone, pval, and parameter data (not used for boundary conditions).

Notes

Parameters are supported in Flopy only when reading in existing models. Parameter values are converted to native values in Flopy and the connection to “parameters” is thus nonexistent.

`static load(f, npar, verbose=False)`

Load property parameters from an existing package.

Parameters

- `f` (*file handle*) -
- `npar` (*int*) - The number of parameters.
- `verbose` (*bool*) - Boolean flag to control output. (default is `False`)

Returns

- `list` (*list object of unique par_types in file f*)
- `dictionary` (*dictionary object with parameters in file f*)

Examples

```
>>>par_types, parm_dict = flopy.modflow.mfpar.ModflowPar.load(f, np)
static parameter_fill(model, shape, findkey, parm_dict, findlayer=None)
Fill an array with parameters using zone, mult, and pval data.
```

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **shape** (*tuple*) - The shape of the returned data array. Typically shape is (nrow, ncol)
- **findkey** (*string*) - the parameter array to be constructed,
- **parm_dict** (*dict*) - dictionary that includes all of the parameter data for a package
- **findlayer** (*int*) - Layer that will be filled. Not required for array boundary condition data.

Returns

data - Filled array resulting from applications of zone, mult, pval, and parameter data.

Return type

numpy array

Examples

for lpf and upw:

```
>>> data = flopy.modflow.mfpar.ModflowPar.parameter_fill(m, (nrow, ncol), 'vkcb
→ ',
>>> .....parm_dict,
→ findlayer=1)
```

```
set_mult(model, ext_unit_dict)
```

Load an existing mult package and set mult data for a model.

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

Examples

```
>>> ml.mfpar.set_mult(ml, ext_unit_dict)
```

set_pval(*model*, *ext_unit_dict*)

Load an existing pval package and set pval data for a model.

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

Examples

```
>>> ml.mfpar.set_pval(ml, ext_unit_dict)
```

set_zone(*model*, *ext_unit_dict*)

Load an existing zone package and set zone data for a model.

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

Examples

```
>>> ml.mfpar.set_zone(ml, ext_unit_dict)
```

flopy.modflow.mfparbc module

mfparbc module. Contains the ModflowParBc class. Note that the user can access the ModflowParBc class as *flopy.modflow.ModflowParBc*.

class ModflowParBc(*bc_parms*)

Bases: *object*

Class for loading boundary condition parameter data for MODFLOW packages that use list data (WEL, GHB, DRN, etc.). This Class is also used to create hfb6 data from hfb parameters. Class also includes methods to create data arrays using pval and boundary condition parameter data.

Notes

Parameters are supported in FloPy only when reading in existing models. Parameter values are converted to native values in FloPy and the connection to “parameters” is thus nonexistent.

get(*fkey*)

overload get to return a value from the bc_parms dictionary

classmethod load(*f*, *npar*, *dt*, *model*, *ext_unit_dict*=None, *verbose*=False)

Load bc property parameters from an existing bc package that uses list data (e.g. WEL, RIV, etc.).

Parameters

- **f** (*file handle*) -
- **npar** (*int*) - The number of parameters.
- **dt** (*numpy.dtype*) - numpy.dtype for the particular list boundary condition.
- **verbose** (*bool*) - Boolean flag to control output. (default is False)

Returns

dictionary

Return type

dictionary object with parameters in file f

Examples

static loadarray(*f*, *npar*, *verbose*=False)

Load bc property parameters from an existing bc package that uses array data (e.g. RCH, EVT).

Parameters

- **f** (*file handle*) -
- **npar** (*int*) - The number of parameters.
- **verbose** (*bool*) - Boolean flag to control output. (default is False)

Returns

dictionary

Return type

dictionary object with parameters in file f

Examples

```
static parameter_bcfill(model, shape, parm_dict, pak_parms)
```

Fill an array with parameters using zone, mult, and pval data.

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **shape** (*tuple*) - The shape of the returned data array. Typically shape is (nrow, ncol)
- **parm_dict** (*list*) - dictionary of parameter instances
- **pak_parms** (*dict*) - dictionary that includes all of the parameter data for a package

Returns

data - Filled array resulting from applications of zone, mult, pval, and parameter data.

Return type

numpy array

Examples

```
for rch and evt >>> data = flopy.modflow.mfparbc.ModflowParBc.parameter_bcfill(m,
(nrow, ncol), >>> ..... 'rech', parm_dict, pak_parms)
```

flopy.modflow.mfpbc module

```
class ModflowPbc(model, layer_row_column_data=None, layer_row_column_shead_thead=None,
cosines=None, extension='pbc', unitnumber=None, zerobase=True)
```

Bases: *Package*

Periodic boundary condition class

```
write_file()
```

Write the package file.

Return type

None

flopy.modflow.mfpcg module

mfpcg module. Contains the ModflowPcg class. Note that the user can access the ModflowPcg class as *flopy.modflow.ModflowPcg*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowPcg(model, mxiter=50, iter1=30, npcond=1, hclose=1e-05, rclose=1e-05,
relax=1.0, nbpol=0, iprpcg=0, mutpcg=3, damp=1.0, damp1=1.0,
ihcofadd=0, extension='pcg', unitnumber=None, filenames=None)
```

Bases: [Package](#)

MODFLOW Pcg Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.modflow.mf.Modflow](#)) to which this package will be added.
- **mxiter** (*int*) - maximum number of outer iterations. (default is 50)
- **iter1** (*int*) - maximum number of inner iterations. (default is 30)
- **npcond** (*int*) - flag used to select the matrix conditioning method. (default is 1). specify npcond = 1 for Modified Incomplete Cholesky. specify npcond = 2 for Polynomial.
- **hclose** (*float*) - is the head change criterion for convergence. (default is 1e-5).
- **rclose** (*float*) - is the residual criterion for convergence. (default is 1e-5)
- **relax** (*float*) - is the relaxation parameter used with npcond = 1. (default is 1.0)
- **nbpol** (*int*) - is only used when npcond = 2 to indicate whether the estimate of the upper bound on the maximum eigenvalue is 2.0, or whether the estimate will be calculated. nbpol = 2 is used to specify the value is 2.0; for any other value of nbpol, the estimate is calculated. Convergence is generally insensitive to this parameter. (default is 0).
- **iprpcg** (*int*) - solver print out interval. (default is 0).
- **mutpcg** (*int*) - If mutpcg = 0, tables of maximum head change and residual will be printed each iteration. If mutpcg = 1, only the total number of iterations will be printed. If mutpcg = 2, no information will be printed. If mutpcg = 3, information will only be printed if convergence fails. (default is 3).
- **damp** (*float*) - is the steady-state damping factor. (default is 1.)
- **damp1** (*float*) - is the transient damping factor. (default is 1.)
- **ihcofadd** (*int*) - is a flag that determines what happens to an active cell that is surrounded by dry cells. (default is 0). If ihcofadd=0, cell converts to dry regardless of HCOF value. This is the default, which is the way PCG2 worked prior to the addition of this option. If ihcofadd<>0, cell converts to dry only if HCOF has no head-dependent stresses or storage terms.
- **extension** (*list string*) - Filename extension (default is 'pcg')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> pcg = flopy.modflow.ModflowPcg(m)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

`pcg`

Return type

`ModflowPcg` object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> pcg = flopy.modflow.ModflowPcg.load('test.pcg', m)
```

write_file()

Write the package file.

Return type

`None`

`flopy.modflow.mfpcgn` module

`mfpcgn` module. Contains the `ModflowPcgn` class. Note that the user can access the `ModflowStr` class as `flopy.modflow.ModflowPcgn`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowPcgn(model, iter_mo=50, iter_mi=30, close_r=1e-05, close_h=1e-05,
    relax=1.0, ifill=0, unit_pc=None, unit_ts=None, adamp=0, damp=1.0,
    damp_lb=0.001, rate_d=0.1, chglimit=0.0, acnvg=0, cnvg_lb=0.001,
    mcnvg=2, rate_c=-1.0, ipunit=None, extension='pcgn', unitnumber=None,
    filenames=None)
```


Bases: [Package](#)

MODFLOW Pcg Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.modflow.mf.Modflow](#)) to which this package will be added.
- **iter_mo** (*int*) - The maximum number of picard (outer) iterations allowed. For nonlinear problems, this variable must be set to some number greater than one, depending on the problem size and degree of nonlinearity. If iter_mo is set to 1, then the pcgn solver assumes that the problem is linear and the input requirements are greatly truncated. (default is 50)
- **iter_mi** (*int*) - maximum number of pcg (inner) iterations allowed. Generally, this variable is set to some number greater than one, depending on the matrix size, degree of convergence called for, and the nature of the problem. For a nonlinear problem, iter_mi should be set large enough that the pcg iteration converges freely with the relative convergence parameter epsilon described in the Parameters Related to Convergence of Inner Iteration: Line 4 subsection. (default is 30)
- **close_r** (*float*) - The residual-based stopping criterion for iteration. This parameter is used differently, depending on whether it is applied to a linear or nonlinear problem.

If iter_mo = 1: For a linear problem, the variant of the conjugate gradient method outlined in algorithm 2 is employed, but uses the absolute convergence criterion in place of the relative convergence criterion. close_r is used as the value in the absolute convergence criterion for quitting the pcg iterative solver. close_r is compared to the square root of the weighted residual norm. In particular, if the square root of the weighted residual norm is less than close_r, then the linear Pcg iterative solve is said to have converged, causing the pcg iteration to cease and control of the program to pass out of the pcg solver.

If iter_mo > 1: For a nonlinear problem, close_r is used as a criterion for quitting the picard (outer) iteration. close_r is compared to the square root of the inner product of the residuals (the residual norm) as calculated on entry to the pcg solver at the beginning of every picard iteration. if this norm is less than close_r, then the picard iteration is considered to have converged.

- **close_h** (*float*) - close_h is used as an alternate stopping criterion for the picard iteration needed to solve a nonlinear problem. The maximum value of the head change is obtained for each picard iteration, after completion of the inner, pcg iteration. If this maximum head change is less than close_h, then the picard iteration is considered tentatively to have converged. However, as nonlinear problems can demonstrate oscillation in the head solution, the picard iteration is not declared to have converged unless the maximum head change is less than close_h for three picard iterations. If these picard iterations are

- sequential, then a good solution is assumed to have been obtained. If the picard iterations are not sequential, then a warning is issued advising that the convergence is conditional and the user is urged to examine the mass balance of the solution.
- **relax** (*float*) - is the relaxation parameter used with `npcond = 1`. (default is 1.0)
 - **ifill** (*int*) - is the fill level of the mic preconditioner. Preconditioners with fill levels of 0 and 1 are available (`ifill = 0` and `ifill = 1`, respectively). (default is 0)
 - **unit_pc** (*int*) - is the unit number of an optional output file where progress for the inner PCG iteration can be written. (default is 0)
 - **unit_ts** (*int*) - is the unit number of an optional output file where the actual time in the PCG solver is accumulated. (default is 0)
 - **adamp** (*int*) - defines the mode of damping applied to the linear solution. In general, damping determines how much of the head changes vector shall be applied to the hydraulic head vector `hj` in picard iteration `j`. If `adamp = 0`, Ordinary damping is employed and a constant value of damping parameter will be used throughout the picard iteration; this option requires a valid value for `damp`. If `adamp = 1`, Adaptive damping is employed. If `adamp = 2`: Enhanced damping algorithm in which the damping value is increased (but never decreased) provided the picard iteration is proceeding satisfactorily. (default is 0)
 - **damp** (*float*) - is the damping factor. (default is 1.)
 - **damp_lb** (*float*) - is the lower bound placed on the dampening; generally, $0 < \text{damp_lb} < \text{damp}$. (default is 0.001)
 - **rate_d** (*float*) - is a rate parameter; generally, $0 < \text{rate_d} < 1$. (default is 0.1)
 - **chglimit** (*float*) - this variable limits the maximum head change applicable to the updated hydraulic heads in a Picard iteration. If `chglimit = 0.0`, then adaptive damping proceeds without this feature. (default is 0.)
 - **acnvg** (*int*) - defines the mode of convergence applied to the PCG solver. (default is 0)
 - **cnvg_lb** (*int*) - is the minimum value that the relative convergence is allowed to take under the self-adjusting convergence option. `cnvg_lb` is used only in convergence mode `acnvg = 1`. (default is 0.001)
 - **mcnvg** (*float*) - increases the relative PCG convergence criteria by a power equal to `MCNVG`. `MCNVG` is used only in convergence mode `acnvg = 2`. (default is 2)
 - **rate_c** (*float*) - this option results in variable enhancement of epsilon. If $0 < \text{rate_c} < 1$, then enhanced relative convergence is allowed to decrease by increasing `epsilon(j) = epsilon(j-1) + rate_c * epsilon(j-1)`, where `j` is the Picarditeration number;

this change in epsilon occurs so long as the Picard iteration is progressing satisfactorily. If `rate_c` ≤ 0 , then the value of epsilon set by `mcnvg` remains unchanged through the picard iteration. It should be emphasized that `rate_c` must have a value greater than 0 for the variable enhancement to be effected; otherwise epsilon remains constant. `rate_c` is used only in convergence mode `acnvg` = 2. (default is -1.)

- **ipunit** (*int*) - enables progress reporting for the picard iteration. If `ipunit` ≥ 0 , then a record of progress made by the picard iteration for each time step is printed in the MODFLOW Listing file (Harbaugh and others, 2000). This record consists of the total number of dry cells at the end of each time step as well as the total number of PCG iterations necessary to obtain convergence. In addition, if `ipunit` > 0 , then extensive diagnostics for each Picard iteration is also written in comma-separated format to a file whose unit number corresponds to `ipunit`; the name for this file, along with its unit number and type 'data' should be entered in the modflow Name file. If `ipunit` < 0 then printing of all progress concerning the Picard iteration is suppressed, as well as information on the nature of the convergence of the picard iteration. (default is 0)
- **extension** (*list string*) - Filename extension (default is 'pcgn')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If `filenames=None` the package name will be created using the model name and package extension and the pcgn output names will be created using the model name and .pcgni, .pcgnt, and .pcgno extensions. If a single string is passed the package will be set to the string and pcgn output names will be created using the model name and pcgn output extensions. To define the names for all package files (input and output) the length of the list of strings should be 4. Default is None.

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> pcgn = flopy.modflow.ModflowPcgn(m)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array

control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

pcgn

Return type

ModflowPcgn object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> pcgn = flopy.modflow.ModflowPcgn.load('test.pcgn', m)
```

write_file()

Write the package file.

Return type

None

flopy.modflow.mfpks module

mfpks module. Contains the ModflowPks class. Note that the user can access the ModflowPks class as `flopy.modflow.ModflowPks`.

```
class ModflowPks(model, mxiter=100, innerit=50, isolver=1, npc=2, iscl=0, iord=0,
                 ncoresm=1, ncoresv=1, damp=1.0, damp=1.0, relax=0.97, ifill=0,
                 droptol=0.0, hclose=0.001, rclose=0.1, l2norm=None, iprpk=0, mutpk=3,
                 mpi=False, partopt=0, novlapimpsol=1, stenimpsol=2, verbose=0,
                 partdata=None, extension='pks', unitnumber=None, filenames=None)
```

Bases: [Package](#)

MODFLOW Pks Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **mxiter** (*int*) - maximum number of outer iterations. (default is 100)
- **innerit** (*int*) - maximum number of inner iterations. (default is 30)
- **hclose** (*float*) - is the head change criterion for convergence. (default is 1.e-3).
- **rclose** (*float*) - is the residual criterion for convergence. (default is 1.e-1)
- **relax** (*float*) - is the relaxation parameter used with npcond = 1. (default is 1.0)
- . -
- . -

- `.` -
- **iprpks** (*int*) - solver print out interval. (default is 0).
- **mutpks** (*int*) -
 If **mutpcg** = 0, tables of maximum head change and residual will be printed each iteration.
 If **mutpcg** = 1, only the total number of iterations will be printed. If **mutpcg** = 2, no information will be printed. If **mutpcg** = 3, information will only be printed if convergence fails.
 (default is 3).
- **damp** (*float*) - is the steady-state damping factor. (default is 1.)
- **damp_t** (*float*) - is the transient damping factor. (default is 1.)
- **extension** (*list string*) - Filename extension (default is 'pks')
- **unitnumber** (*int*) - File unit number (default is 27).
- **filenames** (*str or list of str*) - Filenames to use for the package. If **filenames**=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> pks = flopy.modflow.ModflowPks(m)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case **ext_unit_dict** is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

Returns

pks

Return type

ModflowPks object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> pks = flopy.modflow.ModflowPks.load('test.pks', m)
```

`write_file()`

Write the package file.

Return type

None

`flopy.modflow.mfpval` module

mfpval module. Contains the `ModflowPval` class. Note that the user can access the `ModflowPval` class as `flopy.modflow.ModflowPval`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowPval(model, pval_dict=None, extension='pval', unitnumber=None,
                  filenames=None)
```

Bases: [Package](#)

MODFLOW Mult Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **pval_dict** (*dict*) - Dictionary with pval data for the model. `pval_dict` is typically instantiated using load method.
- **extension** (*string*) - Filename extension (default is 'pval')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If `filenames=None` the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Parameters are supported in Flopy only when reading in existing models. Parameter values are converted to native values in Flopy and the connection to “parameters” is thus nonexistent.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> pval_dict = flopy.modflow.ModflowZon(m, pval_dict=pval_dict)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

pval

Return type

ModflowPval dict

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> mlt = flopy.modflow.ModflowPval.load('test.pval', m)
```

write_file()

Write the package file.

Return type

None

Notes

Not implemented because parameters are only supported on load

flopy.modflow.mfrch module

mfrch module. Contains the ModflowRch class. Note that the user can access the ModflowRch class as `flopy.modflow.ModflowRch`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowRch(model, nrchop=3, ipakcb=None, rech=0.001, irch=0, extension='rch',
                 unitnumber=None, filenames=None)
```

Bases: [Package](#)

MODFLOW Recharge Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.modflow.mf.Modflow](#)) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0).
- **nrchop** (*int*) - is the recharge option code. 1: Recharge to top grid layer only 2: Recharge to layer defined in irch 3: Recharge to highest active cell (default is 3).
- **rech** (*float or filename or ndarray or dict keyed on kper (zero-based)*) - Recharge flux (default is 1.e-3, which is used for all stress periods)
- **irch** (*int or filename or ndarray or dict keyed on kper (zero-based)*) - Layer (for an unstructured grid) or node (for an unstructured grid) to which recharge is applied in each vertical column (only used when nrchop=2). Default is 0, which is used for all stress periods.
- **extension** (*string*) - Filename extension (default is 'rch')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output name will be created using the model name and .cbc extension (for example, modflowtest.cbc), if ipakcbc is a number greater than zero. If a single string is passed the package will be set to the string and cbc output names will be created using the model name and .cbc extension, if ipakcbc is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

Parameters are supported in Flopy only when reading in existing models. Parameter values are converted to native values in Flopy and the connection to “parameters” is thus nonexistent.

Examples

```
>>> #steady state
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> rch = flopy.modflow.ModflowRch(m, nrchop=3, rech=1.2e-4)
```

```
>>> #transient with time-varying recharge
>>> import flopy
>>> rech = {}
>>> rech[0] = 1.2e-4 #stress period 1 to 4
>>> rech[4] = 0.0 #stress period 5 and 6
>>> rech[6] = 1.2e-3 #stress period 7 to the end
>>> m = flopy.modflow.Modflow()
>>> rch = flopy.modflow.ModflowRch(m, nrchop=3, rech=rech)
```

```
check(f=None, verbose=True, level=1, RTmin=2e-08, RTmax=0.0002, checktype=None)
```

Check package data for common errors.

Parameters

- **f** (*str* or *file handle*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If *f* is None, check method does not write results to a summary file. (default is None)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.
- **RTmin** (*float*) - Minimum product of recharge and transmissivity. Default is 2e-8
- **RTmax** (*float*) - Maximum product of recharge and transmissivity. Default is 2e-4

Return type

None

Notes

Unstructured models not checked for extreme recharge transmissivity ratios.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.rch.check()
```

classmethod `load(f, model, nper=None, ext_unit_dict=None, check=True)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **nper** (*int*) - The number of stress periods. If `nper` is `None`, then `nper` will be obtained from the model object. (default is `None`).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using `EXTERNAL`, or older style array control records, then `f` should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.
- **check** (*boolean*) - Check package data for common errors. (default `True`)

Returns

rch - `ModflowRch` object.

Return type

`ModflowRch` object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> rch = flopy.modflow.ModflowRch.load('test.rch', m)
```

write_file(`check=True, f=None`)

Write the package file.

Parameters

- **check** (*boolean*) - Check package data for common errors. (default `True`)

Return type

`None`

flopy.modflow.mfriv module

mfriv module. Contains the ModflowRiv class. Note that the user can access the ModflowRiv class as `flopy.modflow.ModflowRiv`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowRiv(model, ipakcb=None, stress_period_data=None, dtype=None, extension='riv',
                 options=None, unitnumber=None, filenames=None, **kwargs)
```

Bases: [Package](#)

MODFLOW River Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0).
- **stress_period_data** (*list of boundaries, or recarray of boundaries, or*) - dictionary of boundaries. Each river cell is defined through definition of layer (int), row (int), column (int), stage (float), cond (float), rbot (float). The simplest form is a dictionary with a lists of boundaries for each stress period, where each list of boundaries itself is a list of boundaries. Indices of the dictionary are the numbers of the stress period. This gives the form of:

```
stress_period_data =
{0: [
    [lay, row, col, stage, cond, rbot],
    [lay, row, col, stage, cond, rbot],
    [lay, row, col, stage, cond, rbot]
],
1: [
    [lay, row, col, stage, cond, rbot],
    [lay, row, col, stage, cond, rbot],
    [lay, row, col, stage, cond, rbot]
], ...
kper:
[
    [lay, row, col, stage, cond, rbot],
    [lay, row, col, stage, cond, rbot],
    [lay, row, col, stage, cond, rbot]
]
}
```

Note that if the number of lists is smaller than the number of stress periods, then the last list of rivers will apply until the end of the simulation. Full details of all options to specify stress_period_data can be found in the flopy3 boundaries Notebook in the basic subdirectory of the examples directory.

- **dtype** (*custom datatype of stress_period_data.*) - (default is None) If None the default river datatype will be applied.
- **naux** (*int*) - number of auxiliary variables
- **extension** (*string*) - Filename extension (default is 'riv')
- **options** (*list of strings*) - Package options. (default is None).
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output name will be created using the model name and .cbc extension (for example, modflowtest.cbc), if ipakcbc is a number greater than zero. If a single string is passed the package will be set to the string and cbc output names will be created using the model name and .cbc extension, if ipakcbc is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

mxactr

Maximum number of river cells for a stress period. This is calculated automatically by FloPy based on the information in layer_row_column_data.

Type

int

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lrcd = {}
>>> lrcd[0] = [[2, 3, 4, 15.6, 1050., -4]] #this river boundary will be
>>> #applied to all stress periods
>>> riv = flopy.modflow.ModflowRiv(m, stress_period_data=lrcd)
```

add_record(*kper, index, values*)

check(*f=None, verbose=True, level=1, checktype=None*)

Check package data for common errors.

Parameters

- **f** (*str or file handle*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If f is None, check method does not write results to a summary file. (default is None)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen.

- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.

Return type

None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.riv.check()
```

```
static get_default_dtype(structured=True)
```

```
static get_empty(ncells=0, aux_names=None, structured=True)
```

```
classmethod load(f, model, nper=None, ext_unit_dict=None, check=True)
```

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **nper** (*int*) - The number of stress periods. If nper is None, then nper will be obtained from the model object. (default is None).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.
- **check** (*boolean*) - Check package data for common errors. (default True)

Returns**rch** - ModflowRiv object.**Return type**

ModflowRiv object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> riv = flopy.modflow.ModflowRiv.load('test.riv', m)
```

```
write_file(check=True)
```

Write the package file.

Parameters

check (*boolean*) - Check package data for common errors. (default True)

Return type

None

flopy.modflow.mfsfr2 module

```
class ModflowSfr2(model, nstrm=-2, nss=1, nsfrpar=0, nparseg=0, const=None,
                  dleak=0.0001, ipakcb=None, istcb2=None, isfropt=0, nstrail=10, isuzn=1,
                  nsfrsets=30, irtflg=0, numtim=2, weight=0.75, flwtol=0.0001,
                  reach_data=None, segment_data=None, channel_geometry_data=None,
                  channel_flow_data=None, dataset_5=None, irdflag=0, iptflag=0,
                  reachinput=False, transroute=False, tabfiles=False, tabfiles_dict=None,
                  extension='sfr', unit_number=None, filenames=None, options=None)
```

Bases: [Package](#)

Streamflow-Routing (SFR2) Package Class

Parameters

- **model** (*model object*) - The model object (of type :class:'flopy.modflow.mf.Modflow') to which this package will be added.
- **nstrm** (*integer*) - An integer value that can be specified to be positive or negative. The absolute value of NSTRM is equal to the number of stream reaches (finite-difference cells) that are active during the simulation and the number of lines of data to be included in Item 2, described below. When NSTRM is specified to be a negative integer, it is also used as a flag for changing the format of the data input, for simulating unsaturated flow beneath streams, and (or) for simulating transient streamflow routing (for MODFLOW-2005 simulations only), depending on the values specified for variables ISFROPT and IRTFLG, as described below. When NSTRM is negative, NSFRPAR must be set to zero, which means that parameters cannot be specified. By default, nstrm is set to negative.
- **nss** (*integer*) - An integer value equal to the number of stream segments (consisting of one or more reaches) that are used to define the complete stream network. The value of NSS represents the number of segments that must be defined through a combination of parameters and variables in Item 4 or variables in Item 6.
- **nparseg** (*integer*) - An integer value equal to (or exceeding) the number of stream-segment definitions associated with all parameters. This number can be more than the total number of segments (NSS) in the stream network because the same segment can be defined in multiple parameters, and because parameters can be time-varying. NPARSEG must equal or exceed the sum of NLST x N for all parameters, where N is the greater of 1 and NUMINST; that is, NPARSEG must equal or exceed the total number of repetitions of item 4b. This variable must be zero when NSTRM is negative.

- **const** (*float*) - A real value (or conversion factor) used in calculating stream depth for stream reach. If stream depth is not calculated using Manning's equation for any stream segment (that is, ICALC does not equal 1 or 2), then a value of zero can be entered. If Manning's equation is used, a constant of 1.486 is used for flow units of cubic feet per second, and a constant of 1.0 is used for units of cubic meters per second. The constant must be multiplied by 86,400 when using time units of days in the simulation. An explanation of time units used in MODFLOW is given by Harbaugh and others (2000, p. 10).
- **dleak** (*float*) - A real value equal to the tolerance level of stream depth used in computing leakage between each stream reach and active model cell. Value is in units of length. Usually a value of 0.0001 is sufficient when units of feet or meters are used in model.
- **ipakcb** (*integer*) - An integer value used as a flag for writing stream-aquifer leakage values. If ipakcb > 0, unformatted leakage between each stream reach and corresponding model cell will be saved to the main cell-by-cell budget file whenever when a cell-by-cell budget has been specified in Output Control (see Harbaugh and others, 2000, pages 52-55). If ipakcb = 0, leakage values will not be printed or saved. Printing to the listing file (ipakcb < 0) is not supported.
- **istcb2** (*integer*) - An integer value used as a flag for writing to a separate formatted file all information on inflows and outflows from each reach; on stream depth, width, and streambed conductance; and on head difference and gradient across the streambed. If ISTCB2 > 0, then ISTCB2 also represents the unit number to which all information for each stream reach will be saved to a separate file when a cell-by-cell budget has been specified in Output Control. If ISTCB2 < 0, it is the unit number to which unformatted streamflow out of each reach will be saved to a file whenever the cell-by-cell budget has been specified in Output Control. Unformatted output will be saved to <model name>.sfq.
- **isfropt** (*integer*) - An integer value that defines the format of the input data and whether or not unsaturated flow is simulated beneath streams. Values of ISFROPT are defined as follows

0 No vertical unsaturated flow beneath streams. Streambed elevations,
stream slope, streambed thickness, and streambed hydraulic conductivity are read for each stress period using variables defined in Items 6b and 6c; the optional variables in Item 2 are not used.

1 No vertical unsaturated flow beneath streams. Streambed elevation,
stream slope, streambed thickness, and streambed hydraulic conductivity are read for each reach only once at the beginning of the simulation using optional variables defined in Item 2; Items 6b and 6c are used to define stream width and depth for ICALC = 0 and stream width for ICALC = 1.

2 Streambed and unsaturated-zone properties are read for each reach

only once at the beginning of the simulation using optional variables defined in Item 2; Items 6b and 6c are used to define stream width and depth for ICALC = 0 and stream width for ICALC = 1. When using the LPF Package, saturated vertical hydraulic conductivity for the unsaturated zone is the same as the vertical hydraulic conductivity of the corresponding layer in LPF and input variable UHC is not read.

3 Same as 2 except saturated vertical hydraulic conductivity for the unsaturated zone (input variable UHC) is read for each reach.

4 Streambed and unsaturated-zone properties are read for the beginning and end of each stream segment using variables defined in Items 6b and 6c; the optional variables in Item 2 are not used. Streambed properties can vary each stress period. When using the LPF Package, saturated vertical hydraulic conductivity for the unsaturated zone is the same as the vertical hydraulic conductivity of the corresponding layer in LPF and input variable UHC1 is not read.

5 Same as 4 except saturated vertical hydraulic conductivity for the unsaturated zone (input variable UHC1) is read for each segment at the beginning of the first stress period only.

- **nstrail** (*integer*) - An integer value that is the number of trailing wave increments used to represent a trailing wave. Trailing waves are used to represent a decrease in the surface infiltration rate. The value can be increased to improve mass balance in the unsaturated zone. Values between 10 and 20 work well and result in unsaturated-zone mass balance errors beneath streams ranging between 0.001 and 0.01 percent. Please see Smith (1983) for further details. (default is 10; for MODFLOW-2005 simulations only when isfrop > 1)
- **isuzn** (*integer*) - An integer value that is the maximum number of vertical cells used to define the unsaturated zone beneath a stream reach. If ICALC is 1 for all segments then ISUZN should be set to 1. (default is 1; for MODFLOW-2005 simulations only when isfrop > 1)
- **nsfrsets** (*integer*) - An integer value that is the maximum number of different sets of trailing waves used to allocate arrays. Arrays are allocated by multiplying NSTRAIL by NSFRSETS. A value of 30 is sufficient for problems where the stream depth varies often. NSFRSETS does not affect model run time. (default is 30; for MODFLOW-2005 simulations only when isfrop > 1)
- **irtflg** (*integer*) - An integer value that indicates whether transient streamflow routing is active. IRTFLG must be specified if NSTRM < 0. If IRTFLG > 0, streamflow will be routed using the kinematic-wave equation (see USGS Techniques and Methods 6-D1, p. 68-69); otherwise, IRTFLG should be specified as 0. Transient streamflow routing is only available for MODFLOW-2005; IRTFLG can be left blank for MODFLOW-2000 simulations. (default is 1)
- **numtim** (*integer*) - An integer value equal to the number of sub time steps used to route streamflow. The time step that will be used to route streamflow will be equal to the MODFLOW time step

- divided by NUMTIM. (default is 2; for MODFLOW-2005 simulations only when irtflg > 0)
- **weight** (*float*) - A real number equal to the time weighting factor used to calculate the change in channel storage. WEIGHT has a value between 0.5 and 1. Please refer to equation 83 in USGS Techniques and Methods 6-D1 for further details. (default is 0.75; for MODFLOW-2005 simulations only when irtflg > 0)
 - **flwtol** (*float*) - A real number equal to the streamflow tolerance for convergence of the kinematic wave equation used for transient streamflow routing. A value of 0.00003 cubic meters per second has been used successfully in test simulations (and would need to be converted to whatever units are being used in the particular simulation). (default is 0.0001; for MODFLOW-2005 simulations only when irtflg > 0)
 - **reach_data** (*recarray*) - Numpy record array of length equal to nstrm, with columns for each variable entered in item 2 (see SFR package input instructions). In following flopy convention, layer, row, column and node number (for unstructured grids) are zero-based; segment and reach are one-based.
 - **segment_data** (*recarray*) - Numpy record array of length equal to nss, with columns for each variable entered in items 6a, 6b and 6c (see SFR package input instructions). Segment numbers are one-based.
 - **channel_geometry_data** (*dict of dicts containing lists*) - Optional. Outer dictionary keyed by stress period (0-based); inner dictionaries keyed by segment number (1-based), for which 8-point channel cross section geometries are desired. Inner dict values are lists of shape (2,8) - with the first dimension referring to two lists: one of 8 XCPT values, and the other of 8 ZCPT values. Example structure: {kper: {segment: [[xcpt1...xcpt8],[zcpt1...zcpt8]]}}. Note that for these to be applied, the user must also specify an icalc value of 2 for each corresponding segment in segment_data for the relevant stress periods.
 - **dataset_5** (*dict of lists*) - Optional; will be built automatically from segment_data unless specified. Dict of lists, with key for each stress period. Each list contains the variables [itmp, irdflag, iptflag]. (see SFR documentation for more details):
 - **itmp** (*list of integers (len = NPER)*) - For each stress period, an integer value for reusing or reading stream segment data that can change each stress period. If ITMP = 0 then all stream segment data are defined by Item 4 (NSFRPAR > 0; number of stream parameters is greater than 0). If ITMP > 0, then stream segment data are not defined in Item 4 and must be defined in Item 6 below for a number of segments equal to the value of ITMP. If ITMP < 0, then stream segment data not defined in Item 4 will be reused from the last stress period (Item 6 is not read for the current stress period). ITMP must be defined >= 0 for the first stress period of a simulation.
 - **irdflag** (*int or list of integers (len = NPER)*) - For each stress

period, an integer value for printing input data specified for this stress period. If `IRDFLG = 0`, input data for this stress period will be printed. If `IRDFLG > 0`, then input data for this stress period will not be printed.

- **iptflag** (*int* or *list* of integers (*len* = *NPER*)) - For each stress period, an integer value for printing streamflow- routing results during this stress period. If `IPTFLG = 0`, or whenever the variable `ICBCFL` or “Save Budget” is specified in Output Control, the results for specified time steps during this stress period will be printed. If `IPTFLG > 0`, then the results during this stress period will not be printed.
- **extension** (*string*) - Filename extension (default is ‘sfr’)
- **unit_number** (*int*) - File unit number (default is None).
- **filenames** (*str* or *list* of *str*) - Filenames to use for the package and the output files. If `filenames=None` the package name will be created using the model name and package extension and the cbc output and sfr output name will be created using the model name and .cbc the .sfr.bin/.sfr.out extensions (for example, `modflowtest.cbc`, and `modflowtest.sfr.bin`), if `ipakcbc` and `istcb2` are numbers greater than zero. If a single string is passed the package name will be set to the string and other uzf output files will be set to the model name with the appropriate output file extensions. To define the names for all package files (input and output) the length of the list of strings should be 3. Default is None.

outlets

Contains the outlet for each SFR segment; format is `{per: {segment: outlet}}`
This attribute is created by the `get_outlets()` method.

Type

nested dictionary

outsegs

Each array is of shape `nss rows x maximum of nss columns`. The first column contains the SFR segments, the second column contains the outsegs of those segments; the third column the outsegs of the outsegs, and so on, until all outlets have been encountered, or `nss` is reached. The latter case indicates circular routing. This attribute is created by the `get_outlets()` method.

Type

dictionary of arrays

Notes

Parameters are not supported in FloPy.

MODFLOW-OWHM is not supported.

The Ground-Water Transport (GWT) process is not supported.

Limitations on which features are supported...

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow()
>>> sfr2 = flopy.modflow.ModflowSfr2(m1, ...)
```

assign_layers(*adjust_botms=False, pad=1.0*)

Assigns the appropriate layer for each SFR reach, based on cell bottoms at location of reach.

Parameters

- **adjust_botms** (*bool*) - Streambed bottom elevations below the model bottom will cause an error in MODFLOW. If True, adjust bottom elevations in lowest layer of the model so they are at least pad distance below any co-located streambed elevations.
- **pad** (*scalar*) - Minimum distance below streambed bottom to set any conflicting model bottom elevations.

Notes

Streambed bottom = strtotop - strthick This routine updates the elevations in the botm array of the flopy.model.ModflowDis instance. To produce a new DIS package file, model.write() or flopy.model.ModflowDis.write() must be run.

check(*f=None, verbose=True, level=1, checktype=None*)

Check sfr2 package data for common errors.

Parameters

- **f** (*str or file handle*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If f is None, check method does not write results to a summary file. (default is None)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.

Return type

None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.sfr2.check()
```

property `const`

property `dataset_5`

auto-update itmp so it is consistent with segment_data.

deactivate_ibound_above()

Sets ibound to 0 for all cells above active SFR cells.

Parameters

`none` -

Notes

This routine updates the ibound array of the `flopy.model.ModflowBas6` instance. To produce a new BAS6 package file, `model.write()` or `flopy.model.ModflowBas6.write()` must be run.

default_value = 0.0

property `df`

export(*f*, ***kwargs*)

Method to export a package to netcdf or shapefile based on the extension of the file name (.shp for shapefile, .nc for netcdf)

Parameters

- *f* (*str*) - filename
- *kwargs* (*keyword arguments*) -

modelgrid

[`flopy.discretization.Grid` instance] user supplied modelgrid which can be used for exporting in lieu of the modelgrid associated with the model object

Return type

None or Netcdf object

export_linkages(*f*, ***kwargs*)

Export linework shapefile showing all routing connections between SFR reaches. A length field containing the distance between connected reaches can be used to filter for the longest connections in a GIS.

export_outlets(*f*, ***kwargs*)

Export point shapefile showing locations where streamflow is leaving the model (outset=0).

export_transient_variable(*f*, *varname*, ***kwargs*)

Export point shapefile showing locations with a given segment_data variable applied. For example, segments where streamflow is entering or leaving the upstream end of a stream segment (FLOW) or where RUNOFF is applied. Cell

centroids of the first reach of segments with non-zero terms of varname are exported; values of varname are exported by stress period in the attribute fields (e.g. flow0, flow1, flow2... for FLOW in stress periods 0, 1, 2...

Parameters

- **f** (*str*, *filename*) -
- **varname** (*str*) - Variable in SFR Package dataset 6a (see SFR package documentation)

static get_default_reach_dtype(*structured=True*)

static get_default_segment_dtype()

static get_empty_reach_data(*nreaches=0*, *aux_names=None*, *structured=True*, *default_value=0.0*)

static get_empty_segment_data(*nsegments=0*, *aux_names=None*, *default_value=0.0*)

get_outlets(*level=0*, *verbose=True*)

Traces all routing connections from each headwater to the outlet.

get_slopes(*default_slope=0.001*, *minimum_slope=0.0001*, *maximum_slope=1.0*)

Compute slopes by reach using values in strtotop (streambed top) and rchlen (reach length) columns of reach_data. The slope for a reach n is computed as strtotop(n+1) - strtotop(n) / rchlen(n). Slopes for outlet reaches are set equal to a default value (default_slope). Populates the slope column in reach_data.

Parameters

- **default_slope** (*float*) - Slope value applied to outlet reaches (where water leaves the model). Default value is 0.001
- **minimum_slope** (*float*) - Assigned to reaches with computed slopes less than this value. This ensures that the Manning's equation won't produce unreasonable values of stage (in other words, that stage is consistent with assumption that streamflow is primarily drive by the streambed gradient). Default value is 0.0001.
- **maximum_slope** (*float*) - Assigned to reaches with computed slopes more than this value. Default value is 1.

get_upsegs()

From segment_data, returns nested dict of all upstream segments by segment, by stress period.

Returns

all_upsegs - Nested dictionary of form {stress period: {segment: [list of upsegs]}}

Return type

dict

Notes

This method will not work if there are instances of circular routing.

`get_variable_by_stress_period(varname)`

property graph

Dictionary of routing connections between segments.

`len_const = {1: 1.486, 2: 1.0, 3: 100.0}`

classmethod `load(f, model, nper=None, gwt=False, nsol=1, ext_unit_dict=None)`

Default load method for standard boundary packages.

property nper

`nsfrpar = 0`

property nss

property nstrm

property paths

`plot_path(start_seg=None, end_seg=0, plot_segment_lines=True)`

Plot a profile of streambed elevation and model top along a path of segments.

Parameters

- `start_seg (int)` - Number of first segment in path.
- `end_seg (int)` - Number of last segment in path (defaults to 0/outlet).
- `plot_segment_lines (bool)` - Controls plotting of segment end locations along profile. (default True)

Returns

`ax`

Return type

`matplotlib.axes._subplots.AxesSubplot` object

`renumber_segments()`

Renumber segments so that segment numbering is continuous and always increases in the downstream direction. This may speed convergence of the NWT solver in some situations.

Returns

`r`

Return type

dictionary mapping old segment numbers to new

`repair_outsegs()`

`reset_reaches()`

`set_outreaches()`

Determine the outreach for each SFR reach (requires a reachID column in reach_data). Uses the segment routing specified for the first stress period to route reaches between segments.

```
time_const = {1: 1.0, 2: 60.0, 3: 3600.0, 4: 86400.0, 5: 31557600.0}
```

```
write_file(filename=None)
```

Write the package file.

Return type

None

```
class check(sfrpackage, verbose=True, level=1)
```

Bases: `object`

Check SFR2 package for common errors

Parameters

- **sfrpackage** (*object*) - Instance of Flopy ModflowSfr2 class.
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.

Notes

Daniel Feinstein's top 10 SFR problems (7/16/2014): 1) cell gaps btw adjacent reaches in a single segment 2) cell gaps btw routed segments. possibly because of re-entry problems at domain edge 3) adjacent reaches with STOP sloping the wrong way 4) routed segments with end/start sloping the wrong way 5) STOP>TOP1 violations, i.e., floaters 6) STOP<<TOP1 violations, i.e., exaggerated incisions 7) segments that end within one diagonal cell distance from another segment, inviting linkage 8) circular routing of segments 9) multiple reaches with non-zero conductance in a single cell 10) reaches in inactive cells

Also after running the model they will want to check for backwater effects.

```
elevations(min_strtop=-10, max_strtop=15000)
```

Checks streambed elevations for downstream rises and inconsistencies with model grid

```
for_nans()
```

Check for nans in reach or segment data

```
numbering()
```

Checks for continuity in segment and reach numbering

```
overlapping_conductance(tol=1e-06)
```

Checks for multiple SFR reaches in one cell; and whether more than one reach has Cond > 0

```
routing()
```

Checks for breaks in routing and does comprehensive check for circular routing

```
run_all()
```

```
slope(minimum_slope=0.0001, maximum_slope=1.0)
```

Checks that streambed slopes are greater than or equal to a specified minimum value. Low slope values can cause "backup" or unrealistic stream stages with icalc options where stage is computed.

find_path(*graph*, *start*, *end*=0)

Get a path through the routing network, from a segment to an outlet.

Parameters

- **graph** (*dict*) - Dictionary of seg : outseg numbers
- **start** (*int*) - Starting segment
- **end** (*int*) - Ending segment (default 0)

Returns

path - List of segment numbers along routing path.

Return type

list

flopy.modflow.mfsip module

mfsip module. Contains the ModflowSip class. Note that the user can access the ModflowSip class as *flopy.modflow.ModflowSip*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowSip(model, mxiter=200, nparm=5, accl=1, hclose=1e-05, ipcalc=1, wseed=0,
                 iprsip=0, extension='sip', unitnumber=None, filenames=None)
```

Bases: *Package*

MODFLOW Strongly Implicit Procedure Package Class.

Parameters

- **model** (*model object*) - The model object (of type :class:flopy.modflow.mf.Modflow) to which this package will be added.
- **mxiter** (*integer*) - The maximum number of times through the iteration loop in one time step in an attempt to solve the system of finite-difference equations. (default is 200)
- **nparm** (*integer*) - The number of iteration variables to be used. Five variables are generally sufficient. (default is 5)
- **accl** (*float*) - The acceleration variable, which must be greater than zero and is generally equal to one. If a zero is entered, it is changed to one. (default is 1)
- **hclose** (*float > 0*) - The head change criterion for convergence. When the maximum absolute value of head change from all nodes during an iteration is less than or equal to hclose, iteration stops. (default is 1e-5)
- **ipcalc** (*0 or 1*) - A flag indicating where the seed for calculating iteration variables will come from. 0 is the seed entered by the user will be used. 1 is the seed will be calculated at the start of the simulation from problem variables. (default is 0)
- **wseed** (*float > 0*) - The seed for calculating iteration variables. wseed is always read, but is used only if ipcalc is equal to zero. (default is 0)

- **iprsip** (*integer* > 0) - the printout interval for sip. iprsip, if equal to zero, is changed to 999. The maximum head change (positive or negative) is printed for each iteration of a time step whenever the time step is an even multiple of iprsip. This printout also occurs at the end of each stress period regardless of the value of iprsip. (default is 0)
- **extension** (*string*) - Filename extension (default is 'sip')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str* or *list* of *str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow()
>>> sip = flopy.modflow.ModflowSip(m1, mxiter=100, hclose=0.0001)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename* or *file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

Returns

sip

Return type

ModflowSip object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> sip = flopy.modflow.ModflowSip.load('test.sip', m)
```

write_file()

Write the package file.

Return type

None

flopy.modflow.mfsor module

mfsor module. Contains the ModflowSor class. Note that the user can access the ModflowSor class as *flopy.modflow.ModflowSor*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowSor(model, mxiter=200, accl=1, hclose=1e-05, iprsor=0, extension='sor',
                 unitnumber=None, filenames=None)
```

Bases: [Package](#)

MODFLOW Slice-successive overrelaxation Package Class.

Parameters

- **model** (*model object*) - The model object (of type :class:flopy.modflow.mf.Modflow) to which this package will be added.
- **mxiter** (*integer*) - The maximum number of iterations allowed in a time step. (default is 200)
- **accl** (*float*) - The acceleration variable, which must be greater than zero and is generally between 1. and 2. (default is 1)
- **hclose** (*float > 0*) - The head change criterion for convergence. When the maximum absolute value of head change from all nodes during an iteration is less than or equal to hclose, iteration stops. (default is 1e-5)
- **iprsor** (*integer > 0*) - the printout interval for sor. iprsor, if equal to zero, is changed to 999. The maximum head change (positive or negative) is printed for each iteration of a time step whenever the time step is an even multiple of iprsor. This printout also occurs at the end of each stress period regardless of the value of iprsor. (default is 0)
- **extension** (*string*) - Filename extension (default is 'sor')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow()
>>> sor = flopy.modflow.ModflowSor(m1)
```

```
classmethod load(f, model, ext_unit_dict=None)
```

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

sor

Return type

ModflowSor object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> sor = flopy.modflow.ModflowSor.load('test.sor', m)
```

write_file()

Write the package file.

Return type

None

flopy.modflow.mfstr module

mfstr module. Contains the ModflowStr class. Note that the user can access the ModflowStr class as `flopy.modflow.ModflowStr`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowStr(model, mxacts=0, nss=0, ntrib=0, ndiv=0, icalc=0, const=86400.0,
                 ipakcb=None, istcb2=None, dtype=None, stress_period_data=None,
                 segment_data=None, irdflg=None, iptflg=None, extension='str',
                 unitnumber=None, filenames=None, options=None, **kwargs)
```

Bases: `Package`

MODFLOW Stream Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **mxacts** (*int*) - Maximum number of stream reaches that will be in use during any stress period. (default is 0)
- **nss** (*int*) - Number of stream segments. (default is 0)
- **ntrib** (*int*) - The number of stream tributaries that can connect to one segment. The program is currently dimensioned so that NTRIB cannot exceed 10. (default is 0)

- **ndiv** (*int*) - A flag, which when positive, specifies that diversions from segments are to be simulated. (default is 0)
- **icalc** (*int*) - A flag, which when positive, specifies that stream stages in reaches are to be calculated. (default is 0)
- **const** (*float*) - Constant value used in calculating stream stage in reaches whenever ICALC is greater than 0. This constant is 1.486 for flow units of cubic feet per second and 1.0 for units of cubic meters per second. The constant must be multiplied by 86,400 when using time units of days in the simulation. If ICALC is 0, const can be any real value. (default is 86400.)
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0).
- **istcb2** (*int*) - A flag that is used flag and a unit number for the option to store streamflow out of each reach in an unformatted (binary) file. If istcb2 is greater than zero streamflow data will be saved. (default is None).
- **dtype** (*tuple, list, or numpy array of numpy dtypes*) - is a tuple, list, or numpy array containing the dtype for datasets 6 and 8 and the dtype for datasets 9 and 10 data in stress_period_data and segment_data dictionaries. (default is None)
- **irdflg** (*integer or dictionary*) - is a integer or dictionary containing a integer flag, when positive suppresses printing of the stream input data for a stress period. If an integer is passed, all stress periods will use the same value. If a dictionary is passed, stress periods not in the dictionary will assigned a value of 1. Default is None which will assign a value of 1 to all stress periods.
- **iptflg** (*integer or dictionary*) - is a integer or dictionary containing a integer flag, when positive suppresses printing of stream results for a stress period. If an integer is passed, all stress periods will use the same value. If a dictionary is passed, stress periods not in the dictionary will assigned a value of 1. Default is None which will assign a value of 1 to all stress periods.
- **stress_period_data** (*dictionary of reach data*) - Each dictionary contains a list of str reach data for a stress period.

Each stress period in the dictionary data contains data for datasets 6 and 8.

The value for stress period data for a stress period can be an integer (-1 or 0), a list of lists, a numpy array, or a numpy recarray. If stress period data for a stress period contains an integer, a -1 denotes data from the previous stress period will be reused and a 0 indicates there are no str reaches for this stress period.

Otherwise stress period data for a stress period should contain mxacts or fewer rows of data containing data for each reach. Reach data are specified through definition of layer (int),

row (int), column (int), segment number (int), sequential reach number (int), flow entering a segment (float), stream stage (float), streambed hydraulic conductance (float), streambed bottom elevation (float), streambed top elevation (float), stream width (float), stream slope (float), roughness coefficient (float), and auxiliary variable data for auxiliary variables defined in options (float).

If icalc=0 is specified, stream width, stream slope, and roughness coefficients, are not used and can be any value for each stress period. If data are specified for dataset 6 for a given stress period and icalc>0, then stream width, stream slope, and roughness coefficients should be appropriately set.

The simplest form is a dictionary with a lists of boundaries for each stress period, where each list of boundaries itself is a list of boundaries. Indices of the dictionary are the numbers of the stress period. For example, if mxacts=3 this gives the form of:

```
stress_period_data =
{0: [
    [lay, row, col, seg, reach, flow, stage, cond, sbot, stop,
    ↪width, slope, rough],
    [lay, row, col, seg, reach, flow, stage, cond, sbot, stop,
    ↪width, slope, rough],
    [lay, row, col, seg, reach, flow, stage, cond, sbot, stop,
    ↪width, slope, rough]]
],
1: [
    [lay, row, col, seg, reach, flow, stage, cond, sbot, stop,
    ↪width, slope, rough],
    [lay, row, col, seg, reach, flow, stage, cond, sbot, stop,
    ↪width, slope, rough],
    [lay, row, col, seg, reach, flow, stage, cond, sbot, stop,
    ↪width, slope, rough]]
], ...
kper:
[
    [lay, row, col, seg, reach, flow, stage, cond, sbot, stop,
    ↪width, slope, rough],
    [lay, row, col, seg, reach, flow, stage, cond, sbot, stop,
    ↪width, slope, rough],
    [lay, row, col, seg, reach, flow, stage, cond, sbot, stop,
    ↪width, slope, rough]]
]
}
```

- **segment_data** (*dictionary of str segment data*) - Each dictionary contains a list of segment str data for a stress period.

Each stress period in the dictionary data contains data for datasets 9, and 10. Segment data for a stress period are ignored if a integer value is specified for stress period data.

The value for segment data for a stress period can be an integer (-1 or 0), a list of lists, a numpy array, or a numpy recarray.

If segment data for a stress period contains an integer, a -1 denotes data from the previous stress period will be reused and a 0 indicates there are no str segments for this stress period.

Otherwise stress period data for a stress period should contain nss rows of data containing data for each segment. Segment data are specified through definition of itrib (int) data for up to 10 tributaries and iupseg (int) data.

If ntrib=0 is specified, itrib values are not used and can be any value for each stress period. If data are specified for dataset 6 for a given stress period and ntrib>0, then itrib data should be specified for columns 0:ntrib.

If ndiv=0 is specified, iupseg values are not used and can be any value for each stress period. If data are specified for dataset 6 for a given stress period and ndiv>0, then iupseg data should be specified for the column in the dataset [10].

The simplest form is a dictionary with a lists of boundaries for each stress period, where each list of boundaries itself is a list of boundaries. Indices of the dictionary are the numbers of the stress period. For example, if nss=2 and ntrib>0 and/or ndiv>0 this gives the form of:

```
segment_data =
{0: [
    [itrib1, itrib2, itrib3, itrib4, itrib5, itrib6, itrib7,
    ↪itrib8, itrib9, itrib10, iupseg],
    [itrib1, itrib2, itrib3, itrib4, itrib5, itrib6, itrib7,
    ↪itrib8, itrib9, itrib10, iupseg],
    ],
1: [
    [itrib1, itrib2, itrib3, itrib4, itrib5, itrib6, itrib7,
    ↪itrib8, itrib9, itrib10, iupseg],
    [itrib1, itrib2, itrib3, itrib4, itrib5, itrib6, itrib7,
    ↪itrib8, itrib9, itrib10, iupseg],
    ], ...
kper:
    [
        [itrib1, itrib2, itrib3, itrib4, itrib5, itrib6, itrib7,
        ↪itrib8, itrib9, itrib10, iupseg],
        [itrib1, itrib2, itrib3, itrib4, itrib5, itrib6, itrib7,
        ↪itrib8, itrib9, itrib10, iupseg],
        ]
}
```

- **options** (*list of strings*) - Package options. Auxiliary variables included as options should be constructed as options=['AUXILIARY IFACE', 'AUX xyz']. Either 'AUXILIARY' or 'AUX' can be specified (case insensitive). (default is None).
- **extension** (*string*) - Filename extension (default is 'str')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package

name will be created using the model name and package extension and the cbc output and str output name will be created using the model name and .cbc the .sfr.bin/.sfr.out extensions (for example, modflowtest.cbc, and modflowtest.str.bin), if ipakcbc and istcb2 are numbers greater than zero. If a single string is passed the package will be set to the string and cbc and sf routput names will be created using the model name and .cbc and .str.bin/.str.out extensions, if ipakcbc and istcb2 are numbers greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 3. Default is None.

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> strd = {}
>>> strd[0] = [[2, 3, 4, 15.6, 1050., -4]] #this str boundary will be
>>> #applied to all stress periods
>>> str = flopy.modflow.ModflowStr(m, stress_period_data=strd)
```

```
static get_default_dtype(structured=True)
```

```
static get_empty(ncells=0, nss=0, aux_names=None, structured=True)
```

```
classmethod load(f, model, nper=None, ext_unit_dict=None)
```

Load an existing package.

Parameters

- **f** (filename or file handle) - File to load.
- **model** (model object) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **nper** (int) - The number of stress periods. If nper is None, then nper will be obtained from the model object. (default is None).
- **ext_unit_dict** (dictionary, optional) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

str - ModflowStr object.

Return type

ModflowStr object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> strm = flopy.modflow.ModflowStr.load('test.str', m)
```

write_file()

Write the package file.

Return type

None

flopy.modflow.mfsub module

mfsub module. Contains the ModflowSub class. Note that the user can access the ModflowSub class as *flopy.modflow.ModflowSub*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowSub(model, ipakcb=None, isuboc=0, idsave=None, idrest=None, idbit=None,
                  nndb=1, ndb=1, nmz=1, nn=20, ac1=0.0, ac2=1.0, itmin=5, ln=0, ldn=0,
                  rnb=1, hc=100000.0, sfe=0.0001, sfv=0.001, com=0.0, dp=[[1e-06, 6e-06,
                  0.0006]], dstart=1.0, dhc=100000.0, dcom=0.0, dz=1.0, nz=1, ids15=None,
                  ids16=None, extension='sub', unitnumber=None, filenames=None)
```

Bases: [Package](#)

MODFLOW SUB Package Class.

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0).
- **isuboc** (*int*) - isuboc is a flag used to control output of information generated by the SUB Package. (default is 0).
- **idsave** (*int*) - idsave is a flag and a unit number on which restart records for delay interbeds will be saved at the end of the simulation. (default is 0).
- **idrest** (*int*) - idrest is a flag and a unit number on which restart records for delay interbeds will be read in at the start of the simulation (default is 0).
- **idbit** (*int*) - idrest is an optional flag that defines if iteration will be used for the delay bed solution of heads. If idbit is 0 or less than 0, iteration will not be used (this is identical to the approach used in MODFLOW-2005 versions less than 1.13. if idbit is greater than 0, iteration of the delay bed solution will continue until convergence is achieved. (default is 0).
- **nndb** (*int*) - nndb is the number of systems of no-delay interbeds. (default is 1).

- **ndb** (*int*) - ndb is the number of systems of delay interbeds. (default is 1).
- **nmz** (*int*) - nmz is the number of material zones that are needed to define the hydraulic properties of systems of delay interbeds. Each material zone is defined by a combination of vertical hydraulic conductivity, elastic specific storage, and inelastic specific storage. (default is 1).
- **nn** (*int*) - nn is the number of nodes used to discretize the half space to approximate the head distributions in systems of delay interbeds. (default is 20).
- **ac1** (*float*) - ac1 is an acceleration parameter. This parameter is used to predict the aquifer head at the interbed boundaries on the basis of the head change computed for the previous iteration. A value of 0.0 results in the use of the aquifer head at the previous iteration. Limited experience indicates that optimum values may range from 0.0 to 0.6. (default is 0).
- **ac2** (*float*) - ac2 is an acceleration parameter. This acceleration parameter is a multiplier for the head changes to compute the head at the new iteration. Values are normally between 1.0 and 2.0, but the optimum is probably closer to 1.0 than to 2.0. However this parameter also can be used to help convergence of the iterative solution by using values between 0 and 1. (default is 1.0).
- **itmin** (*int*) - ITMIN is the minimum number of iterations for which one-dimensional equations will be solved for flow in interbeds when the Strongly Implicit Procedure (SIP) is used to solve the ground-water flow equations. If the current iteration level is greater than ITMIN and the SIP convergence criterion for head closure (HCLOSE) is met at a particular cell, the one-dimensional equations for that cell will not be solved. The previous solution will be used. The value of ITMIN is not used if a solver other than SIP is used to solve the ground-water flow equations. (default is 5).
- **ln** (*int or array of ints (nndb)*) - ln is a one-dimensional array specifying the model layer assignments for each system of no-delay interbeds. (default is 0).
- **ldn** (*int or array of ints (ndb)*) - ldn is a one-dimensional array specifying the model layer assignments for each system of delay interbeds. (default is 0).
- **rnb** (*float or array of floats (ndb, nrow, ncol)*) - rnb is an array specifying the factor nequiv at each cell for each system of delay interbeds. The array also is used to define the areal extent of each system of interbeds. For cells beyond the areal extent of the system of interbeds, enter a number less than 1.0 in the corresponding element of this array. (default is 1).
- **hc** (*float or array of floats (nndb, nrow, ncol)*) - hc is an array specifying the preconsolidation head or preconsolidation stress in terms of head in the aquifer for systems of no-delay interbeds. For any model cells in which specified HC is greater than the

- corresponding value of starting head, the value of HC will be set to that of starting head. (default is 100000).
- **sfe** (*float* or array of floats (*nndb*, *nrow*, *ncol*)) - sfe is an array specifying the dimensionless elastic skeletal storage coefficient for systems of no-delay interbeds. (default is 1.e-4).
 - **sfv** (*float* or array of floats (*nndb*, *nrow*, *ncol*)) - sfv is an array specifying the dimensionless inelastic skeletal storage coefficient for systems of no-delay interbeds. (default is 1.e-3).
 - **com** (*float* or array of floats (*nndb*, *nrow*, *ncol*)) - com is an array specifying the starting compaction in each system of no-delay interbeds. Compaction values computed by the package are added to values in this array so that printed or stored values of compaction and land subsidence may include previous components. Values in this array do not affect calculations of storage changes or resulting compaction. For simulations in which output values are to reflect compaction and subsidence since the start of the simulation, enter zero values for all elements of this array. (default is 0).
 - **dp** (*list* or array of floats (*nmz*, 3)) - Data item includes nmz records, each with a value of vertical hydraulic conductivity, elastic specific storage, and inelastic specific storage. (default is [1.e-6, 6.e-6, 6.e-4]).
 - **dstart** (*float* or array of floats (*ndb*, *nrow*, *ncol*)) - dstart is an array specifying starting head in interbeds for systems of delay interbeds. For a particular location in a system of interbeds, the starting head is applied to every node in the string of nodes that approximates flow in half of a doubly draining interbed. (default is 1).
 - **dhc** (*float* or array of floats (*ndb*, *nrow*, *ncol*)) - dhc is an array specifying the starting preconsolidation head in interbeds for systems of delay interbeds. For a particular location in a system of interbeds, the starting preconsolidation head is applied to every node in the string of nodes that approximates flow in half of a doubly draining interbed. For any location at which specified starting preconsolidation head is greater than the corresponding value of the starting head, Dstart, the value of the starting preconsolidation head will be set to that of the starting head. (default is 100000).
 - **dcom** (*float* or array of floats (*ndb*, *nrow*, *ncol*)) - dcom is an array specifying the starting compaction in each system of delay interbeds. Compaction values computed by the package are added to values in this array so that printed or stored values of compaction and land subsidence may include previous components. Values in this array do not affect calculations of storage changes or resulting compaction. For simulations in which output values are to reflect compaction and subsidence since the start of the simulation, enter zero values for all elements of this array. (default is 0).

- **dz** (*float* or array of floats (*ndb*, *nrow*, *ncol*)) - dz is an array specifying the equivalent thickness for a system of delay interbeds. (default is 1).
- **nz** (*int* or array of ints (*ndb*, *nrow*, *ncol*)) - nz is an array specifying the material zone numbers for systems of delay interbeds. The zone number for each location in the model grid selects the hydraulic conductivity, elastic specific storage, and inelastic specific storage of the interbeds. (default is 1).
- **ids15** (*list* or array of ints (12)) - Format codes and unit numbers for subsidence, compaction by model layer, compaction by interbed system, vertical displacement, no-delay preconsolidation, and delay preconsolidation will be printed. If ids15 is None and isuboc>0 then print code 0 will be used for all data which is output to the binary subsidence output file (unit=1051). The 12 entries in ids15 correspond to ifm1, iun1, ifm2, iun2, ifm3, iun3, ifm4, iun4, ifm5, iun5, ifm6, and iun6 variables. (default is None).
- **ids16** (*list* or array of ints (*isuboc*, 17)) - Stress period and time step range and print and save flags used to control printing and saving of information generated by the SUB Package during program execution. Each row of ids16 corresponds to isp1, isp2, its1, its2, ifl1, ifl2, ifl3, ifl4, ifl5, ifl6, ifl7, ifl8, ifl9, ifl10, ifl11, ifl12, and ifl13 variables for isuboc entries. isp1, isp2, its1, and its2 are stress period and time step ranges. ifl1 and ifl2 control subsidence printing and saving. ifl3 and ifl4 control compaction by model layer printing and saving. ifl5 and ifl6 control compaction by interbed system printing and saving. ifl7 and ifl8 control vertical displacement printing and saving. ifl9 and ifl10 control critical head for no-delay interbeds printing and saving. ifl11 and ifl12 control critical head for delay interbeds printing and saving. ifl13 controls volumetric budget for delay interbeds printing. If ids16 is None and isuboc>0 then all available subsidence output will be printed and saved to the binary subsidence output file (unit=1051). (default is None).
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str* or *list* of *str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output name and other sub output files will be created using the model name and .cbc and swt output extensions (for example, modflowtest.cbc), if ipakcbc and other sub output files (dataset 15) are numbers greater than zero. If a single string is passed the package name will be set to the string and other sub output files will be set to the model name with the appropriate output file extensions. To define the names for all package files (input and output) the length of the list of strings should be 9. Default is None.

Notes

Parameters are supported in Flopy only when reading in existing models. Parameter values are converted to native values in Flopy and the connection to “parameters” is thus nonexistent. Parameters are not supported in the SUB Package.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> sub = flopy.modflow.ModflowSub(m)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

`sub`

Return type

ModflowSub object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> sub = flopy.modflow.ModflowSub.load('test.sub', m)
```

write_file(*check=False, f=None*)

Write the package file.

Return type

None

flopy.modflow.mfswi2 module

mfswi2 module. Contains the ModflowSwi2 class. Note that the user can access the ModflowSwi2 class as `flopy.modflow.ModflowSwi2`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowSwi2(model, nsrf=1, istrat=1, nob=0, iswizt=None, ipakcb=None, iswiobs=0,
                  options=None, nsolver=1, iprsol=0, mutsol=3, solver2params={'damp':
1.0, 'damp1': 1.0, 'iter1': 20, 'mxiter': 100, 'nbpol': 2, 'npcond': 1,
'rclose': 0.0001, 'relax': 1.0, 'zclose': 0.001}, toeslope=0.05,
tipslope=0.05, alpha=None, beta=0.1, nadptmx=1, nadptmn=1, adptfct=1.0,
nu=0.025, zeta=[0.0], ssz=0.25, isource=0, obsnam=None, obslrc=None,
npln=None, extension='swi2', unitnumber=None, filenames=None)
```

Bases: [Package](#)

MODFLOW SWI2 Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **nsrf** (*int*) - number of active surfaces (interfaces). This equals the number of zones minus one. (default is 1).
- **istrat** (*int*) - flag indicating the density distribution. (default is 1).
- **iswizt** (*int*) - unit number for zeta output. (default is None).
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is None).
- **iswiobs** (*int*) - flag and unit number SWI2 observation output. (default is 0).
- **options** (*list of strings*) - Package options. If 'adaptive' is one of the options adaptive SWI2 time steps will be used. (default is None).
- **nsolver** (*int*) - DE4 solver is used if nsolver=1. PCG solver is used if nsolver=2. (default is 1).
- **iprsol** (*int*) - solver print out interval. (default is 0).
- **mutsol** (*int*) - If MUTSOL = 0, tables of maximum head change and residual will be printed each iteration. If MUTSOL = 1, only the total number of iterations will be printed. If MUTSOL = 2, no information will be printed. If MUTSOL = 3, information will only be printed if convergence fails. (default is 3).
- **solver2parameters** (*dict*) - only used if nsolver = 2
 - mxiter**
[int] maximum number of outer iterations. (default is 100)
 - iter1**
[int] maximum number of inner iterations. (default is 20)

npcond

[int] flag used to select the matrix conditioning method. (default is 1). specify NPCOND = 1 for Modified Incomplete Cholesky. specify NPCOND = 2 for Polynomial.

zclose

[float] is the ZETA change criterion for convergence. (default is 1e-3).

rclose

[float] is the residual criterion for convergence. (default is 1e-4)

relax

[float] is the relaxation parameter used with NPCOND = 1. (default is 1.0)

nbpol

[int] is only used when NPCOND = 2 to indicate whether the estimate of the upper bound on the maximum eigenvalue is 2.0, or whether the estimate will be calculated. NBPOL = 2 is used to specify the value is 2.0; for any other value of NBPOL, the estimate is calculated. Convergence is generally insensitive to this parameter. (default is 2).

damp

[float] is the steady-state damping factor. (default is 1.)

damppt

[float] is the transient damping factor. (default is 1.)

- **toeslope** (*float*) - Maximum slope of toe cells. (default is 0.05)
- **tipslope** (*float*) - Maximum slope of tip cells. (default is 0.05)
- **alpha** (*float*) - fraction of threshold used to move the tip and toe to adjacent empty cells when the slope exceeds user-specified TOESLOPE and TIPSLOPE values. (default is None)
- **beta** (*float*) - Fraction of threshold used to move the toe to adjacent non-empty cells when the surface is below a minimum value defined by the user-specified TOESLOPE value. (default is 0.1).
- **nadptmx** (*int*) - only used if adaptive is True. Maximum number of SWI2 time steps per MODFLOW time step. (default is 1).
- **nadptmn** (*int*) - only used if adaptive is True. Minimum number of SWI2 time steps per MODFLOW time step. (default is 1).
- **adptfct** (*float*) - is the factor used to evaluate tip and toe thicknesses and control the number of SWI2 time steps per MODFLOW time step. When the maximum tip or toe thickness exceeds the product of TOESLOPE or TIPSLOPE the cell size and ADPTFCT, the number of SWI2 time steps are increased to a value less than or equal to NADPT. When the maximum tip or toe thickness is less than the product of TOESLOPE or TIPSLOPE the cell size and ADPTFCT, the number of SWI2 time steps is decreased in the next MODFLOW time step to a value greater than or equal to 1. ADPTFCT must be greater than 0.0 and is reset to 1.0 if NADPTMX is equal to NADPTMN. (default is 1.0).

- **nu** (*array of floats*) - if `istart = 1`, density of each zone (`nsrf + 1` values). if `istrat = 0`, density along top of layer, each surface, and bottom of layer (`nsrf + 2` values). (default is `0.025`)
- **zeta** (*list of floats or list of array of floats [(nlay, nrow, ncol),]*) - (`nlay, nrow, ncol`) initial elevations of the active surfaces. The list should contain an entry for each surface and be of size `nsrf`. (default is `[0.]`)
- **ssz** (*float or array of floats (nlay, nrow, ncol)*) - effective porosity. (default is `0.25`)
- **isource** (*integer or array of integers (nlay, nrow, ncol)*) - Source type of any external sources or sinks, specified with any outside package (i.e. WEL Package, RCH Package, GHB Package). (default is `0`).

If `ISOURCE > 0` sources and sinks have the same fluid density as the zone `ISOURCE`. If such a zone is not present in the cell, sources and sinks have the same fluid density as the active zone at the top of the aquifer. If `ISOURCE = 0` sources and sinks have the same fluid density as the active zone at the top of the aquifer. If `ISOURCE < 0` sources have the same fluid density as the zone with a number equal to the absolute value of `ISOURCE`. Sinks have the same fluid density as the active zone at the top of the aquifer. This option is useful for the modeling of the ocean bottom where infiltrating water is salt, yet exfiltrating water is of the same type as the water at the top of the aquifer.

- **obsnam** (*list of strings*) - names for nob observations.
- **obslrc** (*list of lists*) - zero-based [layer, row, column] lists for nob observations.
- **extension** (*string*) - Filename extension (default is `'swi2'`)
- **npln** (*int*) - Deprecated - use `nsrf` instead.
- **unitnumber** (*int*) - File unit number (default is `None`).
- **filenames** (*str or list of str*) - Filenames to use for the package and the zeta, cbc, obs output files. If `filenames=None` the package name will be created using the model name and package extension and the output file names will be created using the model name and output extensions. If a single string is passed the package will be set to the string and output names will be created using the model name and zeta, cbc, and observation extensions. To define the names for all package files (input and output) the length of the list of strings should be 4. Default is `None`.

Notes

Parameters are supported in Flopy only when reading in existing models. Parameter values are converted to native values in Flopy and the connection to “parameters” is thus nonexistent.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> swi2 = flopy.modflow.ModflowSwi2(m)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

`swi2`

Return type

ModflowSwi2 object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> swi2 = flopy.modflow.ModflowSwi2.load('test.swi2', m)
```

write_file(*check=True, f=None*)

Write the package file.

Parameters

check (*boolean*) - Check package data for common errors. (default True)

Return type

None

flopy.modflow.mfswr1 module

mfswr module. Contains the ModflowSwr1 class. Note that the user can access the ModflowSwr1 class as `flopy.modflow.ModflowSwr1`.

Additional information for this MODFLOW process can be found at the [Online MODFLOW Guide](#).

```
class ModflowSwr1(model, extension='swr', unitnumber=None, filenames=None)
```

Bases: [Package](#)

MODFLOW Surface-Water Routing Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **extension** (*string*) - Filename extension (default is 'swr')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

SWR1 Class is only used to write SWR1 filename to name file. Full functionality still needs to be implemented.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> swr = flopy.modflow.ModflowSwr1(m)
```

```
classmethod load(f, model, ext_unit_dict=None)
```

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type: class:`flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

swr - ModflowSwr1 object (of type `flopy.modflow.mfbas.ModflowSwr1`)

Return type

ModflowSwt1 object

Notes

Load method still needs to be implemented.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> swr = flopy.modflow.ModflowSwt1.load('test.swr', m)
```

write_file()

Write the package file.

Return type

None

flopy.modflow.mfswt module

mfswt module. Contains the ModflowSwt class. Note that the user can access the ModflowSub class as *flopy.modflow.ModflowSwt*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowSwt(model, ipakcb=None, iswtoc=0, nsystm=1, ithk=0, ivalid=0, istpcs=1,
    icrcc=0, lnwt=0, izcfl=0, izcfm=0, iglfl=0, iglfm=0, iestfl=0,
    iestfm=0, ipcsfl=0, ipcsfm=0, istfl=0, istfm=0, gl0=0.0, sgm=1.7,
    sgs=2.0, thick=1.0, sse=1.0, ssv=1.0, cr=0.01, cc=0.25, void=0.82,
    sub=0.0, pcsoff=0.0, pcs=0.0, ids16=None, ids17=None, extension='swt',
    unitnumber=None, filenames=None)
```

Bases: [Package](#)

MODFLOW SUB-WT Package Class.

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0).
- **iswtoc** (*int*) - iswtoc is a flag used to control output of information generated by the SUB Package. (default is 0).
- **nsystm** (*int*) - nsystm is the number of systems of interbeds. (default is 1).
- **ithk** (*int*) - ithk is a flag to determine how thicknesses of compressible sediments vary in response to changes in saturated thickness. If ithk < 1, thickness of compressible sediments is constant. If ithk > 0, thickness of compressible sediments varies in response to changes in saturated thickness. (default is 1).

- **ivoid** (*int*) - ivoid is a flag to determine how void ratios of compressible sediments vary in response to changes in saturated thickness. If ivoid < 1, void ratio will be treated as a constant. If ivoid > 0, void ratio will be treated as a variable. (default is 0).
- **nn** (*int*) - nn is the number of nodes used to discretize the half space to approximate the head distributions in systems of delay interbeds. (default is 20).
- **istpcs** (*int*) - istpcs is a flag to determine how initial preconsolidation stress will be obtained. If istpcs does not equal 0, an array of offset values will be read in for each model layer. The offset values will be added to the initial effective stress to get initial preconsolidation stress. If istpcs = 0, an array with initial preconsolidation stress values will be read. (default is 1).
- **icrcc** (*int*) - icrcc is a flag to determine how recompression and compression indices will be obtained. If ICRCC is not equal to 0, arrays of elastic specific storage and inelastic skeletal specific storage will be read for each system of interbeds; the recompression index and compression index will not be read. If icrcc = 0, arrays of recompression index and compression index will be read for each system of interbeds; elastic skeletal specific storage and inelastic skeletal specific storage will not be read. (default is 0).
- **lnwt** (*int or array of ints (nsystem)*) - lnwt is a one-dimensional array specifying the model layer assignments for each system of interbeds. (default is 0).
- **izcfl** (*int*) - izcfl is a flag to specify whether or not initial calculated values of layer-center elevation will be printed. (default is 0).
- **izcfm** (*int*) - izcfm is is a code for the format in which layer-center elevation will be printed. (default is 0).
- **iglfl** (*int*) - iglfl is a flag to specify whether or not initial calculated values of geostatic stress will be printed. (default is 0).
- **iglfm** (*int*) - iglfm is a code for the format in which geostatic stress will be printed. (default is 0).
- **iestfl** (*int*) - iestfl is a flag to specify whether or not initial calculated values of effective stress will be printed. (default is 0).
- **iestfm** (*int*) - iestfm is a code for the format in which effective stress will be printed. (default is 0).
- **ipcsfl** (*int*) - ipcsfl is a flag to specify whether or not initial calculated values of preconsolidation stress will be printed. (default is 0).
- **ipcsfm** (*int*) - ipcsfm is a code for the format in which preconsolidation stress will be printed. (default is 0).

- **istfl** (*int*) - istfl is a flag to specify whether or not initial equivalent storage properties will be printed for each system of interbeds. If icrcc is not equal to 0, the equivalent storage properties that can be printed are recompression and compression indices (cr and cc), which are calculated from elastic and inelastic skeletal specific storage (sske and sskv). If icrcc = 0, equivalent storage properties that can be printed are elastic and inelastic skeletal specific storage, which are calculated from the recompression and compression indices. (default is 0).
- **istfm** (*int*) - istfm is a code for the format in which equivalent storage properties will be printed. (default is 0).
- **gl0** (*float* or array of floats (*nrow*, *ncol*)) - gl0 is an array specifying the geostatic stress above model layer 1. If the top of model layer 1 is the land surface, enter values of zero for this array. (default is 0.).
- **sgm** (*float* or array of floats (*nrow*, *ncol*)) - sgm is an array specifying the specific gravity of moist or unsaturated sediments. (default is 1.7).
- **sgs** (*float* or array of floats (*nrow*, *ncol*)) - sgs is an array specifying the specific gravity of saturated sediments. (default is 2.).
- **thick** (*float* or array of floats (*nsystem*, *nrow*, *ncol*)) - thick is an array specifying the thickness of compressible sediments. (default is 1.).
- **sse** (*float* or array of floats (*nsystem*, *nrow*, *ncol*)) - sse is an array specifying the initial elastic skeletal specific storage of compressible beds. sse is not used if icrcc = 0. (default is 1.).
- **ssv** (*float* or array of floats (*nsystem*, *nrow*, *ncol*)) - ssv is an array specifying the initial inelastic skeletal specific storage of compressible beds. ssv is not used if icrcc = 0. (default is 1.).
- **cr** (*float* or array of floats (*nsystem*, *nrow*, *ncol*)) - cr is an array specifying the recompression index of compressible beds. cr is not used if icrcc is not equal to 0. (default is 0.01).
- **cc** (*float* or array of floats (*nsystem*, *nrow*, *ncol*)) - cc is an array specifying the compression index of compressible beds cc is not used if icrcc is not equal to 0. (default is 0.25).
- **void** (*float* or array of floats (*nsystem*, *nrow*, *ncol*)) - void is an array specifying the initial void ratio of compressible beds. (default is 0.82).
- **sub** (*float* or array of floats (*nsystem*, *nrow*, *ncol*)) - sub is an array specifying the initial compaction in each system of interbeds. Compaction values computed by the package are added to values in this array so that printed or stored values of compaction and land subsidence may include previous components. Values in this array do not affect calculations of storage changes or resulting compaction. For simulations in which output values

will reflect compaction and subsidence since the start of the simulation, enter zero values for all elements of this array. (default is 0.).

- **pcsoff** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - pcsoff is an array specifying the offset from initial effective stress to initial preconsolidation stress at the bottom of the model layer in units of height of a column of water. pcsoff is not used if istpcs=0. (default is 0.).
- **pcs** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - pcs is an array specifying the initial preconsolidation stress, in units of height of a column of water, at the bottom of the model layer. pcs is not used if istpcs is not equal to 0. (default is 0.).
- **ids16** (*list* or array of ints (26)) - Format codes and unit numbers for swtsidence, compaction by model layer, compaction by interbed system, vertical displacement, preconsolidation stress, change in preconsolidation stress, geostatic stress, change in geostatic stress, effective stress, void ration, thickness of compressible sediments, and layer-center elevation will be printed. If ids16 is None and iswtoc>0 then print code 0 will be used for all data which is output to the binary swtsidence output file (unit=1054). The 26 entries in ids16 correspond to ifm1, iun1, ifm2, iun2, ifm3, iun3, ifm4, iun4, ifm5, iun5, ifm6, iun6, ifm7, iun7, ifm8, iun8, ifm9, iun9, ifm10, iun11, ifm12, iun12, ifm13, and iun13 variables. (default is None).
- **ids17** (*list* or array of ints (*iswtoc*, 30)) - Stress period and time step range and print and save flags used to control printing and saving of information generated by the SUB-WT Package during program execution. Each row of ids17 corresponds to isp1, isp2, its1, its2, ifl1, ifl2, ifl3, ifl4, ifl5, ifl6, ifl7, ifl8, ifl9, ifl10, ifl11, ifl12, ifl13, ifl14, ifl15, ifl16, ifl17, ifl18, ifl19, ifl20, ifl21, ifl22, ifl23, ifl24, ifl25, and ifl26 variables for iswtoc entries. isp1, isp2, its1, and its2 are stress period and time step ranges. ifl1 and ifl2 control subsidence printing and saving. ifl3 and ifl4 control compaction by model layer printing and saving. ifl5 and ifl6 control compaction by interbed system printing and saving. ifl7 and ifl8 control vertical displacement printing and saving. ifl9 and ifl10 control preconsolidation stress printing and saving. ifl11 and ifl12 control change in preconsolidation stress printing and saving. ifl13 and ifl14 control geostatic stress printing and saving. ifl15 and ifl16 control change in geostatic stress printing and saving. ifl17 and ifl18 control effective stress printing and saving. ifl19 and ifl20 control change in effective stress printing and saving. ifl21 and ifl22 control void ratio printing and saving. ifl23 and ifl24 control compressible bed thickness printing and saving. ifl25 and ifl26 control layer-center elevation printing and saving. If ids17 is None and iswtoc>0 then all available subsidence output will be printed and saved to the binary subsidence output file (unit=1054). (default is None).
- **unitnumber** (*int*) - File unit number (default is None).

- **filenames** (*str* or *list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output name and other swt output files will be created using the model name and .cbc and swt output extensions (for example, modflowtest.cbc), if ipakcbc and other swt output files (dataset 16) are numbers greater than zero. If a single string is passed the package name will be set to the string and other swt output files will be set to the model name with the appropriate output file extensions. To define the names for all package files (input and output) the length of the list of strings should be 15. Default is None.

Notes

Parameters are supported in Flopy only when reading in existing models. Parameter values are converted to native values in Flopy and the connection to “parameters” is thus nonexistent. Parameters are not supported in the SUB-WT Package.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> swt = flopy.modflow.ModflowSwt(m)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename* or *file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

Returns

swt

Return type

ModflowSwt object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> swt = flopy.modflow.ModflowSwt.load('test.swt', m)
```

write_file(*f=None*)

Write the package file.

Return type

None

flopy.modflow.mfupw module

mfupw module. Contains the ModflowUpw class. Note that the user can access the ModflowUpw class as *flopy.modflow.ModflowUpw*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowUpw(model, laytyp=0, layavg=0, chani=1.0, layvka=0, laywet=0, ipakcb=None,
                 hdry=-1e+30, iphdry=0, hk=1.0, hani=1.0, vka=1.0, ss=1e-05, sy=0.15,
                 vkcb=0.0, noparcheck=False, extension='upw', unitnumber=None,
                 filenames=None)
```

Bases: [Package](#)

Upstream weighting package class

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0).
- **hdry** (*float*) - Is the head that is assigned to cells that are converted to dry during a simulation. Although this value plays no role in the model calculations, it is useful as an indicator when looking at the resulting heads that are output from the model. HDry is thus similar to HNOFLO in the Basic Package, which is the value assigned to cells that are no-flow cells at the start of a model simulation. (default is -1.e30).
- **iphdry** (*int*) - iphdry is a flag that indicates whether groundwater head will be set to hdry when the groundwater head is less than 0.0001 above the cell bottom (units defined by lenuni in the discretization package). If iphdry=0, then head will not be set to hdry. If iphdry>0, then head will be set to hdry. If the head solution from one simulation will be used as starting heads for a subsequent simulation, or if the Observation Process is used (Harbaugh and others, 2000), then hdry should not be printed to the output file for dry cells (that is, the upw package input variable should be set as iphdry=0). (default is 0)

- **noparcheck** (*bool*) - noparcheck turns off the checking that a value is defined for all cells when parameters are used to define layer data.
- **laytyp** (*int* or array of ints (*nlay*)) - Layer type (default is 0).
- **layavg** (*int* or array of ints (*nlay*)) - Layer average (default is 0). 0 is harmonic mean 1 is logarithmic mean 2 is arithmetic mean of saturated thickness and logarithmic mean of hydraulic conductivity
- **chani** (*float* or array of floats (*nlay*)) - contains a value for each layer that is a flag or the horizontal anisotropy. If CHANI is less than or equal to 0, then variable HANI defines horizontal anisotropy. If CHANI is greater than 0, then CHANI is the horizontal anisotropy for the entire layer, and HANI is not read. If any HANI parameters are used, CHANI for all layers must be less than or equal to 0. Use as many records as needed to enter a value of CHANI for each layer. The horizontal anisotropy is the ratio of the hydraulic conductivity along columns (the Y direction) to the hydraulic conductivity along rows (the X direction).
- **layvka** (*int* or array of ints (*nlay*)) - a flag for each layer that indicates whether variable VKA is vertical hydraulic conductivity or the ratio of horizontal to vertical hydraulic conductivity.
- **laywet** (*int* or array of ints (*nlay*)) - contains a flag for each layer that indicates if wetting is active. laywet should always be zero for the UPW Package because all cells initially active are wettable.
- **hk** (*float* or array of floats (*nlay, nrow, ncol*)) - is the hydraulic conductivity along rows. HK is multiplied by horizontal anisotropy (see CHANI and HANI) to obtain hydraulic conductivity along columns. (default is 1.0).
- **hani** (*float* or array of floats (*nlay, nrow, ncol*)) - is the ratio of hydraulic conductivity along columns to hydraulic conductivity along rows, where HK of item 10 specifies the hydraulic conductivity along rows. Thus, the hydraulic conductivity along columns is the product of the values in HK and HANI. (default is 1.0).
- **vka** (*float* or array of floats (*nlay, nrow, ncol*)) - is either vertical hydraulic conductivity or the ratio of horizontal to vertical hydraulic conductivity depending on the value of LAYVKA. (default is 1.0).
- **ss** (*float* or array of floats (*nlay, nrow, ncol*)) - is specific storage unless the STORAGECOEFFICIENT option is used. When STORAGECOEFFICIENT is used, Ss is confined storage coefficient. (default is 1.e-5).
- **sy** (*float* or array of floats (*nlay, nrow, ncol*)) - is specific yield. (default is 0.15).
- **vkcb** (*float* or array of floats (*nlay, nrow, ncol*)) - is the vertical hydraulic conductivity of a Quasi-three-dimensional

- confining bed below a layer. (default is 0.0).
- **extension** (*string*) - Filename extension (default is 'upw')
 - **unitnumber** (*int*) - File unit number (default is None).
 - **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output name will be created using the model name and .cbc extension (for example, modflowtest.cbc), if ipakcbc is a number greater than zero. If a single string is passed the package will be set to the string and cbc output name will be created using the model name and .cbc extension, if ipakcbc is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lpf = flopy.modflow.ModflowLpf(m)
```

classmethod `load(f, model, ext_unit_dict=None, check=True)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.
- **check** (*boolean*) - Check package data for common errors. (default True)

Returns

dis - ModflowLpf object.

Return type

ModflowUPW object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> upw = flopy.modflow.ModflowUpw.load('test.upw', m)
```

`write_file(check=True, f=None)`

Write the package file.

Parameters

check (*boolean*) - Check package data for common errors. (default True)

Return type

None

`flopy.modflow.mfuzf1` module

mfuzf1 module. Contains the `ModflowUzf1` class. Note that the user can access the `ModflowUzf1` class as `flopy.modflow.ModflowUzf1`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowUzf1(model, nuztop=1, iuzfopt=0, irunflg=0, ietflg=0, ipakcb=None,
    iuzfcb2=None, ntrail2=10, nsets=20, surfdep=1.0, iuzfbnd=1, irunbnd=0,
    vks=1e-06, eps=3.5, thts=0.35, thtr=0.15, thti=0.2, specifythtr=False,
    specifythti=False, nosurfleak=False, finf=1e-08, pet=5e-08, extdp=15.0,
    extwc=0.1, air_entry=0.0, hroot=0.0, rootact=0.0, nwt_11_fmt=False,
    specifysurfk=False, rejectsurfk=False, seepsurfk=False, etsquare=None,
    netflux=None, capillaryuzet=False, nuzgag=None, uzgag=None,
    extension='uzf', unitnumber=None, filenames=None, options=None,
    surfk=0.1)
```

Bases: [Package](#)

MODFLOW Unsaturated Zone Flow 1 Boundary Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **nuztop** (*integer*) - used to define which cell in a vertical column that recharge and discharge is simulated. (default is 1)
 - 1 Recharge to and discharge from only the top model layer.** This option assumes land surface is defined as top of layer 1.
 - 2 Recharge to and discharge from the specified layer in variable IUZFBND.** This option assumes land surface is defined as top of layer specified in IUZFBND.
 - 3 Recharge to and discharge from the highest active cell in each vertical column.** Land surface is determined as top of layer specified in IUZFBND. A constant head node intercepts any recharge and prevents deeper percolation.
- **iuzfopt** (*integer*) - equal to 1 or 2. A value of 1 indicates that the vertical hydraulic conductivity will be specified within the

UZF1 Package input file using array VKS. A value of 2 indicates that the vertical hydraulic conductivity will be specified within either the BCF or LPF Package input file. (default is 0)

- **irunflg** (*integer*) - specifies whether ground water that discharges to land surface will be routed to stream segments or lakes as specified in the IRUNBND array (IRUNFLG not equal to zero) or if ground-water discharge is removed from the model simulation and accounted for in the ground-water budget as a loss of water (IRUNFLG=0). The Streamflow-Routing (SFR2) and(or) the Lake (LAK3) Packages must be active if IRUNFLG is not zero. (default is 0)
- **ietflg** (*integer*) - specifies whether or not evapotranspiration (ET) will be simulated. ET will not be simulated if IETFLG is zero, otherwise it will be simulated. (default is 0)
- **ipakcb** (*integer*) - flag for writing ground-water recharge, ET, and ground-water discharge to land surface rates to a separate unformatted file using subroutine UBUDSV. If ipakcb>0, it is the unit number to which the cell-by-cell rates will be written when 'SAVE BUDGET' or a non-zero value for ICBCFL is specified in Output Control. If ipakcb less than or equal to 0, cell-by-cell rates will not be written to a file. (default is 57)
- **iuzfcb2** (*integer*) - flag for writing ground-water recharge, ET, and ground-water discharge to land surface rates to a separate unformatted file using module UBDSV3. If IUZFCB2>0, it is the unit number to which cell-by-cell rates will be written when 'SAVE BUDGET' or a non-zero value for ICBCFL is specified in Output Control. If IUZFCB2 less than or equal to 0, cell-by-cell rates will not be written to file. (default is 0)
- **ntrail2** (*integer*) - equal to the number of trailing waves used to define the water-content profile following a decrease in the infiltration rate. The number of trailing waves varies depending on the problem, but a range between 10 and 20 is usually adequate. More trailing waves may decrease mass-balance error and will increase computational requirements and memory usage. (default is 10)
- **nsets** (*integer*) - equal to the number of wave sets used to simulate multiple infiltration periods. The number of wave sets should be set to 20 for most problems involving time varying infiltration. The total number of waves allowed within an unsaturated zone cell is equal to NTRAIL2*NSETS2. An error will occur if the number of waves in a cell exceeds this value. (default is 20)
- **surfdep** (*float*) - The average height of undulations, D (Figure 1 in UZF documentation), in the land surface altitude. (default is 1.0)
- **iuzfbnd** (*integer*) - used to define the aerial extent of the active model in which recharge and discharge will be simulated. (default is 1)
- **irunbnd** (*integer*) - used to define the stream segments within

- the Streamflow-Routing (SFR2) Package or lake numbers in the Lake (LAK3) Package that overland runoff from excess infiltration and ground-water discharge to land surface will be added. A positive integer value identifies the stream segment and a negative integer value identifies the lake number. (default is 0)
- **vks** (*float*) - used to define the saturated vertical hydraulic conductivity of the unsaturated zone (LT-1). (default is 1.0E-6)
 - **eps** (*float*) - values for each model cell used to define the Brooks-Corey epsilon of the unsaturated zone. Epsilon is used in the relation of water content to hydraulic conductivity (Brooks and Corey, 1966). (default is 3.5)
 - **thts** (*float*) - used to define the saturated water content of the unsaturated zone in units of volume of water to total volume (L3L-3). (default is 0.35)
 - **thtr** (*float*) - used to define the residual water content for each vertical column of cells in units of volume of water to total volume (L3L-3). THTR is the irreducible water content and the unsaturated water content cannot drain to water contents less than THTR. This variable is not included unless the key word SPECIFYTHTR is specified. (default is 0.15)
 - **thti** (*float*) - used to define the initial water content for each vertical column of cells in units of volume of water at start of simulation to total volume (L3L-3). THTI should not be specified for steady-state simulations. (default is 0.20)
 - **row_col_iftunit_iuzopt** (*list*) - used to specify where information will be printed for each time step. row and col are zero-based. IUZOPT specifies what that information will be. (default is [])
IUZOPT is
 - 1 Prints time, ground-water head, and thickness of unsaturated zone, and cumulative volumes of infiltration, recharge, storage, change in storage and ground-water discharge to land surface.
 - 2 Same as option 1 except rates of infiltration, recharge, change in storage, and ground-water discharge also are printed.
 - 3 Prints time, ground-water head, thickness of unsaturated zone, followed by a series of depths and water contents in the unsaturated zone.
 - **nwt_11_fmt** (*boolean*) - flag indicating whether or not to utilize a newer (MODFLOW-NWT version 1.1 or later) format style, i.e., uzfl optional variables appear line-by-line rather than in a specific order on a single line. True means that optional variables (e.g., SPECIFYTHTR, SPECIFYTHTI, NOSURFLEAK) appear on new lines. True also supports a number of newer optional variables (e.g., SPECIFYSURFK, REJECTSURFK, SEEPSURFK). False means that optional variables appear on one line. (default is False)
 - **specifythtr** (*boolean*) - key word for specifying optional input variable THTR (default is 0)
 - **specifythti** (*boolean*) - key word for specifying optional input variable THTI. (default is 0)

- **nosurfleak** (*boolean*) - key word for inactivating calculation of surface leakage. (default is 0)
- **specifysurfk** (*boolean*) - (MODFLOW-NWT version 1.1 and MODFLOW-2005 1.12 or later) An optional character variable. When SPECIFYSURFK is specified, the variable SURFK is specified in Data Set 4b.
- **rejectsurfk** (*boolean*) - (MODFLOW-NWT version 1.1 and MODFLOW-2005 1.12 or later) An optional character variable. When REJECTSURFK is specified, SURFK instead of VKS is used for calculating rejected infiltration. REJECTSURFK only is included if SPECIFYSURFK is included.
- **seepsurfk** (*boolean*) - (MODFLOW-NWT version 1.1 and MODFLOW-2005 1.12 or later) An optional character variable. When SEEPSURFK is specified, SURFK instead of VKS is used for calculating surface leakage. SEEPSURFK only is included if SPECIFYSURFK is included.
- **etsquare** (*float (smoothfact)*) - (MODFLOW-NWT version 1.1 and MODFLOW-2005 1.12 or later) An optional character variable. When ETSQUARE is specified, groundwater ET is simulated using a constant potential ET rate, and is smoothed over a specified smoothing interval. This option is recommended only when using the NWT solver.

etsquare is activated in flopy by specifying a real value for smoothfact (default is None). For example, if the interval factor (smoothfact) is specified as smoothfact=0.1 (recommended), then the smoothing interval will be calculated as: $SMOOTHINT = 0.1 * EXTDP$ and is applied over the range for groundwater head (h): * $h < CELTOP - EXTDP$, ET is zero; * $CELTOP - EXTDP < h < CELTOP - EXTDP + SMOOTHINT$, ET is smoothed; $CELTOP - EXTDP + SMOOTHINT < h$, ET is equal to potential ET.

- **uzgage** (*dict of lists or list of lists*) - Dataset 8 in UZF Package documentation. Each entry in the dict is keyed by iftunit.

Dict of lists: If iftunit is negative, the list is empty. If iftunit is positive, the list includes [IUZROW, IUZCOL, IUZOPT] List of lists: Lists follow the format described in the documentation: [[IUZROW, IUZCOL, IFTUNIT, IUZOPT]] or [[-IFTUNIT]]

- **netflux** (*list of [Unitrech (int), Unitdis (int)]*) - (MODFLOW-NWT version 1.1 and MODFLOW-2005 1.12 or later) An optional character variable. When NETFLUX is specified, the sum of recharge (L3/T) and the sum of discharge (L3/T) is written to separate unformatted files using module UBDSV3.

netflux is activated in flopy by specifying a list for Unitrech and Unitdis (default is None). Unitrech and Unitdis are the unit numbers to which these values are written when "SAVE BUDGET" is specified in Output Control. Values written to Unitrech are the sum of recharge values for the UZF, SFR2, and LAK packages, and values written to Unitdis are the sum of discharge values for the UZF, SFR2, and LAK packages. Values are averaged over the period between output times.

[NETFLUX unitrech unitdis]

- **finf** (*float*, 2-D array, or *dict* of {*kper*:value}) - where *kper* is the zero-based stress period to assign a value to. Value should be cast-able to Util2d instance can be a scalar, list, or ndarray is the array value is constant in time. Used to define the infiltration rates (LT-1) at land surface for each vertical column of cells. If FINF is specified as being greater than the vertical hydraulic conductivity then FINF is set equal to the vertical unsaturated hydraulic conductivity. Excess water is routed to streams or lakes when IRUNFLG is not zero, and if SFR2 or LAK3 is active. (default is 1.0E-8)
- **pet** (*float*, 2-D array, or *dict* of {*kper*:value}) - where *kper* is the zero-based stress period to assign a value to. Value should be cast-able to Util2d instance can be a scalar, list, or ndarray is the array value is constant in time. Used to define the ET demand rates (L1T-1) within the ET extinction depth interval for each vertical column of cells. (default is 5.0E-8)
- **extdp** (*float*, 2-D array, or *dict* of {*kper*:value}) - where *kper* is the zero-based stress period to assign a value to. Value should be cast-able to Util2d instance can be a scalar, list, or ndarray is the array value is constant in time. Used to define the ET extinction depths. The quantity of ET removed from a cell is limited by the volume of water stored in the unsaturated zone above the extinction depth. If ground water is within the ET extinction depth, then the rate removed is based on a linear decrease in the maximum rate at land surface and zero at the ET extinction depth. The linear decrease is the same method used in the Evapotranspiration Package (McDonald and Harbaugh, 1988, chap. 10). (default is 15.0)
- **extwc** (*float*, 2-D array, or *dict* of {*kper*:value}) -

where *kper* is the zero-based stress period

to assign a value to. Value should be cast-able to Util2d instance can be a scalar, list, or ndarray is the array value is constant in time. Used to define the extinction water content below which ET cannot be removed from the unsaturated zone. EXTWC must have a value between (THTS-Sy) and THTS, where Sy is the specific yield specified in either the LPF or BCF Package. (default is 0.1)

air_entry

[*float*, 2-D array, or *dict* of {*kper*: value}] where *kper* is the zero-based stress period. Used to define the air entry pressure of the unsaturated zone in MODFLOW-NWT simulations

- **hroot** (*float*, 2-D array, or *dict* of {*kper*: value}) - where *kper* is the zero-based stress period. Used to define the pressure potential at roots in the unsaturated zone in MODFLOW-NWT simulations
- **rootact** (*float*, 2-D array, or *dict* of {*kper*: value}) - where *kper* is the zero-based stress period. Used to define the root activity in the unsaturated zone in MODFLOW-NWT simulations

- **uzfbud_ext** (*list*) - appears to be used for sequential naming of budget output files (default is [])
- **extension** (*string*) - Filename extension (default is 'uzf')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output, uzf output, and uzf observation names will be created using the model name and .cbc, uzfcb2.bin, and .uzf#.out extensions (for example, modflowtest.cbc, and modflowtest.uzfcd2.bin), if ipakcbc, iuzfcb2, and len(uzgag) are numbers greater than zero. For uzf observations the file extension is created using the uzf observation file unit number (for example, for uzf observations written to unit 123 the file extension would be .uzf123.out). If a single string is passed the package name will be set to the string and other uzf output files will be set to the model name with the appropriate output file extensions. To define the names for all package files (input and output) the length of the list of strings should be 3 + len(uzgag). Default is None.
- **surfk** (*float*) - An optional array of positive real values used to define the hydraulic conductivity (LT-1). SURFK is used for calculating the rejected infiltration and/or surface leakage. IF SURFK is set greater than VKS then it is set equal to VKS. Only used if SEEPSURFK is True.

nuzgag

equal to the number of cells (one per vertical column) that will be specified for printing detailed information on the unsaturated zone water budget and water content. A gage also may be used to print the budget summed over all model cells. (default is None)

Type

integer (deprecated - counter is set based on length of uzgag)

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> m1 = flopy.modflow.Modflow()
>>> uzf = flopy.modflow.ModflowUzf1(m1, ...)
```

classmethod load(*f, model, ext_unit_dict=None, check=False*)

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.

- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

uzf - ModflowUZF1 object.

Return type

ModflowUZF1 object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> uzf = flopy.modflow.ModflowUZF1.load('test.uzf', m)
```

property nuzgag

Number of uzf gages

Returns

nuzgag - Number of uzf gages

Return type

`int`

property uzgag

Get the uzf gage data

Returns

uzgag - Dictionary containing uzf gage data for each gage

Return type

`dict`

write_file(*f=None*)

Write the package file.

Return type

`None`

flopy.modflow.mfwel module

mfwel module. Contains the ModflowWel class. Note that the user can access the ModflowWel class as `flopy.modflow.ModflowWel`.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

```
class ModflowWel(model, ipakcb=None, stress_period_data=None, dtype=None, extension='wel',
                  options=None, binary=False, unitnumber=None, filenames=None,
                  add_package=True)
```


Bases: [Package](#)

MODFLOW Well Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.modflow.mf.Modflow](#)) to which this package will be added.
- **ipakcb** (*int*) - A flag that is used to determine if cell-by-cell budget data should be saved. If ipakcb is non-zero cell-by-cell budget data will be saved. (default is 0).
- **stress_period_data** (*list of boundaries, or recarray of boundaries, or*) - dictionary of boundaries Each well is defined through definition of layer (int), row (int), column (int), flux (float). The simplest form is a dictionary with a lists of boundaries for each stress period, where each list of boundaries itself is a list of boundaries. Indices of the dictionary are the numbers of the stress period. This gives the form of:

```
stress_period_data = {0: [
    [lay, row, col, flux], [lay, row, col, flux], [lay,
    row, col, flux] ],
    1: [
    [lay, row, col, flux], [lay, row, col, flux], [lay,
    row, col, flux] ], ...
    kper:
    [ [lay, row, col, flux], [lay, row, col, flux], [lay,
    row, col, flux] ]
}
```

Note that if the number of lists is smaller than the number of stress periods, then the last list of wells will apply until the end of the simulation. Full details of all options to specify stress_period_data can be found in the flopy3 boundaries Notebook in the basic subdirectory of the examples directory

- **dtype** (*custom datatype of stress_period_data.*) - If None the default well datatype will be applied (default is None).
- **extension** (*string*) - Filename extension (default is 'wel')
- **options** (*list of strings*) - Package options (default is None).
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the cbc output name will be created using the model name and .cbc extension (for example, modflowtest.cbc), if ipakcbc is a number greater than zero. If a single string is passed the package will be set to the string and cbc output names will be created using the model name and .cbc extension, if ipakcbc is a number greater than zero. To define the names for all package files (input and

output) the length of the list of strings should be 2. Default is None.

- **add_package** (*bool*) - Flag to add the initialised package object to the parent model object. Default is True.

mxactw

Maximum number of wells for a stress period. This is calculated automatically by FloPy based on the information in stress_period_data.

Type

int

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> lrcq = {0:[[2, 3, 4, -100.]], 1:[[2, 3, 4, -100.]]}
>>> wel = flopy.modflow.ModflowWel(m, stress_period_data=lrcq)
```

add_record(*kper*, *index*, *values*)

static get_default_dtype(*structured=True*)

static get_empty(*ncells=0*, *aux_names=None*, *structured=True*)

classmethod load(*f*, *model*, *nper=None*, *ext_unit_dict=None*, *check=True*)

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **nper** (*int*) - The number of stress periods. If nper is None, then nper will be obtained from the model object. (default is None).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case ext_unit_dict is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

Returns

wel - ModflowWel object.

Return type

ModflowWel object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> wel = flopy.modflow.ModflowWel.load('test.wel', m)
```

write_file(*f=None*)

Write the package file.

Parameters

f - (str) optional file name

Return type

None

flopy.modflow.mfzon module

mfzon module. Contains the ModflowZone class. Note that the user can access the ModflowZone class as *flopy.modflow.ModflowZone*.

Additional information for this MODFLOW package can be found at the [Online MODFLOW Guide](#).

class ModflowZon(*model*, *zone_dict=None*, *extension='zon'*, *unitnumber=None*, *filenames=None*)

Bases: *Package*

MODFLOW Zone Package Class.

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **zone_dict** (*dict*) - Dictionary with zone data for the model. zone_dict is typically instantiated using load method.
- **extension** (*string*) - Filename extension (default is 'zon')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Parameters are supported in Flopy only when reading in existing models. Parameter values are converted to native values in Flopy and the connection to "parameters" is thus nonexistent.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> zonedict = flopy.modflow.ModflowZon(m, zone_dict=zone_dict)
```

classmethod `load(f, model, nrow=None, ncol=None, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **nrow** (*int*) - number of rows. If not specified it will be retrieved from the model object. (default is None).
- **ncol** (*int*) - number of columns. If not specified it will be retrieved from the model object. (default is None).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

zone

Return type

ModflowZone dict

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> zon = flopy.modflow.ModflowZon.load('test.zon', m)
```

write_file()

Write the package file.

Return type

None

Notes

Not implemented because parameters are only supported on load

8.2.3 MT3DMS Packages

Contents:

`flopy.mt3d.mt` module

```
class Mt3dList(model, extension='list', listunit=7)
```

Bases: `Package`

List package class

`write_file()`

Every Package needs its own `write_file` function

```
class Mt3dms(modelname='mt3dtest', namefile_ext='nam', modflowmodel=None,
            ftlfilename='mt3d_link.ftl', ftlfree=False, version='mt3dms', exe_name='mt3dms',
            structured=True, listunit=16, ftlunit=10, model_ws='.', external_path=None,
            verbose=False, load=True, silent=0)
```

Bases: `BaseModel`

MT3DMS Model Class.

Parameters

- **modelname** (*str*, default "mt3dtest") - Name of model. This string will be used to name the MODFLOW input that are created with `write_model`.
- **namefile_ext** (*str*, default "nam") - Extension for the namefile.
- **modflowmodel** (`flopy.modflow.mf.Modflow`) - This is a flopy Modflow model object upon which this Mt3dms model is based.
- **ftlfilename** (*str*, default "mt3d_link.ftl") - Name of flow-transport link file.
- **ftlfree** (*TYPE*, default False) - If flow-link transport file is formatted (True) or unformatted (False, default).
- **version** (*str*, default "mt3dms") - Mt3d version. Choose one of: "mt3dms" (default) or "mt3d-usgs".
- **exe_name** (*str*, default "mt3dms") - The name of the executable to use.
- **structured** (*bool*, default True) - Specify if model grid is structured (default) or unstructured.
- **listunit** (*int*, default 16) - Unit number for the list file.
- **ftlunit** (*int*, default 10) - Unit number for flow-transport link file.
- **model_ws** (*str*, optional) - Model workspace. Directory name to create model data sets. Default is the present working directory.
- **external_path** (*str*, optional) - Location for external files.
- **verbose** (*bool*, default False) - Print additional information to the screen.
- **load** (*bool*, default True) - Load model.

- **silent** (*int*, default 0) - Silent option.

Notes

Examples

```
>>> import flopy
>>> m = flopy.mt3d.mt.Mt3dms()
```

get_nrow_ncol_nlay_nper()

classmethod load(*f*, version='mt3dms', exe_name='mt3dms', verbose=False, model_ws='.', load_only=None, forgive=False, modflowmodel=None)

Load an existing model.

Parameters

- **f** (*str*) - Path to MT3D name file to load.
- **version** (*str*, default "mt3dms") - Mt3d version. Choose one of: "mt3dms" (default) or "mt3d-usgs".
- **exe_name** (*str*, default "mt3dms") - The name of the executable to use.
- **verbose** (*bool*, default False) - Print information on the load process if True.
- **model_ws** (*str*, default ".") - Model workspace path. Default is the current directory.
- **load_only** (*list of str*, optional) - Packages to load (e.g. ['btn', 'adv']). Default None means that all packages will be loaded.
- **forgive** (*bool*, default False) - Option to raise exceptions on package load failure, which can be useful for debugging.
- **modflowmodel** (*flopy.modflow.mf.Modflow*, optional) - This is a flopy Modflow model object upon which this Mt3dms model is based.

Return type

flopy.mt3d.mt.Mt3dms

Notes

The load method does not retain the name for the MODFLOW-generated FTL file. This can be added manually after the MT3D model has been loaded. The syntax for doing this manually is `mt.ftlfilename = 'example.ftl'`.

Examples

```
>>> import flopy
>>> mt = flopy.mt3d.mt.Mt3dms.load('example.nam')
>>> mt.ftlfilename = 'example.ftl'
```

static load_mas(*fname*)

Load an mt3d mas file and return a numpy recarray

Parameters

fname (*str*) - name of MT3D mas file

Returns

r

Return type

np.ndarray

static load_obs(*fname*)

Load an mt3d obs file and return a numpy recarray

Parameters

fname (*str*) - name of MT3D obs file

Returns

r

Return type

np.ndarray

load_results(*kwargs*)**

property mcomp

property modelgrid

property modeltime

property ncol

property ncomp

property nlay

property nper

property nrow

property nrow_ncol_nlay_nper

property solver_tols

property sr

write_name_file()

Write the name file.

flopy.mt3d.mtadv module

```
class Mt3dAdv(model, mixelm=3, percel=0.75, mxpart=800000, nadvfd=1, itrack=3, wd=0.5,
             dceps=1e-05, nplane=2, npl=10, npf=40, npmin=5, npmax=80, nlsink=0,
             npsink=15, dchmoc=0.0001, extension='adv', unitnumber=None, filenames=None)
```

Bases: [Package](#)

MT3DMS Advection Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.mt3d.mt.Mt3dms](#)) to which this package will be added.
- **mixelm** (*int*) - MIXELM is an integer flag for the advection solution option. MIXELM = 0, the standard finite-difference method with upstream or central-in-space weighting, depending on the value of NADVFD; = 1, the forward-tracking method of characteristics (MOC); = 2, the backward-tracking modified method of characteristics (MMOC); = 3, the hybrid method of characteristics (HMOC) with MOC or MMOC automatically and dynamically selected; = -1, the third-order TVD scheme (ULTIMATE).
- **percel** (*float*) - PERCEL is the Courant number (i.e., the number of cells, or a fraction of a cell) advection will be allowed in any direction in one transport step. For implicit finite-difference or particle-tracking-based schemes, there is no limit on PERCEL, but for accuracy reasons, it is generally not set much greater than one. Note, however, that the PERCEL limit is checked over the entire model grid. Thus, even if PERCEL > 1, advection may not be more than one cell's length at most model locations. For the explicit finite-difference or the third-order TVD scheme, PERCEL is also a stability constraint which must not exceed one and will be automatically reset to one if a value greater than one is specified.
- **mxpart** (*int*) - MXPART is the maximum total number of moving particles allowed and is used only when MIXELM = 1 or 3.
- **nadvfd** (*int*) - NADVFD is an integer flag indicating which weighting scheme should be used; it is needed only when the advection term is solved using the implicit finite-difference method. NADVFD = 0 or 1, upstream weighting (default); = 2, central-in-space weighting.
- **itrack** (*int*) - ITRACK is a flag indicating which particle-tracking algorithm is selected for the Eulerian-Lagrangian methods. ITRACK = 1, the first-order Euler algorithm is used. = 2, the fourth-order Runge-Kutta algorithm is used; this option is computationally demanding and may be needed only when PERCEL is set greater than one. = 3, the hybrid first- and fourth-order algorithm is used; the Runge-Kutta algorithm is used in sink/source cells and the cells next to sinks/sources while the Euler algorithm is used elsewhere.
- **wd** (*float*) - is a concentration weighting factor between 0.5 and 1. It is used for operator splitting in the particle-tracking-based methods. The value of 0.5 is generally adequate. The value of WD may be adjusted to achieve better mass balance.

Generally, it can be increased toward 1.0 as advection becomes more dominant.

- **dceps** (*float*) - is a small Relative Cell Concentration Gradient below which advective transport is considered
- **nplane** (*int*) - NPLANE is a flag indicating whether the random or fixed pattern is selected for initial placement of moving particles. If NPLANE = 0, the random pattern is selected for initial placement. Particles are distributed randomly in both the horizontal and vertical directions by calling a random number generator (Figure 18b). This option is usually preferred and leads to smaller mass balance discrepancy in nonuniform or diverging/converging flow fields. If NPLANE > 0, the fixed pattern is selected for initial placement. The value of NPLANE serves as the number of vertical 'planes' on which initial particles are placed within each cell block (Figure 18a). The fixed pattern may work better than the random pattern only in relatively uniform flow fields. For two-dimensional simulations in plan view, set NPLANE = 1. For cross sectional or three-dimensional simulations, NPLANE = 2 is normally adequate. Increase NPLANE if more resolution in the vertical direction is desired.
- **npl** (*int*) - NPL is the number of initial particles per cell to be placed at cells where the Relative Cell Concentration Gradient is less than or equal to DCEPS. Generally, NPL can be set to zero since advection is considered insignificant when the Relative Cell Concentration Gradient is less than or equal to DCEPS. Setting NPL equal to NPH causes a uniform number of particles to be placed in every cell over the entire grid (i.e., the uniform approach).
- **nph** (*int*) - NPH is the number of initial particles per cell to be placed at cells where the Relative Cell Concentration Gradient is greater than DCEPS. The selection of NPH depends on the nature of the flow field and also the computer memory limitation. Generally, a smaller number should be used in relatively uniform flow fields and a larger number should be used in relatively nonuniform flow fields. However, values exceeding 16 in two-dimensional simulation or 32 in three-dimensional simulation are rarely necessary. If the random pattern is chosen, NPH particles are randomly distributed within the cell block. If the fixed pattern is chosen, NPH is divided by NPLANE to yield the number of particles to be placed per vertical plane, which is rounded to one of the values shown in Figure 30.
- **npmin** (*int*) - is the minimum number of particles allowed per cell. If the number of particles in a cell at the end of a transport step is fewer than NPMIN, new particles are inserted into that cell to maintain a sufficient number of particles. NPMIN can be set to zero in relatively uniform flow fields and to a number greater than zero in diverging/converging flow fields. Generally, a value between zero and four is adequate.
- **npmax** (*int*) - NPMAX is the maximum number of particles allowed per cell. If the number of particles in a cell exceeds NPMAX, all particles are removed from that cell and replaced by a new set of

particles equal to NPH to maintain mass balance. Generally, NPMAX can be set to approximately two times of NPH.

- **interp** (*int*) - is a flag indicating the concentration interpolation method for use in the MMOC scheme. Currently, only linear interpolation is implemented.
- **nlsink** (*int*) - is a flag indicating whether the random or fixed pattern is selected for initial placement of particles to approximate sink cells in the MMOC scheme. The convention is the same as that for NPLANE. It is generally adequate to set NLSINK equivalent to NPLANE.
- **npsink** (*int*) - is the number of particles used to approximate sink cells in the MMOC scheme. The convention is the same as that for NPH. It is generally adequate to set NPSINK equivalent to NPH.
- **dchmoc** (*float*) - DCHMOC is the critical Relative Concentration Gradient for controlling the selective use of either MOC or MMOC in the HMOC solution scheme. The MOC solution is selected at cells where the Relative Concentration Gradient is greater than DCHMOC. The MMOC solution is selected at cells where the Relative Concentration Gradient is less than or equal to DCHMOC.
- **extension** (*string*) - Filename extension (default is 'adv')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Examples

```
>>> import flopy
>>> m = flopy.mt3d.Mt3dms()
>>> adv = flopy.mt3d.Mt3dAdv(m)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.mt3d.Mt3dms`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

adv - `Mt3dAdv` object.

Return type

Mt3dAdv object

Examples

```
>>> import flopy
>>> mt = flopy.mt3d.Mt3dms()
>>> adv = flopy.mt3d.Mt3dAdv.load('test.adv', m)
```

write_file()

Write the package file

Return type

None

flopy.mt3d.mtbtn module

mtbtn module. Contains the Mt3dBtn class. Note that the user can access the Mt3dBtn class as *flopy.mt3d.Mt3dBtn*.

Additional information for this MT3DMS package can be found in the MT3DMS User's Manual.

```
class Mt3dBtn(model, MFStyleArr=False, DRYCell=False, Legacy99Stor=False, FTLPrint=False,
              NoWetDryPrint=False, OmitDryBud=False, AltWTSorb=False, nlay=None,
              nrow=None, ncol=None, nper=None, ncomp=1, mcomp=1, tunit='D', lunit='M',
              munit='KG', laycon=None, delr=None, delc=None, htop=None, dz=None,
              prsity=0.3, icbund=1, sconc=0.0, cinact=1e+30, thkmin=0.01, ifmtcn=0,
              ifmtnp=0, ifmtrf=0, ifmtdp=0, savucn=True, nprs=0, timprs=None, obs=None,
              nprobs=1, chkmas=True, nprmas=1, perlen=None, nstp=None, tsmult=None,
              ssflag=None, dt0=0, mxstrn=50000, ttsmult=1.0, ttsmax=0,
              species_names=None, extension='btn', unitnumber=None, filenames=None,
              **kwargs)
```

Bases: [Package](#)

Basic Transport Package Class.

Parameters

- **model** (*model object*) - The model object (of type *flopy.mt3dms.mt.Mt3dms*) to which this package will be added.
- **MFStyleArr** (*str*) - Specifies whether or not to read arrays using the MODFLOW array reader format or the original MT3DMS array reader
- **DRYCell** (*str*) - Specifies whether or not to route mass through dry cells. When MF-NWT is used to generate the flow-transport link file, this is a distinct possibility.
- **Legacy99Stor** (*str*) - Specifies whether or not to use the storage formulation used in MT3DMS
- **FTLPrint** (*str*) - Specifies if flow-transport link terms (cell-by-cell flows) should be echoed to the MT3D-USGS listing file.

- **NoWetDryPrint** (*str*) - Specifies whether or not to suppress wet/dry messaging in the MT3D-USGS listing file.
- **OmitDryBudg** (*str*) - Specifies whether or not to include the mass flux terms through dry cells in the mass budget written to the listing file.
- **AltWTSorb** (*str*) - Specifies whether or not to use the MT3DMS formulation (this keyword omitted) for the solid phase, whereby the entire cell thickness is available for interacting with the aqueous phase, even though the aqueous phase may only occupy a portion of the cell's thickness. When used, only the saturated portion of the cell is available for sorbing
- **ncomp** (*int*) - The total number of chemical species in the simulation. (default is None, will be changed to 1 if sconc is single value)
- **mcomp** (*int*) - The total number of 'mobile' species (default is 1). mcomp must be equal or less than ncomp.
- **tunit** (*str*) - The name of unit for time (default is 'D', for 'days'). Used for identification purposes only.
- **lunit** (*str*) - The name of unit for length (default is 'M', for 'meters'). Used for identification purposes only.
- **munit** (*str*) - The name of unit for mass (default is 'KG', for 'kilograms'). Used for identification purposes only.
- **prsim** (*float* or array of floats (*nlay, nrow, ncol*)) - The effective porosity of the porous medium in a single porosity system, or the mobile porosity in a dual-porosity medium (the immobile porosity is defined through the Chemical Reaction Package. (default is 0.25).
- **icbund** (*int* or array of ints (*nlay, nrow, ncol*)) - The icbund array specifies the boundary condition type for solute species (shared by all species). If icbund = 0, the cell is an inactive concentration cell; If icbund < 0, the cell is a constant-concentration cell; If icbund > 0, the cell is an active concentration cell where the concentration value will be calculated. (default is 1).
- **sconc** (*float*, array of (*nlay, nrow, ncol*), or filename) - sconc is the starting concentration for the first species. To specify starting concentrations for other species in a multi-species simulation, include additional keywords, such as sconc2, sconc3, and so forth.
- **cinact** (*float*) - The value for indicating an inactive concentration cell. (default is 1e30).
- **thkmin** (*float*) - The minimum saturated thickness in a cell, expressed as the decimal fraction of its thickness, below which the cell is considered inactive. (default is 0.01).
- **ifmtcn** (*int*) - A flag/format code indicating how the calculated concentration should be printed to the standard output text file. Format codes for printing are listed in Table 3 of the

MT3DMS manual. If `ifmtcn > 0` printing is in wrap form; `ifmtcn < 0` printing is in strip form; if `ifmtcn = 0` concentrations are not printed. (default is 0).

- **ifmtnp** (*int*) - A flag/format code indicating how the number of particles should be printed to the standard output text file. The convention is the same as for `ifmtcn`. (default is 0).
- **ifmtrf** (*int*) - A flag/format code indicating how the calculated retardation factor should be printed to the standard output text file. The convention is the same as for `ifmtcn`. (default is 0).
- **ifmtdp** (*int*) - A flag/format code indicating how the distance-weighted dispersion coefficient should be printed to the standard output text file. The convention is the same as for `ifmtcn`. (default is 0).
- **savucn** (*bool*) - A logical flag indicating whether the concentration solution should be saved in an unformatted file. (default is True).
- **nprs** (*int*) - A flag indicating (i) the frequency of the output and (ii) whether the output frequency is specified in terms of total elapsed simulation time or the transport step number. If `nprs > 0` results will be saved at the times as specified in `timprs`; if `nprs = 0`, results will not be saved except at the end of simulation; if `NPRS < 0`, simulation results will be saved whenever the number of transport steps is an even multiple of `nprs`. (default is 0).
- **timprs** (*list of floats*) - The total elapsed time at which the simulation results are saved. The number of entries in `timprs` must equal `nprs`. (default is None).
- **obs** (*array of int*) - An array with the cell indices (layer, row, column) for which the concentration is to be printed at every transport step. (default is None). `obs` indices must be entered as zero-based numbers as a 1 is added to them before writing to the `btn` file.
- **nprobs** (*int*) - An integer indicating how frequently the concentration at the specified observation points should be saved. (default is 1).
- **chkmas** (*bool*) - A logical flag indicating whether a one-line summary of mass balance information should be printed. (default is True).
- **nprmas** (*int*) - An integer indicating how frequently the mass budget information should be saved. (default is 1).
- **dt0** (*float*) - The user-specified initial transport step size within each time-step of the flow solution. (default is 0).
- **mxstrn** (*int*) - The maximum number of transport steps allowed within one time step of the flow solution. (default is 50000).
- **ttsmult** (*float*) - The multiplier for successive transport steps within a flow time-step if the GCG solver is used and the solution option for the advection term is the standard finite-difference method. (default is 1.0).

- **ttsmax** (*float*) - The maximum transport step size allowed when transport step size multiplier TTSMULT > 1.0. (default is 0).
- **species_names** (*list of str*) - A list of names for every species in the simulation.
- **extension** (*string*) - Filename extension (default is 'btn')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Examples

```
>>> import flopy
>>> mt = flopy.mt3dms.Mt3dms()
>>> btn = flopy.mt3dms.Mt3dBtn(mt)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.mt3d.Mt3dms`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

btn - `Mt3dBtn` object.

Return type

`Mt3dBtn` object

Examples

```
>>> import flopy
>>> mt = flopy.mt3d.Mt3dms()
>>> btn = flopy.mt3d.Mt3dBtn.load('test.btn', mt)
```

setmodflowvars(*nlay, nrow, ncol, nper, laycon, delr, delc, htop, dz, perlen, nstp, tsmult*)

Set these variables from the MODFLOW model, if it exists

`write_file()`

Write the package file

Return type

None

`flopy.mt3d.mtcts` module

`class Mt3dCts`

Bases: *Package*

MT3D-USGS Contaminant Treatment System package class

Parameters

- **model** (*model object*) - The model object (of type `flopy.mt3dms.mt.Mt3dms`) to which this package will be added.
- **mxcts** (*int*) - The maximum number of contaminant transport systems implemented in a simulation.
- **ictsout** (*int*) - The unit number on which well-by-well output information is written. The default file extension assigned to the output file is TSO
- **mxext** (*int*) - The maximum number of extraction wells specified as part of a contaminant treatment system
- **mxinj** (*int*) - The maximum number of injection wells specified as part of a contaminant treatment system
- **mxwel** (*int*) - The maximum number of wells in the flow model. MXWEL is recommended to be set equal to MXWEL as specified in the WEL file
- **iforce** (*int*) - A flag to force concentration in treatment systems to satisfy specified concentration/mass values based on the treatment option selected without considering whether treatment is necessary or not. This flag is ignored if 'no treatment' option is selected.

0 Concentration for all injection wells is set to satisfy

treatment levels only if blended concentration exceeds the desired concentration/mass level for a treatment system. If the blended concentration in a treatment system is less than the specified concentration/mass level, then injection wells inject water with blended concentrations.

1 Concentration for all injection wells is forced to satisfy

specified concentration/mass values.

- **ncts** (*int*) - The number of contaminant treatment systems. If NCTS ≥ 0 , NCTS is the number of contaminant treatment systems. If NCTS = -1, treatment system information from the previous stress period is reused for the current stress period.
- **icts** (*int*) - The contaminant treatment system index number.
- **next** (*int*) - The number of extraction wells for the treatment system number ICTS.

- **ninj** (*int*) - The number of injection wells for the treatment system number ICTS.
- **itrtnj** (*int*) - Is the level of treatment provided for the treatment system number ICTS. Each treatment system blends concentration collected from all extraction wells contributing to the treatment system and assigns a treated concentration to all injection wells associated with that treatment system based on the treatment option selected

0 no treatment is provided 1 same level of treatment is provided to all injection wells. 2 different level of treatment can be provided to each individual injection well.
- **qincts** (*float*) - The external flow entering a treatment system. External flow may be flow entering a treatment system that is not a part of the model domain but plays an important role in influencing the blended concentration of a treatment system
- **cincts** (*float*) - The concentration with which the external flow enters a treatment system
- **ioptinj** (*int*) - Is a treatment option. Negative values indicate removal of concentration/mass and positive values indicate addition of concentration/mass.
 - 1 Percentage concentration/mass addition/removal is performed.**
Percentages must be specified as fractions. Example, for 50% concentration/mass removal is desired, -0.5 must be specified.
 - 2 Concentration is added/removed from the blended concentration.**
Specified concentration CMCHGINJ is added to the blended concentration. If the specified concentration removal, CMCHGINJ, is greater than the blended concentration, the treated concentration is set to zero.
 - 3 Mass is added/removed from the blended concentration.**
Specified mass CMCHGINJ is added to the blended concentration. If the specified mass removal, CMCHGINJ, is greater than the blended total mass, the treated concentration is set to zero.
 - 4 Specified concentration is set equal to the entered value**
CMCHGINJ. A positive value is expected for CMCHGINJ with this option.
- **cmchginj** (*float*) - Is the addition, removal, or specified concentration/mass values set for the treatment system. Concentration/mass is added, removed, or used as specified concentrations depending on the treatment option IOPTINJ. Note that concentration/mass values as specified by CMCHGINJ are enforced if the option IFORCE is set to 1. If IFORCE is set to 0, then CMCHGINJ is enforced only when the blended concentration exceeds the specified concentration CNTE.

- **cnte** (*float*) - The concentration that is not to be exceeded for a treatment system. Treatment is applied to blended concentration only if it exceeds CNTE, when IFORCE is set to 0.
- **kinj** (*int*) - Layer index for a CTS injection well
- **iinj** (*int*) - Row index for a CTS injection well
- **jinj** (*int*) - Column index for a CTS injection well
- **iwinj** (*int*) - The well index number. This number corresponds to the well number as it appears in the WEL file of the flow model.
- **qoutcts** (*float*) - the flow rate of outflow from a treatment system to an external sink. This flow rate must be specified to maintain an overall treatment system mass balance. QOUTCTS must be set equal to total inflow into a treatment system minus total outflow to all injection wells for a treatment system

Notes

Parameters are not supported in FloPy.

Examples

```
>>>
>>>
>>>
>>>
```

```
static get_default_CTS_dtype(ncomp=1, iforce=0)
```

Construct a dtype for the recarray containing the list of cts systems

```
classmethod load(f, model, nlay=None, nrow=None, ncol=None, nper=None, ncomp=None,
                 ext_unit_dict=None)
```

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.mt3d.mt.Mt3dms*) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

Returns

cts - Mt3dCts object

Return type

Mt3dCts object

Examples

```
>>>
```

`flopy.mt3d.mtdsp` module

```
class Mt3dDsp(model, al=0.01, trpt=0.1, trpv=0.01, dmcoef=1e-09, extension='dsp',
              multiDiff=False, nocross=False, unitnumber=None, filenames=None, **kwargs)
```

Bases: [Package](#)

MT3DMS Dispersion Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.mt3d.mt.Mt3dms](#)) to which this package will be added.
- **al** (*float or array of floats (nlay, nrow, ncol)*) - AL is the longitudinal dispersivity, for every cell of the model grid (unit, L). (default is 0.01)
- **trpt** (*float or array of floats (nlay)*) - is a 1D real array defining the ratio of the horizontal transverse dispersivity to the longitudinal dispersivity. Each value in the array corresponds to one model layer. Some recent field studies suggest that TRPT is generally not greater than 0.1. (default is 0.1)
- **trpv** (*float or array of floats (nlay)*) - is the ratio of the vertical transverse dispersivity to the longitudinal dispersivity. Each value in the array corresponds to one model layer. Some recent field studies suggest that TRPT is generally not greater than 0.01. Set TRPV equal to TRPT to use the standard isotropic dispersion model (Equation 10 in Chapter 2). Otherwise, the modified isotropic dispersion model is used (Equation 11 in Chapter 2). (default is 0.01)
- **dmcoef** (*float or array of floats (nlay) or (nlay, nrow, ncol) if the*) - multiDiff option is used. DMCOEF is the effective molecular diffusion coefficient (unit, L²T⁻¹). Set DMCOEF = 0 if the effect of molecular diffusion is considered unimportant. Each value in the array corresponds to one model layer. The value for dmcoef applies only to species 1. See kwargs for entering dmcoef for other species. (default is 1.e-9).
- **multiDiff** (*boolean*) - To activate the component-dependent diffusion option, a keyword input record must be inserted to the beginning of the Dispersion (DSP) input file. The symbol \$ in the first column of an input line signifies a keyword input record containing one or more predefined keywords. Above the keyword input record, comment lines marked by the symbol # in the first column are allowed. Comment lines are processed but have no effect on the simulation. Furthermore, blank lines are also acceptable above the keyword input record. Below the keyword input record, the format of the DSP input file must remain unchanged from the previous versions except for the diffusion coefficient as explained below. If no keyword input record is specified, the input file remains backward compatible with all

previous versions of MT3DMS. The predefined keyword for the component-dependent diffusion option is `MultiDiffusion`. The keyword is case insensitive so `'MultiDiffusion'` is equivalent to either `'Multidiffusion'` or `'multidiffusion'`. If this keyword is specified in the keyword input record that has been inserted into the beginning of the DSP input file, the component-dependent diffusion option has been activated and the user needs to specify one diffusion coefficient for each mobile solute component and at each model cell. This is done by specifying one mobile component at a time, from the first component to the last component (`MCOMP`). For each mobile component, the real array reader utility (`RARRAY`) is used to input the 3-D diffusion coefficient array, one model layer at a time. (default is `False`)

- **`nocross`** (*boolean*) - To disable cross-dispersion terms, a keyword input record must be input record must be inserted to the beginning of the Dispersion (DSP) input file. The symbol `$` in the first column of an input line signifies a keyword input record containing one or more predefined keywords. Above the keyword input record, comment lines marked by the symbol `#` in the first column are allowed. Comment lines are processed but have no effect on the simulation. Furthermore, blank lines are also acceptable above the keyword input record. Below the keyword input record, the format of the DSP input file must remain unchanged from the previous versions except for the diffusion coefficient as explained below. If no keyword input record is specified, the input file remains backward compatible with all previous versions of MT3DMS. The predefined keyword to turn off calculation of the cross- dispersion terms is `'nocross'`. The keyword is case insensitive so `'Nocross'` is equivalent to either `'NOCROSS'` or `'nocross'`. (default is `False`, which means cross-dispersion coefficients are calculated).
- **`extension`** (*string*) - Filename extension (default is `'dsp'`)
- **`unitnumber`** (*int*) - File unit number (default is `None`).
- **`filenames`** (*str or list of str*) - Filenames to use for the package. If `filenames=None` the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is `None`.
- **`kwargs`** (*dictionary*) - If a multi-species simulation, then `dmcoef` values can be specified for other species as `dmcoef2`, `dmcoef3`, etc. For example: `dmcoef1=1.e-10`, `dmcoef2=4.e-10`, ... If a value is not specified, then `dmcoef` is set to `0.0`.

Notes

Examples

```
>>> import flopy
>>> m = flopy.mt3d.Mt3dms()
>>> dsp = flopy.mt3d.Mt3dDsp(m)
```

classmethod `load(f, model, nlay=None, nrow=None, ncol=None, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.mt3d.Mt.Mt3dms`) to which this package will be added.
- **nlay** (*int*) - number of model layers. If None it will be retrieved from the model.
- **nrow** (*int*) - number of model rows. If None it will be retrieved from the model.
- **ncol** (*int*) - number of model columns. If None it will be retrieved from the model.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

dsk - Mt3dDsp object.

Return type

Mt3dDsp object

Examples

```
>>> import flopy
>>> mt = flopy.mt3d.Mt3dms()
>>> dsp = flopy.mt3d.Mt3dAdv.load('test.dsp', m)
```

write_file()

Write the package file

Return type

None

flopy.mt3d.mtgcg module

```
class Mt3dGcg(model, mxiter=1, iter1=50, isolve=3, ncrs=0, accl=1, cclose=1e-05,
              iprgcg=0, extension='gcg', unitnumber=None, filenames=None)
```

Bases: [Package](#)

MT3DMS Generalized Conjugate Gradient Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.mt3d.mt.Mt3dms](#)) to which this package will be added.
- **mxiter** (*int*) - is the maximum number of outer iterations; it should be set to an integer greater than one only when a nonlinear sorption isotherm is included in simulation. (default is 1)
- **iter1** (*int*) - is the maximum number of inner iterations; a value of 30-50 should be adequate for most problems. (default is 50)
- **isolve** (*int*) - is the type of preconditioners to be used with the Lanczos/ORTHOMIN acceleration scheme: = 1, Jacobi = 2, SSOR = 3, Modified Incomplete Cholesky (MIC) (MIC usually converges faster, but it needs significantly more memory) (default is 3)
- **ncrs** (*int*) - is an integer flag for treatment of dispersion tensor cross terms: = 0, lump all dispersion cross terms to the right-hand-side (approximate but highly efficient). = 1, include full dispersion tensor (memory intensive). (default is 0)
- **accl** (*float*) - is the relaxation factor for the SSOR option; a value of 1.0 is generally adequate. (default is 1)
- **cclose** (*float*) - is the convergence criterion in terms of relative concentration; a real value between 10⁻⁴ and 10⁻⁶ is generally adequate. (default is 1.E-5)
- **iprgcg** (*int*) - IPRGCG is the interval for printing the maximum concentration changes of each iteration. Set IPRGCG to zero as default for printing at the end of each stress period. (default is 0)
- **extension** (*string*) - Filename extension (default is 'gcg')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package. If filenames=None the package name will be created using the model name and package extension. If a single string is passed the package will be set to the string. Default is None.

Notes

Examples

```
>>> import flopy
>>> m = flopy.mt3d.Mt3dms()
>>> gcg = flopy.mt3d.Mt3dGcg(m)
```

classmethod `load(f, model, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.mt3d.Mt3dms`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

gcg - Mt3dGcg object.

Return type

Mt3dGcg object

Examples

```
>>> import flopy
>>> mt = flopy.mt3d.Mt3dms()
>>> gcg = flopy.mt3d.Mt3dGcg.load('test.gcg', m)
```

unitnumber = 35

write_file()

Write the package file

Return type

None

flopy.mt3d.mtlkt module

```
class Mt3dLkt(model, nlkinit=0, mxlkb=0, icbclk=None, ietlak=0, coldlak=0.0,
              lk_stress_period_data=None, dtype=None, extension='lkt', unitnumber=None,
              filenames=None, iprn=-1, **kwargs)
```

Bases: `Package`

MT3D-USGS LaKe Transport package class

Parameters

- **model** (*model object*) - The model object (of type `flopy.mt3dms.Mt3dms`) to which this package will be added.

- **nlkinit** (*int*) - is equal to the number of simulated lakes as specified in the flow simulation
- **mxlkb** (*int*) - must be greater than or equal to the sum total of boundary conditions applied to each lake
- **icbclk** (*int*) - is equal to the unit number on which lake-by-lake transport information will be printed. This unit number must appear in the NAM input file required for every MT3D-USGS simulation.
- **ietlak** (*int*) - specifies whether or not evaporation as simulated in the flow solution will act as a mass sink. = 0, Mass does not exit the model via simulated lake evaporation != 0, Mass may leave the lake via simulated lake evaporation
- **coldlak** (*array of floats*) - is a vector of real numbers representing the initial concentrations in the simulated lakes. The length of the vector is equal to the number of simulated lakes, NLKINIT. Initial lake concentrations should be in the same order as the lakes appearing in the LAK input file corresponding to the MODFLOW simulation.
- **ntmp** (*int*) - is an integer value corresponding to the number of specified lake boundary conditions to follow. For the first stress period, this value must be greater than or equal to zero, but may be less than zero in subsequent stress periods.
- **ilkbc** (*int*) - is the lake number for which the current boundary condition will be specified
- **ilkbctyp** (*int*) -

specifies what the boundary condition type is for ilakbc

- 1 a precipitation boundary. If precipitation directly to lakes**
is simulated in the flow model and a non-zero concentration (default is zero) is desired, use ISFBCTYP = 1;
 - 2 a runoff boundary condition that is not the same thing as**
runoff simulated in the UZF1 package and routed to a lake (or stream) using the IRNBND array. Users who specify runoff in the LAK input via the RNF variable appearing in record set 9a and want to assign a non-zero concentration (default is zero) associated with this specified source, use ISFBCTYP=2;
 - 3 a Pump boundary condition. Users who specify a withdrawal**
from a lake via the WTHDRW variable appearing in record set 9a and want to assign a non-zero concentration (default is zero) associated with this specified source, use ISFBCTYP=2;
 - 4 an evaporation boundary condition. In models where evaporation**
is simulated directly from the surface of the lake, users can use this boundary condition to specify a non-zero concentration (default is zero) associated with the evaporation losses.
- **extension** (*string*) - Filename extension (default is 'lkt')

- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str* or *list* of *str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the lake output name will be created using the model name and lake concentration observation extension (for example, modflowtest.cbc and modflowtest.lkcobs.out), if icbclk is a number greater than zero. If a single string is passed the package will be set to the string and lake concentration observation output name will be created using the model name and .lkcoobs.out extension, if icbclk is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> mt = flopy.mt3d.Mt3dms()
>>> lkt = flopy.mt3d.Mt3dLkt(mt)
```

static `get_default_dtype(ncomp=1)`

Construct a dtype for the recarray containing the list of boundary conditions interacting with the lake (i.e., pumps, specified runoff...)

classmethod `load(f, model, nlak=None, nper=None, ncomp=None, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename* or *file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.mt3d.Mt3dms`) to which this package will be added.
- **nlak** (*int*) - number of lakes to be simulated
- **nper** (*int*) - number of stress periods
- **ncomp** (*int*) - number of species to be simulated
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

lkt - MT3D-USGS object.

Return type

MT3D-USGS object

Examples

```
>>> import flopy
>>> datadir = 'examples/data/mt3d_test/mfnwt_mt3dusgs/lkt'
>>> mt = flopy.mt3d.Mt3dms.load(
...     'lkt_mt.nam', exe_name='mt3d-usgs_1.0.00.exe',
...     model_ws=datadir, load_only='btn')
>>> lkt = flopy.mt3d.Mt3dLkt.load('test.lkt', mt)
```

write_file()

Write the package file

Return type

None

flopy.mt3d.mtphc module

```
class Mt3dPhc(model, os=2, temp=25, asbin=0, eps_aqu=0, eps_ph=0, scr_output=1,
              cb_offset=0, smse=['pH', 'pe'], mine=[], ie=[], surf=[], mobkin=[],
              minkin=[], surfkin=[], imobkin=[], extension='phc', unitnumber=None,
              filenames=None)
```

Bases: *Package*

PHC package class for PHT3D

unitnumber = 38

write_file()

Write the package file

Return type

None

flopy.mt3d.mtrct module

```
class Mt3dRct(model, isothm=0, ireact=0, igetsc=1, rhob=None, prsity2=None, srconc=None,
              sp1=None, sp2=None, rcl=None, rc2=None, extension='rct', unitnumber=None,
              filenames=None, **kwargs)
```

Bases: *Package*

Chemical reaction package class.

Parameters

- **model** (*model object*) - The model object (of type flopy.mt3dms.Mt3dms) to which this package will be added.
- **isothm** (*int*) - isothm is a flag indicating which type of sorption (or dual-domain mass transfer) is simulated: isothm = 0, no sorption is simulated; isothm = 1, linear isotherm (equilibrium-controlled); isothm = 2, Freundlich isotherm (equilibrium-controlled); isothm = 3, Langmuir isotherm (equilibrium-controlled); isothm = 4, first-order kinetic sorption (nonequilibrium); isothm = 5, dual-domain mass transfer (without sorption); isothm = 6, dual-domain mass transfer (with sorption). (default is 0).

- **ireact** (*int*) - ireact is a flag indicating which type of kinetic rate reaction is simulated: ireact = 0, no kinetic rate reaction is simulated; ireact = 1, first-order irreversible reaction, ireact = 100, zero-order reactions (decay or production). Note that this reaction package is not intended for modeling chemical reactions between species. An add-on reaction package developed specifically for that purpose may be used. (default is 0).
- **igetsc** (*int*) - igetsc is an integer flag indicating whether the initial concentration for the nonequilibrium sorbed or immobile phase of all species should be read when nonequilibrium sorption (isothm = 4) or dual-domain mass transfer (isothm = 5 or 6) is simulated: igetsc = 0, the initial concentration for the sorbed or immobile phase is not read. By default, the sorbed phase is assumed to be in equilibrium with the dissolved phase (isothm = 4), and the immobile domain is assumed to have zero concentration (isothm = 5 or 6). igetsc > 0, the initial concentration for the sorbed phase or immobile liquid phase of all species will be read. (default is 1).
- **rhob** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - rhob is the bulk density of the aquifer medium (unit, ML⁻³). rhob is used if isothm = 1, 2, 3, 4, or 6. If rhob is not user-specified and isothm is not 5 then rhob is set to 1.8e3. (default is None)
- **prsyty2** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - prsyty2 is the porosity of the immobile domain (the ratio of pore spaces filled with immobile fluids over the bulk volume of the aquifer medium) when the simulation is intended to represent a dual-domain system. prsyty2 is used if isothm = 5 or 6. If prsyty2 is not user-specified and isothm = 5 or 6 then prsyty2 is set to 0.1. (default is None)
- **srconc** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - srconc is the user-specified initial concentration for the sorbed phase of the first species if isothm = 4 (unit, MM⁻¹). Note that for equilibrium-controlled sorption, the initial concentration for the sorbed phase cannot be specified. srconc is the user-specified initial concentration of the first species for the immobile liquid phase if isothm = 5 or 6 (unit, ML⁻³). If srconc is not user-specified and isothm = 4, 5, or 6 then srconc is set to 0. (default is None).
- **sp1** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - sp1 is the first sorption parameter for the first species. The use of sp1 depends on the type of sorption selected (the value of isothm). For linear sorption (isothm = 1) and nonequilibrium sorption (isothm = 4), sp1 is the distribution coefficient (Kd) (unit, L3M⁻¹). For Freundlich sorption (isothm = 2), sp1 is the Freundlich equilibrium constant (Kf) (the unit depends on the Freundlich exponent a). For Langmuir sorption (isothm = 3), sp1 is the Langmuir equilibrium constant (Kl) (unit, L3M⁻¹). For dual-domain mass transfer without sorption (isothm = 5), sp1 is not used, but still must be entered. For dual-domain mass transfer with sorption (isothm = 6), sp1 is also the distribution coefficient (Kd) (unit, L3M⁻¹). If sp1 is not specified and

- isothm > 0 then sp1 is set to 0. (default is None).
- **sp2** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - sp2 is the second sorption or dual-domain model parameter for the first species. The use of sp2 depends on the type of sorption or dual-domain model selected. For linear sorption (isothm = 1), sp2 is read but not used. For Freundlich sorption (isothm = 2), sp2 is the Freundlich exponent a . For Langmuir sorption (isothm = 3), sp2 is the total concentration of the sorption sites available (S) (unit, MM-1). For nonequilibrium sorption (isothm = 4), sp2 is the first-order mass transfer rate between the dissolved and sorbed phases (unit, T-1). For dual-domain mass transfer (isothm = 5 or 6), sp2 is the first-order mass transfer rate between the two domains (unit, T-1). If sp2 is not specified and isothm > 0 then sp2 is set to 0. (default is None).
 - **rc1** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - rc1 is the first-order reaction rate for the dissolved (liquid) phase for the first species (unit, T-1). rc1 is not used ireact = 0. If a dual-domain system is simulated, the reaction rates for the liquid phase in the mobile and immobile domains are assumed to be equal. If rc1 is not specified and ireact > 0 then rc1 is set to 0. (default is None).
 - **rc2** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - rc2 is the first-order reaction rate for the sorbed phase for the first species (unit, T-1). rc2 is not used ireact = 0. If a dual-domain system is simulated, the reaction rates for the sorbed phase in the mobile and immobile domains are assumed to be equal. Generally, if the reaction is radioactive decay, rc2 should be set equal to rc1, while for biodegradation, rc2 may be different from rc1. Note that rc2 is read but not used, if no sorption is included in the simulation. If rc2 is not specified and ireact > 0 then rc2 is set to 0. (default is None).
 - **extension** (*string*) - Filename extension (default is 'rct')
 - **unitnumber** (*int*) -
File unit number. If file unit number is None then an unused unit number if used. (default is None).
 - **srconcn** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - srconcn is the user-specified initial concentration for the sorbed phase of species *n*. If srconcn is not passed as a ****kwarg** and isothm = 4, 5, or 6 then srconcn for species *n* is set to 0. See description of srconcn for a more complete description of srconcn.
 - **sp1n** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - sp1n is the first sorption parameter for species *n*. If sp1n is not passed as a ****kwarg** and isothm > 0 then sp1 for species *n* is set to 0. See description of sp1 for a more complete description of sp1n.
 - **sp2n** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - sp2n is the second sorption or dual-domain model parameter for species *n*. If sp2n is not passed as a ****kwarg** and isothm > 0 then sp2 for species *n* is set to 0. See description of sp2 for a more complete description of sp2n.

- **rc1n** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - rc1n is the first-order reaction rate for the dissolved (liquid) phase for species *n*. If rc1n is not passed as a ****kwarg** and *ireact* > 0 then rc1 for species *n* is set to 0. See description of rc1 for a more complete description of rc1n.
- **rc2n** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - rc2n is the first-order reaction rate for the sorbed phase for species *n*. If rc2n is not passed as a ****kwarg** and *ireact* > 0 then rc2 for species *n* is set to 0. See description of rc2 for a more complete description of rc2n.

Notes

Examples

```
>>> import flopy
>>> mt = flopy.mt3dms.Mt3dms()
>>> rct = flopy.mt3dms.Mt3dRct(mt)
```

```
classmethod load(f, model, nlay=None, nrow=None, ncol=None, ncomp=None,
                 ext_unit_dict=None)
```

Load an existing package.

Parameters

- **f** (*filename* or *file handle*) - File to load.
- **model** (*model object*) - The model object (of type *flopy.mt3d.mt.Mt3dms*) to which this package will be added.
- **nlay** (*int*) - Number of model layers in the reaction package. If *nlay* is not specified, the number of layers in the passed model object is used. (default is None).
- **nrow** (*int*) - Number of model rows in the reaction package. If *nrow* is not specified, the number of rows in the passed model object is used. (default is None).
- **ncol** (*int*) - Number of model columns in the reaction package. If *ncol* is not specified, the number of columns in the passed model object is used. (default is None).
- **ext_unit_dict** (*dictionary*, *optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function *flopy.utils.mfreadnam.parsefilenamefile*.

Returns

rct - Mt3dRct object.

Return type

Mt3dRct object

Examples

```
>>> import flopy
>>> mt = flopy.mt3d.Mt3dms()
>>> rct = flopy.mt3d.Mt3dRct.load('test.rct', mt)
```

write_file()

Write the package file

Return type

None

flopy.mt3d.mtsft module

```
class Mt3dSft(model, nsfinit=0, mxsfbc=0, icbcfsf=0, ioutobs=0, ietsfr=0, isfsolv=1,
             wimp=0.5, wups=1.0, cclosesf=1e-06, mxitersf=10, crntsf=1.0, iprtxmd=0,
             coldsf=0.0, dispsf=0.0, nobssf=0, obs_sf=None, sf_stress_period_data=None,
             unitnumber=None, filenames=None, dtype=None, extension='sft', **kwargs)
```

Bases: [Package](#)

MT3D-USGS StreamFlow Transport package class

Parameters

- **model** (*model object*) - The model object (of type `flopy.mt3dms.Mt3dms`) to which this package will be added.
- **nsfinit** (*int*) - Is the number of simulated stream reaches (in SFR2, the number of stream reaches is greater than or equal to the number of stream segments). This is equal to NSTRM found on the first line of the SFR2 input file. If NSFINIT > 0 then surface-water transport is solved in the stream network while taking into account groundwater exchange and precipitation and evaporation sources and sinks. Otherwise, if NSFINIT < 0, the surface-water network as represented by the SFR2 flow package merely acts as a boundary condition to the groundwater transport problem; transport in the surface-water network is not simulated.
- **mxsfbc** (*int*) - Is the maximum number of stream boundary conditions.
- **icbcfsf** (*int*) - Is an integer value that directs MT3D-USGS to write reach-by-reach concentration information to unit ICBCSF.
- **ioutobs** (*int*) - Is the unit number of the output file for simulated concentrations at specified gage locations. The NAM file must also list the unit number to which observation information will be written.
- **ietsfr** (*int*) - Specifies whether or not mass will exit the surface-water network with simulated evaporation. If IETSFR = 0, then mass does not leave via stream evaporation. If IETSFR > 0, then mass is allowed to exit the simulation with the simulated evaporation.
- **isfsolv** (*int*) - Specifies the numerical technique that will be used to solve the transport problem in the surface water network. The first release of MT3D-USGS (version 1.0) only allows for

a finite-difference formulation and regardless of what value the user specifies, the variable defaults to 1, meaning the finite-difference solution is invoked.

- **wimp** (*float*) - Is the stream solver time weighting factor. Ranges between 0.0 and 1.0. Values of 0.0, 0.5, or 1.0 correspond to explicit, Crank-Nicolson, and fully implicit schemes, respectively.
- **wups** (*float*) - Is the space weighting factor employed in the stream network solver. Ranges between 0.0 and 1.0. Values of 0.0 and 1.0 correspond to a central-in-space and upstream weighting factors, respectively.
- **cclosesf** (*float*) - Is the closure criterion for the SFT solver
- **mxitersf** (*int*) - Limits the maximum number of iterations the SFT solver can use to find a solution of the stream transport problem.
- **crntsf** (*float*) - Is the Courant constraint specific to the SFT time step, its value has no bearing upon the groundwater transport solution time step.
- **iprtxmd** (*int*) - A flag to print SFT solution information to the standard output file. IPRTXMD = 0 means no SFT solution information is printed; IPRTXMD = 1 means SFT solution summary information is printed at the end of every MT3D-USGS outer iteration; and IPRTXMD = 2 means SFT solution details are written for each SFT outer iteration that calls the xMD solver that solved SFT equations.
- **coldsf** (*array of floats*) - Represents the initial concentrations in the surface water network. The length of the array is equal to the number of stream reaches and starting concentration values should be entered in the same order that individual reaches are entered for record set 2 in the SFR2 input file. To specify starting concentrations for other species in a multi-species simulation, include additional keywords, such as coldsf2, coldsf3, and so forth.
- **dispsf** (*array of floats*) - Is the dispersion coefficient [L² T⁻¹] for each stream reach in the simulation and can vary for each simulated component of the simulation. That is, the length of the array is equal to the number of simulated stream reaches times the number of simulated components. Values of dispersion for each reach should be entered in the same order that individual reaches are entered for record set 2 in the SFR2 input file. To specify dispsf for other species in a multi-species simulation, include additional keywords, such as dispsf2, dispsf3, and so forth.
- **nobssf** (*int*) - Specifies the number of surface flow observation points for monitoring simulated concentrations in streams.
- **isobs** (*int*) - The segment number for each stream flow concentration observation point.
- **irobs** (*int*) - The reach number for each stream flow concentration observation point.

- **ntmp** (*int*) - The number of specified stream boundary conditions to follow. For the first stress period, this value must be greater than or equal to zero, but may be less than zero in subsequent stress periods.
- **isegbc** (*int*) - Is the segment number for which the current boundary condition will be applied.
- **irchbc** (*int*) - Is the reach number for which the current boundary condition will be applied.
- **isfbctyp** (*int*) -

Specifies, for ISEGBC/IRCHBC, what the boundary condition type is

0 A headwater boundary. That is, for streams entering at the

boundary of the simulated domain that need a specified concentration, use ISFBCTYP = 0

1 a precipitation boundary. If precipitation directly to

channels is simulated in the flow model and a non-zero concentration (default is zero) is desired, use ISFBCTYP = 1

2 a runoff boundary condition that is not the same thing as

runoff simulated in the UZF1 package and routed to a stream (or lake) using the IRNBND array. Users who specify runoff in the SFR2 input via the RUNOFF variable appearing in either record sets 4b or 6a and want to assign a non-zero concentration (default is zero) associated with this specified source, use ISFBCTYP=2;

3 a constant-concentration boundary. Any ISEGBC/IRCHBC

combination may set equal to a constant concentration boundary condition.

4 a pumping boundary condition. 5 an evaporation boundary condition. In models where

evaporation is simulated directly from the surface of the channel, users can use this boundary condition to specify a non-zero concentration (default is zero) associated with the evaporation losses.

- **cbcsf** (*float*) - Is the specified concentration associated with the current boundary condition entry. Repeat CBCSF for each simulated species (NCOMP).
- **extension** (*string*) - Filename extension (default is 'sft')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the sfr output name will be created using the model name and lake concentration observation extension (for example, modflowtest.cbc and modflowtest.sftcobs.out), if ioutobs is a number greater than zero. If a single string is passed the package will be set to the string and sfr concentration observation output name will be created using the model name and .sftcobs.out extension, if

ioutobs is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> datadir = 'examples/data/mt3d_test/mfnwt_mt3dusgs/sft_crnkNic'
>>> mf = flopy.modflow.Modflow.load(
...     'CrnkNic.nam', model_ws=datadir, load_only=['dis', 'bas6'])
>>> sfr = flopy.modflow.ModflowSfr2.load('CrnkNic.sfr2', mf)
>>> chk = sfr.check()
>>> # initialize an MT3D-USGS model
>>> mt = flopy.mt3d.Mt3dms.load(
...     'CrnkNic.mtnam', exe_name='mt3d-usgs-1.0.00.exe',
>>>     model_ws=datadir, load_only='btn')
>>> sft = flopy.mt3d.Mt3dSft.load(mt, 'CrnkNic.sft')
```

static `get_default_dtype(ncomp=1)`

Construct a dtype for the recarray containing the list of surface water boundary conditions.

classmethod `load(f, model, nsfinit=None, nper=None, ncomp=None, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.mt3d.Mt3dms`) to which this package will be added.
- **nsfinit** (*int*) - number of simulated stream reaches in the surface-water transport process.
- **isfsolv** (*int*) - Specifies the numerical technique that will be used to solve the transport problem in the surface water network. The first release of MT3D-USGS (version 1.0) only allows for a finite-difference formulation and regardless of what value the user specifies, the variable defaults to 1, meaning the finite-difference solution is invoked.
- **wimp** (*float*) - Is the stream solver time weighting factor. Ranges between 0.0 and 1.0. Values of 0.0, 0.5, or 1.0 correspond to explicit, Crank-Nicolson, and fully implicit schemes, respectively.
- **wups** (*float*) - Is the space weighting factor employed in the stream network solver. Ranges between 0.0 and 1.0. Values of 0.0 and 1.0 correspond to a central-in-space and upstream weighting factors, respectively.

- **cclosesf** (*float*) - Is the closure criterion for the SFT solver
- **mxitsrf** (*int*) - Limits the maximum number of iterations the SFT solver can use to find a solution of the stream transport problem.
- **crnssf** (*float*) - Is the Courant constraint specific to the SFT time step, its value has no bearing upon the groundwater transport solution time step.
- **iprtxmd** (*int*) - a flag to print SFT solution information to the standard output file. IPRTXMD can equal 0, 1, or 2, and will write increasing amounts of solver information to the standard output file, respectively.

Returns

sft - MT3D-USGS object

Return type

MT3D-USGS object

Examples

```
>>> import os
>>> import flopy
>>> mf = flopy.modflow.Modflow.load('CrnkNic_mf.nam',
...                               load_only=['dis', 'bas6'])
>>> sfr = flopy.modflow.ModflowSfr2.load('CrnkNic.sfr2', mf)
>>> mt = flopy.mt3d.Mt3dms.load('CrnkNic_mt.nam', load_only='btn')
>>> sft = flopy.mt3d.Mt3dSft.load('CrnkNic.sft', mt)
```

write_file()

Write the package file

Return type

None

Examples

```
>>> import flopy
>>> datadir = '.examples/data/mt3d_test/mfnwt_mt3dusgs/sft_crnkNic'
>>> mt = flopy.mt3d.Mt3dms.load(
...     'CrnkNic.mtnam', exe_name='mt3d-usgs_1.0.00.exe',
...     model_ws=datadir, verbose=True)
>>> mt.name = 'CrnkNic_rewrite'
>>> mt.sft.dispsf.fmtin = '(10F12.2)'
>>> mt.write_input()
```

flopy.mt3d.mtssm module

```
class Mt3dSsm(model, crch=None, cevt=None, mxss=None, stress_period_data=None,
              dtype=None, extension='ssm', unitnumber=None, filenames=None, **kwargs)
```

Bases: [Package](#)

MT3DMS Source and Sink Mixing Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.mt3d.mt.Mt3dms](#)) to which this package will be added.
- **crch** ([Transient2d](#), *scalar, array of floats, or dictionary*) - CRCH is the concentration of recharge for species 1. If the recharge flux is positive, it acts as a source whose concentration can be specified as desired. If the recharge flux is negative, it acts as a sink (discharge) whose concentration is always set equal to the concentration of groundwater at the cell where discharge occurs. Note that the location and flow rate of recharge/discharge are obtained from the flow model directly through the unformatted flow-transport link file. crch can be specified as an array, if the array is constant for the entire simulation. If crch changes by stress period, then the user must provide a dictionary, where the key is the stress period number (zero based) and the value is the recharge array. The recharge concentration can be specified for additional species by passing additional arguments to the Mt3dSsm constructor. For example, to specify the recharge concentration for species two one could use `crch2={0: 0., 1: 10*np.ones((nrow, ncol), dtype=float)}` as and additional keyword argument that is passed to Mt3dSsm when making the ssm object.
- **cevt** ([Transient2d](#), *scalar, array of floats, or dictionary*) - is the concentration of evapotranspiration flux for species 1. Evapotranspiration is the only type of sink whose concentration may be specified externally. Note that the concentration of a sink cannot be greater than that of the aquifer at the sink cell. Thus, if the sink concentration is specified greater than that of the aquifer, it is automatically set equal to the concentration of the aquifer. Also note that the location and flow rate of evapotranspiration are obtained from the flow model directly through the unformatted flow-transport link file. For multi-species simulations, see crch for a description of how to specify additional concentrations arrays for each species.
- **stress_period_data** (*dictionary*) - Keys in the dictionary are stress zero-based stress period numbers; values in the dictionary are recarrays of SSM boundaries. The dtype for the recarray can be obtained using `ssm.dtype` (after the ssm package has been created). The default dtype for the recarray is `np.dtype([('k', int), ('i', int), ('j', int), ('css', np.float32), ('itype', int), ((cssms(n), float), n=1, ncomp)])` If there are more than one component species, then additional entries will be added to the dtype as indicated by `cssm(n)`. Note that if the number of dictionary entries is less than the number of stress periods, then the last recarray of boundaries will apply until the end

of the simulation. Full details of all options to specify `stress_period_data` can be found in the `flopy3_multi-component_SSM` ipython notebook in the Notebook subdirectory of the examples directory. `CSS` is the specified source concentration or mass-loading rate, depending on the value of `ITYPE`, in a single-species simulation, (For a multispecies simulation, `CSS` is not used, but a dummy value still needs to be entered here.) Note that for most types of sources, `CSS` is interpreted as the source concentration with the unit of mass per unit volume (ML-3), which, when multiplied by its corresponding flow rate (L3T-1) from the flow model, yields the mass-loading rate (MT-1) of the source. For a special type of sources (`ITYPE = 15`), `CSS` is taken directly as the mass-loading rate (MT-1) of the source so that no flow rate is required from the flow model. Furthermore, if the source is specified as a constant-concentration cell (`itype = -1`), the specified value of `CSS` is assigned directly as the concentration of the designated cell. If the designated cell is also associated with a sink/source term in the flow model, the flow rate is not used. `itype` is an integer indicating the type of the point source. An `itype` dictionary can be retrieved from the `ssm` object as `itype = mt3d.Mt3dSsm.itype_dict()` (`CSSMS(n)`, `n=1`, `NCOMP`) defines the concentrations of a point source for multispecies simulation with `NCOMP>1`. In a multispecies simulation, it is necessary to define the concentrations of all species associated with a point source. As an example, if a chemical of a certain species is injected into a multispecies system, the concentration of that species is assigned a value greater than zero while the concentrations of all other species are assigned zero. `CSSMS(n)` can be entered in free format, separated by a comma or space between values. Several important notes on assigning concentration for the constant-concentration condition (`ITYPE = -1`) are listed below: The constant-concentration condition defined in this input file takes precedence to that defined in the Basic Transport Package input file. In a multiple stress period simulation, a constant-concentration cell, once defined, will remain a constant-concentration cell in the duration of the simulation, but its concentration value can be specified to vary in different stress periods. In a multispecies simulation, if it is only necessary to define different constant-concentration conditions for selected species at the same cell location, specify the desired concentrations for those species, and assign a negative value for all other species. The negative value is a flag used by `MT3DMS` to skip assigning the constant-concentration condition for the designated species.

- **dtype** (*np.dtype*) - dtype to use for the recarray of boundaries. If left as `None` (the default) then the dtype will be automatically constructed.
- **extension** (*string*) - Filename extension (default is 'ssm')
- **unitnumber** (*int*) - File unit number (default is `None`).
- **filenames** (*str or list of str*) - Filenames to use for the package. If `filenames=None` the package name will be created using the model name and package extension. If a single string

is passed the package will be set to the string. Default is None.

Notes

Examples

```
>>> import flopy
>>> m = flopy.mt3d.Mt3dms()
>>> itype = mt3d.Mt3dSsm.itype_dict()
>>> ssm_data = {}
>>> ssm_data[0] = [(4, 4, 4, 1.0, itype['GHB'], 1.0, 100.0)]
>>> ssm_data[5] = [(4, 4, 4, 0.5, itype['GHB'], 0.5, 200.0)]
>>> ssm = flopy.mt3d.Mt3dSsm(m, stress_period_data=ssm_data)
```

from_package(package, ncomp_aux_names)

read the point source and sink info from a package ncomp_aux_names (list): the aux variable names in the package that are the component concentrations

static get_default_dtype(ncomp=1)

Construct a dtype for the recarray containing the list of sources and sinks

static itype_dict()

classmethod load(f, model, nlay=None, nrow=None, ncol=None, nper=None, ncomp=None, ext_unit_dict=None)

Load an existing package.

Parameters

- **f** (filename or file handle) - File to load.
- **model** (model object) - The model object (of type `flopy.mt3d.Mt.Mt3dms`) to which this package will be added.
- **ext_unit_dict** (dictionary, optional) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

ssm - Mt3dSsm object.

Return type

Mt3dSsm object

Examples

```
>>> import flopy
>>> mt = flopy.mt3d.Mt3dms()
>>> ssm = flopy.mt3d.Mt3dSsm.load('test.ssm', mt)
```

write_file()

Write the package file

Return type

None

```
class SsmPackage(label="", instance=None, needTFstr=False)
```

Bases: `object`

flopy.mt3d.mttob module

```
class Mt3dTob(model, outnam='tob_output', CScale=1.0, FluxGroups=[], FScale=1.0,
              iOutFlux=0, extension='tob', unitnumber=None, filenames=None)
```

Bases: `Package`

Transport Observation package class

```
write_file()
```

Write the package file

Return type

None

flopy.mt3d.mtuzt module

```
class Mt3dUzt(model, icbcuz=None, iet=0, iuzfbnd=None, cuzinf=None, cuzet=None,
              cgwet=None, extension='uzt', unitnumber=None, filenames=None, **kwargs)
```

Bases: `Package`

MT3D-USGS Unsaturated-Zone Transport package class

Parameters

- **model** (*model object*) - The model object (of type `flopy.mt3dms.Mt3dms`) to which this package will be added.
- **icbcuz** (*int*) - Is the unit number to which unsaturated-zone concentration will be written out.
- **iet** (*int*) - Is a flag that indicates whether or not ET is being simulated in the UZF1 flow package (`=0` indicates that ET is not being simulated). If ET is not being simulated, IET informs FMI package not to look for UZET and GWET arrays in the flow-transport link file.
- **iuzfbnd** (*array of ints*) - Specifies which row/column indices variably-saturated transport will be simulated in.
 - >0 indicates variably-saturated transport will be simulated; `=0` indicates variably-saturated transport will not be simulated; <0 Corresponds to `IUZFBND < 0` in the UZF1 input package, meaning
 - that user-supplied values for FINF are specified
 - recharge and therefore transport through the
 - unsaturated zone is not simulated.
- **incuzinf** (*int*) - (This value is repeated for each stress period as explained next) A flag indicating whether an array containing the concentration of infiltrating water (FINF) for each simulated species (ncomp) will be read for the current stress period.

If INCUZINF ≥ 0 , an array containing the concentration of infiltrating flux for each species will be read. If INCUZINF < 0 , the concentration of infiltrating flux will be reused from the previous stress period. If INCUZINF < 0 is specified for the first stress period, then by default the concentration of positive infiltrating flux (source) is set equal to zero. There is no possibility of a negative infiltration flux being specified. If infiltrating water is rejected due to an infiltration rate exceeding the vertical hydraulic conductivity, or because saturation is reached in the unsaturated zone and the water table is therefore at land surface, the concentration of the runoff will be equal to CUZINF specified next. The runoff is routed if IRNBND is specified in the MODFLOW simulation.

- **cuzinf** (*array of floats*) - Is the concentration of the infiltrating flux for a particular species. An array for each species will be read.
- **incuzet** (*int*) - (This value is repeated for each stress period as explained next) A flag indicating whether an array containing the concentration of evapotranspiration flux originating from the unsaturated zone will be read for the current stress period. If INCUZET ≥ 0 , an array containing the concentration of evapotranspiration flux originating from the unsaturated zone for each species will be read. If INCUZET < 0 , the concentration of evapotranspiration flux for each species will be reused from the last stress period. If INCUZET < 0 is specified for the first stress period, then by default, the concentration of negative evapotranspiration flux (sink) is set equal to the aquifer concentration, while the concentration of positive evapotranspiration flux (source) is set to zero.
- **cuzet** (*array of floats*) - Is the concentration of ET fluxes originating from the unsaturated zone. As a default, this array is set equal to 0 and only overridden if the user specifies INCUZET > 1 . If empirical evidence suggest volatilization of simulated constituents from the unsaturated zone, this may be one mechanism for simulating this process, though it would depend on the amount of simulated ET originating from the unsaturated zone. An array for each species will be read.
- **incgwet** (*int*) - (This value is repeated for each stress period as explained next) Is a flag indicating whether an array containing the concentration of evapotranspiration flux originating from the saturated zone will be read for the current stress period. If INCGWET ≥ 0 , an array containing the concentration of evapotranspiration flux originating from the saturated zone for each species will be read. If INCGWET < 0 , the concentration of evapotranspiration flux for each species will be reused from the last stress period. If INCUZET < 0 is specified for the first stress period, then by default, the concentration of negative evapotranspiration flux (sink) is set to the aquifer concentration, while the concentration of positive evapotranspiration flux (source) is set to zero.
- **cgwet** (*array of floats*) - Is the concentration of ET fluxes

originating from the saturated zone. As a default, this array is set equal to 0 and only overridden if the user specifies INCUZET > 1. An array for each species will be read.

- **extension** (*string*) - Filename extension (default is 'uzt')
- **unitnumber** (*int*) - File unit number (default is None).
- **filenames** (*str or list of str*) - Filenames to use for the package and the output files. If filenames=None the package name will be created using the model name and package extension and the uzf output name will be created using the model name and uzf concentration observation extension (for example, modflowtest.cbc and modflowtest.uzcobs.out), if icbcuz is a number greater than zero. If a single string is passed the package will be set to the string and uzf concentration observation output name will be created using the model name and .uzcobs.out extension, if icbcuz is a number greater than zero. To define the names for all package files (input and output) the length of the list of strings should be 2. Default is None.

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> datadir = 'examples/data/mt3d_test/mfnwt_mt3dusgs/keat_uzf'
>>> mt = flopy.mt3d.Mt3dms.load(
...     'Keat_UZF_mt.nam', exe_name='mt3d-usgs_1.0.00.exe',
...     model_ws=datadir, load_only='btn')
>>> uzt = flopy.mt3d.Mt3dUzt('Keat_UZF.uzt', mt)
```

classmethod `load(f, model, nlay=None, nrow=None, ncol=None, nper=None, ncomp=None, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.mt3d.Mt3dms`) to which this package will be added.
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

uzt - Mt3dUzt object.

Return type

Mt3dSsm object

Examples

```
>>> import flopy
>>> mt = flopy.mt3d.Mt3dms()
>>> uzt = flopy.mt3d.Mt3dUzt.load('test.uzt', mt)
```

write_file()

Write the package file

Return type

None

8.2.4 SEAWAT Packages

Contents:

flopy.seawat.swt module

```
class Seawat(modelname='swttest', namefile_ext='nam', modflowmodel=None, mt3dmodel=None,
             version='seawat', exe_name='swtv4', structured=True, listunit=2, model_ws='.',
             external_path=None, verbose=False, load=True, silent=0)
```

Bases: [BaseModel](#)

SEAWAT Model Class.

Parameters

- **modelname** (*str*, default "swttest") - Name of model. This string will be used to name the SEAWAT input that are created with `write_model`.
- **namefile_ext** (*str*, default "nam") - Extension for the namefile.
- **modflowmodel** ([Modflow](#), default None) - Instance of a Modflow object.
- **mt3dmodel** ([Mt3dms](#), default None) - Instance of a Mt3dms object.
- **version** (*str*, default "seawat") - Version of SEAWAT to use. Valid versions are "seawat" (default).
- **exe_name** (*str*, default "swtv4") - The name of the executable to use.
- **structured** (*bool*, default True) - Specify if model grid is structured (default) or unstructured.
- **listunit** (*int*, default 2) - Unit number for the list file.
- **model_ws** (*str*, default ".") - Model workspace. Directory name to create model data sets. Default is the present working directory.
- **external_path** (*str*, optional) - Location for external files.
- **verbose** (*bool*, default False) - Print additional information to the screen.
- **load** (*bool*, default True) - Load model.
- **silent** (*int*, default 0) - Silent option.

Notes

Examples

```
>>> import flopy
>>> m = flopy.seawat.swt.Seawat()
```

change_model_ws(*new_pth=None, reset_external=False*)

Change the model work space.

Parameters

new_pth (*str* or *PathLike*) - Path of the new model workspace. If this path does not exist, it will be created. If no value (*None*) is given, the default is the present working directory.

Returns

val - Can be used to see what packages are in the model, and can then be used with `get_package` to pull out individual packages.

Return type

list of strings

get_ifrefm()

get_nrow_ncol_nlay_nper()

classmethod load(*f: str, version='seawat', exe_name: str | PathLike = 'swtv4', verbose=False, model_ws: str | PathLike = '.', load_only=None*)

Load an existing model.

Parameters

- **f** (*str*) - Name of SEAWAT name file to load.
- **version** (*str*, default "seawat") - Version of SEAWAT to use. Valid versions are "seawat" (default).
- **exe_name** (*str*, default "swtv4") - The name of the executable to use.
- **verbose** (*bool*, default *False*) - Print additional information to the screen.
- **model_ws** (*str* or *PathLike*, default ".") - Model workspace. Directory to create model data sets. Default is the present working directory.
- **load_only** (*list* of *str*, optional) - Packages to load (e.g. ["lpf", "adv"]). Default *None* means that all packages will be loaded.

Return type

flopy.seawat.swt.Seawat

Examples

```
>>> import flopy
>>> m = flopy.seawat.swt.Seawat.load(f)
```

property mcomp

property modelgrid

property modeltime

property ncol

property ncomp

property nlay

property nper

property nrow

property nrow_ncol_nlay_nper

write_name_file()

Write the name file

Return type

None

class SeawatList(model, extension='list', listunit=7)

Bases: [Package](#)

List Package class

write_file()

Every Package needs its own write_file function

flopy.seawat.swtvdf module

```
class SeawatVdf(model, mtdnconc=1, mfnadvfd=1, nswtcpl=1, iwttable=1, densemin=0,
                densemax=0, dnscri=0.01, denseref=1.0, denseslp=0.025, crhoref=0,
                firstdt=0.001, indense=1, dense=1.0, nsrhoeos=1, drhodprhd=0.00446,
                prhdref=0.0, extension='vdf', unitnumber=None, filenames=None, **kwargs)
```

Bases: [Package](#)

SEAWAT Variable-Density Flow Package Class.

Parameters

- **model** (model object) - The model object (of type [flopy.seawat.swt.Seawat](#)) to which this package will be added.
- **mt3drhoflg** (mtdnconc (or) - is the MT3DMS species number that will be used in the equation of state to compute fluid density. This input variable was formerly referred to as MTDNCONC (Langevin and others, 2003). If MT3DRHOFLG = 0, fluid density is specified using items 6 and 7, and flow will be uncoupled with transport if the IMT Process is active. If MT3DRHOFLG > 0, fluid density is calculated using the MT3DMS species number that corresponds with

MT3DRHOFLG. A value for MT3DRHOFLG greater than zero indicates that flow will be coupled with transport. If MT3DRHOFLG = -1, fluid density is calculated using one or more MT3DMS species. Items 4a, 4b, and 4c will be read instead of item 4. The dependence of fluid density on pressure head can only be activated when MT3DRHOFLG = -1. A value for MT3DRHOFLG of -1 indicates that flow will be coupled with transport.

- **mfadvfd** (*int*) - is a flag that determines the method for calculating the internodal density values used to conserve fluid mass. If MFADVFD = 2, internodal conductance values used to conserve fluid mass are calculated using a central-in-space algorithm. If MFADVFD \neq 2, internodal conductance values used to conserve fluid mass are calculated using an upstream-weighted algorithm.
- **nswtcpl** (*int*) - is a flag used to determine the flow and transport coupling procedure. If NSWTCPL = 0 or 1, flow and transport will be explicitly coupled using a one-timestep lag. The explicit coupling option is normally much faster than the iterative option and is recommended for most applications. If NSWTCPL > 1, NSWTCPL is the maximum number of non-linear coupling iterations for the flow and transport solutions. SEAWAT-2000 will stop execution after NSWTCPL iterations if convergence between flow and transport has not occurred. If NSWTCPL = -1, the flow solution will be recalculated only for: The first transport step of the simulation, or The last transport step of the MODFLOW timestep, or The maximum density change at a cell is greater than DNSCRIT.
- **iwtable** (*int*) - is a flag used to activate the variable-density water-table corrections (Guo and Langevin, 2002, eq. 82). If IWTABLE = 0, the water-table correction will not be applied. If IWTABLE > 0, the water-table correction will be applied.
- **densemin** (*float*) - is the minimum fluid density. If the resulting density value calculated with the equation of state is less than DENSEMIN, the density value is set to DENSEMIN. If DENSEMIN = 0, the computed fluid density is not limited by DENSEMIN (this is the option to use for most simulations). If DENSEMIN > 0, a computed fluid density less than DENSEMIN is automatically reset to DENSEMIN.
- **densemmax** (*float*) - is the maximum fluid density. If the resulting density value calculated with the equation of state is greater than DENSEMAX, the density value is set to DENSEMAX. If DENSEMAX = 0, the computed fluid density is not limited by DENSEMAX (this is the option to use for most simulations). If DENSEMAX > 0, a computed fluid density larger than DENSEMAX is automatically reset to DENSEMAX.
- **dnscrit** (*float*) - is a user-specified density value. If NSWTCPL is greater than 1, DNSCRIT is the convergence criterion, in units of fluid density, for convergence between flow and transport. If the maximum fluid density difference between two consecutive implicit coupling iterations is not less than DNSCRIT, the program will continue to iterate on the flow and transport equations, or will terminate if NSWTCPL is reached. If NSWTCPL

- is -1, DNSCRIT is the maximum density threshold, in units of fluid density. If the fluid density change (between the present transport timestep and the last flow solution) at one or more cells is greater than DNSCRIT, then SEAWAT_V4 will update the flow field (by solving the flow equation with the updated density field).
- **denseref** (*float*) - is the fluid density at the reference concentration, temperature, and pressure. For most simulations, DENSEREF is specified as the density of freshwater at 25 degrees C and at a reference pressure of zero.
 - **drhodc** (*float*) - formerly referred to as DENSESLP (Langevin and others, 2003), is the slope of the linear equation of state that relates fluid density to solute concentration. In SEAWAT_V4, separate values for DRHODC can be entered for as many MT3DMS species as desired. If DRHODC is not specified for a species, then that species does not affect fluid density. Any measurement unit can be used for solute concentration, provided DENSEREF and DRHODC are set properly. DRHODC can be approximated by the user by dividing the density difference over the range of end-member fluids by the difference in concentration between the end-member fluids.
 - **drhodprhd** (*float*) - is the slope of the linear equation of state that relates fluid density to the height of the pressure head (in terms of the reference density). Note that DRHODPRHD can be calculated from the volumetric expansion coefficient for pressure using equation 15. If the simulation is formulated in terms of kilograms and meters, DRHODPRHD has an approximate value of $4.46 \times 10^{-3} \text{ kg/m}^4$. A value of zero, which is typically used for most problems, inactivates the dependence of fluid density on pressure.
 - **prhdref** (*float*) - is the reference pressure head. This value should normally be set to zero.
 - **nsrhoeos** (*int*) - is the number of MT3DMS species to be used in the equation of state for fluid density. This value is read only if MT3DRHOFLG = -1.
 - **mtrhospec** (*int*) - is the MT3DMS species number corresponding to the adjacent DRHODC and CRHOREF.
 - **crhoref** (*float*) - is the reference concentration (C0) for species, MTRHOSPEC. For most simulations, CRHOREF should be specified as zero. If MT3DRHOFLG > 0, CRHOREF is assumed to equal zero (as was done in previous versions of SEAWAT).
 - **firstdt** (*float*) - is the length of the first transport timestep used to start the simulation if both of the following two conditions are met: 1. The IMT Process is active, and 2. transport timesteps are calculated as a function of the user-specified Courant number (the MT3DMS input variable, PERCEL, is greater than zero).
 - **indense** (*int*) - is a flag. INDENSE is read only if MT3DRHOFLG is equal to zero. If INDENSE < 0, values for the DENSE array will be reused from the previous stress period. If it is the first stress

period, values for the DENSE array will be set to DENSEREF. If `INDENSE = 0`, values for the DENSE array will be set to DENSEREF. If `INDENSE >= 1`, values for the DENSE array will be read from item 7. If `INDENSE = 2`, values read for the DENSE array are assumed to represent solute concentration, and will be converted to density values using the equation of state.

- **dense** (`Transient3d`) - A float or array of floats (`nlay`, `nrow`, `ncol`) should be assigned as values to a dictionary related to keys of period number. `dense` is the fluid density array read for each layer using the MODFLOW-2000 U2DREL array reader. The DENSE array is read only if `MT3DRHOFLG` is equal to zero. The DENSE array may also be entered in terms of solute concentration, or any other units, if `INDENSE` is set to 2 and the constants used in the density equation of state are specified appropriately.
- **extension** (`string`) - Filename extension (default is 'vdf')
- **unitnumber** (`int`) - File unit number (default is 37).

Notes

In `swt_4 mtdnconc` became `mt3drhoflg`. If the latter one is defined in `kwargs`, it will overwrite `mtdnconc`. Same goes for `denseslp`, which has become `drhodc`.

When loading an existing SEAWAT model that has DENSE specified as concentrations, the load process will convert those concentrations into density values using the equation of state. This is only relevant when `mtdnconc` (or `mt3drhoflg`) is set to zero.

Examples

```
>>> import flopy
>>> m = flopy.seawat.Seawat()
>>> lpf = flopy.seawat.SeawatVdf(m)
```

```
classmethod load(f, model, nper=None, ext_unit_dict=None)
```

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.seawat.swt.Seawat`) to which this package will be added.
- **nper** (`int`) - The number of stress periods. If `nper` is `None`, then `nper` will be obtained from the model object. (default is `None`).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then `f` should be a file handle. In this case `ext_unit_dict` is required, which can be constructed using the function `flopy.utils.mfreadnam.parsefilenamefile`.

Returns

vdf - SeawatVdf object.

Return type

SeawatVdf object

Examples

```
>>> import flopy
>>> mf = flopy.modflow.Modflow()
>>> dis = flopy.modflow.ModflowDis(mf)
>>> mt = flopy.mt3d.Mt3dms()
>>> swt = flopy.seawat.Seawat(modflowmodel=mf, mt3dmsmodel=mt)
>>> vdf = flopy.seawat.SeawatVdf.load('test.vdf', m)
```

unitnumber = 37

write_file()

Write the package file

Return type

None

flopy.seawat.swtvsc module

```
class SeawatVsc(model, mt3dmuflg=-1, viscmn=0.0, viscmx=0.0, viscref=0.0008904,
               nsmueos=0, mutempopt=2, mtmuspec=1, dmudc=1.923e-06, cmuref=0.0,
               mtmutempspec=1, amucoeff=None, invisc=-1, visc=-1, extension='vsc',
               unitnumber=None, filenames=None, **kwargs)
```

Bases: [Package](#)

SEAWAT Viscosity Package Class.

Parameters

- **model** (*model object*) - The model object (of type [flopy.seawat.swt.Seawat](#)) to which this package will be added.
- **mt3drhoflg** (*mt3dmuflg (or)*) - is the MT3DMS species number that will be used in the equation to compute fluid viscosity. If MT3DMUFLG >= 0, fluid density is calculated using the MT3DMS species number that corresponds with MT3DMUFLG. If MT3DMUFLG = -1, fluid viscosity is calculated using one or more MT3DMS species.
- **viscmn** (*float*) - is the minimum fluid viscosity. If the resulting viscosity value calculated with the equation is less than VISCMIN, the viscosity value is set to VISCMIN. If VISCMIN = 0, the computed fluid viscosity is not limited by VISCMIN (this is the option to use for most simulations). If VISCMIN > 0, a computed fluid viscosity less than VISCMIN is automatically reset to VISCMIN.
- **viscmx** (*float*) - is the maximum fluid viscosity. If the resulting viscosity value calculated with the equation is greater than VISCMAx, the viscosity value is set to VISCMAx. If VISCMAx = 0, the computed fluid viscosity is not limited by VISCMAx (this

- is the option to use for most simulations). If `VISCMAX > 0`, a computed fluid viscosity larger than `VISCMAX` is automatically reset to `VISCMAX`.
- **visceref** (*float*) - is the fluid viscosity at the reference concentration and reference temperature. For most simulations, `VISCREF` is specified as the viscosity of freshwater.
 - **dmudc** (*float, or list of floats (of size nsmueos) if nsmueos > 1*) - is the slope of the linear equation that relates fluid viscosity to solute concentration.
 - **nmueos** (*int*) - is the number of MT3DMS species to be used in the linear equation for fluid viscosity (this number does not include the temperature species if the nonlinear option is being used). This value is read only if `MT3DMUFLG = -1`. A value of zero indicates that none of the MT3DMS species have a linear effect on fluid viscosity (the nonlinear temperature dependence may still be activated); nothing should be entered for item 3c in this case.
 - **mutempopt** (*int*) - is a flag that specifies the option for including the effect of temperature on fluid viscosity. If `MUTEMPOPT = 0`, the effect of temperature on fluid viscosity is not included or is a simple linear relation that is specified in item 3c. If `MUTEMPOPT = 1`, fluid viscosity is calculated using equation 18. The size of the `AMUCOEFF` array in item 3e is 4 (`MUNCOEFF = 4`). If `MUTEMPOPT = 2`, fluid viscosity is calculated using equation 19. The size of the `AMUCOEFF` array in item 3e is 5 (`MUNCOEFF = 5`). If `MUTEMPOPT = 3`, fluid viscosity is calculated using equation 20. The size of the `AMUCOEFF` array in item 3e is 2 (`MUNCOEFF = 2`). If `NSMUEOS` and `MUTEMPOPT` are both set to zero, all fluid viscosities are set to `VISCREF`.
 - **mtmuspec** (*int, or list of ints (of size nsmueos) if nsmueos > 1*) - is the MT3DMS species number corresponding to the adjacent `DMUDC` and `CMUREF`.
 - **dmudc** - is the slope of the linear equation that relates fluid viscosity to solute concentration.
 - **cmuref** (*float, or list of floats (of size nsmueos) if nsmueos > 1*) - is the reference concentration.
 - **mtmuspectemp** (*int*) - is the MT3DMS species number that corresponds to temperature. This value must be between 1 and `NCOMP` and should not be listed in `MTMUSPEC` of item 3c.
 - **amucoeff** (*float*) - is the coefficient array of size `MUNCOEFF`. `AMUCOEFF` is `A` in equations 18, 19, and 20.
 - **muncoeff** (*int*) - is the size of the `AMUCOEFF` array.
 - **invisc** (*int*) - is a flag. `INVISC` is read only if `MT3DMUFLG` is equal to zero. If `INVISC < 0`, values for the `VISC` array will be reused from the previous stress period. If it is the first stress period, values for the `VISC` array will be set to `VISCREF`. If `INVISC = 0`, values for the `VISC` array will be set to `VISCREF`. If `INVISC >= 1`, values for the `VISC` array will be read from item 5. If `INVISC = 2`, values read for the `VISC` array are assumed to

represent solute concentration, and will be converted to viscosity values.

- **visc** (*float* or array of floats (*nlay*, *nrow*, *ncol*)) - is the fluid viscosity array read for each layer using the MODFLOW-2000 U2DREL array reader. The VISC array is read only if MT3DMUFLG is equal to zero. The VISC array may also be entered in terms of solute concentration (or any other units) if INVISC is set to 2, and the simple linear expression in item 3 can be used to represent the relation to viscosity.
- **extension** (*string*) - Filename extension (default is 'vsc')
- **unitnumber** (*int*) - File unit number (default is 38).

Notes

Examples

```
>>> import flopy
>>> m = flopy.seawat.Seawat()
>>> vsc = flopy.modflow.SeawatVsc(m)
```

classmethod `load(f, model, nper=None, ext_unit_dict=None)`

Load an existing package.

Parameters

- **f** (*filename or file handle*) - File to load.
- **model** (*model object*) - The model object (of type `flopy.seawat.swt.Seawat`) to which this package will be added.
- **nper** (*int*) - The number of stress periods. If *nper* is *None*, then *nper* will be obtained from the model object. (default is *None*).
- **ext_unit_dict** (*dictionary, optional*) - If the arrays in the file are specified using EXTERNAL, or older style array control records, then *f* should be a file handle. In this case *ext_unit_dict* is required, which can be constructed using the function `flopy.utils.mfreadnam.parsenamefile`.

Returns

vsc - SeawatVsc object.

Return type

SeawatVsc object

Examples

```
>>> import flopy
>>> mf = flopy.modflow.Modflow()
>>> dis = flopy.modflow.ModflowDis(mf)
>>> mt = flopy.mt3d.Mt3dms()
>>> swt = flopy.seawat.Seawat(modflowmodel=mf, mt3dmsmodel=mt)
>>> vdf = flopy.seawat.SeawatVsc.load('test.vsc', m)
```

unitnumber = 38

write_file()

Write the package file

Return type

None

8.2.5 MODPATH 7 Packages

Contents:

flopy.modpath.mp7 module

mp7 module. Contains the Modpath7List and Modpath7 classes.

```
class Modpath7(modelname='modpath7test', simfile_ext='mpsim', namefile_ext='mpnam',
               version='modpath7', exe_name='mp7', flowmodel=None, headfilename=None,
               budgetfilename=None, model_ws=None, verbose=False)
```

Bases: [BaseModel](#)

Modpath 7 class.

Parameters

- **modelname** (*str*, default "modpath7test") - Basename for MODPATH 7 input and output files.
- **simfile_ext** (*str*, default "mpsim") - Filename extension of the MODPATH 7 simulation file.
- **namefile_ext** (*str*, default "mpnam") - Filename extension of the MODPATH 7 namefile.
- **version** (*str*, default "modpath7") - String that defines the MODPATH version. Valid versions are "modpath7" (default).
- **exe_name** (*str*, default "mp7") - The name of the executable to use.
- **flowmodel** (*flopy.modflow.Modflow* or *flopy.mf6.MFModel* object) - MODFLOW model object.
- **headfilename** (*str*, optional) - Filename of the MODFLOW output head file. If headfilename is not provided then it will be set from the flowmodel.
- **budgetfilename** (*str*, optional) - Filename of the MODFLOW output cell-by-cell budget file. If budgetfilename is not provided then it will be set from the flowmodel.

- **model_ws** (*str*, default ".") - Model workspace. Directory name to create model data sets. Default is the current working directory.
- **verbose** (*bool*, default *False*) - Print additional information to the screen.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('mf2005.nam')
>>> mp = flopy.modpath.Modpath7('mf2005_mp', flowmodel=m)
```

```
classmethod create_mp7(modelname='modpath7test', trackdir='forward', flowmodel=None,
                        exe_name='mp7', model_ws='.', verbose=False,
                        columncelldivisions=2, rowcelldivisions=2,
                        layercelldivisions=2, nodes=None)
```

Create a default MODPATH 7 model using a passed flowmodel with 8 particles in user-specified node locations or every active model cell.

Parameters

- **modelname** (*str*) - Basename for MODPATH 7 input and output files (default is 'modpath7test').
- **trackdir** (*str*) - Keyword that defines the MODPATH particle tracking direction. Available trackdir's are 'backward' and 'forward'. (default is 'forward')
- **flowmodel** (*flopy.modflow.Modflow* or *flopy.mf6.MFModel* object) - MODFLOW model
- **exe_name** (*str*) - The name of the executable to use (the default is 'mp7').
- **model_ws** (*str*) - model workspace. Directory name to create model data sets. (default is the current working directory).
- **verbose** (*bool*) - Print additional information to the screen (default is *False*).
- **columncelldivisions** (*int*) - Number of particles in a cell in the column (x-coordinate) direction (default is 2).
- **rowcelldivisions** (*int*) - Number of particles in a cell in the row (y-coordinate) direction (default is 2).
- **layercelldivisions** (*int*) - Number of particles in a cell in the layer (z-coordinate) direction (default is 2).
- **nodes** (*int*, *list* of *ints*, *tuple* of *ints*, or *np.ndarray*) - Nodes (zero-based) with particles. If (default is node 0).

Returns

mp

Return type

Modpath7 object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('mf2005.nam')
>>> mp = flopy.modpath.Modpath7.create_mp7(flowmodel=m)
```

property hdry

property hnoflo

property laytyp

write_name_file()

Write the name file

Return type

None

class Modpath7List(model, extension='list', unitnumber=None)

Bases: *Package*

List package class

write_file()

Every Package needs its own write_file function

flopy.modpath.mp7bas module

mp7bas module. Contains the Modpath7Bas class.

class Modpath7Bas(model, porosity=0.3, defaultiface=None, extension='mpbas')

Bases: *Package*

MODPATH 7 Basic Package Class.

Parameters

- **model** (*model object*) - The model object (of type flopy.modpath.Modpath7) to which this package will be added.
- **porosity** (*float or array of floats (nlay, nrow, ncol)*) - The porosity array (the default is 0.30).
- **defaultiface** (*dict*) - Dictionary with keys that are the text string used by MODFLOW in the budget output file to label flow rates for a stress package and the values are the cell face (iface) on which to assign flows (the default is None).
- **extension** (*str, optional*) - File extension (default is 'mpbas').

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('mf2005.nam')
>>> mp = flopy.modpath.Modpath7('mf2005_mp', flowmodel=m)
>>> mpbas = flopy.modpath.Modpath7Bas(mp)
```

write_file(*check=False*)

Write the package file

Parameters

check (*boolean*) - Check package data for common errors. (default False)

Return type

None

flopy.modpath.mp7particledata module

mp7particledata module. Contains the **ParticleData**, **CellDataType**, **FaceDataType**, and **NodeParticleData** classes.

class CellDataType(*drape=0, columncelldivisions=3, rowcelldivisions=3, layercelldivisions=3*)

Bases: `object`

Cell data type class to create a MODPATH 7 particle location template for input style 2, 3, and 4 in cells (templatesubdivisiontype = 2).

Parameters

- **drape** (*int*) - Drape indicates how particles are treated when starting locations are specified for cells that are dry. If drape is 0, Particles are placed in the specified cell. If the cell is dry at the time of release, the status of the particle is set to unreleased and removed from the simulation. If drape is 1, particles are placed in the upper most active grid cell directly beneath the specified layer, row, column or node location (default is 0).
- **columncelldivisions** (*int*) - Number of particles in a cell in the column (x-coordinate) direction (default is 3).
- **rowcelldivisions** (*int*) - Number of particles in a cell in the row (y-coordinate) direction (default is 3).
- **layercelldivisions** (*int*) - Number of particles in a cell in the layer (z-coordinate) direction (default is 3).

Examples

```
>>> import flopy
>>> cd = flopy.modpath.CellDataType()
```

```
write(f=None)
```

Parameters

f (*fileobject*) - Fileobject that is open with write access

```
class FaceDataType(drape=0, verticaldivisions1=3, horizontaldivisions1=3,
                  verticaldivisions2=3, horizontaldivisions2=3, verticaldivisions3=3,
                  horizontaldivisions3=3, verticaldivisions4=3, horizontaldivisions4=3,
                  rowdivisions5=3, columndivisions5=3, rowdivisions6=3,
                  columndivisions6=3)
```

Bases: `object`

Face data type class to create a MODPATH 7 particle location template for input style 2, 3, and 4 on cell faces (templatesubdivisiontype = 2).

Parameters

- **drape** (*int*) - Drape indicates how particles are treated when starting locations are specified for cells that are dry. If drape is 0, Particles are placed in the specified cell. If the cell is dry at the time of release, the status of the particle is set to unreleased and removed from the simulation. If drape is 1, particles are placed in the upper most active grid cell directly beneath the specified layer, row, column or node location (default is 0).
- **verticaldivisions1** (*int*) - The number of vertical subdivisions that define the two-dimensional array of particles on cell face 1 (default is 3).
- **horizontaldivisions1** (*int*) - The number of horizontal subdivisions that define the two-dimensional array of particles on cell face 1 (default is 3).
- **verticaldivisions2** (*int*) - The number of vertical subdivisions that define the two-dimensional array of particles on cell face 2 (default is 3).
- **horizontaldivisions2** (*int*) - The number of horizontal subdivisions that define the two-dimensional array of particles on cell face 2 (default is 3).
- **verticaldivisions3** (*int*) - The number of vertical subdivisions that define the two-dimensional array of particles on cell face 3 (default is 3).
- **horizontaldivisions3** (*int*) - The number of horizontal subdivisions that define the two-dimensional array of particles on cell face 3 (default is 3).
- **verticaldivisions4** (*int*) - The number of vertical subdivisions that define the two-dimensional array of particles on cell face 4 (default is 3).

- **horizontaldivisions4** (*int*) - The number of horizontal subdivisions that define the two-dimensional array of particles on cell face 4 (default is 3).
- **rowdivisions5** (*int*) - The number of row subdivisions that define the two-dimensional array of particles on the bottom cell face (face 5) (default is 3).
- **columndivisions5** (*int*) - The number of column subdivisions that define the two-dimensional array of particles on the bottom cell face (face 5) (default is 3).
- **rowdivisions6** (*int*) - The number of row subdivisions that define the two-dimensional array of particles on the top cell face (face 6) (default is 3).
- **columndivisions6** (*int*) - The number of column subdivisions that define the two-dimensional array of particles on the top cell face (face 6) (default is 3).

Examples

```
>>> import flopy
>>> fd = flopy.modpath.FaceDataType()
```

```
write(f=None)
```

Parameters

f (*fileobject*) - Fileobject that is open with write access

class LRCParticleData(*subdivisiondata=None, lrcregions=None*)

Bases: `object`

Layer, row, column particle data template class to create MODPATH 7 particle location input style 2 on cell faces (`templatesubdivisiontype = 1`) and/or in cells (`templatesubdivisiontype = 2`). Particle locations for this template are specified by layer, row, column regions.

Parameters

- **subdivisiondata** (*FaceDataType, CellDataType or list of FaceDataType*) - and/or *CellDataType* types *FaceDataType*, *CellDataType*, or a list of *FaceDataType* and/or *CellDataType*s that are used to create one or more particle templates in a particle group. If *subdivisiondata* is *None*, a default *CellDataType* with 27 particles per cell will be created (default is *None*).
- **lrcregions** (*list of lists tuples or np.ndarrays*) - Layer, row, column (zero-based) regions with particles created using the specified template parameters. A region is defined as a list/tuple of *minlayer*, *minrow*, *mincolumn*, *maxlayer*, *maxrow*, *maxcolumn* values. If *subdivisiondata* is a list, a list/tuple or array of layer, row, column regions with the same length as *subdivisiondata* must be provided. If *lrcregions* is *None*, particles will be placed in the first model cell (default is *None*).

Examples

```
>>> import flopy
>>> pg = flopy.modpath.LRCParticleData(lrcregions=[0, 0, 0, 3, 10, 10])
```

```
write(f=None)
```

Parameters

f (*fileobject*) - Fileobject that is open with write access

class NodeParticleData(*subdivisiondata=None, nodes=None*)

Bases: `object`

Node particle data template class to create MODPATH 7 particle location input style 3 on cell faces (*templatesubdivisiontype* = 1) and/or in cells (*templatesubdivisiontype* = 2). Particle locations for this template are specified by nodes.

Parameters

- **subdivisiondata** (*FaceDataType, CellDataType or list of FaceDataType*) - and/or *CellDataType* types *FaceDataType*, *CellDataType*, or a list of *FaceDataType* and/or *CellDataType*s that are used to create one or more particle templates in a particle group. If *subdivisiondata* is *None*, a default *CellDataType* with 27 particles per cell will be created (default is *None*).
- **nodes** (*int, list of ints, tuple of ints, or np.ndarray*) - Nodes (zero-based) with particles created using the specified template parameters. If *subdivisiondata* is a list, a list of nodes with the same length as *subdivisiondata* must be provided. If *nodes* is *None*, particles will be placed in the first model cell (default is *None*).

Examples

```
>>> import flopy
>>> pg = flopy.modpath.NodeParticleData(nodes=[100, 101])
```

```
write(f=None)
```

Parameters

f (*fileobject*) - Fileobject that is open with write access

class ParticleData(*partlocs=None, structured=False, particleids=None, localx=None, locally=None, localz=None, timeoffset=None, drape=None*)

Bases: `object`

Class to create the most basic particle data type (starting location input style 1). Input style 1 is the most general input style and provides the most flexibility in customizing starting locations.

Parameters

- **partlocs** (*list/tuple of int, list/tuple of list/tuple, or np.ndarray*) - Particle locations (zero-based) that are either layer, row, column locations or nodes.

- **structured** (*bool*) - Boolean defining if a structured (True) or unstructured particle recarray will be created (default is False).
- **particleids** (*list, tuple, or np.ndarray*) - Particle ids for the defined particle locations. If particleids is None, MODPATH 7 will define the particle ids to each particle location. If particleids is provided a particle id must be provided for each partloc (default is None).
- **localx** (*float, list, tuple, or np.ndarray*) - Local x-location of the particle in the cell. If a single value is provided all particles will have the same localx position. If a list, tuple, or np.ndarray is provided a localx position must be provided for each partloc. If localx is None, a value of 0.5 (center of the cell) will be used (default is None).
- **localy** (*float, list, tuple, or np.ndarray*) - Local y-location of the particle in the cell. If a single value is provided all particles will have the same localy position. If a list, tuple, or np.ndarray is provided a localy position must be provided for each partloc. If localy is None, a value of 0.5 (center of the cell) will be used (default is None).
- **localz** (*float, list, tuple, or np.ndarray*) - Local z-location of the particle in the cell. If a single value is provided all particles will have the same localz position. If a list, tuple, or np.ndarray is provided a localz position must be provided for each partloc. If localy is None, a value of 0.5 (center of the cell) will be used (default is None).
- **timeoffset** (*float, list, tuple, or np.ndarray*) - Timeoffset of the particle relative to the release time. If a single value is provided all particles will have the same timeoffset. If a list, tuple, or np.ndarray is provided a timeoffset must be provided for each partloc. If timeoffset is None, a value of 0. (equal to the release time) will be used (default is None).
- **drape** (*int, list, tuple, or np.ndarray*) - Drape indicates how particles are treated when starting locations are specified for cells that are dry. If drape is 0, Particles are placed in the specified cell. If the cell is dry at the time of release, the status of the particle is set to unreleased and removed from the simulation. If drape is 1, particles are placed in the upper most active grid cell directly beneath the specified layer, row, column or node location. If a single value is provided all particles will have the same drape value. If a list, tuple, or np.ndarray is provided a drape value must be provided for each partloc. If drape is None, a value of 0 will be used (default is None).

Examples

```
>>> import flopy
>>> locs = [(0, 0, 0), (1, 0, 0), (2, 0, 0)]
>>> pd = flopy.modpath.ParticleData(locs, structured=True, drape=0,
...                                localx=0.5, localy=0.5, localz=1)

write(f=None)
```

Parameters

f (*fileobject*) - Fileobject that is open with write access

flopy.modpath.mp7particlegroup module

mp7particlegroup module. Contains the **ParticleGroup**, and **ParticleGroupNodeTemplate** classes.

```
class ParticleGroup(particlegroupname='PG1', filename=None, releasedata=0.0,
                    particledata=None)
```

Bases: **_Modpath7ParticleGroup**

ParticleGroup class to create MODPATH 7 particle group data for location input style 1. Location input style 1 is the most general type of particle group that requires the user to define the location of all particles and associated data (relative release time, drape, and optionally particle ids). **Particledata** locations can be specified by layer, row, column (locationstyle=1) or nodes (locationstyle=2) and are created with the **ParticleData** class.

Parameters

- **particlegroupname** (*str*) - Name of particle group (default is 'PG1')
- **filename** (*str*) - Name of the external file that will contain the particle data. If filename is '' or None the particle information for the particle group will be written to the MODPATH7 simulation file (default is None).
- **releasedata** (*float, int, list, or tuple*) - If releasedata is a float or an int or a list/tuple with a single float or int, releaseoption is set to 1 and release data is the particle release time (default is 0.0).
- **particledata** (**ParticleData**) - **ParticleData** instance with particle data. If particledata is None, a **ParticleData** instance will be created with a node-based particle in the center of the first node in the model (default is None).

Examples

```
>>> import flopy
>>> p = [(2, 0, 0), (0, 20, 0)]
>>> p = flopy.modpath.ParticleData(p)
>>> pg = flopy.modpath.ParticleGroup(particledata=p)
```

```
write(fp=None, ws='.')
```

Write MODPATH 7 particle data items 1 through 5

Parameters

- **fp** (*fileobject*) - Fileobject that is open with write access
- **ws** (*str*) - Workspace for particle data

```
class ParticleGroupLRCTemplate(particlegroupname='PG1', filename=None, releasedata=(0.0,),
                                particledata=None)
```

Bases: `_ParticleGroupTemplate`

Layer, row, column particle template class to create MODPATH 7 particle location input style 2. Particle locations for this template are specified by layer, row, column regions.

Parameters

- **particlegroupname** (*str*) - Name of particle group
- **filename** (*str*) - Name of the external file that will contain the particle data. If filename is '' or None the particle information for the particle group will be written to the MODPATH7 simulation file.
- **releasedata** (*float, int, list, or tuple*) - If releasedata is a float or an int or a list/tuple with a single float or int, releaseoption is set to 1 and release data is the particle release time.
- **particledata** - LRCParticleData object with input style 2 face and/or node particle data. If particledata is None a default LRCParticleData object is created (default is None).

```
write(fp=None, ws='.')
```

Parameters

- **fp** (*fileobject*) - Fileobject that is open with write access
- **ws** (*str*) - Workspace for particle data

```
class ParticleGroupNodeTemplate(particlegroupname='PG1', filename=None,
                                releasedata=(0.0,), particledata=None)
```

Bases: `_ParticleGroupTemplate`

Node particle template class to create MODPATH 7 particle location input style 3. Particle locations for this template are specified by nodes.

Parameters

- **particlegroupname** (*str*) - Name of particle group
- **filename** (*str*) - Name of the external file that will contain the particle data. If filename is '' or None the particle information

for the particle group will be written to the MODPATH7 simulation file.

- **releasedata** (*float, int, list, or tuple*) - If releasedata is a float or an int or a list/tuple with a single float or int, releaseoption is set to 1 and release data is the particle release time.
- **particledata** - NodeParticleData object with input style 3 face and/or node particle data. If particledata is None a default NodeParticleData object is created (default is None).

write(*fp=None, ws='.'*)

Parameters

- **fp** (*fileobject*) - Fileobject that is open with write access
- **ws** (*str*) - Workspace for particle data

flopy.modpath.mp7sim module

mpsim module. Contains the ModpathSim class. Note that the user can access the ModpathSim class as *flopy.modpath.ModpathSim*.

Additional information for this MODFLOW/MODPATH package can be found at the [Online MODFLOW Guide](#).

```
class Modpath7Sim(model, mpnamefilename=None, listingfilename=None,
                  endpointfilename=None, pathlinefilename=None, timeseriesfilename=None,
                  tracefilename=None, simulationtype='pathline',
                  trackingdirection='forward', weaksinkoption='stop_at',
                  weaksourceoption='stop_at', budgetoutputoption='no',
                  traceparticledata=None, budgetcellnumbers=None, referencetime=None,
                  stoptimeoption='extend', stoptime=None, timepointdata=None,
                  zonedataoption='off', stopzone=None, zones=0,
                  retardationfactoroption='off', retardation=1.0, particlegroups=None,
                  extension='mpsim')
```

Bases: [Package](#)

MODPATH Simulation File Package Class.

Parameters

- **model** (*model object*) - The model object (of type flopy.modpath.Modpath7) to which this package will be added.
- **mpnamefilename** (*str*) - Filename of the MODPATH 7 name file. If mpnamefilename is not defined it will be generated from the model name (default is None).
- **listingfilename** (*str*) - Filename of the MODPATH 7 listing file. If listingfilename is not defined it will be generated from the model name (default is None).
- **endpointfilename** (*str*) - Filename of the MODPATH 7 endpoint file. If endpointfilename is not defined it will be generated from the model name (default is None).

- **pathlinefilename** (*str*) - Filename of the MODPATH 7 pathline file. If pathlinefilename is not defined it will be generated from the model name (default is None).
- **timeseriesfilename** (*str*) - Filename of the MODPATH 7 timeseries file. If timeseriesfilename is not defined it will be generated from the model name (default is None).
- **tracefilename** (*str*) - Filename of the MODPATH 7 tracefile file. If tracefilename is not defined it will be generated from the model name (default is None).
- **simulationtype** (*str*) - MODPATH 7 simulation type. Valid simulation types are 'endpoint', 'pathline', 'timeseries', or 'combined' (default is 'pathline').
- **trackingdirection** (*str*) - MODPATH 7 tracking direction. Valid tracking directions are 'forward' or 'backward' (default is 'forward').
- **weaksinkoption** (*str*) - MODPATH 7 weak sink option. Valid weak sink options are 'pass_through' or 'stop_at' (default value is 'stop_at').
- **weaksourceoption** (*str*) - MODPATH 7 weak source option. Valid weak source options are 'pass_through' or 'stop_at' (default value is 'stop_at').
- **budgetoutputoption** (*str*) - MODPATH 7 budget output option. Valid budget output options are 'no' - individual cell water balance errors are not computed and budget record headers are not printed, 'summary' - a summary of individual cell water balance errors for each time step is printed in the listing file without record headers, or 'record_summary' - a summary of individual cell water balance errors for each time step is printed in the listing file with record headers (default is 'summary').
- **traceparticledata** (*list or tuple*) - List or tuple with two ints that define the particle group and particle id (zero-based) of the specified particle that is followed in detail. If traceparticledata is None, trace mode is off (default is None).
- **budgetcellnumbers** (*int, list of ints, tuple of ints, or np.ndarray*) - Cell numbers (zero-based) for which detailed water budgets are computed. If budgetcellnumbers is None, detailed water budgets are not calculated (default is None).
- **referencetime** (*float, list, or tuple*) - Specified reference time if a float or a list/tuple with a single float value is provided (reference time option 1). Otherwise a list or tuple with a zero-based stress period (int) and time step (int) and a float defining the relative time position in the time step is provided (reference time option 2). If referencetime is None, reference time is set to 0 (default is None).
- **stoptimeoption** (*str*) - String indicating how a particle tracking simulation is terminated based on time. If stop time option is 'total', particles will be stopped at the end of the final time step if 'forward' tracking is simulated or at the beginning of

the first time step if backward tracking. If stop time option is 'extend', initial or final steady-state time steps will be extended and all particles will be tracked until they reach a termination location. If stop time option is 'specified', particles will be tracked until they reach a termination location or the specified stop time is reached (default is 'extend').

- **stoptime** (*float*) - User-specified value of tracking time at which to stop a particle tracking simulation. Stop time is only used if the stop time option is 'specified'. If stoptime is None and the stop time option is 'specified' particles will be terminated at the end of the last time step if 'forward' tracking or the beginning of the first time step if 'backward' tracking (default is None).
- **timepointdata** (*list* or *tuple*) - List or tuple with 2 items that is only used if simulationtype is 'timeseries' or 'combined'. If the second item is a float then the timepoint data corresponds to time point option 1 and the first entry is the number of time points (timepointcount) and the second entry is the time point interval. If the second item is a list, tuple, or np.ndarray then the timepoint data corresponds to time point option 2 and the number of time points entries (timepointcount) in the second item and the second item is an list, tuple, or array of user-defined time points. If Timepointdata is None, time point option 1 is specified and the total simulation time is split into 100 intervals (default is None).
- **zonedataoption** (*str*) - If zonedataoption is 'off', zone array data are not read and a zone value of 1 is applied to all cells. If zonedataoption is 'on', zone array data are read (default is 'off').
- **stopzone** (*int*) - A zero-based specified integer zone value that indicates an automatic stopping location for particles and is only used if zonedataoption is 'on'. A value of -1 indicates no automatic stop zone is used. Stopzone values less than -1 are not allowed. If stopzone is None, stopzone is set to -1 (default is None).
- **zones** (*float* or array of floats (*nlay, nrow, ncol*)) - Array of zero-based positive integer zones that are only used if zonedataoption is 'on' (default is 0).
- **retardationfactoroption** (*str*) - If retardationfactoroption is 'off', retardation array data are not read and a retardation factor of 1 is applied to all cells. If retardationfactoroption is 'on', retardation factor array data are read (default is 'off').
- **retardation** (*float* or array of floats (*nlay, nrow, ncol*)) - Array of retardation factors that are only used if retardationfactoroption is 'on' (default is 1).
- **particlegroups** (*ParticleGroup* or *list* of *ParticleGroups*) - ParticleGroup or list of ParticlesGroups that contain data for individual particle groups. If None is specified, a particle in the center of node 0 will be created (default is None).

- **extension** (*string*) - Filename extension (default is 'mpsim')

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('mf2005.nam')
>>> mp = flopy.modpath.Modpath7('mf2005_mp', flowmodel=m)
>>> mpsim = flopy.modpath.Modpath7Sim(mp)
```

write_file(*check=False*)

Write the package file

Parameters

check (*boolean*) - Check package data for common errors. (default False)

Return type

None

class budgetOpt(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: [Enum](#)

Enumeration of different budget output options

no = 0

record_summary = 2

summary = 1

class onoffOpt(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: [Enum](#)

Enumeration of on-off options

off = 1

on = 2

class simType(*value, names=None, *, module=None, qualname=None, type=None, start=1, boundary=None*)

Bases: [Enum](#)

Enumeration of different simulation types

combined = 4

endpoint = 1

pathline = 2

timeseries = 3

sim_enum_error(*v, s, e*)

Standard enumeration error format error :param v: Enumeration value :type v: str
:param s: User-defined value :type s: str :param e: Enumeration class :type e: Enum class

```
class stopOpt(value, names=None, *, module=None, qualname=None, type=None, start=1,
              boundary=None)
```

Bases: [Enum](#)

Enumeration of different stop time options

extend = 2

specified = 3

total = 1

```
class trackDir(value, names=None, *, module=None, qualname=None, type=None, start=1,
               boundary=None)
```

Bases: [Enum](#)

Enumeration of different tracking directions

backward = 2

forward = 1

```
class weakOpt(value, names=None, *, module=None, qualname=None, type=None, start=1,
              boundary=None)
```

Bases: [Enum](#)

Enumeration of different weak sink and source options

pass_through = 1

stop_at = 2

8.2.6 MODPATH 6 Packages

Contents:

[flopy.modpath.mp6 module](#)

```
class Modpath6(modelname='modpathtest', simfile_ext='mpsim', namefile_ext='mpnam',
                version='modpath', exe_name: str | PathLike = 'mp6', modflowmodel=None,
                dis_file: str | PathLike | None = None, dis_unit=87, head_file: str |
                PathLike | None = None, budget_file: str | PathLike | None = None,
                model_ws: str | PathLike | None = None, external_path: str | PathLike |
                None = None, verbose=False, load=True, listunit=7)
```

Bases: [BaseModel](#)

Modpath6 class.

Parameters

- **modelname** ([str](#), default "modpathtest") - Basename for MODPATH 6 input and output files.
- **simfile_ext** ([str](#), default "mpsim") - Filename extension of the MODPATH 6 simulation file.
- **namefile_ext** ([str](#), default "mpnam") - Filename extension of the MODPATH 6 namefile.

- **version** (*str*, default "modpath") - String that defines the MODPATH version. Valid versions are "modpath" (default).
- **exe_name** (*str* or *PathLike*, default "mp6") - The name of the executable to use.
- **modflowmodel** (*flopy.modflow.Modflow*) - MODFLOW model object with one of LPF, BCF6, or UPW packages.
- **dis_file** (*str* or *PathLike*) - Required dis file name.
- **dis_unit** (*int*, default 87) - Optional dis file unit number.
- **head_file** (*str* or *PathLike*) - Required filename of the MODFLOW output head file.
- **budget_file** (*str* or *PathLike*) - Required filename of the MODFLOW output cell-by-cell budget file.
- **model_ws** (*str* or *PathLike*, optional) - Model workspace. Directory name to create model data sets. Default is the current working directory.
- **external_path** (*str* or *PathLike*, optional) - Location for external files.
- **verbose** (*bool*, default *False*) - Print additional information to the screen.
- **load** (*bool*, default *True*) - Load model.
- **listunit** (*int*, default 7) - LIST file unit number.

create_mpsim(*simtype*=*pathline*, *trackdir*=*forward*, *packages*=*WEL*, *start_time*=0, *default_ifaces*=*None*, *ParticleColumnCount*=4, *ParticleRowCount*=4, *MinRow*=0, *MinColumn*=0, *MaxRow*=*None*, *MaxColumn*=*None*)

Create a MODPATH simulation file using available MODFLOW boundary package data.

Parameters

- **simtype** (*str*) -
 Keyword defining the MODPATH simulation type. Available *simtype*'s are 'endpoint', 'pathline', and 'timeseries'. (default is 'PATHLINE')
- **trackdir** (*str*) - Keyword that defines the MODPATH particle tracking direction. Available *trackdir*'s are 'backward' and 'forward'. (default is 'forward')
- **packages** (*None*, *str* or *list* of *strings*) - Keyword defining the modflow packages used to create initial particle locations. Supported packages are 'WEL', 'MNW2' and 'RCH'. (default is 'WEL').
- **start_time** (*float* or *tuple*) - Sets the value of MODPATH reference time relative to MODFLOW time. *float* : value of MODFLOW simulation time at which to start the particle tracking simulation.

Sets the value of MODPATH ReferenceTimeOption to 1.

tuple

[(period, step, time fraction) MODFLOW stress period, time

step and fraction] between 0 and 1 at which to start the particle tracking simulation. Sets the value of MODPATH ReferenceTimeOption to 2.

- **default_ifaces** (*list*) - List of cell faces (1-6; see MODPATH6 manual, fig. 7) on which to start particles. (default is None, meaning ifaces will vary depending on packages argument above)
- **ParticleRowCount** (*int*) - Rows of particles to start on each cell index face (iface).
- **ParticleColumnCount** (*int*) - Columns of particles to start on each cell index face (iface).

Returns

mpsim

Return type

ModpathSim object

getmf()

getsim()

property mf

next_ext_unit()

Function to encapsulate next_ext_unit attribute

property sim

write_name_file()

Write the name file

Return type

None

class Modpath6List(*model*, *extension*='list', *listunit*=7)

Bases: [Package](#)

List package class

write_file()

Every Package needs its own write_file function

flopy.modpath.mp6bas module

mpbas module. Contains the ModpathBas class. Note that the user can access the ModpathBas class as *flopy.modflow.ModpathBas*.

Additional information for this MODFLOW/MODPATH package can be found at the [Online MODFLOW Guide](#).

class Modpath6Bas(*model*, *hnoflo*=-9999.0, *hdry*=-8888.0, *def_face_ct*=0, *bud_label*=None, *def_iface*=None, *laytyp*=None, *ibound*=None, *prsity*=0.3, *prsityCB*=0.3, *extension*='mpbas', *unitnumber*=86)

Bases: [Package](#)

MODPATH Basic Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modpath.mp.Modpath`) to which this package will be added.
- **hnoflo** (*float*) - Head value assigned to inactive cells (default is -9999.).
- **hdry** (*float*) - Head value assigned to dry cells (default is -8888.).
- **def_face_ct** (*int*) - Number fo default iface codes to read (default is 0).
- **bud_label** (*str or list of strs*) - MODFLOW budget item to which a default iface is assigned.
- **def_iface** (*int or list of ints*) - Cell face (iface) on which to assign flows from MODFLOW budget file.
- **laytyp** (*None, int or list of ints*) - MODFLOW layer type (0 is convertible, 1 is confined). If *None*, read from modflow model
- **ibound** (*None or array of ints, optional*) - The ibound array (the default is 1). If *None*, pull from parent modflow model
- **prsimty** (*array of ints, optional*) - The porosity array (the default is 0.30).
- **prsimtyCB** (*array of ints, optional*) - The porosity array for confining beds (the default is 0.30).
- **extension** (*str, optional*) - File extension (default is 'mpbas').

heading

Text string written to top of package input file.

Type

`str`

Notes

Examples

```
>>> import flopy
>>> m = flopy.modpath.Modpath6()
>>> mpbas = flopy.modpath.Modpath6Bas(m)
```

write_file()

Write the package file

Return type

None

flopy.modpath.mp6sim module

mpsim module. Contains the ModpathSim class. Note that the user can access the ModpathSim class as `flopy.modpath.ModpathSim`.

Additional information for this MODFLOW/MODPATH package can be found at the [Online MODFLOW Guide](#).

```
class Modpath6Sim(model, mp_name_file='mp.nam', mp_list_file='mp.list', option_flags=[1, 2,
1, 1, 1, 2, 2, 1, 2, 1, 1, 1], ref_time=0, ref_time_per_stp=[0, 0,
1.0], stop_time=None, group_name='group_1', group_placement=[[1, 1, 1,
0, 1, 1]], release_times=[[1, 1]], group_region=[[1, 1, 1, 1, 1, 1]],
mask_nlay=[1], mask_layer=[1], mask_1lay=[1], face_ct=[1], ifaces=[[6,
1, 1]], part_ct=[[1, 1, 1]], time_ct=1, release_time_incr=1,
time_pts=[1], particle_cell_cnt=[[2, 2, 2]], cell_bd_ct=1, bud_loc=[[1,
1, 1, 1]], trace_id=1, stop_zone=1, zone=1, retard_fac=1.0,
retard_fcCB=1.0, strt_file=None, extension='mpsim')
```

Bases: [Package](#)

MODPATH Simulation File Package Class.

Parameters

- **model** (*model object*) - The model object (of type `flopy.modpath.mp.Modpath`) to which this package will be added.
- **extension** (*string*) - Filename extension (default is 'mpsim')

heading

Text string written to top of package input file.

Type

`str`

Notes

Examples

```
>>> import flopy
>>> m = flopy.modpath.Modpath6()
>>> dis = flopy.modpath.Modpath6Sim(m)
```

check(*f=None, verbose=True, level=1, checktype=None*)

Check package data for common errors.

Parameters

- **f** (*str or file handle*) - String defining file name or file handle for summary file of check method output. If a string is passed a file handle is created. If *f* is None, check method does not write results to a summary file. (default is None)
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If *level=0*, summary checks are performed. If *level=1*, full checks are performed.

Return type
None

Examples

`write_file()`

Write the package file

Return type
None

```
class StartingLocationsFile(model, inputstyle=1, extension='loc', verbose=False,
                             use_pandas=False)
```

Bases: [Package](#)

Class for working with MODPATH Starting Locations file for particles.

Parameters

- **model** (*Modpath object*) - The model object (of type `flopy.modpath.mp.Modpath`) to which this package will be added.
- **inputstyle** (*1*) - Input style described in MODPATH6 manual (currently only input style 1 is supported)
- **extension** (*string*) - Filename extension (default is 'loc')
- **use_pandas** (*bool, default False*) - If True and pandas is available use pandas to write the particle locations >2x speed

`static get_dtypes()`

Build numpy dtype for the MODPATH 6 starting locations file.

```
static get_empty_starting_locations_data(npt=0, default_xloc0=0.5,
                                         default_yloc0=0.5, default_zloc0=0.0)
```

get an empty recarray for particle starting location info.

Parameters

- **npt** (*int*) - Number of particles. Particles in array will be numbered consecutively from 1 to npt.

```
write_file(data=None, float_format='{:.8f}')
```

Every Package needs its own write_file function

8.3 Flopy Utilities

8.3.1 Model Utilities (including binary file readers)

Contents:

flopy.utils.binaryfile module

Module to read MODFLOW binary output files. The module contains four important classes that can be accessed by the user.

- `HeadFile` (Binary head file. Can also be used for drawdown)
- `HeadUFile` (Binary MODFLOW-USG unstructured head file)
- `UcnFile` (Binary concentration file from MT3DMS)
- `CellBudgetFile` (Binary cell-by-cell flow file)

class `BinaryHeader`(*bintype=None*, *precision='single'*)

Bases: `Header`

The `binary_header` class is a class to create headers for MODFLOW binary files.

Parameters

- **`bintype`** (*str*) - is the type of file being opened (head and ucn file currently supported)
- **`precision`** (*str*) - is the precision of the floating point data in the file

static `create`(*bintype=None*, *precision='single'*, ***kwargs*)

Create a binary header

static `set_dtype`(*bintype=None*, *precision='single'*)

Set the dtype

`set_values`(***kwargs*)

Set values using kwargs

class `BinaryLayerFile`(*filename: str | PathLike*, *precision*, *verbose*, *kwargs*)

Bases: `LayerFile`

The `BinaryLayerFile` class is the super class from which specific derived classes are formed. This class should not be instantiated directly

`get_databytes`(*header*)

Parameters

`header` (`datafile.Header`) - header object

Returns

`databytes` - size of the data array, in bytes, following the header

Return type

`int`

`get_ts`(*idx*)

Get a time series from the binary file.

Parameters

`idx` (*tuple of ints*, or a list of a tuple of ints) - `idx` can be (layer, row, column) or it can be a list in the form [(layer, row, column), (layer, row, column), ...]. The layer, row, and column values must be zero based.

Returns

`out` - Array has size (ntimes, ncells + 1). The first column in the data array will contain time (totim).

Return type
numpy array

Notes

The layer, row, and column values must be zero-based, and must be within the following ranges: $0 \leq k < nlay$; $0 \leq i < nrow$; $0 \leq j < ncol$

Examples

exception BudgetIndexError

Bases: `Exception`

class CellBudgetFile(filename: `str` | `PathLike`, precision='auto', verbose=False, **kwargs)

Bases: `object`

CellBudgetFile Class.

Parameters

- **filename** (`str` or `PathLike`) - Path of the cell budget file
- **precision** (`string`) - 'single' or 'double'. Default is 'single'.
- **verbose** (`bool`) - Write information to the screen. Default is False.

Notes

Examples

```
>>> import flopy.utils.binaryfile as bf
>>> cbb = bf.CellBudgetFile('mymodel.cbb')
>>> cbb.list_records()
>>> rec = cbb.get_data(kstpiper=(0,0), text='RIVER LEAKAGE')
```

close()

Close the file handle

get_data(idx=None, kstpiper=None, totim=None, text=None, paknam=None, full3D=False)

Get data from the binary budget file.

Parameters

- **idx** (`int` or `list`) - The zero-based record number. The first record is record 0.
- **kstpiper** (`tuple of ints`) - A tuple containing the time step and stress period (kstp, kper). The kstp and kper values are zero based.
- **totim** (`float`) - The simulation time.
- **text** (`str`) - The text identifier for the record. Examples include 'RIVER LEAKAGE', 'STORAGE', 'FLOW RIGHT FACE', etc.

- **full3D** (*boolean*) - If true, then return the record as a three dimensional numpy array, even for those list-style records written as part of a 'COMPACT BUDGET' MODFLOW budget file. (Default is False.)

Returns

recordlist - A list of budget objects. The structure of the returned object depends on the structure of the data in the cbb file.

If full3D is True, then this method will return a numpy masked array of size (nlay, nrow, ncol) for those list-style 'COMPACT BUDGET' records written by MODFLOW.

Return type

list of records

Notes

Examples

get_indices(text=None)

Get a list of indices for a selected record name

Parameters

text (*str*) - The text identifier for the record. Examples include 'RIVER LEAKAGE', 'STORAGE', 'FLOW RIGHT FACE', etc.

Returns

out - indices of selected record name in budget file.

Return type

tuple

get_kstpkper()

Get a list of unique stress periods and time steps in the file

Returns

out - List of unique kstp, kper combinations in binary file. kstp and kper values are zero-based.

Return type

list of (kstp, kper) tuples

get_nrecords()

Return the number of records in the file

Returns

out - Number of records in the file.

Return type

int

get_position(idx, header=False)

Get the starting position of the data or header for a specified record number in the binary budget file.

Parameters

- **idx** (*int*) - The zero-based record number. The first record is record 0.
- **header** (*bool*) - If True, the position of the start of the header data is returned. If False, the position of the start of the data is returned (default is False).

Returns

ipos - The position of the start of the data in the cell budget file or the start of the header.

Return type

int64

get_record(*idx*, *full3D=False*)

Get a single data record from the budget file.

Parameters

- **idx** (*int*) - The zero-based record number. The first record is record 0.
- **full3D** (*boolean*) - If true, then return the record as a three dimensional numpy array, even for those list-style records written as part of a 'COMPACT BUDGET' MODFLOW budget file. (Default is False.)

Returns

record - The structure of the returned object depends on the structure of the data in the cbb file. Compact list data are returned as

If full3D is True, then this method will return a numpy masked array of size (nlay, nrow, ncol) for those list-style 'COMPACT BUDGET' records written by MODFLOW.

Return type

a single data record

Notes**Examples**

get_residual(*totim*, *scaled=False*)

Return an array the size of the model grid containing the flow residual calculated from the budget terms. Residual will not be correct unless all flow terms are written to the budget file.

Parameters

- **totim** (*float*) - Simulation time for which to calculate the residual. This value must be precise, so it is best to get it from the `get_times` method.
- **scaled** (*bool*) - If True, then divide the residual by the total cell inflow

Returns

residual - The flow residual for the cell of shape (nlay, nrow, ncol)

Return type

np.ndarray

get_times()

Get a list of unique times in the file

Returns

out - List contains unique simulation times (totim) in binary file.

Return type

list of floats

get_ts(idx, text=None, times=None)

Get a time series from the binary budget file.

Parameters

- **idx** (*tuple of ints, or a list of a tuple of ints*) - idx can be (layer, row, column) or it can be a list in the form [(layer, row, column), (layer, row, column), ...]. The layer, row, and column values must be zero based.
- **text** (*str*) - The text identifier for the record. Examples include 'RIVER LEAKAGE', 'STORAGE', 'FLOW RIGHT FACE', etc.
- **times** (*iterable of floats*) - List of times to from which to get time series.

Returns

out - Array has size (ntimes, ncells + 1). The first column in the data array will contain time (totim).

Return type

numpy array

Notes

The layer, row, and column values must be zero-based, and must be within the following ranges: $0 \leq k < \text{nlay}$; $0 \leq i < \text{nrow}$; $0 \leq j < \text{ncol}$

Examples**get_unique_package_names(decode=False)**

Get a list of unique package names in the file

Parameters

decode (*bool*) - Optional boolean used to decode byte strings (default is False).

Returns

names - List of unique package names in the binary file.

Return type

list of strings

get_unique_record_names(*decode=False*)

Get a list of unique record names in the file

Parameters

decode (*bool*) - Optional boolean used to decode byte strings (default is False).

Returns

names - List of unique text names in the binary file.

Return type

list of strings

list_records()

Print a list of all of the records in the file

list_unique_packages()

Print a list of unique package names

list_unique_records()

Print a list of unique record names

class HeadFile(*filename: str | PathLike, text='head', precision='auto', verbose=False, **kwargs*)

Bases: *BinaryLayerFile*

HeadFile Class.

Parameters

- **filename** (*str* or *PathLike*) - Path of the concentration file
- **text** (*string*) - Name of the text string in the head file. Default is 'head'
- **precision** (*string*) - 'auto', 'single' or 'double'. Default is 'auto'.
- **verbose** (*bool*) - Write information to the screen. Default is False.

Notes

The HeadFile class provides simple ways to retrieve 2d and 3d head arrays from a MODFLOW binary head file and time series arrays for one or more cells.

The BinaryLayerFile class is built on a record array consisting of headers, which are record arrays of the modflow header information (kstp, kper, pertim, totim, text, nrow, ncol, ilay) and long integers, which are pointers to first bytes of data for the corresponding data array.

Examples

```
>>> import flopy.utils.binaryfile as bf
>>> hdoobj = bf.HeadFile('model.hds', precision='single')
>>> hdoobj.list_records()
>>> rec = hdoobj.get_data(kstpkper=(1, 50))
```

```
>>> ddnoobj = bf.HeadFile('model.ddn', text='drawdown', precision='single')
>>> ddnoobj.list_records()
>>> rec = ddnoobj.get_data(totim=100.)
```

```
class HeadUFile(filename: str | PathLike, text='headu', precision='auto', verbose=False,
                **kwargs)
```

Bases: `BinaryLayerFile`

Unstructured MODFLOW-USG HeadUFile Class.

Parameters

- **filename** (*string*) - Name of the concentration file
- **text** (*string*) - Name of the text string in the head file. Default is 'headu'
- **precision** (*string*) - 'auto', 'single' or 'double'. Default is 'auto'.
- **verbose** (*bool*) - Write information to the screen. Default is False.

Notes

The HeadUFile class provides simple ways to retrieve a list of head arrays from a MODFLOW-USG binary head file and time series arrays for one or more cells.

The BinaryLayerFile class is built on a record array consisting of headers, which are record arrays of the modflow header information (kstp, kper, pertim, totim, text, nrow, ncol, ilay) and long integers, which are pointers to first bytes of data for the corresponding data array. For unstructured grids, nrow and ncol are the starting and ending node numbers for layer, ilay. This class overrides methods in the parent class so that the proper sized arrays are created.

When the get_data method is called for this class, a list of one-dimensional arrays will be returned, where each array is the head array for a layer. If the heads for a layer were not saved, then None will be returned for that layer.

Examples

```
>>> import flopy.utils.binaryfile as bf
>>> hdoobj = bf.HeadUFile('model.hds')
>>> hdoobj.list_records()
>>> usgheads = hdoobj.get_data(kstpkper=(1, 50))
```

```
get_databytes(header)
```

Parameters

header (`datafile.Header`) - header object

Returns

databytes - size of the data array, in bytes, following the header

Return type

`int`

`get_ts(idx)`

Get a time series from the binary HeadUFile

Parameters

idx (`int` or *list of ints*) - idx can be nodenumber or it can be a list in the form [nodenumber, nodenumber, ...]. The nodenumber, values must be zero based.

Returns

out - Array has size (ntimes, ncells + 1). The first column in the data array will contain time (totim).

Return type

numpy array

`class UcnFile(filename, text='concentration', precision='auto', verbose=False, **kwargs)`

Bases: `BinaryLayerFile`

UcnFile Class.

Parameters

- **filename** (*string*) - Name of the concentration file
- **text** (*string*) - Name of the text string in the ucn file. Default is 'CONCENTRATION'
- **precision** (*string*) - 'auto', 'single' or 'double'. Default is 'auto'.
- **verbose** (*bool*) - Write information to the screen. Default is False.

Notes

The UcnFile class provides simple ways to retrieve 2d and 3d concentration arrays from a MT3D binary head file and time series arrays for one or more cells.

The BinaryLayerFile class is built on a record array consisting of headers, which are record arrays of the modflow header information (kstp, kper, pertim, totim, text, nrow, ncol, ilay) and long integers, which are pointers to first bytes of data for the corresponding data array.

Examples

```
>>> import flopy.utils.binaryfile as bf
>>> ucobj = bf.UcnFile('MT3D001.UCN', precision='single')
>>> ucobj.list_records()
>>> rec = ucobj.get_data(kstpkper=(1,1))
```

binaryread(file, vartype, shape=(1,), charlen=16)

Uses numpy to read from binary file. This was found to be faster than the struct approach and is used as the default.

binaryread_struct(file, vartype, shape=(1,), charlen=16)

Read text, a scalar value, or an array of values from a binary file.

file

[file object] is an open file object

vartype

[type] is the return variable type: str, numpy.int32, numpy.float32, or numpy.float64

shape

[tuple] is the shape of the returned array (shape(1,) returns a single value) for example, shape = (nlay, nrow, ncol)

charlen

[int] is the length of the text string. Note that string arrays cannot be returned, only multi-character strings. Shape has no affect on strings.

get_headfile_precision(filename: str | PathLike)

Determine precision of a MODFLOW head file.

Parameters

- **filename** (str or PathLike) -
- **precision.** (Path of binary MODFLOW file to determine) -

Returns

- **result** (str)
- Result will be unknown, single, or double

join_struct_arrays(arrays)

Simple function that can join two numpy structured arrays.

write_budget(fbin, data, kstp=1, kper=1, text='FLOW-JA-FACE', imeth=1, delt=1.0, pertim=1.0, totim=1.0, text1id1='GWF-1', text2id1='GWF-1', text1id2='GWF-1', text2id2='NPF')

write_head(fbin, data, kstp=1, kper=1, pertim=1.0, totim=1.0, text='HEAD', ilay=1)

flopy.utils.check module

```
class check(package, f: str | PathLike | None = None, verbose=True, level=1,
            property_threshold_values={})
```

Bases: `object`

Check package for common errors

Parameters

- **package** (*object*) - Instance of Package class.
- **f** (*str* or *PathLike*, *optional*) - Path to the summary file. If no path is provided, a summary file is not created and results are only written to stdout.
- **verbose** (*bool*) - Boolean flag used to determine if check method results are written to the screen
- **level** (*int*) - Check method analysis level. If level=0, summary checks are performed. If level=1, full checks are performed.
- **property_threshold_values** (*dict*) -

hk

[tuple] Reasonable minimum/maximum hydraulic conductivity value; values below this will be flagged. Default is (1e-11, 1e5), after Bear, 1972 (see https://en.wikipedia.org/wiki/Hydraulic_conductivity) and Schwartz and Zhang (2003, Table 4.4).

vka

[tuple] Reasonable minimum/maximum hydraulic conductivity value; Default is (1e-11, 1e5), after Bear, 1972 (see https://en.wikipedia.org/wiki/Hydraulic_conductivity) and Schwartz and Zhang (2003, Table 4.4).

vkcb

[tuple] Reasonable minimum/maximum hydraulic conductivity value for quasi-3D confining bed; Default is (1e-11, 1e5), after Bear, 1972 (see https://en.wikipedia.org/wiki/Hydraulic_conductivity) and Schwartz and Zhang (2003, Table 4.4).

sy

[tuple] Reasonable minimum/maximum specific yield values; Default is (0.01,0.5) after Anderson, Woessner and Hunt (2015, Table 5.2).

sy

[tuple] Reasonable minimum/maximum specific storage values; Default is (3.3e-6, 2e-2) after Anderson, Woessner and Hunt (2015, Table 5.2).

Notes

Anderson, M.P, Woessner, W.W. and Hunt, R.J., 2015. **Applied Groundwater**

Modeling: Simulation of Flow and Advective Transport, Elsevier, 564p.

Bear, J., 1972. Dynamics of Fluids in Porous Media. Dover Publications. Schwartz, F.W. and Zhang, H., 2003. Fundamentals of Groundwater, Wiley, 583 p.

append_passed(*message*)

Add a check to the passed list if it isn't already in there.

bc_stage_names = {'DRN': 'elev', 'GHB': 'bhead'}

get_active(*include_cbd=False*)

Returns a boolean array of active cells for the model.

Parameters

include_cbd (*boolean*) - If True, active is of same dimension as the thickness array in the DIS module (includes quasi 3-D confining beds). Default False.

Returns

active - True where active.

Return type

3-D boolean array

get_neighbors(*a*)

For a structured grid, this returns the 6 neighboring values for each value in *a*. For an unstructured grid, this returns the *n_max* neighboring values, where *n_max* is the maximum number of nodal connections for any node within the model; nodes with less than *n_max* connections are assigned *np.nan* for indices above the number of connections for that node.

Parameters

- **a** (*3-D Model array in layer, row, column order array, even for an*) -
- **instance** (*unstructured grid; for*) -
- **array** (*a Util3d*) -
- **flopy.modflow.ModflowBas.ibound**. ((*e.g.*) -

Returns

neighbors - Array of neighbors, where axis 0 contains the *n* neighboring values for each value in *a*, and subsequent axes are in layer, row, column order. “*n*” is 6 for a structured grid, and “*n*” is *n_max* for an unstructured grid, as described above. Nan is returned for values at edges.

Return type

4-D array

isvalid(*inds*)

Check that indices are valid for model grid

Parameters

inds (*tuple or lists or arrays; or a 1-D array*) - (*k, i, j*) for structured grids; (*node*) for unstructured.

Returns

isvalid - True for each index in inds that is valid for the model grid.

Return type

1-D boolean array

```
package_check_levels = {'sfr': 1}
```

```
print_summary(cols=None, delimiter=',', float_format='{:.6f}')
```

```
property_threshold_values = {'hani': None, 'hk': (1e-11, 100000.0), 'k': (1e-11, 100000.0), 'k22': (1e-11, 100000.0), 'k33': (1e-11, 100000.0), 'ss': (1e-06, 0.01), 'sy': (0.01, 0.5), 'vka': (1e-11, 100000.0), 'vkcb': (1e-11, 100000.0)}
```

```
remove_passed(message)
```

Remove a check to the passed list if it failed in any stress period.

```
solver_packages = {'mf2005': ['DE4', 'SIP', 'GMG', 'PCG', 'PCGN'], 'mf2k': ['DE4', 'SIP', 'SOR', 'GMG', 'PCG', 'PCGN'], 'mfntw': ['DE4', 'SIP', 'PCG', 'NWT'], 'mfusg': ['SMS']}
```

```
stress_period_data_values(stress_period_data, criteria, col=None, error_name="", error_type='Warning')
```

If criteria contains any true values, return the error_type, package name, k,i,j indices, values, and description of error for each row in stress_period_data where criteria=True.

```
summarize(scrub: bool = False)
```

```
thin_cell_threshold = 1.0
```

```
values(a, criteria, error_name="", error_type='Warning')
```

If criteria contains any true values, return the error_type, package name, indices, array values, and description of error for each True value in criteria.

```
view_summary_array_fields(fields)
```

```
fields_view(arr, fields)
```

creates view of array that only contains the fields in fields. <https://stackoverflow.com/q/15182381/>

```
class mf6check(package, f=None, verbose=True, level=1, property_threshold_values={})
```

Bases: [check](#)

```
get_active(include_cbd=False)
```

Returns a boolean array of active cells for the model.

Parameters

include_cbd (boolean) - Does not apply to MF6 models, always false.

Returns

active - True where active.

Return type

3-D boolean array

flopy.utils.compare module

```
compare(namefile1: str | PathLike = None, namefile2: str | PathLike = None,
        precision='auto', max_cumpd=0.01, max_incpd=0.01, htol=0.001, outfile1: str |
        PathLike | None = None, outfile2: str | PathLike | None = None, files1: str |
        PathLike | List[str | PathLike] | None = None, files2: str | PathLike | List[str
        | PathLike] | None = None)
```

Compare the budget and head results for two MODFLOW-based model simulations.

Parameters

- **namefile1** (*str* or *PathLike*, optional) - namefile path for base model
- **namefile2** (*str* or *PathLike*, optional) - namefile path for comparison model
- **precision** (*str*) - precision for binary head file (“auto”, “single”, or “double”) default is “auto”
- **max_cumpd** (*float*) - maximum percent discrepancy allowed for cumulative budget terms (default is 0.01)
- **max_incpd** (*float*) - maximum percent discrepancy allowed for incremental budget terms (default is 0.01)
- **htol** (*float*) - maximum allowed head difference (default is 0.001)
- **outfile1** (*str* or *PathLike*, optional) - budget comparison output file name. If outfile1 is None, no budget comparison output is saved. (default is None)
- **outfile2** (*str* or *PathLike*, optional) - head comparison output file name. If outfile2 is None, no head comparison output is saved. (default is None)
- **files1** (*str*, *PathLike*, or *list*, optional) - base model output file. If files1 is not None, results will be extracted from files1 and namefile1 will not be used. (default is None)
- **files2** (*str*, *PathLike*, or *list*, optional) - comparison model output file. If files2 is not None, results will be extracted from files2 and namefile2 will not be used. (default is None)

Returns

success - boolean indicating if the budget and head differences are less than max_cumpd, max_incpd, and htol.

Return type

bool

```
compare_budget(namefile1: str | PathLike | None, namefile2: str | PathLike | None,
                max_cumpd=0.01, max_incpd=0.01, outfile: str | PathLike | None = None,
                files1: str | PathLike | List[str | PathLike] | None = None, files2: str
                | PathLike | List[str | PathLike] | None = None)
```

Compare the budget results from two simulations.

Parameters

- **namefile1** (*str* or *PathLike*, optional) - namefile path for base model

- **namefile2** (*str* or *PathLike*, optional) - namefile path for comparison model
- **max_cumpd** (*float*) - maximum percent discrepancy allowed for cumulative budget terms (default is 0.01)
- **max_incpd** (*float*) - maximum percent discrepancy allowed for incremental budget terms (default is 0.01)
- **outfile** (*str* or *PathLike*, optional) - budget comparison output file name. If outfile is None, no comparison output is saved. (default is None)
- **files1** (*str*, *PathLike*, or *list*, optional) - base model output file. If files1 is not None, results will be extracted from files1 and namefile1 will not be used. (default is None)
- **files2** (*str*, *PathLike*, or *list*, optional) - comparison model output file. If files2 is not None, results will be extracted from files2 and namefile2 will not be used. (default is None)

Returns

success - boolean indicating if the difference between budgets are less than max_cumpd and max_incpd

Return type

bool

```
compare_concentrations(namefile1: str | PathLike, namefile2: str | PathLike,
                        precision='auto', ctol=0.001, outfile: str | PathLike | None =
                        None, files1: str | PathLike | List[str | PathLike] | None =
                        None, files2: str | PathLike | List[str | PathLike] | None =
                        None, difftol=False, verbose=False)
```

Compare the mt3dms and mt3dusgs concentration results from two simulations.

Parameters

- **namefile1** (*str* or *PathLike*) - namefile path for base model
- **namefile2** (*str* or *PathLike*) - namefile path for comparison model
- **precision** (*str*) - precision for binary head file ("auto", "single", or "double") default is "auto"
- **ctol** (*float*) - maximum allowed concentration difference (default is 0.001)
- **outfile** (*str* or *PathLike*, optional) - concentration comparison output file name. If outfile is None, no comparison output is saved. (default is None)
- **files1** (*str*, *PathLike*, or *list*, optional) - base model output file. If files1 is not None, results will be extracted from files1 and namefile1 will not be used. (default is None)
- **files2** (*str*, *PathLike*, or *list*, optional) - comparison model output file. If files2 is not None, results will be extracted from files2 and namefile2 will not be used. (default is None)
- **difftol** (*bool*) - boolean determining if the absolute value of the concentration difference greater than ctol should be evaluated (default is False)

- **verbose** (*bool*) - boolean indicating if verbose output should be written to the terminal (default is False)

Returns

success - boolean indicating if the concentration differences are less than `ctol`.

Return type

bool

```
compare_heads(namefile1: str | PathLike | None, namefile2: str | PathLike | None,
precision='auto', text='head', text2=None, htol=0.001, outfile: str |
PathLike | None = None, files1: str | PathLike | List[str | PathLike] |
None = None, files2: str | PathLike | List[str | PathLike] | None = None,
diff_tol=False, verbose=False, exfile: str | PathLike | None = None,
exarr=None, maxerr=None)
```

Compare the head results from two simulations.

Parameters

- **namefile1** (*str* or *PathLike*) - namefile path for base model
- **namefile2** (*str* or *PathLike*) - namefile path for comparison model
- **precision** (*str*) - precision for binary head file ("auto", "single", or "double") default is "auto"
- **htol** (*float*) - maximum allowed head difference (default is 0.001)
- **outfile** (*str* or *PathLike*) - head comparison output file name. If outfile is *None*, no comparison output is saved. (default is *None*)
- **files1** (*str* or *PathLike*, or *List* of *str* or *PathLike*) - base model output files. If files1 is not *None*, results will be extracted from files1 and namefile1 will not be used. (default is *None*)
- **files2** (*str* or *PathLike*, or *List* of *str* or *PathLike*) - comparison model output files. If files2 is not *None*, results will be extracted from files2 and namefile2 will not be used. (default is *None*)
- **diff_tol** (*bool*) - boolean determining if the absolute value of the head difference greater than htol should be evaluated (default is False)
- **verbose** (*bool*) - boolean indicating if verbose output should be written to the terminal (default is False)
- **exfile** (*str* or *PathLike*, optional) - path to a file with exclusion array data. Head differences will not be evaluated where exclusion array values are greater than zero. (default is *None*)
- **exarr** (*numpy.ndarray*) - exclusion array. Head differences will not be evaluated where exclusion array values are greater than zero. (default is *None*).
- **maxerr** (*int*) - maximum number of head difference greater than htol that should be reported. If maxerr is *None*, all head difference greater than htol will be reported. (default is *None*)

Returns

success - boolean indicating if the head differences are less than `htol`.

Return type

`bool`

```
compare_stages(namefile1: str | PathLike = None, namefile2: str | PathLike = None,  
                files1: str | PathLike | List[str | PathLike] | None = None, files2: str  
                | PathLike | List[str | PathLike] | None = None, htol=0.001, outfile: str  
                | PathLike | None = None, difftol=False, verbose=False)
```

Compare SWR process stage results from two simulations.

Parameters

- **namefile1** (*str* or *PathLike*) - namefile path for base model
- **namefile2** (*str* or *PathLike*) - namefile path for comparison model
- **precision** (*str*) - precision for binary head file (“auto”, “single”, or “double”) default is “auto”
- **htol** (*float*) - maximum allowed stage difference (default is 0.001)
- **outfile** (*str* or *PathLike*, *optional*) - head comparison output file name. If outfile is None, no comparison output is saved. (default is None)
- **files1** (*str*, *PathLike*, or *list*, *optional*) - base model output file. If files1 is not None, results will be extracted from files1 and namefile1 will not be used. (default is None)
- **files2** (*str*, *PathLike*, or *list*, *optional*) - comparison model output file. If files2 is not None, results will be extracted from files2 and namefile2 will not be used. (default is None)
- **difftol** (*bool*) - boolean determining if the absolute value of the stage difference greater than htol should be evaluated (default is False)
- **verbose** (*bool*) - boolean indicating if verbose output should be written to the terminal (default is False)

Returns

success - boolean indicating if the stage differences are less than `htol`.

Return type

`bool`

```
compare_swrbudget(namefile1: str | PathLike | None, namefile2: str | PathLike | None,  
                   max_cumpd=0.01, max_incpd=0.01, outfile: str | PathLike | None = None,  
                   files1: str | PathLike | List[str | PathLike] | None = None, files2:  
                   str | PathLike | List[str | PathLike] | None = None)
```

Compare the SWR budget results from two simulations.

Parameters

- **namefile1** (*str* or *PathLike*, *optional*) - namefile path for base model
- **namefile2** (*str* or *PathLike*, *optional*) - namefile path for comparison model

- **max_cumpd** (*float*) - maximum percent discrepancy allowed for cumulative budget terms (default is 0.01)
- **max_incpd** (*float*) - maximum percent discrepancy allowed for incremental budget terms (default is 0.01)
- **outfile** (*str* or *PathLike*, *optional*) - budget comparison output file name. If outfile is None, no comparison output is saved. (default is None)
- **files1** (*str*, *PathLike*, or *list*, *optional*) - base model output file. If files1 is not None, results will be extracted from files1 and namefile1 will not be used. (default is None)
- **files2** (*str*, *PathLike*, or *list*, *optional*) - comparison model output file. If files2 is not None, results will be extracted from files2 and namefile2 will not be used. (default is None)

Returns

success - boolean indicating if the difference between budgets are less than max_cumpd and max_incpd

Return type

bool

`eval_bud_diff(fpth: str | PathLike, b0, b1, ia=None, dtol=1e-06)`

flopy.utils.crs module

Utilities related to coordinate reference system handling.

get_authority_crs(crs)

Try to get the authority representation for a coordinate reference system (CRS), for more robust comparison with other CRS objects.

Parameters

crs (*pyproj.CRS*) - Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg "EPSG:26916") or a WKT string.

Returns

authority_crs - CRS instance initialized with the name and authority code (e.g. `epsg: 5070`) produced by `pyproj.crs.CRS.to_authority()`

Return type

pyproj.CRS instance

Notes

`pyproj.crs.CRS.to_authority()` will return None if a matching authority name and code can't be found. In this case, the input crs instance will be returned.

References

<http://pyproj4.github.io/pyproj/stable/api/crs/crs.html>

get_crs(*prjfile*=None, *crs*=None, ***kwargs*)

Helper function to produce a `pyproj.CRS` object from various input. Longer-term, this would just handle the `crs` and `prjfile` arguments, but in the near term, we need to warn users about deprecating the `prj`, `epsg`, `proj4` and `wkt_string` inputs.

Parameters

- **prjfile** (*str* or *pathlike*, optional) - `_description_`, by default None
- **prj** (*str* or *pathlike*, optional) - Deprecated since version 3.4. use `prjfile` instead.
- **epsg** (*int*, optional) - Deprecated since version 3.4. use `crs` instead.
- **proj4** (*str*, optional) - Deprecated since version 3.4. use `crs` instead.
- **crs** (`pyproj.CRS`, optional if `prjfile` is specified) - Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg “EPSG:26916”) or a WKT string.
- **wkt_string** (*str*, optional) - Deprecated since version 3.4. use `crs` instead.

Returns

`crs`

Return type

`pyproj.CRS` instance

get_shapefile_crs(*shapefile*)

Get the coordinate reference system for a shapefile.

Parameters

shapefile (*str* or *pathlike*) - Path to a shapefile or an associated projection (.prj) file.

Returns

`crs`

Return type

`pyproj.CRS` instance

flopy.utils.cvfdutil module**class Point(x, y)**Bases: `object`**area_of_polygon(x, y)**

Calculates the signed area of an arbitrary polygon given its vertices <https://stackoverflow.com/a/4682656/> (Joe Kington) http://softsurfer.com/Archive/algorithm_0101/algorithm_0101.htm#2D%20Polygons

centroid_of_polygon(points)

<https://stackoverflow.com/a/14115494/> (mgamba)

get_disv_gridprops(verts, iverts, xcyc=None)

Calculate disv grid properties from verts and iverts

Parameters

- **verts** (`ndarray`) - 2d array of x, y vertices
- **iverts** (`list`) - list of size ncpl, with a list of vertex numbers for each cell

Returns

gridprops - Dictionary containing entries that can be passed directly into the modflow6 disv package.

Return type

`dict`

gridlist_to_disv_gridprops(gridlist)

Take a list of flopy structured model grids and convert them into a dictionary that can be passed into the modflow6 disv package. Cells from a child grid will be patched in to make a single set of vertices. Cells will be numbered according to consecutive numbering of active cells in the grid list.

Parameters

gridlist (`list`) - List of flopy.discretization.modelgrid. Must be of type structured grids

Returns

gridprops - Dictionary containing entries that can be passed directly into the modflow6 disv package.

Return type

`dict`

gridlist_to_verts(gridlist)

Take a list of flopy structured model grids and convert them into vertices. The idomain can be set to remove cells in a parent grid. Cells from a child grid will be patched in to make a single set of vertices. Cells will be numbered according to consecutive numbering of active cells in the grid list.

Parameters

gridlist (`list`) - List of flopy.discretization.modelgrid. Must be of type structured grids

Returns

verts, iverts - vertices and list of cells and which vertices comprise the cells

Return type

np.ndarray, list

isBetween(a, b, c, epsilon=0.001)

segment_face(ivert, ivlist1, ivlist2, vertices)

Check the vertex lists for cell 1 and cell 2. Add a new vertex to cell 1 if necessary.

Parameters

- **ivert** (*int*) - vertex number to check
- **ivlist1** (*list*) - list of vertices for cell 1. Add a new vertex to this cell if needed.
- **ivlist2** (*list*) - list of vertices for cell2.
- **vertices** (*ndarray*) - array of x, y vertices

Returns

segmented - Return True if a face in cell 1 was split up by adding a new vertex

Return type

bool

shapefile_to_cvfd(shp, **kwargs)

shapefile_to_xcyc(shp)

Get cell centroid coordinates

Parameters

shp (*string*) - Name of shape file

Returns

xcyc - x, y coordinates of all polygons in shp

Return type

ndarray

shared_face(ivlist1, ivlist2)

to_cvfd(vertdict, nodestart=None, nodestop=None, skip_hanging_node_check=False, verbose=False)

Convert a vertex dictionary into verts and iverts

Parameters

- **vertdict** - vertdict is a dictionary {icell: [(x1, y1), (x2, y2), (x3, y3), ...]}
- **nodestart** (*int*) - starting node number. (default is zero)
- **nodestop** (*int*) - ending node number up to but not including. (default is len(vertdict))
- **skip_hanging_node_check** (*bool*) - skip the hanging node check. this may only be necessary for quad-based grid refinement. (default is False)
- **verbose** (*bool*) - print messages to the screen. (default is False)

Returns

- **verts** (*ndarray*) - array of x, y vertices

- **iverts** (*list*) - list containing a list for each cell

flopy.utils.datafile module

Module to read MODFLOW output files. The module contains shared abstract classes that should not be directly accessed.

class Header(*filetype=None, precision='single'*)

Bases: `object`

The header class is an abstract base class to create headers for MODFLOW files

get_dtype()

Return the dtype

get_names()

Return the dtype names

get_values()

Return the header values

class LayerFile(*filename: str | PathLike, precision, verbose, kwargs*)

Bases: `object`

The LayerFile class is the abstract base class from which specific derived classes are formed. LayerFile This class should not be instantiated directly.

close()

Close the file handle.

get_alldata(*mflay=None, nodata=-9999*)

Get all of the data from the file.

Parameters

- **mflay** (*integer*) - MODFLOW zero-based layer number to return. If None, then all all layers will be included. (Default is None.)
- **nodata** (*float*) - The nodata value in the data array. All array values that have the nodata value will be assigned `np.nan`.

Returns

data - Array has size (ntimes, nlay, nrow, ncol) if mflay is None or it has size (ntimes, nrow, ncol) if mlay is specified.

Return type

numpy array

Notes

Examples

```
get_data(kstpkper=None, idx=None, totim=None, mflay=None)
```

Get data from the file for the specified conditions.

Parameters

- **idx** (*int*) - The zero-based record number. The first record is record 0.
- **kstpkper** (*tuple of ints*) - A tuple containing the time step and stress period (kstp, kper). These are zero-based kstp and kper values.
- **totim** (*float*) - The simulation time.
- **mflay** (*integer*) - MODFLOW zero-based layer number to return. If None, then all all layers will be included. (Default is None.)

Returns

data - Array has size (nlay, nrow, ncol) if mflay is None or it has size (nrow, ncol) if mlay is specified.

Return type

numpy array

Notes

if both kstpkper and totim are None, will return the last entry

Examples

```
get_kstpkper()
```

Get a list of unique stress periods and time steps in the file

Returns

out - List of unique kstp, kper combinations in binary file. kstp and kper values are presently zero-based.

Return type

list of (kstp, kper) tuples

```
get_times()
```

Get a list of unique times in the file

Returns

out - List contains unique simulation times (totim) in binary file.

Return type

list of floats

```
list_records()
```

Print a list of all of the records in the file `obj.list_records()`

plot(axes=None, kstpker=None, totim=None, mflay=None, filename_base=None, **kwargs)

Plot 3-D model output data in a specific location in LayerFile instance

Parameters

- **axes** (*list of matplotlib.pyplot.axis*) - List of matplotlib.pyplot.axis that will be used to plot data for each layer. If axes=None axes will be generated. (default is None)
- **kstpker** (*tuple of ints*) - A tuple containing the time step and stress period (kstp, kper). These are zero-based kstp and kper values.
- **totim** (*float*) - The simulation time.
- **mflay** (*int*) - MODFLOW zero-based layer number to return. If None, then all all layers will be included. (default is None)
- **filename_base** (*str*) - Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)
- ****kwargs** (*dict*) -

pcolor

[bool] Boolean used to determine if matplotlib.pyplot.pcolormesh plot will be plotted. (default is True)

colorbar

[bool] Boolean used to determine if a color bar will be added to the matplotlib.pyplot.pcolormesh. Only used if pcolor=True. (default is False)

contour

[bool] Boolean used to determine if matplotlib.pyplot.contour plot will be plotted. (default is False)

clabel

[bool] Boolean used to determine if matplotlib.pyplot.clabel will be plotted. Only used if contour=True. (default is False)

grid

[bool] Boolean used to determine if the model grid will be plotted on the figure. (default is False)

masked_values

[list] List of unique values to be excluded from the plot.

file_extension

[str] Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')

Return type

None

Notes

Examples

```
>>> import flopy
>>> hdoobj = flopy.utils.HeadFile('test.hds')
>>> times = hdoobj.get_times()
>>> hdoobj.plot(totim=times[-1])
```

```
to_shapefile(filename: str | PathLike, kstpker=None, totim=None, mflay=None,
              attrib_name='lf_data', verbose=False)
```

Export model output data to a shapefile at a specific location
in LayerFile instance.

Parameters

- **filename** (*str* or *PathLike*) - Shapefile path to write
- **kstpker** (*tuple of ints*) - A tuple containing the time step and stress period (kstp, kper). These are zero-based kstp and kper values.
- **totim** (*float*) - The simulation time.
- **mflay** (*integer*) - MODFLOW zero-based layer number to return. If None, then layer 1 will be written
- **attrib_name** (*str*) - Base name of attribute columns. (default is 'lf_data')
- **verbose** (*bool*) - Whether to print verbose output

Return type

None

Notes

Examples

```
>>> import flopy
>>> hdoobj = flopy.utils.HeadFile('test.hds')
>>> times = hdoobj.get_times()
>>> hdoobj.to_shapefile('test_heads_sp6.shp', totim=times[-1])
```

flopy.utils.datautil module

```
class ArrayIndexIter(array_shape, index_as_tuple=False)
```

Bases: `object`

```
class ConstIter(value)
```

Bases: `object`

```
class DatumUtil
```

Bases: `object`

```

    static is_basic_type(obj)

    static is_float(str)

    static is_int(str)

class FileIter(file_path)
    Bases: object
    close()

class Multilist(mdlist=None, shape=None, callback=None)
    Bases: object

    Class for storing objects in an n-dimensional list which can be iterated through as
    a single list.

    Parameters

        • mdlist (list) - multi-dimensional list to initialize the
          multi-list. either mdlist or both shape and callback must be
          specified

        • shape (tuple) - shape of the multi-list

        • callback (method) - callback method that takes a location in the
          multi-list (tuple) and returns an object to be stored at that
          location in the multi-list

increment_dimension : (dimension, callback)
    increments the size of one of the two dimensions of the multi-list

build_list : (callback)
    builds a multi-list of shape self.list_shape, constructing objects for the list
    using the supplied callback method

first_item : () : object
    gets the first entry in the multi-list

get_total_size : () : int
    returns the total number of entries in the multi-list

in_shape : (indexes) : boolean
    returns whether a tuple of indexes are valid indexes for the shape of the
    multi-list

inc_shape_idx : (indexes) : tuple
    given a tuple of indexes pointing to an entry in the multi-list, returns a
    tuple of indexes pointing to the next entry in the multi-list

first_index : () : tuple
    returns a tuple of indexes pointing to the first entry in the multi-list

indexes : (start_indexes=None, end_indexes=None) : iter(tuple)
    returns an iterator that iterates from the location in the multi-list defined
    by start_indexes to the location in the multi-list defined by end_indexes

elements : () : iter(object)
    returns an iterator that iterates over each object stored in the multi-list

```

```
build_list(callback)

elements()

first_index()

first_item()

get_total_size()

in_shape(indexes)

inc_shape_idx(indexes)

increment_dimension(dimension, callback)

indexes(start_indexes=None, end_indexes=None)

nth_index(n)

class MultiListIter(multi_list, detailed_info=False, iter_leaf_lists=False)
    Bases: object

class NameIter(name, first_not_numbered=True)
    Bases: object

class PathIter(path, first_not_numbered=True)
    Bases: object

class PyListUtil(path=None, max_error=0.01)
    Bases: object

    Class contains miscellaneous methods to work with and compare python lists
    Parameters
        • path (string) - file path to read/write to
        • max_error (float) - maximum acceptable error when doing a compare
          of floating point numbers

is_iterable : (obj : unknown) : boolean
    determines if obj is iterable

is_empty_list : (current_list : list) : boolean
    determines if an n-dimensional list is empty

con_convert : (data : string, data_type : type that has conversion
    operation) : boolean
    returns true if data can be converted into data_type

max_multi_dim_list_size : (current_list : list) : boolean
    determines the max number of items in a multi-dimensional list 'current_list'

first_item : (current_list : list) : variable
    returns the first item in the list 'current_list'

next_item : (current_list : list) : variable
    returns the next item in the list 'current_list'
```

```

array_comp : (first_array : list, second_array : list) : boolean
    compares two lists, returns true if they are identical (with max_error)

spilt_data_line : (line : string) : list
    splits a string apart (using split) and then cleans up the results dealing with
    various MODFLOW input file related delimiters.  returns the delimiter type
    used.

clean_numeric : (text : string) : string
    returns a cleaned up version of 'text' with only numeric characters

save_array_diff : (first_array : list, second_array : list,
                    first_array_name : string, second_array_name : string)
    saves lists 'first_array' and 'second_array' to files first_array_name
    and second_array_name and then saves the difference of the two arrays to
    'debug_array_diff.txt'

save_array(filename: string, multi_array: list)
    saves 'multi_array' to the file 'filename'

array_comp(first_array, second_array)

static clean_numeric(text)

consistent_delim = False

delimiter_list = {':', '1'}

delimiter_used = None

static first_item(current_list)

static has_one_item(current_list)

static is_empty_list(current_list)

static is_iterable(obj)

line_num = 0

static max_multi_dim_list_size(current_list)

static next_item(current_list, new_list=True, nesting_change=0, end_of_list=True)

static next_list(current_list)

numeric_chars = {'-': 0, '.': 0, '0': 0, '1': 0, '2': 0, '3': 0, '4': 0, '5': 0,
                  '6': 0, '7': 0, '8': 0, '9': 0}

quote_list = {'"', "'"}

static reset_delimiter_used()

riv_array_comp(first_array, second_array)

save_array(filename, multi_array)

save_array_diff(first_array, second_array, first_array_name, second_array_name)

```

```

    static split_data_line(line, external_file=False, delimiter_conf_length=15)
clean_filename(file_name)
clean_name(name)
find_keyword(arr_line, keyword_dict)
max_tuple_abs_size(some_tuple)

```

flopy.utils.flopy_io module

Module for input/output utilities

```
flux_to_wel(cbc_file, text, precision='single', model=None, verbose=False)
```

Convert flux in a binary cell budget file to a wel instance

Parameters

- **cbc_file** ((*str*) cell budget file name) -
- **text** ((*str*) text string of the desired flux type (e.g. "drains")) -
- **precision** ((optional *str*) precision of the cell budget file) -
- **model** ((optional) BaseModel instance. If passed, a new ModflowWel) - instance will be added to model
- **verbose** (bool flag passed to CellBudgetFile) -

Return type

flopy.modflow.ModflowWel instance

```
get_next_line(f)
```

Get the next line from a file that is not a blank line

Parameters

f (filehandle) - filehandle to a open file

Returns

line - next non-empty line in a open file

Return type

string

```
get_ts_sp(line)
```

Reader method to get time step and stress period numbers from list files and Modflow other output files

Parameters

line (*str*) - line containing information about the stress period and time step. The line must contain "STRESS PERIOD <x> TIME STEP <y>"

Return type

tuple of stress period and time step numbers

```
get_url_text(url, error_msg=None)
```

Get text from a url.

```
line_parse(line)
```

Convert a line of text into to a list of values. This handles the case where a free formatted MODFLOW input file may have commas in it.

line_strip(line)

Remove comments and replace commas from input text for a free formatted modflow input file

Parameters

line (*str*) - a line of text from a modflow input file

Returns

str

Return type

line with comments removed and commas replaced

loadtxt(file, delimiter=' ', dtype=None, skiprows=0, use_pandas=True, **kwargs)

Use pandas if it is available to load a text file (significantly faster than `n.loadtxt` or `genfromtxt` see <https://stackoverflow.com/q/18259393/>)

Parameters

- **file** (*file* or *str*) - File, filename, or generator to read.
- **delimiter** (*str*, *optional*) - The string used to separate values. By default, this is any whitespace.
- **dtype** (*data-type*, *optional*) - Data-type of the resulting array
- **skiprows** (*int*, *optional*) - Skip the first skiprows lines; default: 0.
- **use_pandas** (*bool*) - If true, the much faster `pandas.read_csv` method is used.
- **kwargs** (*dict*) - Keyword arguments passed to `numpy.loadtxt` or `pandas.read_csv`.

Returns

ra - Numpy record array of file contents.

Return type

`np.recarray`

multi_line_strip(fobj)

Get next line that is not blank or is not a comment line from a free formatted modflow input file

Parameters

fobj (*open file object*) - a line of text from an input file

Returns

str

Return type

line with comments removed and commas replaced

pop_item(line, dtype=<class 'str'>)**read_fixed_var(line, ncol=1, length=10, ipos=None, free=False)**

Parse a fixed format line using user provided data

Parameters

- **line** (*str*) - text string to parse.
- **ncol** (*int*) - number of columns to parse from line. (default is 1)

- **length** (*int*) - length of each column for fixed column widths. (default is 10)
- **ipos** (*list, int, or numpy array*) - user-provided column widths. (default is None)
- **free** (*bool*) - boolean indicating if sting is free format. ncol, length, and ipos are not used if free is True. (default is False)

Returns

out - padded list containing data parsed from the passed text string

Return type

list

relpath_safe(*path: str | PathLike, start: str | PathLike = '.', scrub: bool = False*) → *str*

Return a relative version of the path starting at the given start path. This is impossible on Windows if the paths are on different drives, in which case the absolute path is returned. The builtin `os.path.relpath` raises a `ValueError`, this method is a workaround to avoid interrupting normal control flow (background at <https://bugs.python.org/issue7195>).

This method also truncates/obfuscates absolute paths with usernames.

Parameters

- **path** (*str or PathLike*) - the path to truncate relative to the start path
- **start** (*str or PathLike, default "."*) - the starting path, defaults to the current working directory
- **scrub** (*bool, default False*) - whether to remove the current login name from paths

Returns

- **str** (the relative path, unless the platform is Windows and the *path*)
- is not on the same drive as *start*, in which case the absolute path,
- *with elements before and including usernames removed and obfuscated*

scrub_login(*s: str*) → *str*

Remove the current login name from the given string, replacing any occurrences with "***".

Parameters

s (*str*) - the input string

Return type

the string with login name obfuscated

ulstrd(*f, nlist, ra, model, sfac_columns, ext_unit_dict*)

Read a list and allow for open/close, binary, external, sfac, etc.

Parameters

- **f** (*file handle*) - file handle for where the list is being read from

- **nlist** (*int*) - size of the list (number of rows) to read
- **ra** (*np.recarray*) - A record array of the correct size that will be filled with the list
- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **sfac_columns** (*list*) - A list of strings containing the column names to scale by sfac
- **ext_unit_dict** (*dictionary, optional*) - If the list in the file is specified using EXTERNAL, then in this case ext_unit_dict is required, which can be constructed using the function *flopy.utils.mfreadnam.parsenamefile*.

write_fixed_var(*v*, *length=10*, *ipos=None*, *free=False*, *comment=None*)

Parameters

- **v** (*list, int, float, bool, or numpy array*) - list, int, float, bool, or numpy array containing the data to be written to a string.
- **length** (*int*) - length of each column for fixed column widths. (default is 10)
- **ipos** (*list, int, or numpy array*) - user-provided column widths. (default is None)
- **free** (*bool*) - boolean indicating if a free format string should be generated. length and ipos are not used if free is True. (default is False)
- **comment** (*str*) - comment string to add to the end of the string

Returns

out - fixed or free format string generated using user-provided data

Return type

str

flopy.utils.formattedfile module

Module to read MODFLOW formatted output files. The module contains one important classes that can be accessed by the user.

- FormattedHeadFile (Formatted head file. Can also be used for drawdown)

class FormattedHeadFile(*filename*, *text='head'*, *precision='single'*, *verbose=False*, ***kwargs*)

Bases: *FormattedLayerFile*

FormattedHeadFile Class.

Parameters

- **filename** (*string*) - Name of the formatted head file
- **text** (*string*) - Name of the text string in the formatted head file. Default is 'head'
- **precision** (*string*) - 'single' or 'double'. Default is 'single'.

- **verbose** (*bool*) - Write information to the screen. Default is False.

Notes

The FormattedHeadFile class provides simple ways to retrieve 2d and 3d head arrays from a MODFLOW formatted head file and time series arrays for one or more cells.

The FormattedHeadFile class is built on a record array consisting of headers, which are record arrays of the modflow header information (kstp, kper, pertim, totim, text, nrow, ncol, ilay) and long integers, which are pointers to first bytes of data for the corresponding data array.

FormattedHeadFile can only read formatted head files containing headers. Use the LABEL option in the output control file to generate head files with headers.

Examples

```
>>> import flopy.utils.formattedfile as ff
>>> hdoobj = ff.FormattedHeadFile('model.fhd', precision='single')
>>> hdoobj.list_records()
>>> rec = hdoobj.get_data(kstpkper=(1, 50))
>>> rec2 = ddnoobj.get_data(totim=100.)
```

class FormattedHeader(text_ident, precision='single')

Bases: [Header](#)

The TextHeader class is a class to read in headers from MODFLOW formatted files.

Parameters

- **type** (*text_ident is the text string in the header that identifies the*) -
- **point** (*of data (eg. 'head') precision is the precision of the floating*) -
- **file** (*data in the*) -

read_header(text_file)

Read header information from a formatted file

Parameters

- **of** (*text_file is an open file object currently at the beginning*) -
- **header** (*the*) -

Returns

- **out** (*numpy array of header information*)
- *also stores the header's format string as self.format_string*

class FormattedLayerFile(filename, precision, verbose, kwargs)

Bases: [LayerFile](#)

The FormattedLayerFile class is the super class from which specific derived classes are formed. This class should not be instantiated directly

close()

Close the file handle.

get_ts(idx)

Get a time series from the formatted file.

Parameters

idx (*tuple of ints, or a list of a tuple of ints*) - idx can be (layer, row, column) or it can be a list in the form [(layer, row, column), (layer, row, column), ...]. The layer, row, and column values must be zero based.

Returns

out - Array has size (ntimes, ncells + 1). The first column in the data array will contain time (totim).

Return type

numpy array

Notes

The layer, row, and column values must be zero-based, and must be within the following ranges: $0 \leq k < nlay$; $0 \leq i < nrow$; $0 \leq j < ncol$

Examples

is_float(s)

is_int(s)

flopy.utils.geometry module

Container objects for working with geometric information

class Collection(geometries=())

Bases: `list`

The collection object is container for a group of flopy geometries

This class acts as a base class for MultiPoint, MultiLineString, and MultiPolygon classes. This class can also accept a mix of geometries and act as a stand alone container.

Parameters

geometries (*list*) - list of flopy.util.geometry objects

property bounds

Method to calculate the bounding box of the collection

Return type

tuple (xmin, ymin, xmax, ymax)

plot(ax=None, **kwargs)

Plotting method for collection

Parameters

- **ax** (*matplotlib.axes object*) -

- **kwargs** (*keyword arguments*) - matplotlib keyword arguments

Return type

matplotlib.axes object

class `LineString(coordinates)`

Bases: `Shape`

property `bounds`

has_z = `False`

plot(`ax=None, **kwargs`)

property `pyshp_parts`

shapeType = `3`

type = `'LineString'`

property `x`

property `y`

property `z`

class `MultiLineString(linestrings=())`

Bases: `Collection`

Container for housing and describing multilinestring geometries (e.g. to be read or written to shapefiles or other geographic data formats)

Parameters:

polygons

[list] list of flopy.utils.geometry.LineString objects

class `MultiPoint(points=())`

Bases: `Collection`

Container for housing and describing multipoint geometries (e.g. to be read or written to shapefiles or other geographic data formats)

Parameters:

polygons

[list] list of flopy.utils.geometry.Point objects

class `MultiPolygon(polygons=())`

Bases: `Collection`

Container for housing and describing multipolygon geometries (e.g. to be read or written to shapefiles or other geographic data formats)

Parameters:**polygons**

[list] list of flopy.utils.geometry.Polygon objects

class Point(*coordinates)

Bases: [Shape](#)

property bounds

has_z = False

plot(ax=None, **kwargs)

property pyshp_parts

shapeType = 1

type = 'Point'

property x

property y

property z

class Polygon(exterior, interiors=None)

Bases: [Shape](#)

property bounds

get_patch(**kwargs)

property patch

plot(ax=None, **kwargs)

Plot the feature. :param ax: :type ax: matplotlib.pyplot axes instance
:param Accepts keyword arguments to descartes.PolygonPatch. Requires the:
:param descartes package (pip install descartes).:

property pyshp_parts

shapeType = 5

type = 'Polygon'

class Shape(shapetype, coordinates=None, exterior=None, interiors=None)

Bases: [object](#)

Parent class for handling geo interfacing, do not instantiate directly

Parameters:

type

[str] shapetype string

coordinates

[list or tuple] list of tuple of point or linestring coordinates

exterior

[list or tuple] 2d list of polygon coordinates

interiors

[list or tuple] 2d or 3d list of polygon interiors

static from_geojson(*geo_interface*)

Method to load from geojson

Parameters

geo_interface (*geojson*, *dict*) - geojson compliant representation of a linestring

Return type

Polygon, *LineString*, or *Point*

property geojson

get_polygon_area(*geom*)

Calculate the area of a closed polygon

Parameters

geom (*geospatial representation of polygon*) - accepted types:

vertices np.array([(x, y),...]) geojson.Polygon shapely.Polygon
shapefile.Shape

Returns

area - area of polygon centroid

Return type

float

get_polygon_centroid(*geom*)

Calculate the centroid of a closed polygon

Parameters

geom (*geospatial representation of polygon*) - accepted types:

vertices np.array([(x, y),...]) geojson.Polygon shapely.Polygon
shapefile.Shape

Returns

centroid - (x, y) of polygon centroid

Return type

tuple

is_clockwise(**geom*)

Determine if a ring is defined clockwise

Parameters

***geom** (*geospatial representation of polygon*) - accepted types:

vertices [(x, y),...]) geojson.Polygon shapely.Polygon
shapefile.Shape x and y vertices: [x1, x2, x3], [y1, y2, y3]

Returns

clockwise - True when the ring is defined clockwise, False otherwise

Return type

bool

point_in_polygon(xc, yc, polygon)

Use the ray casting algorithm to determine if a point is within a polygon. Enables very fast intersection calculations!

Parameters

- **xc** (np.ndarray) - 2d array of xpoints
- **yc** (np.ndarray) - 2d array of ypoints
- **polygon** (iterable (list)) - polygon vertices [(x0, y0), ..., (xn, yn)] note: polygon can be open or closed

Returns

mask - True value means point is in polygon!

Return type

np.array

project_point_onto_xc_line(line, pts, d0=0, direction='x')

Method to project points onto a cross sectional line that is defined by distance. Used for plotting MODPATH results on to a cross section!

line

[list or np.ndarray] numpy array of [(x0, y0), (x1, y1)] that defines the line to project on to

pts

[list or np.ndarray] numpy array of [(x, y),] points to be projected

d0 : distance offset along line of min(x1) **direction** : string
projection direction "x" or "y"

Returns

np.ndarray of projected [(x, y),] points

rotate(x, y, xoff, yoff, angrot_radians)

Given x and y array-like values calculate the rotation about an arbitrary origin and then return the rotated coordinates.

transform(x, y, xoff, yoff, angrot_radians, length_multiplier=1.0, inverse=False)

Given x and y array-like values calculate the translation about an arbitrary origin and then return the rotated coordinates.

flopy.utils.geospatial_utils module

class GeoSpatialCollection(obj, shapetype=None)

Bases: object

The GeoSpatialCollection class allows a user to convert between Collection objects from common geospatial libraries.

Parameters

- **obj** (collection object) - obj can accept the following types
str : shapfile path PathLike : shapfile path shapfile.Reader
object list of [shapfile.Shape, shapfile.Shape,]

shapefile.Shapes object flopy.utils.geometry.Collection
 object list of [flopy.utils.geometry, ...] objects
 geojson.GeometryCollection object geojson.FeatureCollection object
 shapely.GeometryCollection object list of [[vertices], ...]

- **shapetype** (*list*) - optional list of shapetypes that is required when vertices are supplied to the class as the obj parameter

property flopy_geometry

Property that returns a flopy.util.geometry.Collection object

Return type

flopy.util.geometry.Collection object

property geojson

Property that returns a geojson.GeometryCollection object to the user

Return type

geojson.GeometryCollection

property points

Property returns a multidimensional list of vertices

Return type

list of vertices

property shape

Property that returns a shapefile.Shapes object

Return type

shapefile.Shapes object

property shapely

Property that returns a shapely.geometry.collection.GeometryCollection object to the user

Return type

shapely.geometry.collection.GeometryCollection object

property shapetype

Returns a list of shapetypes to the user

Return type

list of *str*

class GeoSpatialUtil(*obj*, *shapetype*=None)

Bases: *object*

Geospatial utils are a unifying method to provide conversion between commonly used geospatial input types

Parameters

- **obj** (*geospatial object*) -
 obj can accept any of the following objects:
 shapefile.Shape object flopy.utils.geometry objects list of
 vertices geojson geometry objects shapely.geometry objects
- **shapetype** (*str*) - shapetype is required when a list of vertices is supplied for obj

property flopy_geometry

Returns a flopy geometry object to the user

Return type

`flopy.utils.geometry.<Shape>`

property geojson

Returns a geojson object to the user

Return type

`geojson.<shape>`

property points

Returns a list of vertices to the user

Return type

`list`

property shape

Returns a shapefile.Shape object to the user

Return type

`shapefile.shape`

property shapely

Returns a shapely.geometry object to the user

Return type

`shapely.geometry.<shape>`

property shapetype

Shapetype string for a geometry

Return type

`str`

flopy.utils.get_modflow module

Download and install USGS MODFLOW and related programs.

This script originates from FloPy: <https://github.com/modflowpy/flopy> This file can be downloaded and run independently outside FloPy. It requires Python 3.6 or later, and has no dependencies.

See <https://developer.github.com/v3/repos/releases/> for GitHub Releases API.

run_main(*bindir*, *repo*='executables', *release_id*='latest', *ostag*=None, *subset*=None, *downloads_dir*=None, *force*=False, *quiet*=False, *_is_cli*=False)

Run main method to get MODFLOW and related programs.

Parameters

- **bindir** (*str* or *Path*) - Writable path to extract executables. Auto-select options start with a colon character. See error message or other documentation for further information on auto-select options.
- **repo** (*str*, default "executables") - Name of GitHub repository. Choose one of "executables" (default), "modflow6", or "modflow6-nightly-build".

- **release_id** (*str*, default "latest") - GitHub release ID.
- **ostag** (*str*, optional) - Operating system tag; default is to automatically choose.
- **subset** (*list*, *set* or *str*, optional) - Optional subset of executables to extract, specified as a list (e.g.) ["mfwt", "mp6"] or a comma-separated string "mfwt,mp6".
- **downloads_dir** (*str* or *Path*, optional) - Manually specify directory to download archives. Default is to use home Downloads, if available, otherwise a temporary directory.
- **force** (*bool*, default False) - If True, always download archive. Default False will use archive if previously downloaded in downloads_dir.
- **quiet** (*bool*, default False) - If True, show fewer messages.
- **_is_cli** (*bool*, default False) - Control behavior of method if this is run as a command-line interface or as a Python function.

flopy.utils.gridgen module

```
class Gridgen(modelgrid, model_ws: str | PathLike = '.', exe_name: str | PathLike =
'gridgen', surface_interpolation='replicate', vertical_pass_through=False,
**kwargs)
```

Bases: `object`

Class to work with the gridgen program to create layered quadtree grids.

Parameters

- **modelgrid** (*flopy.discretization.StructuredGrid*) - Flopy StructuredGrid object. Note this also accepts ModflowDis and ModflowGwfdis objects, however it is deprecated and support will be removed in version 3.3.7
- **model_ws** (*str* or *PathLike*) - workspace location for creating gridgen files (default is '.')
- **exe_name** (*str*) - path and name of the gridgen program. (default is gridgen)
- **surface_interpolation** (*str*) - Default gridgen method for interpolating elevations. Valid options include 'replicate' (default) and 'interpolate'
- **vertical_pass_through** (*bool*) - If true, Gridgen's GRID_TO_USGDATA command will connect layers where intermediate layers are inactive. (default is False)
- ****kwargs** -
 - vertical_smoothing_level**
[int] maximum level difference between two vertically adjacent cells. Adjust with caution, as adjustments can cause unexpected results to simulated flows
 - horizontal_smoothing_level**
[int] maximum level difference between two horizontally

adjacent cells. Adjust with caution, as adjustments can cause unexpected results to simulated flows

Notes

For the surface elevations, the top of a layer uses the same surface as the bottom of the overlying layer.

`add_active_domain(feature, layers)`

Parameters

- **feature** (*str* or *list*) -
feature can be:
 a string containing the name of a polygon a list of polygons flopy.utils.geometry.Collection object of Polygons shapely.geometry.Collection object of Polygons geojson.GeometryCollection object of Polygons list of shapefile.Shape objects shapefile.Shapes object
- **layers** (*list*) - A list of layers (zero based) for which this active domain applies.

Return type

None

`add_refinement_features(features, featuretype, level, layers)`

Parameters

- **features** (*str*, *list*, or *collection object*) -
features can be
 a string containing the name of a shapefile a list of points, lines, or polygons flopy.utils.geometry.Collection object a list of flopy.utils.geometry objects shapely.geometry.Collection object geojson.GeometryCollection object a list of shapefile.Shape objects shapefile.Shapes object
- **featuretype** (*str*) - Must be either 'point', 'line', or 'polygon'
- **level** (*int*) - The level of refinement for this features
- **layers** (*list*) - A list of layers (zero based) for which this refinement features applies.

Return type

None

`build(verbose=False)`

Build the quadtree grid

Parameters

- verbose** (*bool*) - If true, print the results of the gridgen command to the terminal (default is False)

Return type

None

export(verbose=False)

Export the quadtree grid to shapefiles, usgdata, and vtk

Parameters

verbose (*bool*) - If true, print the results of the gridgen command to the terminal (default is False)

Return type

None

get_angldegx(fldr=None)

Get the angldegx array

Parameters

fldr (*ndarray*) - Flow direction indicator array. If None, then it is read from gridgen output.

Returns

angldegx - A 1D vector indicating the angle (in degrees) between the x axis and an outward normal to the face.

Return type

ndarray

get_area()

Get the area array

Returns

area - A 1D vector of cell areas of size nodes

Return type

ndarray

get_bot()

Get the bot array

Returns

bot - A 1D vector of cell bottom elevations of size nodes

Return type

ndarray

get_cellxy(ncells)

Parameters

ncells (*int*) - Number of cells for which to create the list of cell centers

Returns

cellxy - x and y cell centers. Shape is (ncells, 2)

Return type

ndarray

get_center(nodenumbr)

Return the cell center x and y coordinates

Parameters

nodenumbr -

Returns

(**x**, **y**)

Return type

tuple

get_cl12()

Get the cl12 array

Returns

cl12 - A 1D vector of the cell connection distances, which are from the center of cell n to its shared face with cell m

Return type

ndarray

get_disu(model, nper=1, perlen=1, nstp=1, tsmult=1, steady=True, itmuni=4, lenuni=2)

Create a MODFLOW-USG DISU flopy object.

Parameters

- **model** (*Flopy model object*) - The Flopy model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **nper** (*int*) - Number of model stress periods (default is 1).
- **perlen** (*float or array of floats (nper)*) - A single value or array of the stress period lengths (default is 1).
- **nstp** (*int or array of ints (nper)*) - Number of time steps in each stress period (default is 1).
- **tsmult** (*float or array of floats (nper)*) - Time step multiplier (default is 1.0).
- **steady** (*boolean or array of boolean (nper)*) - True or False indicating whether or not stress period is steady state (default is True).
- **itmuni** (*int*) - Time units, default is days (4)
- **lenuni** (*int*) - Length units, default is meters (2)

Returns

disu

Return type

Flopy ModflowDisU object.

get_fahl()

Get the fahl array

Returns

fahl - A 1D vector of the cell connection information, which is flow area for a vertical connection and horizontal length for a horizontal connection

Return type

ndarray

get_fldr()

Get the fldr array

Returns

fldr - A 1D vector indicating the direction of the connection

1, 2, and 3 are plus x, y, and z directions. -1, -2, and -3 are negative x, y, and z directions.

Return type
ndarray

get_gridprops_disu5()

Get a dictionary of information needed to create a MODFLOW-USG DISU Package. The returned dictionary can be unpacked directly into the ModflowDisU constructor. The ja dictionary entry will be returned as zero-based.

Returns
gridprops

Return type
dict

get_gridprops_disu6(repair_asymmetry=True)

Get a dictionary of information needed to create a MODFLOW 6 DISU Package. The returned dictionary can be unpacked directly into the ModflowGwfdisu constructor.

Parameters
repair_asymmetry (*bool*) - MODFLOW 6 checks for symmetry in the hwva array, and round off errors in the floating point calculations can result in small errors. If this flag is true, then symmetry will be forced by setting the symmetric counterparts to the same value (the first one encountered).

Returns
gridprops

Return type
dict

get_gridprops_disv()

Get a dictionary of information needed to create a MODFLOW 6 DISV Package. The returned dictionary can be unpacked directly into the ModflowGwfdisv constructor.

Returns
gridprops

Return type
dict

get_gridprops_unstructuredgrid()

Get a dictionary of information needed to create a flopy UnstructuredGrid. The returned dictionary can be unpacked directly into the flopy.discretization.UnstructuredGrid() constructor.

Returns
gridprops

Return type
dict

get_gridprops_vertexgrid()

Get a dictionary of information needed to create a flopy VertexGrid.

The returned dictionary can be unpacked directly into the `flopy.discretization.VertexGrid()` constructor.

Returns
gridprops

Return type
`dict`

get_hwva(*ja=None, ihc=None, fahl=None, top=None, bot=None*)

Get the hwva array

Parameters

- **ja** (*ndarray*) - Cell connectivity. If None, it will be read from gridgen output.
- **ihc** (*ndarray*) - Connection horizontal indicator array. If None it will be read and calculated from gridgen output.
- **fahl** (*ndarray*) - Flow area, horizontal width array required by MODFLOW-USG. If none then it will be read from the gridgen output. Default is None.
- **top** (*ndarray*) - Cell top elevation. If None, it will be read from gridgen output.
- **bot** (*ndarray*) - Cell bottom elevation. If None, it will be read from gridgen output.

Returns
fahl - A 1D vector of the cell connection information, which is flow area for a vertical connection and horizontal length for a horizontal connection

Return type
`ndarray`

get_iac()

Get the iac array

Returns
iac - A 1D vector of the number of connections (plus 1) for each cell

Return type
`ndarray`

get_ihc(*nodelay=None, ia=None, fldr=None*)

Get the ihc array

Parameters
fldr (*ndarray*) - Flow direction indicator array. If None, then it is read from gridgen output.

Returns
ihc - A 1D vector indicating the direction of the connection where 0 is vertical, 1 is a regular horizontal connection and 2 is a vertically staggered horizontal connection.

Return type
`ndarray`

get_ivc(*fldr=None*)

Get the MODFLOW-USG ivc array

Parameters

fldr (*ndarray*) - Flow direction indicator array. If None, then it is read from gridgen output.

Returns

ivc - A 1D vector indicating the direction of the connection where 1 is vertical and 0 is horizontal.

Return type

ndarray

get_ja(*nja=None*)

Get the zero-based ja array

Parameters

nja (*int*) - Number of connections. If None, then it is read from gridgen output.

Returns

ja - A 1D vector of the cell connectivity (one-based)

Return type

ndarray

get_nlay()

Get the number of layers

Returns

nlay

Return type

int

get_nod_reccarray()

Load the qtg.nod file and return as a numpy recarray

Returns

node_ra - Recarray representation of the node file with zero-based indexing

Return type

ndarray

get_nodelay()

Return the nodelay array, which is an array of size nlay containing the number of nodes in each layer.

Returns

nodelay - Number of nodes in each layer

Return type

ndarray

get_nodes()

Get the number of nodes

Returns

nodes

Return type`int`**get_top()**

Get the top array

Returns**top** - A 1D vector of cell top elevations of size nodes**Return type**`ndarray`**get_vertices(nodenumbr)**

Return a list of 5 vertices for the cell. The first vertex should be the same as the last vertex.

Parameters**nodenumbr** -**Returns****list of vertices****Return type**`list`**get_verts_iverts(ncells, verbose=False)**

Return a 2d array of x and y vertices and a list of size ncells that has the list of vertices for each cell.

Parameters

- **ncells** (`int`) - The number of entries in iverts. This should be ncpl for a layered model and nodes for a disu model.
- **verbose** (`bool`) - Print information as its working

Returns**verts, iverts** - **verts** is a 2d array of x and y vertex pairs (nvert, 2) and **iverts** is a list of vertices that comprise each cell**Return type**`tuple`**static gridarray_to_flopyusg_gridarray(nodelay, a)****intersect(features, featuretype, layer)****Parameters**

- **features** (`str` or `list`) - features can be either a string containing the name of a shapefile or it can be a list of points, lines, or polygons
- **featuretype** (`str`) - Must be either 'point', 'line', or 'polygon'
- **layer** (`int`) - Layer (zero based) to intersect with. Zero based.

Returns**result** - Recarray of the intersection properties.

Return type

`np.recarray`

`plot(ax=None, layer=0, edgecolor='k', facecolor='none', cmap='Dark2', a=None, masked_values=None, **kwargs)`

Plot the grid. This method will plot the grid using the shapefile that was created as part of the build method.

Note that the layer option is not working yet.

Parameters

- **ax** (*matplotlib.pyplot axis*) - The plot axis. If not provided it, `plt.gca()` will be used. If there is not a current axis then a new one will be created.
- **layer** (*int*) - Layer number to plot
- **cmap** (*string*) - Name of colormap to use for polygon shading (default is 'Dark2')
- **edgecolor** (*string*) - Color name. (Default is 'scaled' to scale the edge colors.)
- **facecolor** (*string*) - Color name. (Default is 'scaled' to scale the face colors.)
- **a** (*numpy.ndarray*) - Array to plot.
- **masked_values** (*iterable of floats, ints*) - Values to mask.
- **kwargs** (*dictionary*) - Keyword arguments that are passed to `PatchCollection.set(**kwargs)`. Some common kwargs would be 'linewidths', 'linestyles', 'alpha', etc.

Returns

`pc`

Return type

`matplotlib.collections.PatchCollection`

`set_surface_interpolation(isurf, type, elev=None, elev_extent=None)`

Parameters

- **isurf** (*int*) - surface number where 0 is top and `nlay + 1` is bottom
- **type** (*str*) - Must be 'INTERPOLATE', 'REPLICATE' or 'ASCIIGRID'.
- **elev** (*numpy.ndarray of shape (nr, nc) or str*) - Array that is used as an asciigrid. If elev is a string, then it is assumed to be the name of the asciigrid.
- **elev_extent** (*list-like*) - List of `xmin`, `xmax`, `ymin`, `ymax` extents of the elev grid. Must be specified for ASCIIGRID; optional otherwise.

Return type

`None`

features_to_shapefile(*features*, *featuretype*, *filename*: *str* | *PathLike*)

Write a shapefile for the features of type *featuretype*.

Parameters

- **features** (*list*) - point, line, or polygon features. Method accepts feature can be:
a list of geometries `flopy.utils.geometry.Collection`
object `shapely.geometry.Collection` object
`geojson.GeometryCollection` object list of `shapefile.Shape` objects
`shapefile.Shapes` object
- **featuretype** (*str*) - Must be 'point', 'line', 'linestring', or 'polygon'
- **filename** (*str* or *PathLike*) - Path of the shapefile to write

Return type

None

get_ia_from_iac(*iac*)

get_isym(*ia*, *ja*)

is_symmetrical(*isym*, *a*, *atol*=0)

ndarray_to_asciigrid(*fname*: *str* | *PathLike*, *a*, *extent*, *nodata*=1e+30)

readld(*f*, *a*)

Quick file to array reader for reading gridgen output. Much faster than the `readld` function in `util_array`

repair_array_asymmetry(*isym*, *a*, *atol*=0)

flopy.utils.gridintersect module

class GridIntersect(*mfgrid*, *method*=None, *rtree*=True)

Bases: `object`

Class for intersecting shapely geometries (Point, Linestring, Polygon, or their Multi variants) with MODFLOW grids. Contains optimized search routines for structured grids.

Notes

- The STR-tree query is based on the bounding box of the shape or collection, if the bounding box of the shape covers nearly the entire grid, the query won't be able to limit the search space much, resulting in slower performance. Therefore, it can sometimes be faster to intersect each individual shape in a collection than it is to intersect with the whole collection at once.
- Building the STR-tree can take a while for large grids. Once built the intersect routines (for individual shapes) should be pretty fast.
- The optimized routines for structured grids can outperform the shapely routines for point and linestring intersections because of the reduced overhead of building and parsing the STR-tree. However, for polygons the STR-tree implementation is often faster than the optimized structured routines, especially for larger grids.

filter_query_result(*cellids*, *shp*)

Filter array of geometries to obtain grid cells that intersect with shape.

Used to (further) reduce query result to cells that intersect with shape.

Parameters

- **cellids** (*iterable*) - iterable of cellids, query result
- **shp** (*shapely.geometry*) - shapely geometry that is prepared and used to filter query result

Returns

filter or generator containing polygons that intersect with shape

Return type

array_like

intersect(*shp*, *shapetype=None*, *sort_by_cellid=True*, *keepzerolengths=False*, *return_all_intersections=False*, *contains_centroid=False*, *min_area_fraction=None*, *shapely2=True*)

Method to intersect a shape with a model grid.

Parameters

- **shp** (*shapely.geometry*, *geojson object*, *shapefile.Shape*,) - or flopy geometry object
- **shapetype** (*str*, *optional*) - type of shape (i.e. "point", "linestring", "polygon" or their multi-variants), used by GeoSpatialUtil if shp is passed as a list of vertices, default is None
- **sort_by_cellid** (*bool*) - sort results by cellid, ensures cell with lowest cellid is returned for boundary cases when using vertex methods, default is True
- **keepzerolengths** (*bool*) - boolean method to keep zero length intersections for linestring intersection, only used if shape type is "linestring"
- **return_all_intersections** (*bool*, *optional*) - if True, return multiple intersection results for points or linestrings on grid cell boundaries (e.g. returns 2 intersection results if a point lies on the boundary between two grid cells). The default is False. Only used if shape type is "point" or "linestring".
- **contains_centroid** (*bool*, *optional*) - if True, only store intersection result if cell centroid is contained within intersection shape, only used if shape type is "polygon"
- **min_area_fraction** (*float*, *optional*) - float defining minimum intersection area threshold, if intersection area is smaller than min_frac_area * cell_area, do not store intersection result, only used if shape type is "polygon"
- **shapely2** (*bool*, *optional*) - temporary flag to determine whether to use methods optimized for shapely 2.0. Useful for comparison performance between the old (shapely 1.8) and new (shapely 2.0) implementations.

Returns

a record array containing information about the intersection

Return type

`numpy.recarray`

intersects(*shp*, *shapetype=None*)

Return cellids for grid cells that intersect with shape.

Parameters

- **shp** (*shapely.geometry*, *geojson geometry*, *shapefile.shape*,) - or flopy geometry object shape to intersect with the grid
- **shapetype** (*str*, *optional*) - type of shape (i.e. "point", "linestring", "polygon" or their multi-variants), used by GeoSpatialUtil if shp is passed as a list of vertices, default is None

Returns

a record array containing cell IDs of the gridcells the shape intersects with

Return type

`numpy.recarray`

static plot_linestring(*rec*, *ax=None*, *cmap=None*, ***kwargs*)

method to plot the linestring intersection results from the resulting `numpy.recarray`.

Note: only works when recarray has 'intersects' column!

Parameters

- **rec** (`numpy.recarray`) - record array containing intersection results (the resulting shapes)
- **ax** (*matplotlib.pyplot.axes*, *optional*) - axes to plot onto, if not provided, creates a new figure
- **cmap** (*str*) - matplotlib colormap
- ****kwargs** - passed to the plot function

Returns

returns the axes handle

Return type

`matplotlib.pyplot.axes`

static plot_point(*rec*, *ax=None*, ***kwargs*)

method to plot the point intersection results from the resulting `numpy.recarray`.

Note: only works when recarray has 'intersects' column!

Parameters

- **rec** (`numpy.recarray`) - record array containing intersection results
- **ax** (*matplotlib.pyplot.axes*, *optional*) - axes to plot onto, if not provided, creates a new figure

- ****kwargs** - passed to the scatter function

Returns

returns the axes handle

Return type

matplotlib.pyplot.axes

static plot_polygon(rec, ax=None, **kwargs)

method to plot the polygon intersection results from the resulting numpy.recarray.

Note: only works when recarray has 'intersects' column!

Parameters

- **rec** (*numpy.recarray*) - record array containing intersection results (the resulting shapes)
- **ax** (*matplotlib.pyplot.axes, optional*) - axes to plot onto, if not provided, creates a new figure
- ****kwargs** - passed to the plot function

Returns

returns the axes handle

Return type

matplotlib.pyplot.axes

query_grid(shp)

Perform spatial query on grid with shapely geometry. If no spatial query is possible returns all grid cells.

Parameters

shp (*shapely.geometry*) - shapely geometry

Returns

array containing cellids of grid cells in query result

Return type

array_like

static sort_gridshapes(geoms, cellids)

Sort geometries (from i.e. query result) by cell id.

Deprecated since version 3.3.6: sorting is now performed on cellids.

Parameters

geoms (*iterable*) - list or iterable of geometries

Returns

sorted list of gridcells

Return type

list

class ModflowGridIndices

Bases: *object*

Collection of methods that can be used to find cell indices for a structured, but irregularly spaced MODFLOW grid.

static find_position_in_array(arr, x)

If arr has x positions for the left edge of a cell, then return the cell index containing x.

Parameters

- **arr** (*A one dimensional array (such as Xe) that contains*) - coordinates for the left cell edge.
- **x** (*float*) - The x position to find in arr.

static kij_from_nn0(n, nlay, nrow, ncol)

Convert the node number to a zero-based layer, row and column format. Return (k0, i0, j0).

Parameters

- **nodenumber** (*int*) - The cell nodenumber, ranging from 0 to number of nodes - 1.
- **nlay** (*int*) - The number of layers.
- **nrow** (*int*) - The number of rows.
- **ncol** (*int*) - The number of columns.

static kij_from_nodenumber(nodenumber, nlay, nrow, ncol)

Convert the modflow node number to a zero-based layer, row and column format. Return (k0, i0, j0).

Parameters

- **nodenumber** (*int*) - The cell nodenumber, ranging from 1 to number of nodes.
- **nlay** (*int*) - The number of layers.
- **nrow** (*int*) - The number of rows.
- **ncol** (*int*) - The number of columns.

static nn0_from_kij(k, i, j, nrow, ncol)

Calculate the zero-based nodenumber using the zero-based layer, row, and column values. The first node has a value of 0.

Parameters

- **k** (*int*) - The model layer number as a zero-based value.
- **i** (*int*) - The model row number as a zero-based value.
- **j** (*int*) - The model column number as a zero-based value.
- **nrow** (*int*) - The number of model rows.
- **ncol** (*int*) - The number of model columns.

static nodenumber_from_kij(k, i, j, nrow, ncol)

Calculate the nodenumber using the zero-based layer, row, and column values. The first node has a value of 1.

Parameters

- **k** (*int*) - The model layer number as a zero-based value.
- **i** (*int*) - The model row number as a zero-based value.

- **j** (*int*) - The model column number as a zero-based value.
- **nrow** (*int*) - The number of model rows.
- **ncol** (*int*) - The number of model columns.

ignore_shapely2_stree_warning()

ignore_shapely_warnings_for_object_array()

parse_shapely_ix_result(*collection*, *ix_result*, *shptyps=None*)

Recursive function for parsing shapely intersection results. Returns a list of shapely shapes matching shptyps.

Parameters

- **collection** (*list*) - state variable for storing result, generally an empty list
- **ix_result** (*shapely.geometry type*) - any shapely intersection result
- **shptyps** (*str, list of str, or None, optional*) - if None (default), return all types of shapes. if str, return shapes of that type, if list of str, return all types in list

Returns

list containing shapely geometries of type shptyps

Return type

list

flopy.utils.gridutil module

Grid utilities

get_disu_kwargs(*nlay*, *nrow*, *ncol*, *delr*, *delc*, *tp*, *botm*)

Create args needed to construct a DISU package.

Parameters

- **nlay** (*int*) - Number of layers
- **nrow** (*int*) - Number of rows
- **ncol** (*int*) - Number of columns
- **delr** (*numpy.ndarray*) - Column spacing along a row
- **delc** (*numpy.ndarray*) - Row spacing along a column
- **tp** (*int or numpy.ndarray*) - Top elevation(s) of cells in the model's top layer
- **botm** (*numpy.ndarray*) - Bottom elevation(s) of all cells in the model

get_lni(*ncpl*, *nodes*) → *List[Tuple[int, int]]*

Get layer index and within-layer node index (both 0-based).

Node count per layer may be an int or array-like of integers.

An integer ncpl indicates all layers have the same node count. If an integer ncpl is less than any specified node numbers, the grid is understood to have at least enough layers to contain them.

If ncpl is array-like it is understood to describe node count

per zero-indexed layer.

Parameters

- **ncpl** (node count per layer (*int* or array-like of ints)) -
- **nodes** (node numbers (array-like of nodes)) -

Return type

A list of tuples (layer index, node index)

uniform_flow_field(qx, qy, qz, shape, delr=None, delc=None, delv=None)

flopy.utils.lgrutil module

class Lgr(nlapp, nrowp, ncolp, delrp, delcp, topp, botmp, idomainp, ncpp=3, ncppl=1, xllp=0.0, yllp=0.0)

Bases: *object*

property child

Return a SimpleRegularGrid object for the child model

Returns

simple_regular_grid - simple grid object containing grid information for the child

Return type

SimpleRegularGrid

get_delr_delc()

get_exchange_data(angldegx=False, cdist=False)

Get the list of parent/child connections

<cellidm1> <cellidm2> <ihc> <cl1> <cl2> <hwva> <angledegx>

Returns

exglist - list of connections between parent and child

Return type

list

get_idomain()

Return the idomain array for the child model. This will normally be all ones unless the idomain array for the parent model is non-rectangular and irregularly shaped. Then, parts of the child model will have idomain zero cells.

Returns

idomain - idomain array for the child model

Return type

ndarray

get_lower_left()

Return the lower left corner of the child grid

Returns

(xll, yll) - location of lower left corner of the child grid

Return type

tuple

get_parent_connections(*kc, ic, jc*)

Return a list of parent cell indices that are connected to child cell *kc*, *ic*, *jc*.

get_parent_indices(*kc, ic, jc*)

Method returns the parent cell indices for this child. The returned indices are in zero-based indexing.

get_replicated_parent_array(*parent_array*)

Get a two-dimensional array the size of the child grid that has values replicated from the provided parent array.

Parameters

parent_array (*ndarray*) - A two-dimensional array that is the size of the parent model rows and columns.

Returns

child_array - A two-dimensional array that is the size of the child model rows and columns

Return type

ndarray

get_shape()

Return the shape of the child grid

Returns

(*nlay, nrow, ncol*) - shape of the child grid

Return type

tuple

get_top_botm()

property parent

Return a SimpleRegularGrid object for the parent model

Returns

simple_regular_grid - simple grid object containing grid information for the parent

Return type

SimpleRegularGrid

class SimpleRegularGrid(*nlay, nrow, ncol, delr, delc, top, botm, idomain, xorigin, yorigin*)

Bases: *object*

Simple object for representing regular MODFLOW grid information.

Parameters

- **nlay** (*int*) - number of layers
- **nrow** (*int*) - number of rows
- **ncol** (*int*) - number of columns
- **delr** (*ndarray*) - delr array
- **delc** (*ndarray*) - delc array
- **top** (*ndarray*) - top array (nrow, ncol)

- **botm** (*ndarray*) - botm array (nlay, nrow, ncol)
- **idomain** (*ndarray*) - idomain array (nlay, nrow, ncol)
- **xorigin** (*float*) - x location of grid lower left corner
- **yorigin** (*float*) - y location of grid lower left corner

get_gridprops_dis6()

property **modelgrid**

flopy.utils.mflistfile module

This is a set of classes for reading budget information out of MODFLOW-style listing files. Cumulative and incremental budgets are returned as numpy recarrays, which can then be easily plotted.

class ListBudget(*file_name*, *budgetkey*=None, *timeunit*='days')

Bases: *object*

MODFLOW family list file handling

Parameters

- **file_name** (*str*) - the list file name
- **budgetkey** (*str*) - the text string identifying the budget table. (default is None)
- **timeunit** (*str*) - the time unit to return in the recarray. (default is 'days')

Notes

The ListBudget class should not be instantiated directly. Access is through derived classes: MfListBudget (MODFLOW), SwtListBudget (SEAWAT) and SwrListBudget (MODFLOW with the SWR process)

Examples

```
>>> mf_list = MfListBudget("my_model.list")
>>> incremental, cumulative = mf_list.get_budget()
>>> df_inc, df_cumul = mf_list.get_dataframes(start_datetime="10-21-2015")
```

get_budget(*names*=None)

Get the recarrays with the incremental and cumulative water budget items in the list file.

Parameters

names (*str* or *list of strings*) - Selection of column names to return. If names is not None then totim, time_step, stress_period, and selection(s) will be returned. (default is None).

Returns

out - Numpy recarrays with the water budget items in list file. The recarray also includes totim, time_step, and stress_period. A

separate recarray is returned for the incremental and cumulative water budget entries.

Return type
recarrays

Examples

```
>>> mf_list = MfListBudget("my_model.list")
>>> budget = mf_list.get_budget()
```

get_cumulative(names=None)

Get a recarray with the cumulative water budget items in the list file.

Parameters

names (*str or list of strings*) - Selection of column names to return. If names is not None then totim, time_step, stress_period, and selection(s) will be returned. (default is None).

Returns

out - Numpy recarray with the water budget items in list file. The recarray also includes totim, time_step, and stress_period.

Return type
recarray

Examples

```
>>> mf_list = MfListBudget("my_model.list")
>>> cumulative = mf_list.get_cumulative()
```

get_data(kstpkiper=None, idx=None, totim=None, incremental=False)

Get water budget data from the list file for the specified conditions.

Parameters

- **idx** (*int*) - The zero-based record number. The first record is record 0. (default is None).
- **kstpkiper** (*tuple of ints*) - A tuple containing the time step and stress period (kstp, kper). These are zero-based kstp and kper values. (default is None).
- **totim** (*float*) - The simulation time. (default is None).
- **incremental** (*bool*) - Boolean flag used to determine if incremental or cumulative water budget data for the specified conditions will be returned. If incremental=True, incremental water budget data will be returned. If incremental=False, cumulative water budget data will be returned. (default is False).

Returns

data - Array has size (number of budget items, 3). Recarray names are 'index', 'value', 'name'.

Return type
numpy recarray

Notes

if both kstpkper and totim are None, will return the last entry

Examples

```
>>> import matplotlib.pyplot as plt
>>> import flopy
>>> mf_list = flopy.utils.MfListBudget("my_model.list")
>>> data = mf_list.get_data(kstpkper=(0,0))
>>> plt.bar(data['index'], data['value'])
>>> plt.xticks(data['index'], data['name'], rotation=45, size=6)
>>> plt.show()
```

get_dataframes(start_datetime='1-1-1970', diff=False)

Get pandas dataframes with the incremental and cumulative water budget items in the list file.

Parameters

start_datetime (*str*) - If start_datetime is passed as None, the rows are indexed on totim. Otherwise, a DatetimeIndex is set. (default is 1-1-1970).

Returns

out - Pandas dataframes with the incremental and cumulative water budget items in list file. A separate pandas dataframe is returned for the incremental and cumulative water budget entries.

Return type

pandas dataframes

Examples

```
>>> mf_list = MfListBudget("my_model.list")
>>> incrementaldf, cumulativddf = mf_list.get_dataframes()
```

get_incremental(names=None)

Get a recarray with the incremental water budget items in the list file.

Parameters

names (*str* or *list of strings*) - Selection of column names to return. If names is not None then totim, time_step, stress_period, and selection(s) will be returned. (default is None).

Returns

out - Numpy recarray with the water budget items in list file. The recarray also includes totim, time_step, and stress_period.

Return type

recarray

Examples

```
>>> mf_list = MfListBudget("my_model.list")
>>> incremental = mf_list.get_incremental()
```

`get_kstpkper()`

Get a list of unique stress periods and time steps in the list file water budgets.

Returns

out - List of unique kstp, kper combinations in list file. kstp and kper values are zero-based.

Return type

list of (kstp, kper) tuples

Examples

```
>>> mf_list = MfListBudget("my_model.list")
>>> kstpkper = mf_list.get_kstpkper()
```

`get_model_runtime(units='seconds')`

Get the elapsed runtime of the model from the list file.

Parameters

units (*str*) - Units in which to return the runtime. Acceptable values are 'seconds', 'minutes', 'hours' (default is 'seconds')

Returns

out - Floating point value with the runtime in requested units. Returns NaN if runtime not found in list file

Return type

float

Examples

```
>>> mf_list = MfListBudget("my_model.list")
>>> budget = mf_list.get_model_runtime(units='hours')
```

`get_record_names()`

Get a list of water budget record names in the file.

Returns

out - List of unique text names in the binary file.

Return type

list of strings

Examples

```
>>> mf_list = MfListBudget('my_model.list')
>>> names = mf_list.get_record_names()
```

get_reduced_pumping()

Get numpy recarray of reduced pumping data from a list file. Reduced pumping data must have been written to the list file during the model run. Works with MfListBudget and MfusgListBudget.

Returns

A numpy recarray with the reduced pumping data from the list file.

Return type

numpy recarray

Example

```
>>> objLST = MfListBudget("my_model.lst")
>>> raryReducedPpg = objLST.get_reduced_pumping()
>>> dfReducedPpg = pd.DataFrame.from_records(raryReducedPpg)
```

get_times()

Get a list of unique water budget times in the list file.

Returns

out - List contains unique water budget simulation times (totim) in list file.

Return type

list of floats

Examples

```
>>> mf_list = MfListBudget('my_model.list')
>>> times = mf_list.get_times()
```

get_tslens()

Get a list of unique water budget time step lengths in the list file.

Returns

out - List contains unique water budget simulation time step lengths (tslen) in list file.

Return type

list of floats

Examples

```
>>> mf_list = MfListBudget('my_model.list')
>>> ts_lengths = mf_list.get_tslens()
```

isvalid()

Get a boolean indicating if budget data are available in the file.

Returns

out - Boolean indicating if budget data are available in the file.

Return type

boolean

Examples

```
>>> mf_list = MfListBudget('my_model.list')
>>> valid = mf_list.isvalid()
```

set_budget_key()

```
class Mf6ListBudget(file_name, budgetkey=None, timeunit='days')
```

Bases: [ListBudget](#)

set_budget_key()

```
class MfListBudget(file_name, budgetkey=None, timeunit='days')
```

Bases: [ListBudget](#)

set_budget_key()

```
class MfusgListBudget(file_name, budgetkey=None, timeunit='days')
```

Bases: [ListBudget](#)

set_budget_key()

```
class SwrListBudget(file_name, budgetkey=None, timeunit='days')
```

Bases: [ListBudget](#)

set_budget_key()

```
class SwtListBudget(file_name, budgetkey=None, timeunit='days')
```

Bases: [ListBudget](#)

set_budget_key()

flopy.utils.mfreadnam module

mfreadnam module. Contains the NamData class. Note that the user can access the NamData class as `flopy.modflow.NamData`.

Additional information about the MODFLOW name file can be found at the [Online MODFLOW Guide](#).

class `NamData(pkgtype, name, handle, packages)`

Bases: `object`

MODFLOW Namefile Class.

Parameters

- **pkgtype** (*string*) - String identifying the type of MODFLOW package. See the `mfnam_packages` dictionary keys in the model object for a list of supported packages. This dictionary is also passed in as `packages`.
- **name** (*string*) - Filename of the package file identified in the name file
- **handle** (*file handle*) - File handle referring to the file identified by `name`
- **packages** (*dictionary*) - Dictionary of package objects as defined in the `mfnam_packages` attribute of `flopy.modflow.mf.Modflow`.

filehandle

File handle to the package file. Read from `handle`.

Type

file handle

filename

Filename of the package file identified in the name file. Read from `name`.

Type

string

filetype

String identifying the type of MODFLOW package. Read from `pkgtype`.

Type

string

package

Package type. Only assigned if `pkgtype` is found in the keys of `packages`

Type

string

Notes

Examples

`attrs_from_namefile_header(namefile)`

`get_entries_from_namefile(path: str | PathLike, ftype: str = None, unit: int = None, extension: str = None) → List[Tuple]`

Get entries from an MF6 namefile. Can select using FTYPE, UNIT, or file extension. This function only supports MF6 namefiles.

Parameters

- **path** (*str* or *PathLike*) - path to a MODFLOW-based model name file
- **ftype** (*str*) - package type
- **unit** (*int*) - file unit number
- **extension** (*str*) - file extension

Returns

entries - list of tuples containing FTYPE, UNIT, FNAME, STATUS for each namefile entry that meets a user-specified value.

Return type

list of tuples

`get_input_files(namefile)`

Return a list of all the input files in this model. :param namefile: path to a MODFLOW-based model name file :type namefile: str

Returns

filelist - list of MODFLOW-based model input files

Return type

list

`get_mf6_blockdata(f, blockstr)`

Return list with all non comments between start and end of block specified by blockstr. :param f: open file object :type f: file object :param blockstr: name of block to search :type blockstr: str

Returns

data - list of data in specified block

Return type

list

`get_mf6_files(mfnamefile)`

Return a list of all the MODFLOW 6 input and output files in this model. :param mfnamefile: path to the MODFLOW 6 simulation name file :type mfnamefile: str

Returns

- **filelist** (*list*) - list of MODFLOW 6 input files in a simulation
- **outplist** (*list*) - list of MODFLOW 6 output files in a simulation

`get_mf6_ftypes(namefile, ftypekeys)`

Return a list of FTYPES that are in the name file and in ftypekeys. :param namefile: path to a MODFLOW 6 name file :type namefile: str :param ftypekeys: list of desired FTYPES :type ftypekeys: list

Returns

ftypes - list of FTYPES that match ftypekeys in namefile

Return type

`list`

get_mf6_mshape(disfile)

Return the shape of the MODFLOW 6 model. :param disfile: path to a MODFLOW 6 discretization file :type disfile: str

Returns

mshape - tuple with the shape of the MODFLOW 6 model.

Return type

`tuple`

get_mf6_nper(tdisfile)

Return the number of stress periods in the MODFLOW 6 model. :param tdisfile: path to the TDIS file :type tdisfile: str

Returns

nper - number of stress periods in the simulation

Return type

`int`

get_sim_name(namefiles, rootpth=None)

Get simulation name. :param namefiles: path(s) to MODFLOW-based model name files :type namefiles: str or list of strings :param rootpth: optional root directory path (default is None) :type rootpth: str

Returns

simname - list of namefiles without the file extension

Return type

`list`

getfiletypeunit(nf, filetype)

Method to return unit number of a package from a NamData instance

Parameters

- **nf** (*NamData instance*) -
- **filetype** (*string, name of package seeking information for*) -

Returns

cunit

Return type

`int`, unit number corresponding to the package type

parsenamefile(namfilename, packages, verbose=True)

Returns dict from the nam file with NamData keyed by unit number

Parameters

- **namfilename** (*str or Path*) - Name of the MODFLOW namefile to parse.
- **packages** (*dict*) - Dictionary of package objects as defined in the `mfnam_packages` attribute of `flopy.modflow.mf.Modflow`.
- **verbose** (*bool*) - Print messages to screen. Default is True.

Returns

For each file listed in the name file, a `flopy.utils.mfreadnam.NamData` instance is stored in the returned dict keyed by unit number.

Return type

dict

Raises

- **FileNotFoundError** - If namfilename does not exist in the directory.
- **ValueError** - For lines that cannot be parsed.

flopy.utils.modpathfile module

Module to read MODPATH output files. The module contains two important classes that can be accessed by the user.

- EndpointFile (ascii endpoint file)
- PathlineFile (ascii pathline file)

class EndpointFile(filename, verbose=False)

Bases: `object`

EndpointFile Class.

Parameters

- **filename** (*string*) - Name of the endpoint file
- **verbose** (*bool*) - Write information to the screen. Default is False.

Examples

```
>>> import flopy
>>> endobj = flopy.utils.EndpointFile('model.mpend')
>>> e1 = endobj.get_data(partid=1)
```

get_alldata()

Get endpoint data from the endpoint file for all endpoints.

Returns

ra - A numpy recarray with the endpoint particle data

Return type

numpy record array

Notes

Examples

```
>>> import flopy
>>> endobj = flopy.utils.EndpointFile('model.mpend')
>>> e = endobj.get_alldata()
```

get_data(partid=0)

Get endpoint data from the endpoint file for a single particle.

Parameters

partid (*int*) - The zero-based particle id. The first record is record 0. (default is 0)

Returns

ra - A numpy recarray with the endpoint particle data for endpoint partid.

Return type

numpy record array

Notes

Examples

```
>>> import flopy
>>> endobj = flopy.utils.EndpointFile('model.mpend')
>>> e1 = endobj.get_data(partid=1)
```

get_destination_endpoint_data(dest_cells, source=False)

Get endpoint data that terminate in a set of destination cells.

Parameters

- **dest_cells** (*list* or *array of tuples*) - (k, i, j) of each destination cell for MODPATH versions less than MODPATH 7 or node number of each destination cell. (zero based)
- **source** (*bool*) - Boolean to specify if dest_cells applies to source or destination cells (default is False).

Returns

epdest - Slice of endpoint data array (e.g. EndpointFile.get_alldata) containing only endpoint data with final locations in (k,i,j) or (node) dest_cells.

Return type

np.recarray

Examples

```
>>> import flopy
>>> e = flopy.utils.EndpointFile('modpath.endpoint')
>>> e0 = e.get_destination_endpoint_data([(0, 0, 0),
...                                     (1, 0, 0)])
```

get_maxid()

Get the maximum endpoint particle id in the file endpoint file

Returns

out - Maximum endpoint particle id.

Return type

`int`

get_maxtime()

Get the maximum time in the endpoint file

Returns

out - Maximum endpoint time.

Return type

`float`

get_maxtraveltime()

Get the maximum travel time in the endpoint file

Returns

out - Maximum endpoint travel time.

Return type

`float`

```
kijnames = ['k0', 'i0', 'j0', 'node0', 'k', 'i', 'j', 'node', 'particleid',
'particlegroup', 'particleidloc', 'zone0', 'zone']
```

```
write_shapefile(endpoint_data=None, shpname='endpoints.shp', direction='ending',
                mg=None, epsg=None, **kwargs)
```

Write particle starting / ending locations to shapefile.

endpoint_data

[`np.recarray`] Record array of same form as that returned by `EndpointFile.get_alldata`. (if none, `EndpointFile.get_alldata()` is exported).

shpname

[`str`] File path for shapefile

direction

[`str`] String defining if starting or ending particle locations should be considered. (default is 'ending')

mg

[`flopy.discretization.grid` instance] Used to scale and rotate Global x,y,z values in MODPATH Endpoint file.

epsg

[`int`] EPSG code for writing projection (.prj) file. If this is not supplied, the proj4 string or epsg code associated with mg will be used.

kwargs : keyword arguments to flopy.export.shapefile_utils.reccarray2shp

```
class PathlineFile(filename: str | PathLike, verbose=False)
```

Bases: `_ModpathSeries`

PathlineFile Class.

Parameters

- **filename** (*str* or *PathLike*) - Path of the pathline file
- **verbose** (*bool*) - Write information to the screen. Default is False.

Examples

```
>>> import flopy
>>> pthobj = flopy.utils.PathlineFile('model.mppth')
>>> p1 = pthobj.get_data(partid=1)
```

```
get_alldata(totim=None, ge=True)
```

get pathline data from the pathline file for all pathlines and all times.

Parameters

- **totim** (*float*) - The simulation time. All pathline points for particle *partid* that are greater than or equal to (*ge=True*) or less than or equal to (*ge=False*) *totim* will be returned. Default is None
- **ge** (*bool*) - Boolean that determines if pathline times greater than or equal to or less than or equal to *totim* is used to create a subset of pathlines. Default is True.

Returns

plist - A list of numpy recarrays with the x, y, z, time, k, and particleid for all pathlines.

Return type

a list of numpy record array

Examples

```
>>> import flopy.utils.modpathfile as mpf
>>> pthobj = flopy.utils.PathlineFile('model.mppth')
>>> p = pthobj.get_alldata()
```

```
get_data(partid=0, totim=None, ge=True)
```

get pathline data from the pathline file for a single pathline.

Parameters

- **partid** (*int*) - The zero-based particle id. The first record is record 0.
- **totim** (*float*) - The simulation time. All pathline points for particle *partid* that are greater than or equal to (*ge=True*) or less than or equal to (*ge=False*) *totim* will be returned. Default is None

- **ge** (*bool*) - Boolean that determines if pathline times greater than or equal to or less than or equal to totim is used to create a subset of pathlines. Default is True.

Returns

ra - A numpy recarray with the x, y, z, time, k, and particleid for pathline partid.

Return type

numpy record array

Notes**Examples**

```
>>> import flopy.utils.modpathfile as mpf
>>> pthobj = flopy.utils.PathlineFile('model.mppth')
>>> p1 = pthobj.get_data(partid=1)
```

get_destination_pathline_data(dest_cells, to_reccarray=False)

Get pathline data that pass through a set of destination cells.

Parameters

- **dest_cells** (*list* or *array of tuples*) - (k, i, j) of each destination cell for MODPATH versions less than MODPATH 7 or node number of each destination cell. (zero based)
- **to_reccarray** (*bool*) - Boolean that controls returned pthldest. If to_reccarray is True, a single recarray with all of the pathlines that intersect dest_cells are returned. If to_reccarray is False, a list of recarrays (the same form as returned by get_alldata method) that intersect dest_cells are returned (default is False).

Returns

pthldest - Slice of pathline data array (e.g. PathlineFile._data) containing only pathlines that pass through (k,i,j) or (node) dest_cells.

Return type

np.recarray

Examples

```
>>> import flopy
>>> p = flopy.utils.PathlineFile('modpath.pathline')
>>> p0 = p.get_destination_pathline_data([(0, 0, 0),
...                                     (1, 0, 0)])
```

get_maxid()

Get the maximum pathline number in the file pathline file

Returns

out - Maximum pathline number.

Return type

`int`

get_maxtime()

Get the maximum time in pathline file

Returns

out - Maximum pathline time.

Return type

`float`

`kijnames = ['k', 'i', 'j', 'node', 'particleid', 'particlegroup',
'linesegmentindex', 'particleidloc', 'sequencenumber']`

`write_shapefile(pathline_data=None, one_per_particle=True, direction='ending',
shpname='pathlines.shp', mg=None, epsg=None, **kwargs)`

Write pathlines to a shapefile

pathline_data

[`np.recarray`] Record array of same form as that returned by `PathlineFile.get_alldata()`. (if none, `PathlineFile.get_alldata()` is exported).

one_per_particle

[`boolean` (default `True`)] `True` writes a single `LineString` with a single set of attribute data for each particle. `False` writes a record/geometry for each pathline segment (each row in the `PathLine` file). This option can be used to visualize attribute information (time, model layer, etc.) across a pathline in a GIS.

direction

[`str`] String defining if starting or ending particle locations should be included in shapefile attribute information. Only used if `one_per_particle=False`. (default is `'ending'`)

shpname

[`str`] File path for shapefile

mg

[`flopy.discretization.grid` instance] Used to scale and rotate Global `x,y,z` values in MODPATH Pathline file.

epsg

[`int`] EPSG code for writing projection (.prj) file. If this is not supplied, the proj4 string or epsg code associated with `mg` will be used.

kwargs : keyword arguments to `flopy.export.shapefile_utils.recarray2shp`

class TimeseriesFile(filename, verbose=False)

Bases: `_ModpathSeries`

TimeseriesFile Class.

Parameters

- **filename** (*string*) - Name of the timeseries file
- **verbose** (*bool*) - Write information to the screen. Default is `False`.

Examples

```
>>> import flopy
>>> tsobj = flopy.utils.TimeseriesFile('model.timeseries')
>>> ts1 = tsobj.get_data(partid=1)
```

get_alldata(*totim=None, ge=True*)

get timeseries data from the timeseries file for all timeseries and all times.

Parameters

- **totim** (*float*) - The simulation time. All timeseries points for timeseries partid that are greater than or equal to (*ge=True*) or less than or equal to (*ge=False*) *totim* will be returned. Default is *None*
- **ge** (*bool*) - Boolean that determines if timeseries times greater than or equal to or less than or equal to *totim* is used to create a subset of timeseries. Default is *True*.

Returns

tlist - A list of numpy recarrays with the x, y, z, time, k, and particleid for all timeseries.

Return type

a list of numpy record array

Examples

```
>>> import flopy
>>> tsobj = flopy.utils.TimeseriesFile('model.timeseries')
>>> ts = tsobj.get_alldata()
```

get_data(*partid=0, totim=None, ge=True*)

get timeseries data from the timeseries file for a single timeseries particleid.

Parameters

- **partid** (*int*) - The zero-based particle id. The first record is record 0.
- **totim** (*float*) - The simulation time. All timeseries points for particle partid that are greater than or equal to (*ge=True*) or less than or equal to (*ge=False*) *totim* will be returned. Default is *None*
- **ge** (*bool*) - Boolean that determines if timeseries times greater than or equal to or less than or equal to *totim* is used to create a subset of timeseries. Default is *True*.

Returns

ra - A numpy recarray with the x, y, z, time, k, and particleid for timeseries partid.

Return type

numpy record array

Notes

Examples

```
>>> import flopy
>>> tsobj = flopy.utils.TimeseriesFile('model.timeseries')
>>> ts1 = tsobj.get_data(partid=1)
```

get_destination_timeseries_data(dest_cells)

Get timeseries data that pass through a set of destination cells.

Parameters

dest_cells (*list* or *array of tuples*) - (k, i, j) of each destination cell for MODPATH versions less than MODPATH 7 or node number of each destination cell. (zero based)

Returns

tsdest - Slice of timeseries data array (e.g. `TmeseriesFile._data`) containing only timeseries that pass through (k,i,j) or (node) `dest_cells`.

Return type

`np.recarray`

Examples

```
>>> import flopy
>>> ts = flopy.utils.TimeseriesFile('modpath.timeseries')
>>> tss = ts.get_destination_timeseries_data([(0, 0, 0),
...                                           (1, 0, 0)])
```

get_maxid()

Get the maximum timeseries number in the file timeseries file

Returns

out - Maximum pathline number.

Return type

`int`

get_maxtime()

Get the maximum time in timeseries file

Returns

out - Maximum pathline time.

Return type

`float`

`kijnames` = ['k', 'i', 'j', 'node', 'particleid', 'particlegroup', 'particleidloc', 'timestep', 'timestepindex', 'timepointindex']

`write_shapefile(timeseries_data=None, one_per_particle=True, direction='ending', shpname='pathlines.shp', mg=None, epsg=None, **kwargs)`

Write pathlines to a shapefile

timeseries_data

[np.recarray] Record array of same form as that returned by Timeseries.get_alldata(). (if none, Timeseries.get_alldata() is exported).

one_per_particle

[boolean (default True)] True writes a single LineString with a single set of attribute data for each particle. False writes a record/geometry for each pathline segment (each row in the Timeseries file). This option can be used to visualize attribute information (time, model layer, etc.) across a pathline in a GIS.

direction

[str] String defining if starting or ending particle locations should be included in shapefile attribute information. Only used if one_per_particle=False. (default is 'ending')

shpname

[str] File path for shapefile

mg

[flopy.discretization.grid instance] Used to scale and rotate Global x,y,z values in MODPATH Timeseries file.

epsg

[int] EPSG code for writing projection (.prj) file. If this is not supplied, the proj4 string or epsg code associated with mg will be used.

kwargs : keyword arguments to flopy.export.shapefile_utils.recarray2shp

flopy.utils.mtlistfile module

This is a class for reading the mass budget from a (multi-component) mt3d(usgs) run. Also includes support for SFT budget.

class MtListBudget(file_name)

Bases: `object`

MT3D mass budget reader

Parameters

file_name (*str*) - the list file name

Examples

```
>>> mt_list = MtListBudget("my_mt3d.list")
>>> gw_df, sw_df = mt_list.parse(start_datetime="10-21-2015")
```

parse(forgive=True, diff=True, start_datetime=None, time_unit='d')

Main entry point for parsing the list file.

Parameters

- **forgive** (*bool*) - flag to raise exceptions when fail-to-read occurs. Default is True
- **diff** (*bool*) - flag to return dataframes with 'in minus out' columns. Default is True

- **start_datetime** (*str*) - str that can be parsed by `pandas.to_datetime`. Example: '1-1-1970'. Default is None.
- **time_unit** (*str*) - str to pass to `pandas.to_timedelta`. Default is 'd' (days)

Returns

df_gw, df_sw - a dataframe for the groundwater mass and (optionally) surface-water mass budget. If the SFT process is not used, **df_sw** is None.

Return type

`pandas.DataFrame`

flopy.utils.observationfile module

class CsvFile(*csvfile*, *delimiter*=';', *deletechars*="", *replace_space*="")

Bases: `object`

Class for reading csv based output files

Parameters

- **csvfile** (*str*) - csv file name
- **delimiter** (*str*) - optional delimiter for the csv or formatted text file, defaults to ";"
- **deletechars** (*str*) - optional string containing characters that should be deleted from the column names, defaults to ""
- **replace_space** (*str*) - optional string containing the character that will be used to replace the space with in any column names, defaults to ""

property nobs

Method to get the number of observations

Return type

`int`

property obsnames

Method to get the observation names

Return type

`list`

static read_csv(*fobj*, *dtype*, *delimiter*=';', *deletechars*="", *replace_space*="")

Parameters

- **fobj** (*file object*) - open text file object to read
- **dtype** (*np.dtype*) -
- **delimiter** (*str*) - optional delimiter for the csv or formatted text file, defaults to ";"
- **deletechars** (*str*) - optional string containing characters that should be deleted from the column names, defaults to ""

- **replace_space** (*str*) - optional string containing the character that will be used to replace the space with in any column names, defaults to ""

Return type

np.recarray

class HydmodObs(*filename*, *verbose=False*, *hydlbl_len=20*)

Bases: [ObsFiles](#)

HydmodObs Class - used to read binary MODFLOW HYDMOD package output

Parameters

- **filename** (*str*) - Name of the hydmod output file
- **verbose** (*boolean*) - If true, print additional information to to the screen during the extraction. (default is False)
- **hydlbl_len** (*int*) - Length of hydmod labels. (default is 20)

Return type

None

class Mf6Obs(*filename*, *verbose=False*, *isBinary='auto'*)

Bases: [ObsFiles](#)

Mf6Obs Class - used to read ascii and binary MODFLOW6 observation output

Parameters

- **filename** (*str*) - Name of the hydmod output file
- **verbose** (*boolean*) - If true, print additional information to to the screen during the extraction. (default is False)
- **isBinary** (*str*, *bool*) - default is "auto", code will attempt to automatically check if file is binary. User can change this to True or False if the auto check fails to work

Return type

None

class ObsFiles

Bases: [FlopyBinaryData](#)

get_data(*idx=None*, *obsname=None*, *totim=None*)

Get data from the observation file.

Parameters

- **idx** (*int*) - The zero-based record number. The first record is record 0. If idx is None and totim are None, data for all simulation times are returned. (default is None)
- **obsname** (*string*) - The name of the observation to return. If obsname is None, all observation data are returned. (default is None)
- **totim** (*float*) - The simulation time to return. If idx is None and totim are None, data for all simulation times are returned. (default is None)

Returns

data - Array has size (ntimes, nitems). totim is always returned. nitems is 2 if idx or obsname is not None or nobst+1.

Return type

numpy record array

Notes

If both idx and obsname are None, will return all of the observation data.

Examples

```
>>> hyd = HydmodObs("my_model.hyd")
>>> ts = hyd.get_data()
```

```
get_dataframe(start_datetime='1-1-1970', idx=None, obsname=None, totim=None,
              timeunit='D')
```

Get pandas dataframe with the incremental and cumulative water budget items in the hydmod file.

Parameters

- **start_datetime** (*str*) - If start_datetime is passed as None, the rows are indexed on totim. Otherwise, a DatetimeIndex is set. (default is 1-1-1970).
- **idx** (*int*) - The zero-based record number. The first record is record 0. If idx is None and totim are None, a dataframe with all simulation times is returned. (default is None)
- **obsname** (*string*) - The name of the observation to return. If obsname is None, all observation data are returned. (default is None)
- **totim** (*float*) - The simulation time to return. If idx is None and totim are None, a dataframe with all simulation times is returned. (default is None)
- **timeunit** (*string*) - time unit of the simulation time. Valid values are 'S'econds, 'M'inutes, 'H'ours, 'D'ays, 'Y'ears. (default is 'D').

Returns

out - Pandas dataframe of selected data.

Return type

pandas dataframe

Notes

If both `idx` and `obsname` are `None`, will return all of the observation data as a dataframe.

Examples

```
>>> hyd = HydmodObs("my_model.hyd")
>>> df = hyd.get_dataframes()
```

`get_nobs()`

Get the number of observations in the file

Returns

out - A tuple with the number of records and number of flow items in the file. The number of flow items is non-zero only if `swrtype='flow'`.

Return type

tuple of int

`get_ntimes()`

Get the number of times in the file

Returns

out - The number of simulation times (`totim`) in binary file.

Return type

int

`get_obsnames()`

Get a list of observation names in the file

Returns

out - List of observation names in the binary file. `totim` is not included in the list of observation names.

Return type

list of strings

`get_times()`

Get a list of unique times in the file

Returns

out - List contains unique simulation times (`totim`) in binary file.

Return type

list of floats

```
class SwrObs(filename, precision='double', verbose=False)
```

Bases: `ObsFiles`

Read binary SWR observations output from MODFLOW SWR Process observation files

Parameters

- **filename** (*string*) - Name of the cell budget file
- **precision** (*string*) - 'single' or 'double'. Default is 'double'.

- **verbose** (*bool*) - Write information to the screen. Default is False.

Notes

Examples

```
>>> import flopy
>>> so = flopy.utils.SwrObs('mymodel.swr.obs')
```

get_reduced_pumping(*f*, *structured=True*)

Method to read reduced pumping from a list file or an external reduced pumping observation file

Parameters

- **f** (*str*) - file name
- **structured** (*bool*) - boolean flag to indicate if model is Structured or USG model. Defaults to True (structured grid).

Returns

np.recarray

Return type

recarray of reduced pumping records.

get_selection(*data*, *names*)

Parameters

- **data** (*numpy recarray*) - recarray of data to make a selection from
- **names** (*string or list of strings*) - column names to return

Returns

out - recarray with selection

Return type

numpy recarray

flopy.utils.optionblock module

class OptionBlock(*options_line*, *package*, *block=True*)

Bases: **object**

Parent class to for option blocks within Modflow-nwt models. This class contains base information and routines that can be shared throughout all option block classes.

Parameters

- **options_line** (*str*) - single line based options string
- **package** (*flopy.pakbase.Package instance*) - valid packages include ModflowWel, ModflowSfr2, ModflowUzf1
- **block** (*bool*) - flag to write as single line or block type

dtype = 'dtype'

classmethod `load_options(options, package)`

Loader for the options class. Reads in an options block and uses context from option util dictionaries to check the validity of the data

Parameters

- **options** (*str* or *file*) - string path to a file or file object
- **package** (*flopy.package* type) - valid packages include `flopy.modflow.ModflowWel`, `flopy.modflow.ModflowUzf1`, `flopy.modflow.ModflowSfr2`

Return type

`OptionBlock` object

`n_nested = 'nvars'`

`nested = 'nested'`

`optional = 'optional'`

`simple_flag = {'dtype': <class 'numpy.bool_'>, 'nested': False, 'optional': False}`

`simple_float = {'dtype': <class 'float'>, 'nested': False, 'optional': False}`

`simple_int = {'dtype': <class 'int'>, 'nested': False, 'optional': False}`

`simple_str = {'dtype': <class 'str'>, 'nested': False, 'optional': False}`

`simple_tabfile = {'dtype': <class 'numpy.bool_'>, 'nested': True, 'nvars': 2, 'vars': {'maxval': {'dtype': <class 'int'>, 'nested': False, 'optional': False}, 'numtab': {'dtype': <class 'int'>, 'nested': False, 'optional': False}}}`

property `single_line_options`

Method to get the single line representation of the Options Block

Returns

`t`

Return type

(*str*) single line representation of Options

update_from_package(*pak*)

Updater method to check the package and update OptionBlock attribute values based on package values.

Parameters

pak (*flopy.package*) - valid packages include `ModflowWel`, `ModflowSfr2`, and `ModflowUzf1` instances

`vars = 'vars'`

write_options(*f*)

Method to write the options block or options line to an open file object.

Parameters

`f`

[*file*, *str*] open file object, or path to file

class OptionUtilBases: `object`**static isfloat(s)**

Simple method to check that a string is a valid floating point variable

Parameters**s** (`str`) -**Return type**`bool`**static isint(s)**

Simple data check method to check that a string is a valid integer

Parameters**s** (`str`) -**Return type**`bool`**static isvalid(dtype, val)**

Check to see if a dtype is valid before setting as an attribute

Parameters

- **dtype** (`type`) - int, float, str, bool, etc...
- **val** (`string`) -

Return type`bool`**flopy.utils.parse_version module****exception InvalidVersion**Bases: `ValueError`

An invalid version was found, users should refer to PEP 440.

class LegacyVersion(version: `str`)Bases: `_BaseVersion`**property base_version:** `str`**property dev:** `None`**property epoch:** `int`**property is_devrelease:** `bool`**property is_postrelease:** `bool`**property is_prerelease:** `bool`**property local:** `None`**property post:** `None`**property pre:** `None`

```
    property public: str
    property release: None

class Version(version: str)
    Bases: _BaseVersion
    property base_version: str
    property dev: int | None
    property epoch: int
    property is_devrelease: bool
    property is_postrelease: bool
    property is_prerelease: bool
    property local: str | None
    property major: int
    property micro: int
    property minor: int
    property post: int | None
    property pre: tuple[str, int] | None
    property public: str
    property release: tuple[int, ...]

parse(version: str) → LegacyVersion | Version
    Parse the given version string and return either a Version object or a LegacyVersion
    object depending on if the given version is a valid PEP 440 version or a legacy
    version.
```

flopy.utils.postprocessing module

```
get_extended_budget(cbcfile, precision='single', idx=None, kstpkiper=None, totim=None,
                    boundary_ifaces=None, hdsfile=None, model=None)
```

Get the flow rate across cell faces including potential stresses applied along boundaries at a given time. Only implemented for “classical” MODFLOW versions where the budget is recorded as FLOW RIGHT FACE, FLOW FRONT FACE and FLOW LOWER FACE arrays.

Parameters

- **cbcfile** (*str*) - Cell by cell file produced by Modflow.
- **precision** (*str*) - Binary file precision, default is ‘single’.
- **idx** (*int* or *list*) - The zero-based record number.
- **kstpkiper** (*tuple of ints*) - A tuple containing the time step and stress period (kstp, kper). The kstp and kper values are zero based.

- **totim** (*float*) - The simulation time.
- **boundary_ifaces** (*dictionary {str: int or list}*) - A dictionary defining how to treat stress flows at boundary cells. The keys are budget terms corresponding to stress packages (same term as in the overall volumetric budget printed in the listing file). The values are either a single iface number to be applied to all cells for the stress package, or a list of lists describing individual boundary cells in the same way as in the package input plus the iface number appended. The iface number indicates the face to which the stress flow is assigned, following the MODPATH convention (see MODPATH user guide). Example: `boundary_ifaces = { 'RECHARGE': 6, 'RIVER LEAKAGE': 6, 'CONSTANT HEAD': [[lay, row, col, iface], ...], 'WELLS': [[lay, row, col, flux, iface], ...], 'HEAD DEP BOUNDS': [[lay, row, col, head, cond, iface], ...]}`. Note: stresses that are not informed in `boundary_ifaces` are implicitly treated as internally-distributed sinks/sources.
- **hdsfile** (*str*) - Head file produced by MODFLOW (only required if `boundary_ifaces` is used).
- **model** (*flopy.modflow.Modflow object*) - Modflow model instance (only required if `boundary_ifaces` is used).

Returns

(**Qx_ext**, **Qy_ext**, **Qz_ext**) - Flow rates across cell faces. `Qx_ext` is a ndarray of size (nlay, nrow, ncol + 1). `Qy_ext` is a ndarray of size (nlay, nrow + 1, ncol). The sign is such that the y axis is considered to increase in the north direction. `Qz_ext` is a ndarray of size (nlay + 1, nrow, ncol). The sign is such that the z axis is considered to increase in the upward direction.

Return type

tuple

get_gradients(*heads, m, nodata, per_idx=None*)

Calculates the hydraulic gradients from the heads array for each stress period.

Parameters

- **heads** (3 or 4-D *np.ndarray*) - Heads array.
- **m** (*flopy.modflow.Modflow object*) - Must have a `flopy.modflow.ModflowDis` object attached.
- **nodata** (*real*) - HDRY value indicating dry cells.
- **per_idx** (*int or sequence of ints*) - stress periods to return. If `None`, returns all stress periods (default).

Returns

grad - Array of hydraulic gradients

Return type

3 or 4-D *np.ndarray*

get_specific_discharge(*vectors, model, head=None, position='centers'*)

Get the discharge vector at cell centers at a given time. For “classical” MODFLOW versions, we calculate it from the flow rate across cell faces. For MODFLOW 6, we directly take it from MODFLOW output (this requires setting the option “`save_specific_discharge`” in the NPF package).

Parameters

- **vectors** (*tuple*, *np.recarray*) - either a tuple of (flow right face, flow front face, flow lower face) numpy arrays from a MODFLOW-2005 compatible Cell Budget File or a specific discharge recarray from a MODFLOW 6 Cell Budget File
- **model** (*object*) - flopy model object
- **head** (*np.ndarray*) - numpy array of head values for a specific model position : str
Position at which the specific discharge will be calculated. Possible values are “centers” (default), “faces” and “vertices”.

Returns

(**qx**, **qy**, **qz**) - Discharge vector. qx, qy, qz are ndarrays of size (nlay, nrow, ncol) for a structured grid or size (nlay, ncpl) for an unstructured grid. The sign of qy is such that the y axis is considered to increase in the north direction. The sign of qz is such that the z axis is considered to increase in the upward direction. Note: if a head array is provided, inactive and dry cells are set to NaN.

Return type

tuple

get_transmissivities(heads, m, r=None, c=None, x=None, y=None, sctop=None, scbot=None, nodata=-999)

Computes transmissivity in each model layer at specified locations and open intervals. A saturated thickness is determined for each row, column or x, y location supplied, based on the open interval (sctop, scbot), if supplied, otherwise the layer tops and bottoms and the water table are used.

Parameters

- **heads** (2D array OR 3D array) - numpy array of shape nlay by n locations (2D) OR complete heads array of the model for one time (3D)
- **m** (*flopy.modflow.Modflow object*) - Must have dis and lpf or upw packages.
- **r** (1D array-like of ints, of length n locations) - row indices (optional; alternately specify x, y)
- **c** (1D array-like of ints, of length n locations) - column indices (optional; alternately specify x, y)
- **x** (1D array-like of floats, of length n locations) - x locations in real world coordinates (optional)
- **y** (1D array-like of floats, of length n locations) - y locations in real world coordinates (optional)
- **sctop** (1D array-like of floats, of length n locations) - open interval tops (optional; default is model top)
- **scbot** (1D array-like of floats, of length n locations) - open interval bottoms (optional; default is model bottom)

- **nodata** (*numeric*) - optional; locations where heads=nodata will be assigned T=0

Returns

T - Transmissivities in each layer at each location

Return type

2D array of same shape as heads (nlay x n locations)

get_water_table(heads, hdry=-1e+30, hnoflo=1e+30, masked_values=None)

Get a 2D array representing the water table elevation for each stress period in heads array.

Parameters

- **heads** (3 or 4-D *np.ndarray*) - Heads array.
- **hdry** (*real*) - The head that is assigned to cells that are converted to dry during a simulation. By default, -1e30.
- **hnoflo** (*real*) - The value of head assigned to all inactive (no flow) cells throughout the simulation, including vertical pass-through cells in MODFLOW 6. By default, 1e30.
- **masked_values** (*list*) - List of any values (in addition to hdry and hnoflo) that should be masked in the water table calculation.

Returns

wt - for each stress period.

Return type

2 or 3-D *np.ndarray* of water table elevations

flopy.utils.rasters module

class Raster(array, bands, crs, transform, nodataval, driver='GTiff', rio_ds=None)

Bases: *object*

The Raster object is used for cropping, sampling raster values, and re-sampling raster values to grids, and provides methods to plot rasters and histograms of raster digital numbers for visualization and analysis purposes.

Parameters

- **array** (*np.ndarray*) - a three dimensional array of raster values with dimensions defined by (raster band, nrow, ncol)
- **bands** (*tuple*) - a tuple of raster bands
- **crs** (*int*, *string*, *rasterio.crs.CRS object*) - either a epsg code, a proj4 string, or a CRS object
- **transform** (*affine.Affine object*) - affine object, which is used to define geometry
- **nodataval** (*float*) - raster no data value
- **rio_ds** (*DatasetReader object*) - rasterIO dataset Reader object

Notes

Examples

```
>>> from flopy.utils import Raster
>>>
>>> rio = Raster.load("myraster.tif")
```

FLOAT32 = (<class 'float'>, <class 'numpy.float32'>, <class 'numpy.float64'>)

FLOAT64 = (<class 'numpy.float64'>,,)

INT16 = (<class 'numpy.int16'>, <class 'numpy.uint16'>)

INT32 = (<class 'int'>, <class 'numpy.int32'>, <class 'numpy.uint32'>, <class 'numpy.uint64'>, <class 'numpy.uint32'>, <class 'numpy.uint32'>)

INT64 = (<class 'numpy.int64'>, <class 'numpy.uint64'>)

INT8 = (<class 'numpy.int8'>, <class 'numpy.uint8'>)

property bands

Returns a tuple of raster bands

property bounds

Returns a tuple of xmin, xmax, ymin, ymax boundaries

crop(*polygon*, *invert=False*)

Method to crop a new raster object from the current raster object

Parameters

- **polygons** (*list*, *geojson*, *shapely.geometry*, *shapefile.Shape*) - crop method accepts any of these geometries:
a list of (x, y) points, ex. [(x1, y1), ...] geojson Polygon object shapely Polygon object shapefile Polygon shape flopy Polygon shape
- **invert** (*bool*) - Default value is False. If invert is True then the area inside the shapes will be masked out

get_array(*band*, *masked=True*)

Method to get a numpy array corresponding to the provided raster band. Nodata vals are set to np.NaN

Parameters

- **band** (*int*) - band number from the raster
- **masked** (*bool*) - determines if nodatavals will be returned as np.nan to the user

Return type

np.ndarray

histogram(*ax=None*, ***kwargs*)

Method to plot a histogram of digital numbers

Parameters

- **ax** (*matplotlib.pyplot.axes*) - optional matplotlib axes for plotting
- ****kwargs** - matplotlib keyword arguments see matplotlib documentation for valid arguments for histogram

Returns

ax

Return type

matplotlib.pyplot.axes

static load(*raster*: *str* | *PathLike*)

Static method to load a raster file into the raster object

Parameters

raster (*str* or *PathLike*) - The path to the raster file

Return type

A Raster object

property nodatavals

Returns a Tuple of values used to define no data

plot(*ax=None*, *contour=False*, ****kwargs**)

Method to plot raster layers or contours.

Parameters

- **ax** (*matplotlib.pyplot.axes*) - optional matplotlib axes for plotting
- **contour** (*bool*) - flag to indicate creation of contour plot
- ****kwargs** - matplotlib keyword arguments see matplotlib documentation for valid arguments for plot and contour.

Returns

ax

Return type

matplotlib.pyplot.axes

resample_to_grid(*modelgrid*, *band*, *method='nearest'*, *multithread=False*, *thread_pool=2*, *extrapolate_edges=False*)

Method to resample the raster data to a user supplied grid of x, y coordinates.

x, y coordinate arrays should correspond to grid vertices

Parameters

- **modelgrid** (*flopy.Grid* object) - model grid to sample data from
- **band** (*int*) - raster band to re-sample
- **method** (*str*) - resampling methods
linear for bi-linear interpolation
nearest for nearest neighbor
cubic for bi-cubic interpolation
mean for mean sampling

median for median sampling
min for minimum sampling
max for maximum sampling
'mode' for majority sampling

- **multithread** (*bool*) - DEPRECATED boolean flag indicating if multithreading should be used with the mean and median sampling methods
- **thread_pool** (*int*) - DEPRECATED number of threads to use for mean and median sampling
- **extrapolate_edges** (*bool*) - boolean flag indicating if areas without data should be filled using the nearest interpolation method. This option has no effect when using the nearest interpolation method.

Return type

`np.array`

sample_point(*point, band=1)

Method to get nearest raster value at a user provided point

Parameters

- ***point** (*point geometry representation*) - accepted data types:
x, y values : ex. `sample_point(1, 3, band=1)` tuple of x, y:
ex `sample_point((1, 3), band=1)` `shapely.geometry.Point`
`geojson.Point` `flopy.geometry.Point`
- **band** (*int*) - raster band to re-sample

Returns

value

Return type

`float`

sample_polygon(polygon, band, invert=False, **kwargs)

Method to get an unordered list of raster values that are located within a arbitrary polygon

Parameters

- **polygon** (*list, geojson, shapely.geometry, shapefile.Shape*) - `sample_polygon` method accepts any of these geometries:
a list of (x, y) points, ex. `[(x1, y1), ...]` `geojson Polygon` object `shapely Polygon` object `shapefile Polygon` shape `flopy Polygon` shape
- **band** (*int*) - raster band to re-sample
- **invert** (*bool*) - Default value is False. If invert is True then the area inside the shapes will be masked out

Return type

`np.ndarray` of unordered raster values

write(name)

Method to write raster data to a .tif file

Parameters

name (*str*) - output raster .tif file name

property xcenters

Returns a np.ndarray of raster x cell centers

property ycenters

Returns a np.ndarray of raster y cell centers

flopy.utils.reccarray_utils module

create_empty_reccarray(length, dtype, default_value=0)

Create a empty reccarray with a defined default value for floats.

Parameters

- **length** (*int*) - Shape of the empty reccarray.
- **dtype** (*np.dtype*) - dtype of the empty reccarray.
- **default_value** (*float*) - default value to use for floats in reccarray.

Returns

r - Recarray of type dtype with shape length.

Return type

np.reccarray

Examples

```
>>> import numpy as np
>>> import flopy
>>> dtype = np.dtype([('x', np.float32), ('y', np.float32)])
>>> ra = flopy.utils.create_empty_reccarray(10, dtype)
```

ra_slice(ra, cols)

Create a slice of a reccarray

Parameters

- **ra** (*np.reccarray*) - reccarray to extract a limited number of columns from.
- **cols** (*list of str*) - List of key names to extract from ra.

Returns

ra_slice - Slice of ra

Return type

np.reccarray

Examples

```
>>> import flopy
>>> raslice = flopy.utils.ra_slice(ra, ['x', 'y'])
```

recarray(array, dtype)

Convert a list of lists or tuples to a recarray.

Parameters

- **array** (*list of lists*) - list of lists containing data to convert to a recarray. The number of entries in each list in the list must be the same.
- **dtype** (*np.dtype*) - dtype of the array data

Returns

r - Recarray of type dtype with shape equal to the length of array.

Return type

np.recarray

Examples

```
>>> import numpy as np
>>> import flopy
>>> dtype = np.dtype([('x', np.float32), ('y', np.float32)])
>>> arr = [(1., 2.), (10., 20.), (100., 200.)]
>>> ra = flopy.utils.recarray(arr, dtype)
```

flopy.utils.reference module

Module spatial referencing for flopy model objects

class TemporalReference(itmuni=4, start_datetime=None)

Bases: `object`

For now, just a container to hold start time and time units files outside of DIS package.

defaults = {'itmuni': 4, 'start_datetime': '01-01-1970'}

itmuni_text = {0: 'undefined', 1: 'seconds', 2: 'minutes', 3: 'hours', 4: 'days', 5: 'years'}

itmuni_values = {'days': 4, 'hours': 3, 'minutes': 2, 'seconds': 1, 'undefined': 0, 'years': 5}

property model_time_units

flopy.utils.sfroutputfile module

class SfrFile(filename, geometries=None, verbose=False)

Bases: `object`

Read SFR package results from text file (ISTCB2 > 0)

Parameters

- **filename** (*str*) - Name of the sfr output file
- **geometries** (*any*) - Ignored
- **verbose** (*any*) - Ignored

Notes

Indexing starts at one for: layer, row, column, segment, reach. Indexing starts at zero for: i, j, k, and kstpkper.

Examples

```
>>> import flopy
>>> sfq = flopy.utils.SfrFile('mymodel.sfq')
```

property **df**

dtypes = {'column': <class 'int'>, 'layer': <class 'int'>, 'reach': <class 'int'>, 'row': <class 'int'>, 'segment': <class 'int'>}

get_dataframe()

Read the whole text file into a pandas dataframe.

Returns

df - SFR output as a pandas dataframe

Return type

pandas dataframe

static get_nstrm(df)

Get the number of SFR cells from the results dataframe.

Returns

nrch - Number of SFR cells

Return type

`int`

get_results(segment, reach)

Get results for a single reach or sequence of segments and reaches.

Parameters

- **segment** (*int* or *sequence of ints*) - Segment number for each location.
- **reach** (*int* or *sequence of ints*) - Reach number for each location

Returns

results - Dataframe of same format as SfrFile.df, but subset to input locations.

Return type

dataframe

get_times()

Parse the stress period/timestep headers.

Returns

kstp**kper** - list of kstp, kper tuples

Return type

tuple

flopy.utils.swroutputfile module

class SwrBudget(filename, precision='double', verbose=False)

Bases: [SwrFile](#)

Read binary SWR budget output from MODFLOW SWR Process binary output files

Parameters

- **filename** (*string*) - Name of the swr budget output file
- **precision** (*string*) - 'single' or 'double'. Default is 'double'.
- **verbose** (*bool*) - Write information to the screen. Default is False.

Notes

Examples

```
>>> import flopy
>>> stageobj = flopy.utils.SwrStage('mymodel.swr.bud')
```

class SwrExchange(filename, precision='double', verbose=False)

Bases: [SwrFile](#)

Read binary SWR surface-water groundwater exchange output from MODFLOW SWR Process binary output files

Parameters

- **filename** (*string*) - Name of the swr surface-water groundwater exchange output file
- **precision** (*string*) - 'single' or 'double'. Default is 'double'.
- **verbose** (*bool*) - Write information to the screen. Default is False.

Notes

Examples

```
>>> import flopy
>>> stageobj = flopy.utils.SwrStage('mymodel.swr.qaq')
```

```
class SwrFile(filename, swrtype='stage', precision='double', verbose=False)
```

Bases: *FlopyBinaryData*

Read binary SWR output from MODFLOW SWR Process binary output files The SwrFile class is the super class from which specific derived classes are formed. This class should not be instantiated directly

Parameters

- **filename** (*string*) - Name of the swr output file
- **swrtype** (*str*) - swr data type. Valid data types are 'stage', 'budget', 'flow', 'exchange', or 'structure'. (default is 'stage')
- **precision** (*string*) - 'single' or 'double'. Default is 'double'.
- **verbose** (*bool*) - Write information to the screen. Default is False.

Notes

Examples

```
>>> import flopy
>>> so = flopy.utils.SwrFile('mymodel.swr.stage.bin')
```

```
get_connectivity()
```

Get connectivity data from the file.

Returns

data - Array has size (nrecord, 3). None is returned if swrtype is not 'flow'

Return type

numpy array

Notes

Examples

```
get_data(idx=None, kswrkstpkper=None, totim=None)
```

Get data from the file for the specified conditions.

Parameters

- **idx** (*int*) - The zero-based record number. The first record is record 0. (default is None)

- **kswrkstpkiper** (*tuple of ints*) - A tuple containing the swr time step, time step, and stress period (kswr, kstp, kper). These are zero-based kswr, kstp, and kper values. (default is None)
- **totim** (*float*) - The simulation time. (default is None)

Returns

data - Array has size (nitems).

Return type

numpy record array

Notes

if both kswrkstpkiper and totim are None, will return the last entry

Examples**get_kswrkstpkiper()**

Get a list of unique stress periods, time steps, and swr time steps in the file

Returns

out - List of unique kswr, kstp, kper combinations in binary file. kswr, kstp, and kper values are zero-based.

Return type

list of (kswr, kstp, kper) tuples

get_nrecords()

Get the number of records in the file

Returns

out - A tupe with the number of records and number of flow items in the file. The number of flow items is non-zero only if swrtype='flow'.

Return type

tuple of int

get_ntimes()

Get the number of times in the file

Returns

out - The number of simulation times (totim) in binary file.

Return type

int

get_record_names()

Get a list of unique record names in the file

Returns

out - List of unique text names in the binary file.

Return type

list of strings

get_times()

Get a list of unique times in the file

Returns

out - List contains unique simulation times (totim) in binary file.

Return type

list of floats

get_ts(irec=0, iconn=0, klay=0, istr=0)

Get a time series from a swr binary file.

Parameters

- **irec** (*int*) - is the zero-based reach (stage, qm, qaq) or reach group number (budget) to retrieve. (default is 0)
- **iconn** (*int*) - is the zero-based connection number for reach (irch) to retrieve qm data. iconn is only used if qm data is being read. (default is 0)
- **klay** (*int*) - is the zero-based layer number for reach (irch) to retrieve qaq data . klay is only used if qaq data is being read. (default is 0)
- **klay** - is the zero-based structure number for reach (irch) to retrieve structure data . isrt is only used if structure data is being read. (default is 0)

Returns

out - Array has size (ntimes, nitens). The first column in the data array will contain time (totim). nitens is 2 for stage data, 15 for budget data, 3 for qm data, and 11 for qaq data.

Return type

numpy recarray

Notes

The irec, iconn, and klay values must be zero-based.

Examples

```
class SwrFlow(filename, precision='double', verbose=False)
```

Bases: *SwrFile*

Read binary SWR flow output from MODFLOW SWR Process binary output files

Parameters

- **filename** (*string*) - Name of the swr flow output file
- **precision** (*string*) - 'single' or 'double'. Default is 'double'.
- **verbose** (*bool*) - Write information to the screen. Default is False.

Notes

Examples

```
>>> import flopy
>>> stageobj = flopy.utils.SwrStage('mymodel.swr.flow')
```

```
class SwrStage(filename, precision='double', verbose=False)
```

Bases: [SwrFile](#)

Read binary SWR stage output from MODFLOW SWR Process binary output files

Parameters

- **filename** (*string*) - Name of the swr stage output file
- **precision** (*string*) - 'single' or 'double'. Default is 'double'.
- **verbose** (*bool*) - Write information to the screen. Default is False.

Notes

Examples

```
>>> import flopy
>>> stageobj = flopy.utils.SwrStage('mymodel.swr.stg')
```

```
class SwrStructure(filename, precision='double', verbose=False)
```

Bases: [SwrFile](#)

Read binary SWR structure output from MODFLOW SWR Process binary output files

Parameters

- **filename** (*string*) - Name of the swr structure output file
- **precision** (*string*) - 'single' or 'double'. Default is 'double'.
- **verbose** (*bool*) - Write information to the screen. Default is False.

Notes

Examples

```
>>> import flopy
>>> stageobj = flopy.utils.SwrStage('mymodel.swr.str')
```

flopy.utils.triangle module

```
class Triangle(model_ws='.', exe_name='triangle', maximum_area=None, angle=20.0,
               nodes=None, additional_args=None)
```

Bases: `object`

Class to work with the triangle program to unstructured triangular grids.
 Information on the triangle program can be found at <https://www.cs.cmu.edu/~quake/triangle.html>

Parameters

- **model_ws** (*str*) - workspace location for creating triangle files (default is '.')
- **exe_name** (*str*) - path and name of the triangle program. (default is triangle, which means that the triangle program must be in your path)
- **maximum_area** (*float*) - the maximum area for any triangle. The default value is None, which means that the user must specify maximum areas for each region.
- **angle** (*float*) - Triangle will continue to add vertices until no angle is less than this specified value. (default is 20 degrees)
- **nodes** (*ndarray*) - Two dimensional array of shape (npoints, 2) with x and y positions of fixed node locations to include in the resulting triangular mesh. (default is None)
- **additional_args** (*list*) - list of additional command line switches to pass to triangle

Return type

None

add_hole(*hole*)

Add a point that will turn enclosing polygon into a hole

Parameters

hole (*tuple*) - (x, y)

Return type

None

add_polygon(*polygon*)

Add a polygon

Parameters

polygon (*list*, *geojson*, *shapely.geometry*, *shapefile.Shape*) - add polygon method accepts any of these geometries:

a list of (x, y) points
 geojson Polygon object
 shapely Polygon object
 shapefile Polygon shape
 flopy.utils.geometry.Polygon object

Return type

None

add_region(*point*, *attribute*=0, *maximum_area*=None)

Add a point that will become a region with a maximum area, if specified.

Parameters

- **point** (*tuple*) - (x, y)
- **attribute** (*integer or float*) - integer value assigned to output elements
- **maximum_area** (*float*) - maximum area of elements in region

Return type

None

build(*verbose=False*)

Build the triangular mesh

Parameters

verbose (*bool*) - If true, print the results of the triangle command to the terminal (default is False)

Return type

None

clean()

Remove the input and output files created by this class and by the Triangle program

Return type

None

get_attribute_array()

Return an array containing the attribute value for each cell. These are the attribute values that are passed into the `add_region()` method.

Returns

attribute_array

Return type

ndarray

get_boundary_marker_array()

Get an integer array that has boundary markers

Returns

iedge - integer array of size `ncpl` containing a boundary ids. The array contains zeros for cells that do not touch a boundary. The boundary ids are the segment numbers for each segment in each polygon that is added with the `add_polygon` method.

Return type

ndarray

get_cell2d()

Get a list of the information needed for the MODFLOW DISV Package.

Returns

cell2d - innermost list contains cell number, x, y, number of vertices, and then the vertex numbers comprising the cell.

Return type

list (of lists)

get_cell_edge_length(*n, ibm*)

Get the length of the edge for cell *n* that corresponds to boundary marker *ibm*

Parameters

- **n** (*int*) - cell number. $0 \leq n < \text{self.ncpl}$
- **ibm** (*integer*) - boundary marker number

Returns

length - Length of the edge along that boundary marker. Will return None if cell n does not touch boundary marker.

Return type

float

get_edge_cells(ibm)

Get a list of cell numbers that correspond to the specified boundary marker.

Parameters

ibm (*integer*) - boundary marker value

Returns

cell_list - list of zero-based cell numbers

Return type

list

get_vertices()

Get a list of vertices in the form needed for the MODFLOW DISV Package.

Returns

vertices - innermost list contains vertex number, x, and y

Return type

list (of lists)

get_xcyc()

Get a 2-dimensional array of x and y cell center coordinates.

Returns

xcyc - column 0 contains the x coordinates and column 1 contains the y coordinates

Return type

ndarray

label_cells(ax=None, onebased=True, **kwargs)

Label the cells with their cell numbers

Parameters

- **ax** (*matplotlib.pyplot.Axes*) - axis to add the plot to. (default is `plt.gca()`)
- **onebased** (*bool*) - Make the labels one-based if True so that they correspond to what would be written to MODFLOW.
- **kwargs** (*dictionary*) - dictionary of arguments to pass to `ax.text()`

Return type

None

label_vertices(*ax=None, onebased=True, **kwargs*)

Label the mesh vertices with their vertex numbers

Parameters

- **ax** (*matplotlib.pyplot.Axes*) - axis to add the plot to. (default is `plt.gca()`)
- **onebased** (*bool*) - Make the labels one-based if True so that they correspond to what would be written to MODFLOW.
- **kwargs** (*dictionary*) - dictionary of arguments to pass to `ax.text()`

Return type

None

plot(*ax=None, layer=0, edgecolor='k', facecolor='none', cmap='Dark2', a=None, masked_values=None, **kwargs*)

Plot the grid. This method will plot the grid using the shapefile that was created as part of the build method.

Note that the layer option is not working yet.

Parameters

- **ax** (*matplotlib.pyplot axis*) - The plot axis. If not provided it, `plt.gca()` will be used. If there is not a current axis then a new one will be created.
- **layer** (*int*) - Layer number to plot
- **cmap** (*string*) - Name of colormap to use for polygon shading (default is 'Dark2')
- **edgecolor** (*string*) - Color name. (Default is 'scaled' to scale the edge colors.)
- **facecolor** (*string*) - Color name. (Default is 'scaled' to scale the face colors.)
- **a** (*numpy.ndarray*) - Array to plot.
- **masked_values** (*iterable of floats, ints*) - Values to mask.
- **kwargs** (*dictionary*) - Keyword arguments that are passed to `PatchCollection.set(**kwargs)`. Some common kwargs would be 'linewidths', 'linestyles', 'alpha', etc.

Return type

None

plot_boundary(*ibm, ax=None, **kwargs*)

Plot a line and vertices for the specified boundary marker

Parameters

- **ibm** (*integer*) - plot the boundary for this boundary marker
- **ax** (*matplotlib.pyplot.Axes*) - axis to add the plot to. (default is `plt.gca()`)
- **kwargs** (*dictionary*) - dictionary of arguments to pass to `ax.plot()`

Return type

None

plot_centroids(*ax=None, **kwargs*)

Plot the cell centroids

Parameters

- **ax** (*matplotlib.pyplot.Axes*) - axis to add the plot to. (default is `plt.gca()`)
- **kwargs** (*dictionary*) - dictionary of arguments to pass to `ax.plot()`

Return type

None

plot_vertices(*ax=None, **kwargs*)

Plot the mesh vertices

Parameters

- **ax** (*matplotlib.pyplot.Axes*) - axis to add the plot to. (default is `plt.gca()`)
- **kwargs** (*dictionary*) - dictionary of arguments to pass to `ax.plot()`

Return type

None

flopy.utils.util_array module**util_array module.** Contains the **util_2d**, **util_3d** and **transient_2d** classes.

These classes encapsulate modflow-style array inputs away from the individual packages. The end-user should not need to instantiate these classes directly.

class **ArrayFormat**(*u2d, python=None, fortran=None, array_free_format=None*)Bases: `object`

ArrayFormat class for handling various output format types for both MODFLOW and flopy

Parameters

- **u2d** (*Util2d instance*) -
- **python** (*str (optional)*) - python-style output format descriptor
e.g. `{0:15.6e}`
- **fortran** (*str (optional)*) - fortran style output format descriptor
e.g. `(2E15.6)`

fortranfortran format output descriptor (e.g. `(100G15.6)`)**Type**`str`**py**python format output descriptor (e.g. `"{0:15.6E}"`)

Type

`str`

numpy

numpy format output descriptor (e.g. `"%15.6e"`)

Type

`str`

npl

number of items per line of output

Type

`int`

width

the width of the formatted numeric output

Type

`int`

decimal

the number of decimal digits in the numeric output

Type

`int`

format

the output format type e.g. `I`, `G`, `E`, etc

Type

`str`

free

free format flag

Type

`bool`

binary

binary format flag

Type

`bool`

get_default_numpy_fmt : (`dtype` : [`np.int32`, `np.float32`])

a static method to get a default numpy dtype - used for loading

decode_fortran_descriptor : (`fd` : `str`)

a static method to decode fortran descriptors into `npl`, `format`, `width`, `decimal`.

Notes

Examples

property array_free_format

property binary

property decimal

static decode_fortran_descriptor(*fd*)

Decode fortran descriptor

Parameters

fd (*str*) -

Returns

npl, *fmt*, *width*, *decimal*

Return type

int, *str*, *int*, *int*

classmethod float()

property format

property fortran

property free

static get_default_numpy_fmt(*dtype*)

classmethod integer()

property npl

property numpy

property py

property width

class Transient2d(*model*, *shape*, *dtype*, *value*, *name*, *fmtin*=None, *cnstnt*=1.0, *iprn*=-1, *ext_filename*=None, *locat*=None, *bin*=False, *array_free_format*=None)

Bases: DataInterface

Transient2d class for handling time-dependent 2-D model arrays. just a thin wrapper around Util2d

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **shape** (*length 2 tuple*) - shape of the 2-D transient arrays, typically (*nrow*,*ncol*)
- **dtype** (*[np.int32, np.float32, bool]*) - the type of the data

- **value** (*variable*) - the data to be assigned to the 2-D arrays. Typically a dict of {kper:value}, where kper is the zero-based stress period to assign a value to. Value should be cast-able to Util2d instance can be a scalar, list, or ndarray is the array value is constant in time.
- **name** (*string*) - name of the property, used for writing comments to input files and for forming external files names (if needed)
- **fmtin** (*string*) - modflow fmtin variable (optional). (the default is None)
- **cnstnt** (*string*) - modflow cnstnt variable (optional) (the default is 1.0)
- **iprn** (*int*) - modflow iprn variable (optional) (the default is -1)
- **locat** (*int*) - modflow locat variable (optional) (the default is None). If the model instance does not support free format and the external flag is not set and the value is a simple scalar, then locat must be explicitly passed as it is the unit number

to read the array from

- **ext_filename** (*string*) - the external filename to write the array representation to (optional) (the default is None) . If type(value) is a string and is an accessible filename, the ext_filename is reset to value.
- **bin** (*bool*) - flag to control writing external arrays as binary (optional) (the default is False)

transient_2ds

the transient sequence of Util2d objects

Type

dict{kper:Util2d}

get_kper_entry : (itmp,string)

get the itmp value and the Util2d file entry of the value in transient_2ds in bin kper. if kper < min(Transient2d.keys()), return (1,zero_entry<Util2d>). If kper > < min(Transient2d.keys()), but is not found in Transient2d.keys(), return (-1,"")

Notes

Examples

property array

build_transient_sequence()

parse self.__value into a dict{kper:Util2d}

property data_type

property dtype

export(f, **kwargs)

classmethod `from_4d(model, pak_name, m4ds)`

construct a Transient2d instance from a dict(name: (masked) 4d numpy.ndarray
:param model: :type model: flopy.mbase derived type :param pak_name: :type
pak_name: str package name (e.g. RCH) :param m4ds: each ndarray must have
shape (nper,1,nrow,ncol).

if an entire (nrow,ncol) slice is np.NaN, then that kper is skipped.

Return type

Transient2d instance

get_kper_entry(kper)

Get the file entry info for a given kper returns (itmp,file entry string from
Util2d)

get_zero_2d(kper)

static `masked4d_array_to_kper_dict(m4d)`

property `model`

property `name`

plot(filename_base=None, file_extension=None, kper=0, fignum=None, **kwargs)

Plot transient 2-D model input data

Parameters

- **filename_base** (*str*) - Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)
- **file_extension** (*str*) - Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')
- **kper** (*int* or *str*) - model stress period. if 'all' is provided, all stress periods will be plotted
- **fignum** (*list* or *int*) - Figure numbers for plot title
- ****kwargs** (*dict*) -

axes

[list of matplotlib.pyplot.axis] List of matplotlib.pyplot.axis that will be used to plot data for each layer. If axes=None axes will be generated. (default is None)

pcolor

[bool] Boolean used to determine if matplotlib.pyplot.pcolormesh plot will be plotted. (default is True)

colorbar

[bool] Boolean used to determine if a color bar will be added to the matplotlib.pyplot.pcolormesh. Only used if pcolor=True. (default is False)

inactive

[bool] Boolean used to determine if a black overlay in inactive cells in a layer will be displayed. (default is True)

contour

[bool] Boolean used to determine if matplotlib.pyplot.contour plot will be plotted. (default is False)

clabel

[bool] Boolean used to determine if matplotlib.pyplot.clabel will be plotted. Only used if contour=True. (default is False)

grid

[bool] Boolean used to determine if the model grid will be plotted on the figure. (default is False)

masked_values

[list] List of unique values to be excluded from the plot.

kper

[str] MODFLOW zero-based stress period number to return. If kper='all' then data for all stress period will be extracted. (default is zero).

Returns

out - Empty list is returned if filename_base is not None. Otherwise a list of matplotlib.pyplot.axis is returned.

Return type

list

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.rch.rech.plot()
```

property plottable

```
class Transient3d(model, shape, dtype, value, name, fmtin=None, cnstnt=1.0, iprn=-1,
                  ext_filename=None, locat=None, bin=False, array_free_format=None)
```

Bases: DataInterface

Transient3d class for handling time-dependent 3-D model arrays. just a thin wrapper around Util3d

Parameters

- **model** (model object) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **shape** (length 3 tuple) - shape of the 3-D transient arrays, typically (nlay,nrow,ncol)

- **dtype** (`[np.int32, np.float32, bool]`) - the type of the data
- **value** (*variable*) - the data to be assigned to the 3-D arrays. Typically a dict of {kper:value}, where kper is the zero-based stress period to assign a value to. Value should be cast-able to Util2d instance can be a scalar, list, or ndarray is the array value is constant in time.
- **name** (*string*) - name of the property, used for writing comments to input files and for forming external files names (if needed)
- **fmtin** (*string*) - modflow fmtin variable (optional). (the default is None)
- **cnstnt** (*string*) - modflow cnstnt variable (optional) (the default is 1.0)
- **iprn** (*int*) - modflow iprn variable (optional) (the default is -1)
- **locat** (*int*) - modflow locat variable (optional) (the default is None). If the model instance does not support free format and the external flag is not set and the value is a simple scalar, then locat must be explicitly passed as it is the unit number
to read the array from
- **ext_filename** (*string*) - the external filename to write the array representation to (optional) (the default is None) . If type(value) is a string and is an accessible filename, the ext_filename is reset to value.
- **bin** (*bool*) - flag to control writing external arrays as binary (optional) (the default is False)

transient_3ds

the transient sequence of Util3d objects

Type

dict{kper:Util3d}

get_kper_entry : (itmp,string)

get the itmp value and the Util2d file entry of the value in transient_2ds in bin kper. if kper < min(Transient2d.keys()), return (1,zero_entry<Util2d>). If kper > < min(Transient2d.keys()), but is not found in Transient2d.keys(), return (-1,"")

Notes

Examples

property array

build_transient_sequence()

parse self.__value into a dict{kper:Util3d}

property data_type

property dtype

get_kper_entry(kper)

get the file entry info for a given kper returns (itmp,file entry string from Util3d)

get_zero_3d(kper)

property model

property name

property plottable

class Util2d(model, shape, dtype, value, name, fmtin=None, cnstnt=1.0, iprn=-1, ext_filename=None, locat=None, bin=False, how=None, array_free_format=None)

Bases: DataInterface

Util2d class for handling 1- or 2-D model arrays

Parameters

- **model** (*model object*) - The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **shape** (*tuple*) - Shape of the 1- or 2-D array
- **dtype** (*[np.int32, np.float32, bool]*) - the type of the data
- **value** (*variable*) - the data to be assigned to the 1- or 2-D array. can be a scalar, list, ndarray, or filename
- **name** (*string*) - name of the property (optional). (the default is None)
- **fmtin** (*string*) - modflow fmtin variable (optional). (the default is None)
- **cnstnt** (*string*) - modflow cnstnt variable (optional) (the default is 1.0)
- **iprn** (*int*) - modflow iprn variable (optional) (the default is -1)
- **locat** (*int*) - modflow locat variable (optional) (the default is None). If the model instance does not support free format and the external flag is not set and the value is a simple scalar, then locat must be explicitly passed as it is the unit number
to read the array from)
- **ext_filename** (*string*) - the external filename to write the array representation to (optional) (the default is None) . If type(value) is a string and is an accessible filename, the ext_filename is reset to value.
- **bin** (*bool*) - flag to control writing external arrays as binary (optional) (the default is False)

array

the array representation of the 2-D object

Type

np.ndarray

how

the str flag to control how the array is written to the model input files e.g. "constant","internal","external","open/close"

Type

str

format

controls the ASCII representation of the numeric array

Type

ArrayFormat object

get_file_entry : string

get the model input file string including the control record

Notes

If value is a valid filename and model.external_path is None, then a copy of the file is made and placed in model.model_ws directory.

If value is a valid filename and model.external_path is not None, then a copy of the file is made a placed in the external_path directory.

If value is a scalar, it is always written as a constant, regardless of the model.external_path setting.

If value is an array and model.external_path is not None, then the array is written out in the external_path directory. The name of the file that holds the array is created from the name attribute. If the model supports "free format", then the array is accessed via the "open/close" approach. Otherwise, a unit number and filename is added to the name file.

If value is an array and model.external_path is None, then the array is written internally to the model input file.

Examples**all()****property array**

Get the COPY of array representation of value attribute with the effects of the control record multiplier applied.

Returns

array - Copy of the array with the multiplier applied.

Return type

numpy.ndarray

Note: .array is a COPY of the array representation as seen by the model - with the effects of the control record multiplier applied.

```
static array2string(shape, data, fortran_format='(FREE)', python_format=None)
    return a string representation of a (possibly wrapped format) array from a
    file (self.__value) and casts to the proper type (self._dtype) made static to
    support the load functionality this routine now supports fixed format arrays
    where the numbers may touch.

property cnsntn_str

property data_type

property dtype

export(f, **kwargs)

property filename

property format

get_constant_cr(value)

get_external_cr()

get_file_entry(how=None)

get_internal_cr()

get_openclose_cr()

get_value()

property how

classmethod load(f_handle, model, shape, dtype, name, ext_unit_dict=None,
                 array_free_format=None, array_format='modflow')
    functionality to load Util2d instance from an existing model input file.
    external and internal record types must be fully loaded if you are using fixed
    format record types,make sure ext_unit_dict has been initialized from the NAM
    file

static load_bin(shape, file_in, dtype, bintype=None)
    Load unformatted file to a 2-D array
```

Parameters

- **shape** (*tuple of int*) - One or two array dimensions
- **file_in** (*file or str*) - Filename or file handle
- **dtype** (*np.int32 or np.float32*) - Data type of unformatted file and Numpy array; use np.int32 for Fortran's INTEGER, and np.float32 for Fortran's REAL data types.
- **bintype** (*str*) - Normally 'Head'

Notes

This method is similar to MODFLOW's U2DREL and U2DINT subroutines, but only for unformatted files.

Return type

2-D array

static load_block(*shape*, *file_in*, *dtype*)

Load block format from a MT3D file to a 2-D array

Parameters

- **shape** (*tuple of int*) - Array dimensions (nrow, ncol)
- **file_in** (*file or str*) - Filename or file handle
- **dtype** (*np.int32 or np.float32*) -

Return type

2-D array

static load_txt(*shape*, *file_in*, *dtype*, *fmtin*)

Load formatted file to a 1-D or 2-D array

Parameters

- **shape** (*tuple of int*) - One or two array dimensions
- **file_in** (*file or str*) - Filename or file handle
- **dtype** (*np.int32 or np.float32*) -
- **fmtin** (*str*) - Fortran array format descriptor, '(FREE)' or e.g. '(10G11.4)'

Notes

This method is similar to MODFLOW's U1DREL, U1DINT, U2DREL and U2DINT subroutines, but only for formatted files.

Return type

1-D or 2-D array

property model

property model_file_path

where the model expects the file to be

Returns

file_path (*str*)

Return type

path relative to the name file

property name

static parse_control_record(*line*, *current_unit=None*, *dtype=<class 'numpy.float32'>*, *ext_unit_dict=None*, *array_format=None*)

parses a control record when reading an existing file rectifies fixed to free format *current_unit* (optional) indicates the unit number of the file being parsed

parse_value(value)

parses and casts the raw value into an acceptable format for __value lot of defense here, so we can make assumptions later

plot(title=None, filename_base=None, file_extension=None, fignum=None, **kwargs)

Plot 2-D model input data

Parameters

- **title** (*str*) - Plot title. If a plot title is not provide one will be created based on data name (self._name). (default is None)
- **filename_base** (*str*) - Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)
- **file_extension** (*str*) - Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')
- ****kwargs** (*dict*) -

axes

[list of matplotlib.pyplot.axis] List of matplotlib.pyplot.axis that will be used to plot data for each layer. If axes=None axes will be generated. (default is None)

pcolor

[bool] Boolean used to determine if matplotlib.pyplot.pcolormesh plot will be plotted. (default is True)

colorbar

[bool] Boolean used to determine if a color bar will be added to the matplotlib.pyplot.pcolormesh. Only used if pcolor=True. (default is False)

inactive

[bool] Boolean used to determine if a black overlay in inactive cells in a layer will be displayed. (default is True)

contour

[bool] Boolean used to determine if matplotlib.pyplot.contour plot will be plotted. (default is False)

clabel

[bool] Boolean used to determine if matplotlib.pyplot.clabel will be plotted. Only used if contour=True. (default is False)

grid

[bool] Boolean used to determine if the model grid will be plotted on the figure. (default is False)

masked_values

[list] List of unique values to be excluded from the plot.

Returns

out - Empty list is returned if filename_base is not None.
Otherwise a list of matplotlib.pyplot.axis is returned.

Return type

list

Notes**Examples**

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.dis.top.plot()
```

property **plottable**

property **python_file_path**

where python is going to write the file :returns: **file_path (str)** :rtype:
path relative to python: includes model_ws

set_fmtin(fmtin)

property **string**

get the string representation of value attribute

Note: the string representation DOES NOT include the effects of the control
record multiplier - this method is used primarily for writing model input files

sum()

unique()

property **vtype**

static write_bin(shape, file_out, data, bintype=None, header_data=None)

static write_txt(shape, file_out, data, fortran_format='(FREE)', python_format=None)

class Util3d(model, shape, dtype, value, name, fmtin=None, cnstnt=1.0, iprn=-1,
locat=None, ext_unit_dict=None, array_free_format=None)

Bases: DataInterface

Util3d class for handling 3-D model arrays. just a thin wrapper around

Util2d

Parameters

- **model** (model object) - The model object (of type `flopy.modflow.mf.Modflow`) to which this package will be added.
- **shape** (length 3 tuple) - shape of the 3-D array, typically (nlay,nrow,ncol)
- **dtype** (`[np.int32, np.float32, bool]`) - the type of the data
- **value** (variable) - the data to be assigned to the 3-D array. can be a scalar, list, or ndarray

- **name** (*string*) - name of the property, used for writing comments to input files
- **fmtin** (*string*) - modflow fmtin variable (optional). (the default is None)
- **cnstnt** (*string*) - modflow cnstnt variable (optional) (the default is 1.0)
- **iprn** (*int*) - modflow iprn variable (optional) (the default is -1)
- **locat** (*int*) - modflow locat variable (optional) (the default is None). If the model instance does not support free format and the external flag is not set and the value is a simple scalar, then locat must be explicitly passed as it is the unit number to read the array from
- **ext_filename** (*string*) - the external filename to write the array representation to (optional) (the default is None) . If type(value) is a string and is an accessible filename, the ext_filename is reset to value.
- **bin** (*bool*) - flag to control writing external arrays as binary (optional) (the default is False)

array

the array representation of the 3-D object

Type

np.ndarray

get_file_entry : string

get the model input file string including the control record for the entire 3-D property

Notes

Examples

property array

Return a numpy array of the 3D shape. If an unstructured model, then return an array of size nodes.

build_2d_instances()

property data_type

property dtype

export(*f*, ***kwargs*)

get_file_entry()

get_value()

classmethod load(*f_handle*, *model*, *shape*, *dtype*, *name*, *ext_unit_dict*=None, *array_format*=None)

property model

property name

`plot(filename_base=None, file_extension=None, mflay=None, fignum=None, **kwargs)`

Plot 3-D model input data

Parameters

- **filename_base** (*str*) - Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)
- **file_extension** (*str*) - Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')
- **mflay** (*int*) - MODFLOW zero-based layer number to return. If None, then all all layers will be included. (default is None)
- ****kwargs** (*dict*) -

axes

[list of matplotlib.pyplot.axis] List of matplotlib.pyplot.axis that will be used to plot data for each layer. If axes=None axes will be generated. (default is None)

pcolor

[bool] Boolean used to determine if matplotlib.pyplot.pcolormesh plot will be plotted. (default is True)

colorbar

[bool] Boolean used to determine if a color bar will be added to the matplotlib.pyplot.pcolormesh. Only used if pcolor=True. (default is False)

inactive

[bool] Boolean used to determine if a black overlay in inactive cells in a layer will be displayed. (default is True)

contour

[bool] Boolean used to determine if matplotlib.pyplot.contour plot will be plotted. (default is False)

clabel

[bool] Boolean used to determine if matplotlib.pyplot.clabel will be plotted. Only used if contour=True. (default is False)

grid

[bool] Boolean used to determine if the model grid will be plotted on the figure. (default is False)

masked_values

[list] List of unique values to be excluded from the plot.

Returns

out - Empty list is returned if filename_base is not None.
Otherwise a list of matplotlib.pyplot.axis is returned.

Return type

list

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.lpf.hk.plot()
```

property plottable

new_u2d(old_util2d, value)

read1d(f, a)

Fill the 1d array, a, with the correct number of values. Required in case lpf 1d arrays (chani, layvka, etc) extend over more than one line

flopy.utils.util_list module

util_list module. Contains the mflist class.

This classes encapsulates modflow-style list inputs away from the individual packages. The end-user should not need to instantiate this class directly.

some more info

class MfList(package, data=None, dtype=None, model=None, list_free_format=None, binary=False)

Bases: DataInterface, DataListInterface

a generic object for handling transient boundary condition lists

Parameters

- **package** (package object) - The package object (of type *flopy.pakbase.Package*) to which this MfList will be added.
- **data** (varies) - the data of the transient list (optional). (the default is None)

mxact

the max number of active bc for any stress period

Type

int

add_record(kper,index,value) : None

add a record to stress period kper at index location

write_transient(f) : None

write the transient sequence to the model input file f

check_kij() : None

checks for boundaries outside of model domain - issues warnings only

Notes

Examples

`add_record(kper, index, values)`

`append(other)`

append the recarrays from one MfList to another :param other: that corresponds with self :type other: variable: an item that can be cast in to an MfList

Returns

dict of {kper

Return type

recarray}

property array

`attribute_by_kper(attr, function=<function mean>, idx_val=None)`

property binary

`check_kij()`

property data

property data_type

property df

`drop(fields)`

drop fields from an MfList

Parameters

fields (*list or set of field names to drop*) -

Returns

dropped

Return type

MfList without the dropped fields

property dtype

`export(f, **kwargs)`

property fmt_string

Returns a C-style fmt string for numpy savetxt that corresponds to the dtype

classmethod from_4d(model, pak_name, m4ds)

construct an MfList instance from a dict of (attribute_name,masked 4D ndarray :param model: :type model: mbase derived type :param pak_name: :type pak_name: str package name (e.g GHB) :param m4ds: :type m4ds: {attribute name:4d masked numpy.ndarray}

Return type

MfList instance

`get_dataframe(squeeze=False)`

Cast recarrays for stress periods into single dataframe containing all stress periods.

Parameters

squeeze (*bool*) - Reduce number of rows in dataframe to only include stress periods where a variable changes.

Returns

df - Dataframe of shape `nrow = nper x ncells`, `ncol = nvar`. If the `squeeze` option is chosen, `nper` is the number of stress periods where at least one cells is different, otherwise it is equal to the number of keys in `MfList.data`.

Return type

dataframe

Notes

Requires `pandas`.

`get_empty(ncell=0)`

`get_filename(kper)`

`get_filenames()`

`get_indices()`

a helper function for plotting - get all unique indices

`get_itmp(kper)`

`static masked4D_arrays_to_stress_period_data(dtype, m4ds)`

convert a dictionary of 4-dim masked arrays to
a `stress_period_data` style dict of recarray

Parameters

- **dtype** (*numpy dtype*) -
- **m4ds** (*dict {name:masked numpy 4-dim ndarray}*) -

Returns

dict {kper

Return type

recarray}

property `masked_4D_arrays`

`masked_4D_arrays_itr()`

property `mg`

property `model`

property `mxact`

property name

property package

plot(key=None, names=None, kper=0, filename_base=None, file_extension=None, mflay=None, **kwargs)

Plot stress period boundary condition (MfList) data for a specified stress period

Parameters

- **key** (*str*) - MfList dictionary key. (default is None)
- **names** (*list*) - List of names for figure titles. (default is None)
- **kper** (*int*) - MODFLOW zero-based stress period number to return. (default is zero)
- **filename_base** (*str*) - Base file name that will be used to automatically generate file names for output image files. Plots will be exported as image files if file_name_base is not None. (default is None)
- **file_extension** (*str*) - Valid matplotlib.pyplot file extension for savefig(). Only used if filename_base is not None. (default is 'png')
- **mflay** (*int*) - MODFLOW zero-based layer number to return. If None, then all layers will be included. (default is None)
- ****kwargs** (*dict*) -

axes

[list of matplotlib.pyplot.axis] List of matplotlib.pyplot.axis that will be used to plot data for each layer. If axes=None axes will be generated. (default is None)

pcolor

[bool] Boolean used to determine if matplotlib.pyplot.pcolormesh plot will be plotted. (default is True)

colorbar

[bool] Boolean used to determine if a color bar will be added to the matplotlib.pyplot.pcolormesh. Only used if pcolor=True. (default is False)

inactive

[bool] Boolean used to determine if a black overlay in inactive cells in a layer will be displayed. (default is True)

contour

[bool] Boolean used to determine if matplotlib.pyplot.contour plot will be plotted. (default is False)

clabel

[bool] Boolean used to determine if

matplotlib.pyplot.clabel will be plotted. Only used if contour=True. (default is False)

grid

[bool] Boolean used to determine if the model grid will be plotted on the figure. (default is False)

masked_values

[list] List of unique values to be excluded from the plot.

Returns

out - Empty list is returned if filename_base is not None. Otherwise a list of matplotlib.pyplot.axis is returned.

Return type

list

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.wel.stress_period_data.plot(ml.wel, kper=1)
```

property plottable

to_array(kper=0, mask=False)

Convert stress period boundary condition (MfList) data for a specified stress period to a 3-D numpy array

Parameters

- **kper** (*int*) - MODFLOW zero-based stress period number to return. (default is zero)
- **mask** (*boolean*) - return array with np.NaN instead of zero

Returns

out - Dictionary of 3-D numpy arrays containing the stress period data for a selected stress period. The dictionary keys are the MfList dtype names for the stress period data ('cond', 'flux', 'bhead', etc.).

Return type

dict of numpy.ndarrays

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> v = ml.wel.stress_period_data.to_array(kper=1)
```

to_shapefile(*filename*, *kper*=None)

Export stress period boundary condition (MfList) data for a specified stress period

Parameters

- **filename** (*str*) - Shapefile name to write
- **kper** (*int*) - MODFLOW zero-based stress period number to return. (default is None)

Return type

None

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.wel.to_shapefile('test_hk.shp', kper=1)
```

property vtype

write_transient(*f*, *single_per*=None, *forceInternal*=False, *write_header*=True, *cln_data*=None)

flopy.utils.utils_def module

Generic classes and utility functions

class FlopyBinaryData

Bases: `object`

The FlopyBinaryData class is a class to that defines the data types for integer, floating point, and character data in MODFLOW binary files. The FlopyBinaryData class is the super class from which the specific derived classes are formed. This class should not be instantiated directly.

read_integer()

read_real()

read_record(*count*, *dtype*=None)

read_text(*nchar*=20)

set_float(*precision*)

get_dis(*model*)

Returns dis or disu object from a given model object.

get_open_file_object(*fname_or_fobj*, *read_write*='rw')

Returns an open file object for either a file name or open file object.

get_pak_vals_shape(*model*, *vals*)

Function to define shape of package input data for Util2d.

Parameters

- **model** (*flopy model object*) -
- **vals** (*Package input values (dict of arrays or scalars, or ndarray, or) - single scalar*).

Returns

shape - shape of input data for Util2d

Return type

tuple

get_unitnumber_from_ext_unit_dict(*model*, *pak_class*, *ext_unit_dict*=None, *ipakcb*=0)

For a given modflow package, defines input file unit number, plus package input and (optionally) output (budget) save file names.

Parameters

- **model** (*model object*) - model for which the unit number is sought.
- **pak_class** (*modflow package class for which the unit number is sought.*) -
- **ext_unit_dict** (*external unit dictionary, optional.*) - If not provided, unitnumber and filenames will be returned as None.
- **ipakcb** (*int, optional*) - Modflow package unit number on which budget is saved. Default is 0, in which case the returned output file is None.

Returns

- **unitnumber** (*int*) - file unit number for the given modflow package (or None)
- **filenames** (*list*) - list of [package input file name, budget file name],

get_util2d_shape_for_layer(*model*, *layer*=0)

Define nrow and ncol for array (Util2d) shape of a given layer in structured and/or unstructured models.

Parameters

- **model** (*model object*) - model for which Util2d shape is sought.
- **layer** (*int*) - layer (base 0) for which Util2d shape is sought.

Returns

(nrow,ncol) - util2d shape for the given layer

Return type

tuple of ints

totim_to_datetime(*totim*, *start*='1-1-1970', *timeunit*='D')

Parameters

- **totim** (*list or numpy array*) -
- **start** (*str*) - Starting date for simulation. (default is 1-1-1970).

- **timeunit** (*string*) - time unit of the simulation time. Valid values are 'S'econds, 'M'inutes, 'H'ours, 'D'ays, 'Y'ears. (default is 'D').

Returns

out - datetime object calculated from start and totim values

Return type

`list`

type_from_iterable(*_iter*, *index*=0, *_type*=<class 'int'>, *default_val*=0)

Returns value of specified type from iterable.

Parameters

- **_iter** (*iterable*) -
- **index** (*int*) - Iterable index to try to convert
- **_type** (*Python type*) -
- **default_val** (*default value (0)*) -

Returns

val

Return type

value of type *_type*, or *default_val*

flopy.utils.utl_import module

get_version(*module*: *module*) → *str*

import_optional_dependency(*name*: *str*, *error_message*: *str* = "", *errors*: *str* = 'raise', *min_version*: *str* | *None* = *None*)

Import an optional dependency.

By default, if a dependency is missing an ImportError with a nice message will be raised. If a dependency is present, but too old, we raise.

Parameters

- **name** (*str*) - The module name.
- **error_message** (*str*) - Additional text to include in the ImportError message.
- **errors** (*str* {'raise', 'warn', 'ignore'}) - What to do when a dependency is not found or its version is too old.
 - raise : Raise an ImportError
 - warn : Only applicable when a module's version is too old. Warns that the version is too old and returns None
 - ignore: If the module is not installed, return None, otherwise, return the module, even if the version is too old. It's expected that users validate the version locally when using errors="ignore" (see. io/html.py)
 - silent: Same as "ignore" except warning message is not written to the screen.

- **min_version** (*str*, default *None*) - Specify a minimum version that is different from the global FloPy minimum version required.

Returns

maybe_module - The imported module, when found and the version is correct. *None* is returned when the package is not found and *errors* is *False*, or when the package's version is too old and *errors* is 'warn'.

Return type

Optional[ModuleType]

flopy.utils.voronoi module

class VoronoiGrid(*tri*, ****kwargs**)

Bases: *object*

FloPy VoronoiGrid helper class for creating a voronoi model grid from an array of input points that define cell centers. The class handles boundary cells by closing polygons along the edge, something that cannot be done directly with the *scipy.spatial.Voronoi* class.

Parameters

- **input** (*flopy.utils.Triangle*) - Constructed and built flopy *Triangle* object.
- **kwargs** (*dict*) - List of additional keyword arguments that will be passed through to *scipy.spatial.Voronoi*. For circular shaped model grids, the *qhull_options='Qz'* option has been found to work well.

Notes

When using *VoronoiGrid*, the construction order used for the *Triangle* grid matters. The first *add_polygon()* call must be to add the model domain. Then *add_polygon()* must be used to add any holes. Lastly, *add_polygon()* can be used to add regions. This *VoronoiGrid* class uses this order to find model edges that require further work for defining and closing edge model cells.

get_disu5_gridprops()

get_disu6_gridprops()

get_disv_gridprops()

Get a dictionary of arguments that can be passed in to the *flopy.mf6.ModflowGwfdisv* class.

Returns

disv_gridprops - Dictionary of arguments than can be unpacked into the *flopy.mf6.ModflowGwfdisv* constructor

Return type

dict

get_gridprops_unstructuredgrid()

Get a dictionary of information needed to create a flopy *UnstructuredGrid*. The returned dictionary can be unpacked directly into the *flopy.discretization.UnstructuredGrid()* constructor.

Returns
gridprops

Return type
`dict`

get_gridprops_vertexgrid()

Get a dictionary of information needed to create a flopy VertexGrid.
 The returned dictionary can be unpacked directly into the
 flopy.discretization.VertexGrid() constructor.

Returns
gridprops

Return type
`dict`

get_patch_collection(ax=None, **kwargs)

Get a matplotlib patch collection representation of the voronoi grid

Parameters

- **ax** (*matplotlib.pyplot.Axes*) - axes to plot the patch collection
- **kwargs** (*dict*) - Additional keyword arguments to pass to the flopy.plot.plot_cvfd function that returns a patch collection from verts and iverts

Returns
pc - patch collection of model

Return type
`matplotlib.collections.PatchCollection`

plot(ax=None, plot_title=True, **kwargs)

Plot the voronoi model grid

Parameters

- **ax** (*matplotlib.pyplot.Axes*) - axes to plot the voronoi grid
- **plot_title** (*bool*) - Add the number of cells and number of vertices as a plot title
- **kwargs** (*dict*) - Additional keyword arguments to pass to self.get_patch_collection

Returns
ax - axes that contains the voronoi model grid

Return type
`matplotlib.pyplot.Axes`

get_sorted_vertices(icell_vertices, vertices)

get_valid_faces(vor)

point_in_cell(point, vertices)

sort_vertices(vlist)

tri2vor(tri, **kwargs)

This is the workhorse for the VoronoiGrid class for creating a voronoi grid from a constructed and built flopy Triangle grid.

Parameters

- **tri** (*flopy.utils.Triangle*) -
- **voronoi** (*Flopy triangle object is used to construct the complementary*) -
- **diagram.** -

Returns

verts, iverts

Return type

ndarray, list of lists

flopy.utils.zonbud module

class ZBNetOutput(zones, time, arrays, zone_array, flux=True)

Bases: *object*

Class that holds zonebudget netcdf output and allows export utilities to recognize the output data type.

Parameters

- **zones** (*np.ndarray*) - array of zone numbers
- **time** (*np.ndarray*) - array of totim
- **arrays** (*dict*) - dictionary of budget term arrays. axis 0 is totim, axis 1 is zones
- **flux** (*bool*) - boolean flag to indicate if budget data is a flux “L³/T”(True, default) or if the data have been processed to volumetric values “L³” (False)

class ZoneBudget(cbc_file, z, kstpker=None, totim=None, aliases=None, verbose=False, **kwargs)

Bases: *object*

ZoneBudget class

Parameters

- **cbc_file** (*str* or *CellBudgetFile* object) - The file name or CellBudgetFile object for which budgets will be computed.
- **z** (*ndarray*) - The array containing to zones to be used.
- **kstpker** (*tuple of ints*) - A tuple containing the time step and stress period (kstp, kper). The kstp and kper values are zero based.
- **totim** (*float*) - The simulation time.
- **aliases** (*dict*) - A dictionary with key, value pairs of zones and aliases. Replaces the corresponding record and field names with the aliases provided. When using this option in conjunction with a list of zones, the zone(s) passed may either be all strings (aliases), all integers, or mixed.

Return type
None

Examples

```
>>> from flopy.utils.zonbud import ZoneBudget
>>> zon = ZoneBudget.read_zone_file('zone_input_file')
>>> zb = ZoneBudget('zonebudtest.cbc', zon, kstpker=(0, 0))
>>> zb.to_csv('zonebudtest.csv')
>>> zb_mgd = zb * 7.48052 / 1000000
```

copy()

Return a deepcopy of the object.

export(f, ml, **kwargs)

Method to export a netcdf file, or add zonebudget output to an open netcdf file instance

Parameters

- **f** (*str* or *flopy.export.netcdf.NetCdf* object) -
- **ml** (*flopy.modflow.Modflow* or *flopy.mf6.ModflowGwf* object) -
- ****kwargs** - **logger** : *flopy.export.netcdf.Logger* instance
masked_vals : list
list of values to mask

Return type

flopy.export.netcdf.NetCdf object

get_budget(names=None, zones=None, net=False, pivot=False)

Get a list of zonebudget record arrays.

Parameters

- **names** (*list* of *strings*) - A list of strings containing the names of the records desired.
- **zones** (*list* of *ints* or *strings*) - A list of integer zone numbers or zone names desired.
- **net** (*boolean*) - If True, returns net IN-OUT for each record.
- **pivot** (*boolean*) - If True, returns data in a more user friendly format

Returns

budget_list - A list of the zonebudget record arrays.

Return type

list of record arrays

Examples

```
>>> names = ['FROM_CONSTANT_HEAD', 'RIVER_LEAKAGE_OUT']
>>> zones = ['ZONE_1', 'ZONE_2']
>>> zb = ZoneBudget('zonebudtest.cbc', zon, kstpkper=(0, 0))
>>> bud = zb.get_budget(names=names, zones=zones)
```

```
get_dataframes(start_datetime=None, timeunit='D', index_key='totim', names=None,
               zones=None, net=False, pivot=False)
```

Get pandas dataframes.

Parameters

- **start_datetime** (*str*) - Datetime string indicating the time at which the simulation starts.
- **timeunit** (*str*) - String that indicates the time units used in the model.
- **index_key** (*str*) - Indicates the fields to be used (in addition to “record”) in the resulting DataFrame multi-index.
- **names** (*list of strings*) - A list of strings containing the names of the records desired.
- **zones** (*list of ints or strings*) - A list of integer zone numbers or zone names desired.
- **net** (*boolean*) - If True, returns net IN-OUT for each record.
- **pivot** (*bool*) - If True, returns dataframe in a more user friendly format

Returns

df - Pandas DataFrame with the budget information.

Return type

Pandas DataFrame

Examples

```
>>> from flopy.utils.zonbud import ZoneBudget
>>> zon = ZoneBudget.read_zone_file('zone_input_file')
>>> zb = ZoneBudget('zonebudtest.cbc', zon, kstpkper=(0, 0))
>>> df = zb.get_dataframes()
```

```
get_model_shape()
```

Get model shape

Returns

- **nlay** (*int*) - Number of layers
- **nrow** (*int*) - Number of rows
- **ncol** (*int*) - Number of columns

get_record_names(stripped=False)

Get a list of water budget record names in the file.

Returns

out - List of unique text names in the binary file.

Return type

list of strings

Examples

```
>>> zb = ZoneBudget('zonebudtest.cbc', zon, kstpkper=(0, 0))
>>> recnames = zb.get_record_names()
```

get_volumetric_budget(modeltime, recarray=None, extrapolate_kper=False)

Method to generate a volumetric budget table based on flux information

Parameters

- **modeltime** (*flopy.discretization.ModelTime* object) - ModelTime object for calculating volumes
- **recarray** (*np.recarray*) - optional, user can pass in a numpy recarray to calculate volumetric budget. recarray must be pivoted before passing to get_volumetric_budget
- **extrapolate_kper** (*bool*) - flag to determine if we fill in data gaps with other timestep information from the same stress period. if True, we assume that flux is constant throughout a stress period and the pandas dataframe returned contains a volumetric budget per stress period
if False, calculates volumes from available flux data

Return type

pd.DataFrame

classmethod read_output(fname, net=False, dataframe=False, **kwargs)

Method to read a zonebudget output file into a recarray or pandas dataframe

Parameters

- **fname** (*str*) - zonebudget output file name
- **net** (*bool*) - boolean flag for net budget
- **dataframe** (*bool*) - boolean flag to return a pandas dataframe
- ****kwargs** - pivot : bool

start_datetime

[str] Datetime string indicating the time at which the simulation starts. Can be used when pandas dataframe is requested

timeunit

[str] String that indicates the time units used in the model.

Return type

np.recarray

classmethod `read_zone_file(fname)`

Method to read a zonebudget zone file into memory

Parameters

fname (*str*) - zone file name

Returns

zones

Return type

np.array

to_csv(fname)

Saves the budget record arrays to a formatted comma-separated values file.

Parameters

fname (*str*) - The name of the output comma-separated values file.

Return type

None

classmethod `write_zone_file(fname, array, fmtin=None, iprn=None)`

Saves a numpy array in a format readable by the zonebudget program executable.

File format: line 1: nlay, nrow, ncol line 2: INTERNAL (format) line 3:
begin data . . .

example from NACP: 19 250 500 INTERNAL (10I7) 199 199 199 199 199 199 199 199
199
199 ...

Parameters

- **array** (*array*) - The array of zones to be written.
- **fname** (*str*) - The path and name of the file to be written.
- **fmtin** (*int*) - The number of values to write to each line.
- **iprn** (*int*) - Padding space to add between each value.

class `ZoneBudget6(name='zonebud', model_ws='.', exe_name='zbud6', extension='.zbnam')`

Bases: `object`

Model class for building, editing and running MODFLOW 6 zonebudget

Parameters

- **name** (*str*) - model name for zonebudget
- **model_ws** (*str*) - path to model
- **exe_name** (*str*) - executable name
- **extension** (*str*) - name file extension

add_package(pkg_name, pkg)

Method to add a package to the ZoneBudget6 object

Parameters

- **pkg_name** (*str*) - three letter package abbreviation
- **pkg** (*str* or *object*) - either a package file name or package object

change_model_name(name)

Method to change the model name for writing a zonebudget model.

Parameters

name (*str*) - new model name

change_model_ws(model_ws)

Method to change the model ws for writing a zonebudget model.

Parameters

model_ws (*str*) - new model directory

export(f, ml, **kwargs)

Method to export a netcdf file, or add zonebudget output to an open netcdf file instance

Parameters

- **f** (*str*, *PathLike*, or *flopy.export.netcdf.NetCdf* object) - The file to export to
- **ml** (*flopy.modflow.Modflow* or *flopy.mf6.ModflowGwf* object) -
- ****kwargs** - **logger** : *flopy.export.netcdf.Logger* instance
masked_vals : list
list of values to mask

Return type

flopy.export.netcdf.NetCdf object

get_budget(f=None, names=None, zones=None, net=False, pivot=False)

Method to read and get zonebudget output

Parameters

- **f** (*str*) - zonebudget output file name
- **names** (*list of strings*) - A list of strings containing the names of the records desired.
- **zones** (*list of ints or strings*) - A list of integer zone numbers or zone names desired.
- **net** (*boolean*) - If True, returns net IN-OUT for each record.
- **pivot** (*bool*) - Method to pivot recordarray into a more user friendly method for working with data

Return type

np.recarray

get_dataframes(start_datetime=None, timeunit='D', index_key='totim', names=None, zones=None, net=False, pivot=False)

Get pandas dataframes.

Parameters

- **start_datetime** (*str*) - Datetime string indicating the time at which the simulation starts.
- **timeunit** (*str*) - String that indicates the time units used in the model.

- **index_key** (*str*) - Indicates the fields to be used (in addition to “record”) in the resulting DataFrame multi-index.
- **names** (*list of strings*) - A list of strings containing the names of the records desired.
- **zones** (*list of ints or strings*) - A list of integer zone numbers or zone names desired.
- **net** (*boolean*) - If True, returns net IN-OUT for each record.
- **pivot** (*bool*) - If True, returns data in a more user friendly fashion

Returns

df - Pandas DataFrame with the budget information.

Return type

Pandas DataFrame

Examples

```
>>> from flopy.utils.zonbud import ZoneBudget6
>>> zb6 = ZoneBudget6.load("my_nam_file", model_ws="my_model_ws")
>>> zb6.run_model()
>>> df = zb6.get_dataframes()
```

get_volumetric_budget(*modeltime*, *recarray=None*, *extrapolate_kper=False*)

Method to generate a volumetric budget table based on flux information

Parameters

- **modeltime** (*flopy.discretization.ModelTime object*) - ModelTime object for calculating volumes
- **recarray** (*np.recarray*) - optional, user can pass in a numpy recarray to calculate volumetric budget. recarray must be pivoted before passing to get_volumetric_budget
- **extrapolate_kper** (*bool*) - flag to determine if we fill in data gaps with other timestep information from the same stress period. if True, we assume that flux is constant throughout a stress period and the pandas dataframe returned contains a volumetric budget per stress period
if False, calculates volumes from available flux data

Return type

pd.DataFrame

static load(*nam_file*, *model_ws*: *str* | *PathLike* = '.')

Method to load a zonebudget model from namefile

Parameters

- **nam_file** (*str*) - zonebudget name file
- **model_ws** (*str* or *PathLike*, default ".") - model workspace path

Return type

ZoneBudget6 object

run_model(exe_name=None, nam_file=None, silent=False)

Method to run a zonebudget model

Parameters

- **exe_name** (*str*) - optional zonebudget executable name
- **nam_file** (*str*) - optional zonebudget name file name
- **silent** (*bool*) - optional flag to silence output

Return type

tuple

write_input(line_length=20)

Method to write a ZoneBudget 6 model to file

Parameters

line_length (*int*) - length of line for izeone array

class ZoneFile6(model, izeone, extension='.zon', aliases=None)

Bases: *object*

Class to build, read, write and edit MODFLOW 6 zonebudget zone files

Parameters

- **model** (*ZoneBudget6 object*) - model object
- **izeone** (*np.array*) - numpy array of zone numbers
- **extension** (*str*) - zone file extension name, defaults to ".zon"
- **aliases** (*dict*) - optional dictionary of zone aliases. ex. {1 : "nw_model"}

static load(f: *str* | *PathLike*, model)

Method to load a Zone file for zonebudget 6.

Parameter

f

[*str* or *PathLike*] zone file path

model

[*ZoneBudget6 object*] zonebudget 6 model object

rtype

ZoneFile6 object

property ncells

Method to get number of model cells

write_input(f=None, line_length=20)

Method to write the zonebudget 6 file

Parameters

- **f** (*str*) - zone file name
- **line_length** (*int*) - maximum length of line to write in izeone array

dataframe_to_netcdf_fmt(*df*, *zone_array*, *flux=True*)

Method to transform a volumetric zonebudget dataframe into array format for netcdf.

time is on axis 0 zone is on axis 1

Parameters

- **df** (*pd.DataFrame*) -
- **zone_array** (*np.ndarray*) - zonebudget zones array
- **flux** (*bool*) - boolean flag to indicate if budget data is a flux “L^3/T” (True, default) or if the data have been processed to volumetric values “L^3” (False)

Return type

ZBNetOutput object

sort_tuple(*tup*, *n=2*)

Sort a tuple by the first n values

tup: tuple

input tuple

n

[int] values to sort tuple by (default is 2)

Returns

tup - tuple sorted by the first n values

Return type

tuple

sum_flux_tuples(*fromzones*, *tozones*, *fluxes*)

8.3.2 Plotting Utilities

Contents:

flopy.plot.crosssection module

class PlotCrossSection(*model=None*, *modelgrid=None*, *ax=None*, *line=None*, *extent=None*, *geographic_coords=False*)

Bases: `object`

Class to create a cross sectional plot of a model.

Parameters

- **ax** (*matplotlib.pyplot axis*) - The plot axis. If not provided it, `plt.gca()` will be used.
- **model** (*flopy.modflow object*) - flopy model object. (Default is None)
- **modelgrid** (*flopy.discretization.Grid object*) - can be a StructuredGrid, VertexGrid, or UnstructuredGrid object
- **line** (*dict*) - Dictionary with either “row”, “column”, or “line” key. If key is “row” or “column” key value should be the zero-based row or column index for cross-section. If key is “line” value should be an array of (x, y) tuples with vertices of

cross-section. Vertices should be in map coordinates consistent with xul, yul, and rotation.

- **extent** (*tuple of floats*) - (xmin, xmax, ymin, ymax) will be used to specify axes limits. If None then these will be calculated based on grid, coordinates, and rotation.
- **geographic_coords** (*bool*) - boolean flag to allow the user to plot cross section lines in geographic coordinates. If False (default), cross section is plotted as the distance along the cross section line.

contour_array(a, masked_values=None, head=None, **kwargs)

Contour a two-dimensional array.

Parameters

- **a** (*numpy.ndarray*) - Three-dimensional array to plot.
- **masked_values** (*iterable of floats, ints*) - Values to mask.
- **head** (*numpy.ndarray*) - Three-dimensional array to set top of patches to the minimum of the top of a layer or the head value. Used to create patches that conform to water-level elevations.
- ****kwargs** (*dictionary*) - keyword arguments passed to matplotlib.pyplot.contour

Returns

contour_set

Return type

matplotlib.pyplot.contour

get_extent()

Get the extent of the rotated and offset grid

Returns

tuple

Return type

(xmin, xmax, ymin, ymax)

get_grid_line_collection(kwargs)**

Get a PatchCollection of the grid

Parameters

- ****kwargs** (*dictionary*) - keyword arguments passed to matplotlib.collections.LineCollection

Returns

PatchCollection

Return type

matplotlib.collections.LineCollection

get_grid_patch_collection(plotarray, projpts=None, fill_between=False, **kwargs)

Get a PatchCollection of plotarray in unmasked cells

Parameters

- **plotarray** (*numpy.ndarray*) - One-dimensional array to attach to the Patch Collection.

- **projpts** (*dict*) - dictionary defined by node number which contains model patch vertices.
- **fill_between** (*bool*) - flag to create polygons that mimic the matplotlib fill between method. Only used by the `plot_fill_between` method.
- ****kwargs** (*dictionary*) - keyword arguments passed to `matplotlib.collections.PatchCollection`

Returns
patches

Return type
`matplotlib.collections.PatchCollection`

plot_array(*a*, *masked_values=None*, *head=None*, ****kwargs**)

Plot a three-dimensional array as a patch collection.

Parameters

- **a** (*numpy.ndarray*) - Three-dimensional array to plot.
- **masked_values** (*iterable of floats, ints*) - Values to mask.
- **head** (*numpy.ndarray*) - Three-dimensional array to set top of patches to the minimum of the top of a layer or the head value. Used to create patches that conform to water-level elevations.
- ****kwargs** (*dictionary*) - keyword arguments passed to `matplotlib.collections.PatchCollection`

Returns
patches

Return type
`matplotlib.collections.PatchCollection`

plot_bc(*name=None*, *package=None*, *kper=0*, *color=None*, *head=None*, ****kwargs**)

Plot boundary conditions locations for a specific boundary type from a flopy model

Parameters

- **name** (*string*) - Package name string ('WEL', 'GHB', etc.). (Default is None)
- **package** (*flopy.modflow.Modflow package class instance*) - flopy package class instance. (Default is None)
- **kper** (*int*) - Stress period to plot
- **color** (*string*) - matplotlib color string. (Default is None)
- **head** (*numpy.ndarray*) - Three-dimensional array (structured grid) or Two-dimensional array (vertex grid) to set top of patches to the minimum of the top of a layer or the head value. Used to create patches that conform to water-level elevations.
- ****kwargs** (*dictionary*) - keyword arguments passed to `matplotlib.collections.PatchCollection`

Returns**patches****Return type**`matplotlib.collections.PatchCollection`

plot_endpoint(*ep*, *direction*='ending', *selection*=None, *selection_direction*=None, *method*='cell', *head*=None, ****kwargs**)

plot_fill_between(*a*, *colors*=('blue', 'red'), *masked_values*=None, *head*=None, ****kwargs**)

Plot a three-dimensional array as lines.

Parameters

- **a** (`numpy.ndarray`) - Three-dimensional array to plot.
- **colors** (`list`) - matplotlib fill colors, two required
- **masked_values** (*iterable of floats, ints*) - Values to mask.
- **head** (`numpy.ndarray`) - Three-dimensional array to set top of patches to the minimum of the top of a layer or the head value. Used to create patches that conform to water-level elevations.
- ****kwargs** (*dictionary*) - keyword arguments passed to `matplotlib.pyplot.plot`

Returns**plot****Return type**list containing `matplotlib.fillbetween` objects

plot_grid(****kwargs**)

Plot the grid lines.

Parameters

kwargs (*ax*, *colors*. The remaining *kwargs* are passed into the) - the `LineCollection` constructor.

Returns**lc****Return type**`matplotlib.collections.LineCollection`

plot_ibound(*ibound*=None, *color_noflow*='black', *color_ch*='blue', *color_vpt*='red', *head*=None, ****kwargs**)

Make a plot of *ibound*. If not specified, then pull *ibound* from the `self.model`

Parameters

- **ibound** (`numpy.ndarray`) - *ibound* array to plot. (Default is *ibound* in 'BAS6' package.)
- **color_noflow** (*string*) - (Default is 'black')
- **color_ch** (*string*) - Color for constant heads (Default is 'blue'.)
- **head** (`numpy.ndarray`) - Three-dimensional array to set top of patches to the minimum of the top of a layer or the head

value. Used to create patches that conform to water-level elevations.

- ****kwargs** (*dictionary*) - keyword arguments passed to `matplotlib.collections.PatchCollection`

Returns

patches

Return type

`matplotlib.collections.PatchCollection`

plot_inactive(*ibound=None, color_noflow='black', **kwargs*)

Make a plot of inactive cells. If not specified, then pull `ibound` from the `self.ml`

Parameters

- **ibound** (*numpy.ndarray*) - `ibound` array to plot. (Default is `ibound` in 'BAS6' package.)
- **color_noflow** (*string*) - (Default is 'black')

Returns

quadmesh

Return type

`matplotlib.collections.QuadMesh`

plot_pathline(*pl, travel_time=None, method='cell', head=None, **kwargs*)

Plot the MODPATH pathlines

Parameters

- **pl** (*list of rec arrays or a single rec array*) - `rec` array or list of `rec` arrays is data returned from `modpathfile.PathlineFile.get_data()` or `get_alldata()` methods. Data in `rec` array is 'x', 'y', 'z', 'time', 'k', and 'particleid'.
- **travel_time** (*float or str*) - `travel_time` is a travel time selection for the displayed pathlines. If a float is passed then pathlines with times less than or equal to the passed time are plotted. If a string is passed a variety logical constraints can be added in front of a time value to select pathlines for a select period of time. Valid logical constraints are `<=`, `<`, `==`, `>=`, and `>`. For example, to select all pathlines less than 10000 days `travel_time='< 10000'` would be passed to `plot_pathline`. (default is `None`)
- **method** (*str*) -
 - “cell” shows only pathlines that intersect with a cell
 - ”all” projects all pathlines onto the cross section regardless of whether they intersect with a given cell
- **head** (*np.ndarray*) - optional adjustment to only show pathlines that are `<=` to the top of the water table given a user supplied head array
- **kwargs** (*layer, ax, colors. The remaining kwargs are passed*) - into the `LineCollection` constructor.

Returns**lc****Return type**`matplotlib.collections.LineCollection`**plot_surface**(*a*, *masked_values=None*, ***kwargs*)

Plot a two- or three-dimensional array as line(s).

Parameters

- **a** (*numpy.ndarray*) - Two- or three-dimensional array to plot.
- **masked_values** (*iterable of floats, ints*) - Values to mask.
- ****kwargs** (*dictionary*) - keyword arguments passed to `matplotlib.pyplot.plot`

Returns**plot****Return type**list containing `matplotlib.plot` objects**plot_timeseries**(*ts*, *travel_time=None*, *method='cell'*, *head=None*, ***kwargs*)

Plot the MODPATH timeseries.

Parameters

- **ts** (*list of rec arrays or a single rec array*) - rec array or list of rec arrays is data returned from `modpathfile TimeseriesFile get_data()` or `get_alldata()` methods. Data in rec array is 'x', 'y', 'z', 'time', 'k', and 'particleid'.
- **travel_time** (*float or str*) - `travel_time` is a travel time selection for the displayed pathlines. If a float is passed then pathlines with times less than or equal to the passed time are plotted. If a string is passed a variety logical constraints can be added in front of a time value to select pathlines for a select period of time. Valid logical constraints are `<=`, `<`, `==`, `>=`, and `>`. For example, to select all pathlines less than 10000 days `travel_time='< 10000'` would be passed to `plot_pathline`. (default is None)
- **kwargs** (*layer, ax, colors. The remaining kwargs are passed*) - into the `LineCollection` constructor. If `layer='all'`, pathlines are output for all layers

Returns**lo****Return type**list of `Line2D` objects**plot_vector**(*vx*, *vy*, *vz*, *head=None*, *kstep=1*, *hstep=1*, *normalize=False*, *masked_values=None*, ***kwargs*)

Plot a vector.

Parameters

- **vx** (*np.ndarray*) - x component of the vector to be plotted (non-rotated) array shape must be (nlay, nrow, ncol) for

a structured grid array shape must be (nlay, ncpl) for a unstructured grid

- **vy** (*np.ndarray*) - y component of the vector to be plotted (non-rotated) array shape must be (nlay, nrow, ncol) for a structured grid array shape must be (nlay, ncpl) for a unstructured grid
- **vz** (*np.ndarray*) - y component of the vector to be plotted (non-rotated) array shape must be (nlay, nrow, ncol) for a structured grid array shape must be (nlay, ncpl) for a unstructured grid
- **head** (*numpy.ndarray*) - MODFLOW's head array. If not provided, then the quivers will be plotted in the cell center.
- **kstep** (*int*) - layer frequency to plot (default is 1)
- **hstep** (*int*) - horizontal frequency to plot (default is 1)
- **normalize** (*bool*) - boolean flag used to determine if vectors should be normalized using the vector magnitude in each cell (default is False)
- **masked_values** (*iterable of floats*) - values to mask
- **kwargs** (*matplotlib.pyplot keyword arguments for the*) - plt.quiver method

Returns

quiver - result of the quiver function

Return type

matplotlib.pyplot.quiver

property polygons

Method to return cached matplotlib polygons for a cross section

Returns

dict

Return type

[matplotlib.patches.Polygon]

set_zcentergrid(vs, kstep=1)

Get an array of z elevations at the center of a cell that is based on minimum of cell top elevation (self.elev) or passed vs numpy.ndarray

Parameters

- **vs** (*numpy.ndarray*) - Three-dimensional array to plot.
- **kstep** (*int*) - plotting layer interval

Returns

zcentergrid

Return type

numpy.ndarray

set_zpts(vs)

Get an array of projected vertices corrected with corrected elevations based on minimum of cell elevation (self.elev) or passed vs numpy.ndarray

Parameters

vs (*numpy.ndarray*) - Two-dimensional array to plot.

Returns

zpts

Return type

dict

flopy.plot.map module

class PlotMapView(*model=None, modelgrid=None, ax=None, layer=0, extent=None*)

Bases: *object*

Class to create a map of the model. Delegates plotting functionality based on model grid type.

Parameters

- **modelgrid** (*flopy.discretization.Grid*) - The modelgrid class can be StructuredGrid, VertexGrid, or UnstructuredGrid (Default is None)
- **ax** (*matplotlib.pyplot axis*) - The plot axis. If not provided it, plt.gca() will be used. If there is not a current axis then a new one will be created.
- **model** (*flopy.modflow object*) - flopy model object. (Default is None)
- **layer** (*int*) - Layer to plot. Default is 0. Must be between 0 and nlay - 1.
- **extent** (*tuple of floats*) - (xmin, xmax, ymin, ymax) will be used to specify axes limits. If None then these will be calculated based on grid, coordinates, and rotation.

Notes

contour_array(*a, masked_values=None, **kwargs*)

Contour an array. If the array is three-dimensional, then the method will contour the layer tied to this class (self.layer).

Parameters

- **a** (*numpy.ndarray*) - Array to plot.
- **masked_values** (*iterable of floats, ints*) - Values to mask.
- ****kwargs** (*dictionary*) - keyword arguments passed to matplotlib.pyplot.pcolormesh

Returns

contour_set

Return type

matplotlib.pyplot.contour

property extent

plot_array(a, masked_values=None, **kwargs)

Plot an array. If the array is three-dimensional, then the method will plot the layer tied to this class (self.layer).

Parameters

- **a** (*numpy.ndarray*) - Array to plot.
- **masked_values** (*iterable of floats, ints*) - Values to mask.
- ****kwargs** (*dictionary*) - keyword arguments passed to `matplotlib.pyplot.pcolormesh`

Returns

quadmsh - `matplotlib.collections.PatchCollection`

Return type

`matplotlib.collections.QuadMesh` or

plot_bc(name=None, package=None, kper=0, color=None, plotAll=False, **kwargs)

Plot boundary conditions locations for a specific boundary type from a flopy model

Parameters

- **name** (*string*) - Package name string ('WEL', 'GHB', etc.). (Default is None)
- **package** (*flopy.modflow.Modflow package class instance*) - flopy package class instance. (Default is None)
- **kper** (*int*) - Stress period to plot
- **color** (*string*) - matplotlib color string. (Default is None)
- **plotAll** (*bool*) - Boolean used to specify that boundary condition locations for all layers will be plotted on the current ModelMap layer. (Default is False)
- ****kwargs** (*dictionary*) - keyword arguments passed to `matplotlib.collections.PatchCollection`

Returns

quadmsh

Return type

`matplotlib.collections.QuadMesh`

plot_endpoint(ep, direction='ending', selection=None, selection_direction=None, **kwargs)

Plot the MODPATH endpoints.

Parameters

- **ep** (*rec array*) - A numpy recarray with the endpoint particle data from the MODPATH 6 endpoint file
- **direction** (*str*) - String defining if starting or ending particle locations should be considered. (default is 'ending')
- **selection** (*tuple*) - tuple that defines the zero-base layer, row, column location (l, r, c) to use to make a selection of particle endpoints. The selection could be a well location

to determine capture zone for the well. If selection is None, all particle endpoints for the user-sepcified direction will be plotted. (default is None)

- **selection_direction** (*str*) - String defining is a selection should be made on starting or ending particle locations. If selection is not None and selection_direction is None, the selection direction will be set to the opposite of direction. (default is None)
- **kwargs** (*ax, c, s or size, colorbar, colorbar_label, shrink. The*) - remaining kwargs are passed into the matplotlib scatter method. If colorbar is True a colorbar will be added to the plot. If colorbar_label is passed in and colorbar is True then colorbar_label will be passed to the colorbar set_label() method. If shrink is passed in and colorbar is True then the colorbar size will be set using shrink.

Returns

sp

Return type

matplotlib.pyplot.scatter

plot_grid(kwargs)**

Plot the grid lines.

Parameters

kwargs (*ax, colors. The remaining kwargs are passed into the*) - the LineCollection constructor.

Returns

lc

Return type

matplotlib.collections.LineCollection

plot_ibound(ibound=None, color_noflow='black', color_ch='blue', color_vpt='red', **kwargs)

Make a plot of ibound. If not specified, then pull ibound from the self.ml

Parameters

- **ibound** (*numpy.ndarray*) - ibound array to plot. (Default is ibound in the modelgrid)
- **color_noflow** (*string*) - (Default is 'black')
- **color_ch** (*string*) - Color for constant heads (Default is 'blue'.)
- **color_vpt** (*string*) - Color for vertical pass through cells (Default is 'red')

Returns

quadmesh

Return type

matplotlib.collections.QuadMesh

plot_inactive(*ibound=None*, *color_noflow='black'*, ***kwargs*)

Make a plot of inactive cells. If not specified, then pull *ibound* from the *self.ml*

Parameters

- **ibound** (*numpy.ndarray*) - *ibound* array to plot. (Default is *ibound* in 'BAS6' package.)
- **color_noflow** (*string*) - (Default is 'black')

Returns

quadmesh

Return type

matplotlib.collections.QuadMesh

plot_pathline(*pl*, *travel_time=None*, ***kwargs*)

Plot the MODPATH pathlines.

Parameters

- **pl** (*list of rec arrays or a single rec array*) - *rec* array or list of *rec* arrays is data returned from *modpathfile* *PathlineFile* *get_data()* or *get_alldata()* methods. Data in *rec* array is 'x', 'y', 'z', 'time', 'k', and 'particleid'.
- **travel_time** (*float or str*) - *travel_time* is a travel time selection for the displayed pathlines. If a float is passed then pathlines with times less than or equal to the passed time are plotted. If a string is passed a variety logical constraints can be added in front of a time value to select pathlines for a select period of time. Valid logical constraints are <=, <, ==, >=, and >. For example, to select all pathlines less than 10000 days *travel_time='< 10000'* would be passed to *plot_pathline*. (default is None)
- **kwargs** (*layer, ax, colors. The remaining kwargs are passed*) - into the *LineCollection* constructor. If *layer='all'*, pathlines are output for all layers

Returns

lc

Return type

matplotlib.collections.LineCollection

plot_shapefile(*shp*, ***kwargs*)

Plot a shapefile. The shapefile must be in the same coordinates as the rotated and offset grid.

Parameters

- **shp** (*str, os.PathLike or pyshp shapefile object*) - Path of the shapefile to plot
- **kwargs** (*dictionary*) - Keyword arguments passed to *plotutil.plot_shapefile()*

plot_shapes(*obj*, ***kwargs*)

Plot shapes is a method that facilitates plotting a collection of geospatial objects

Parameters

- **obj** (*collection object*) - obj can accept the following types
 str : shapefile path PathLike : shapefile path
 shapefile.Reader object list of [shapefile.Shape,
 shapefile.Shape,] shapefile.Shapes object
 flopy.utils.geometry.Collection object list
 of [flopy.utils.geometry, ...] objects
 geojson.GeometryCollection object geojson.FeatureCollection
 object shapely.GeometryCollection object list of [[vertices],
 ...]
- **kwargs** (*dictionary*) - keyword arguments passed to
 plotutil.plot_shapefile()

Return type

matplotlib.Collection object

plot_timeseries(ts, travel_time=None, **kwargs)

Plot the MODPATH timeseries.

Parameters

- **ts** (*list of rec arrays or a single rec array*) - rec array
 or list of rec arrays is data returned from modpathfile
 TimeseriesFile get_data() or get_alldata() methods. Data in
 rec array is 'x', 'y', 'z', 'time', 'k', and 'particleid'.
- **travel_time** (*float or str*) - travel_time is a travel time
 selection for the displayed pathlines. If a float is
 passed then pathlines with times less than or equal to the
 passed time are plotted. If a string is passed a variety
 logical constraints can be added in front of a time value to
 select pathlines for a select period of time. Valid logical
 constraints are <=, <, ==, >=, and >. For example, to select
 all pathlines less than 10000 days travel_time='< 10000' would
 be passed to plot_pathline. (default is None)
- **kwargs** (*layer, ax, colors. The remaining kwargs are passed*)
 - into the LineCollection constructor. If layer='all',
 pathlines are output for all layers

Returns

lo

Return type

list of Line2D objects

plot_vector(vx, vy, istep=1, jstep=1, normalize=False, masked_values=None, **kwargs)

Plot a vector.

Parameters

- **vx** (*np.ndarray*) - x component of the vector to be plotted
 (non-rotated) array shape must be (nlay, nrow, ncol) for
 a structured grid array shape must be (nlay, ncpl) for a
 unstructured grid
- **vy** (*np.ndarray*) - y component of the vector to be plotted
 (non-rotated) array shape must be (nlay, nrow, ncol) for

- a structured grid array shape must be (nlay, ncpl) for a unstructured grid
- **istep** (*int*) - row frequency to plot (default is 1)
- **jstep** (*int*) - column frequency to plot (default is 1)
- **normalize** (*bool*) - boolean flag used to determine if vectors should be normalized using the vector magnitude in each cell (default is False)
- **masked_values** (*iterable of floats*) - values to mask
- **kwargs** (*matplotlib.pyplot keyword arguments for the*) - plt.quiver method

Returns

quiver - result of the quiver function

Return type

matplotlib.pyplot.quiver

flopy.plot.plotutil module

Module containing helper functions for plotting model data using ModelMap and ModelCrossSection. Functions for plotting shapefiles are also included.

class PlotUtilities

Bases: `object`

Class which groups a collection of plotting utilities which Flopy and Flopy6 can use to generate map based plots

static **centered_specific_discharge**(*Qx, Qy, Qz, delr, delc, sat_thk*)

DEPRECATED. Use postprocessing.get_specific_discharge() instead.

Using the MODFLOW discharge, calculate the cell centered specific discharge by dividing by the flow width and then averaging to the cell center.

Parameters

- **Qx** (*numpy.ndarray*) - MODFLOW 'flow right face'
- **Qy** (*numpy.ndarray*) - MODFLOW 'flow front face'. The sign on this array will be flipped by this function so that the y axis is positive to north.
- **Qz** (*numpy.ndarray*) - MODFLOW 'flow lower face'. The sign on this array will be flipped by this function so that the z axis is positive in the upward direction.
- **delr** (*numpy.ndarray*) - MODFLOW delr array
- **delc** (*numpy.ndarray*) - MODFLOW delc array
- **sat_thk** (*numpy.ndarray*) - Saturated thickness for each cell

Returns

(**qx, qy, qz**) - Specific discharge arrays that have been interpolated to cell centers.

Return type

`tuple` of `numpy.ndarrays`

static `saturated_thickness(head, top, botm, laytyp, mask_values=None)`

Calculate the saturated thickness.

Parameters

- **head** (*numpy.ndarray*) - head array
- **top** (*numpy.ndarray*) - top array of shape (nrow, ncol)
- **botm** (*numpy.ndarray*) - botm array of shape (nlay, nrow, ncol)
- **laytyp** (*numpy.ndarray*) - confined (0) or convertible (1) of shape (nlay)
- **mask_values** (*list of floats*) - If head is one of these values, then set sat to top - bot

Returns

sat_thk - Saturated thickness of shape (nlay, nrow, ncol).

Return type

numpy.ndarray

class `SwiConcentration(model=None, botm=None, istrat=1, nu=None)`

Bases: *object*

The `binary_header` class is a class to create headers for MODFLOW binary files

calc_conc(zeta, layer=None)

Calculate concentrations for a given time step using passed zeta.

Parameters

- **zeta** (*dictionary of numpy arrays*) - Dictionary of zeta results. zeta keys are zero-based zeta surfaces.
- **layer** (*int*) - Concentration will be calculated for the specified layer. If layer is None, then the concentration will be calculated for all layers. (default is None).

Returns

conc - Calculated concentration.

Return type

numpy array

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('test')
>>> c = flopy.plot.SwiConcentration(model=m)
>>> conc = c.calc_conc(z, layer=0)
```

class `UnstructuredPlotUtilities`

Bases: *object*

Collection of unstructured grid and vertex grid compatible plotting helper functions

static `arctan2(verts, reverse=False)`

Reads 2 dimensional set of verts and orders them using the arctan 2 method

Parameters

verts (*np.array of floats*) - Nx2 array of verts

Returns

verts - Nx2 array of verts

Return type

np.array of float

static `irregular_shape_patch(xverts, yverts=None)`

Patch for vertex cross-section plotting when we have an irregular shape type throughout the model grid or multiple shape types. This method is also used by the model splitter as a helper function for remapping the cell2d array.

Parameters

- **xverts** (*list*) - xvertices
- **yverts** (*list or None*) - yvertices

Return type

xverts, yverts as *np.ndarray*

static `line_intersect_grid(ptsin, xgrid, ygrid)`

Uses cross product method to find which cells intersect with the line and then uses the parameterized line equation to calculate intersection x, y vertex points. Should be quite fast for large model grids!

Parameters

- **pts** (*list*) - list of tuple line vertex pairs (ex. [(1, 0), (10, 0)])
- **xgrid** (*np.array*) - model grid x vertices
- **ygrid** (*np.array*) - model grid y vertices

Returns

vdict

Return type

dict of cell vertices

advanced_package_bc_helper(pkg, modelgrid, kper)

Helper function for plotting boundary conditions from “advanced” packages

Parameters

- **pkg** (*flopy Package objects*) -
- **modelgrid** (*flopy.discretization.Grid object*) -

cvfd_to_patch_collection(verts, iverts)

Create a patch collection from control volume vertices and incidence list

Parameters

- **verts** (*ndarray*) - 2d array of x and y points.
- **iverts** (*list of lists*) - should be of len(ncells) with a list of vertex numbers for each cell

filter_modpath_by_travel_time(recarray, travel_time)

Helper method for filtering particles by travel time. Used in modpath plotting routines

Parameters

- **recarray** (*np.recarray*) - recarray of modpath particle information
- **travel_time** (*str*, *float*) - travel time logical argument to filter modpath output

Return type

np.recarray

intersect_modpath_with_crosssection(recarrays, projpts, xvertices, yvertices, projection, ncpl, method='cell', starting=False)

Method to intersect modpath output with a cross-section

Parameters

- **recarrays** (*list*) - list of numpy recarrays
- **projpts** (*dict*) - dict of crosssectional cell vertices
- **xvertices** (*np.array*) - array of modelgrid xvertices
- **yvertices** (*np.array*) - array of modelgrid yvertices
- **projection** (*str*) - projection direction (x or y)
- **ncpl** (*int*) - number of cells per layer (cross sectional version)
- **method** (*str*) - intersection method ('cell' or 'all')
- **starting** (*bool*) - modpath starting location flag

Returns

dict

Return type

dictionary of intersecting recarrays

parse_modpath_selection_options(ep, direction, selection, selection_direction)

Returns

plot_cvfd(verts, iverts, ax=None, layer=0, cmap='Dark2', edgecolor='scaled', facecolor='scaled', a=None, masked_values=None, **kwargs)

Generic function for plotting a control volume finite difference grid of information.

Parameters

- **verts** (*ndarray*) - 2d array of x and y points.
- **iverts** (*list of lists*) - should be of len(ncells) with a list of vertex number for each cell
- **ax** (*matplotlib.pyplot axis*) - matplotlib.pyplot axis instance. Default is None
- **layer** (*int*) - layer to extract. Used in combination to the optional ncpl parameter. Default is 0
- **cmap** (*string*) - Name of colormap to use for polygon shading (default is 'Dark2')

- **edgecolor** (*string*) - Color name. (Default is 'scaled' to scale the edge colors.)
- **facecolor** (*string*) - Color name. (Default is 'scaled' to scale the face colors.)
- **a** (*numpy.ndarray*) - Array to plot.
- **masked_values** (*iterable of floats, ints*) - Values to mask.
- **kwargs** (*dictionary*) - Keyword arguments that are passed to `PatchCollection.set(**kwargs)`. Some common kwargs would be 'linewidths', 'linestyles', 'alpha', etc.

Returns

`pc`

Return type

`matplotlib.collections.PatchCollection`

Examples

```
plot_shapefile(shp, ax=None, radius=500.0, cmap='Dark2', edgecolor='scaled',
               facecolor='scaled', a=None, masked_values=None, idx=None, **kwargs)
```

Generic function for plotting a shapefile.

Parameters

- **shp** (*string or os.PathLike*) - Path of the shapefile to plot.
- **ax** (*matplotlib.pyplot.axes object*) -
- **radius** (*float*) - Radius of circle for points. (Default is 500.)
- **cmap** (*string*) - Name of colormap to use for polygon shading (default is 'Dark2')
- **edgecolor** (*string*) - Color name. (Default is 'scaled' to scale the edge colors.)
- **facecolor** (*string*) - Color name. (Default is 'scaled' to scale the face colors.)
- **a** (*numpy.ndarray*) - Array to plot.
- **masked_values** (*iterable of floats, ints*) - Values to mask.
- **idx** (*iterable int*) - A list or array that contains shape numbers to include in the patch collection. Return all shapes if not specified.
- **kwargs** (*dictionary*) - Keyword arguments that are passed to `PatchCollection.set(**kwargs)`. Some common kwargs would be 'linewidths', 'linestyles', 'alpha', etc.

Returns

`pc`

Return type

`matplotlib.collections.PatchCollection`

Examples

```
reproject_modpath_to_crosssection(idict, projpts, xypts, projection, modelgrid, ncpl,
                                  geographic_coords, starting=False)
```

Method to reproject modpath points onto cross sectional line

Parameters

- **idict** (*dict*) - dictionary of intersecting points
- **projpts** (*dict*) - dictionary of cross sectional cells
- **xypts** (*dict*) - dictionary of cross sectional line
- **projection** (*str*) - projection direction (x or y)
- **modelgrid** (*Grid object*) - flopy modelgrid object
- **ncpl** (*int*) - number of cells per layer (cross sectional version)
- **geographic_coords** (*bool*) - flag for plotting in geographic coordinates
- **starting** (*bool*) - flag for modpath position

Return type

dictionary of projected modpath lines or points

```
shapefile_extents(shp)
```

Determine the extents of a shapefile

Parameters

shp (*string*) - Name of the shapefile to convert to a PatchCollection.

Returns

extents - tuple with xmin, xmax, ymin, ymax from shapefile.

Return type

tuple

Examples

```
>>> import flopy
>>> fshp = 'myshapefile'
>>> extent = flopy.plot.plotutil.shapefile_extents(fshp)
```

```
shapefile_get_vertices(shp)
```

Get vertices for the features in a shapefile

Parameters

shp (*string*) - Name of the shapefile to extract shapefile feature vertices.

Returns

vertices - Vertices is a list with vertices for each feature in the shapefile. Individual feature vertices are x, y tuples and contained in a list. A list with a single x, y tuple is returned for point shapefiles. A list with multiple x, y tuples is returned for polyline and polygon shapefiles.

Return type

list

Examples

```
>>> import flopy
>>> fshp = 'myshapefile'
>>> lines = flopy.plot.plotutil.shapefile_get_vertices(fshp)
```

shapefile_to_patch_collection(shp: *str* | *PathLike*, radius=500.0, idx=None)

Create a patch collection from the shapes in a shapefile

Parameters

- **shp** (*str* or *PathLike*) - Name of the shapefile to convert to a PatchCollection.
- **radius** (*float*) - Radius of circle for points in the shapefile. (Default is 500.)
- **idx** (*iterable int*) - A list or array that contains shape numbers to include in the patch collection. Return all shapes if not specified.

Returns

pc - Patch collection of shapes in the shapefile

Return type

matplotlib.collections.PatchCollection

flopy.plot.styles module

class styles

Bases: *object*

Styles class for custom matplotlib styling

The class contains both custom styles and plotting methods for custom formatting using a specific matplotlib style

Additional styles can be easily added to the mplstyle folder and accessed using the plt.style.context() method.

classmethod USGSMap()

classmethod USGSPlot()

classmethod add_annotation(ax=None, text="", xy=None, xytext=None, bold=True, italic=True, fontsize=9, ha='left', va='bottom', **kwargs)

Add an annotation to a axis object

Parameters

- **ax** (*axis object*) - matplotlib axis object (default is None)
- **text** (*str*) - text string
- **xy** (*tuple*) - tuple with the location of the annotation (default is None)
- **xytext** (*tuple*) - tuple with the location of the text
- **bold** (*bool*) - boolean indicating if bold font (default is True)

- **italic** (*bool*) - boolean indicating if italic font (default is True)
- **fontsize** (*int*) - font size (default is 9 points)
- **ha** (*str*) - matplotlib horizontal alignment keyword (default is left)
- **va** (*str*) - matplotlib vertical alignment keyword (default is bottom)
- **kwargs** (*dict*) - dictionary with valid matplotlib annotation object keywords

Returns

ann_obj - matplotlib annotation object

Return type

object

```
classmethod add_text(ax=None, text="", x=0.0, y=0.0, transform=True, bold=True,
                    italic=True, fontsize=9, ha='left', va='bottom', **kwargs)
```

Add USGS-style text to a axis object

Parameters

- **ax** (*axis object*) - matplotlib axis object (default is None)
- **text** (*str*) - text string
- **x** (*float*) - x-location of text string (default is 0.)
- **y** (*float*) - y-location of text string (default is 0.)
- **transform** (*bool*) - boolean that determines if a transformed (True) or data (False) coordinate system is used to define the (x, y) location of the text string (default is True)
- **bold** (*bool*) - boolean indicating if bold font (default is True)
- **italic** (*bool*) - boolean indicating if italic font (default is True)
- **fontsize** (*int*) - font size (default is 9 points)
- **ha** (*str*) - matplotlib horizontal alignment keyword (default is left)
- **va** (*str*) - matplotlib vertical alignment keyword (default is bottom)
- **kwargs** (*dict*) - dictionary with valid matplotlib text object keywords

Returns

text_obj - matplotlib text object

Return type

object

```
classmethod graph_legend(ax=None, handles=None, labels=None, **kwargs)
```

Add a USGS-style legend to a matplotlib axis object

Parameters

- **ax** (*axis object*) - matplotlib axis object (default is None)
- **handles** (*list*) - list of legend handles
- **labels** (*list*) - list of labels for legend handles
- **kwargs** (*kwargs*) - matplotlib legend kwargs

Returns

leg - matplotlib legend object

Return type

object

classmethod graph_legend_title(*leg, title=None, fontsize=9*)

Set the legend title for a matplotlib legend object

Parameters

- **leg** (*legend object*) - matplotlib legend object
- **title** (*str*) - title for legend
- **fontsize** (*int*) - fontsize for legend

Returns

leg - matplotlib legend object

Return type

object

classmethod heading(*ax=None, letter=None, heading=None, x=0.0, y=1.01, idx=None, fontsize=9*)

Add a USGS-style heading to a matplotlib axis object

Parameters

- **ax** (*axis object*) - matplotlib axis object (default is None)
- **letter** (*str*) - string that defines the subplot (A, B, C, etc.)
- **heading** (*str*) - text string
- **x** (*float*) - location of the heading in the x-direction in normalized plot dimensions ranging from 0 to 1 (default is 0.00)
- **y** (*float*) - location of the heading in the y-direction in normalized plot dimensions ranging from 0 to 1 (default is 1.01)
- **idx** (*int*) - index for programatically generating the heading letter when letter is None and idx is not None. `idx = 0` will generate A (default is None)

Returns

text - matplotlib text object

Return type

object

classmethod remove_edge_ticks(*ax=None*)

Remove unnecessary ticks on the edges of the plot

Parameters

ax (*axis object*) - matplotlib axis object (default is None)

Returns

ax - matplotlib axis object

Return type

axis object

classmethod **set_font_type**(*family*, *fontname*)

Method to set the matplotlib font type for the current style

Note: this method only works when adding text using the styles methods.

Parameters

- **family** (*str*) - matplotlib.rcParams font.family
- **font** (*str*) - matplotlib.rcParams font.fontname

Return type

None

classmethod **xlabel**(*ax=None*, *label=""*, *bold=False*, *italic=False*, ***kwargs*)

Method to set the xlabel using the styled fontdict

Parameters

- **ax** (*axis object*) - matplotlib axis object (default is None)
- **label** (*str*) - axis label for the chart
- **bold** (*bool*) - flag to switch to boldface text
- **italic** (*bool*) - flag to use italic text
- **kwargs** (*dict*) - keyword arguments for the matplotlib set_xlabel method

Return type

None

classmethod **ylabel**(*ax=None*, *label=""*, *bold=False*, *italic=False*, ***kwargs*)

Method to set the ylabel using the styled fontdict

Parameters

- **ax** (*axis object*) - matplotlib axis object (default is None)
- **label** (*str*) - axis label for the chart
- **bold** (*bool*) - flag to switch to boldface text
- **italic** (*bool*) - flag to use italic text
- **kwargs** (*dict*) - keyword arguments for the matplotlib set_xlabel method

Return type

None

8.3.3 Export Utilities

Contents:

flopy.export.longnames module

Human readable long names for netCDF variables.

flopy.export.metadata module

class `acdd(sciencebase_id, model)`

Bases: `object`

Translate ScienceBase global metadata attributes to CF and ACDD global attributes.

Parameters

- **sciencebase_id** (*str*) - Unique identifier for ScienceBase record (e.g. 582da7efe4b04d580bd37be8)
- **model** (*flopy model object*) - Model object

References

<https://www.sciencebase.gov/catalog/> <https://cfconventions.org/cf-conventions/v1.6.0/cf-conventions.html#description-of-file-contents> https://wiki.esipfed.org/Attribute_Convention_for_Data_Discovery

property `bounds`

property `creator`

property `creator_url`

property `geospatial_bounds`

Describes the data's 2D or 3D geospatial extent in OGC's Well-Known Text (WKT) Geometry format

property `geospatial_bounds_vertical_crs`

The vertical coordinate reference system (CRS) for the Z axis of the point coordinates in the `geospatial_bounds` attribute.

get_sciencebase_metadata(*id*)

Gets metadata json text for given ID from sciencebase.gov; loads into python dictionary. Fetches the reference text using the url: <https://www.sciencebase.gov/catalog/item/<ID>?format=json>

Parameters

ID (*str*) - ScienceBase ID string; e.g. 582da7efe4b04d580bd37be8 for Dane County Model

Returns

metadata - Dictionary of metadata

Return type

`dict`

get_sciencebase_xml_metadata()

Gets xml from sciencebase.gov, using XML url obtained from json using get_sciencebase_metadata().

Parameters

ID (*str*) - ScienceBase ID string; e.g. 582da7efe4b04d580bd37be8 for Dane County Model

Returns

metadata - Dictionary of metadata

Return type

dict

property references**property time_coverage****property vertical_datum**

Try to parse the vertical datum from the xml info

property xmlfile**property xmlroot**

ElementTree root element object for xml metadata

flopy.export.netcdf module**class Logger(filename, echo=False)**

Bases: *object*

Basic class for logging events during the linear analysis calculations if filename is passed, then an file handle is opened

Parameters

filename (*bool* or *string*) - if string, it is the log file to write. If a bool, then log is written to the screen. **echo** (*bool*): a flag to force screen output

items

tracks when something is started. If a log entry is not in items, then it is treated as a new entry with the string being the key and the datetime as the value. If a log entry is in items, then the end time and delta time are written and the item is popped from the keys

Type

dict

log(phrase)

log something that happened

Parameters

phrase (*str*) - the thing that happened

warn(message)

Write a warning to the log file

Parameters

message (*str*) - the warning text

```
class NetCdf(output_filename: str | PathLike, model, time_values=None, z_positive='up',
            verbose=None, prj=None, logger=None, forgive=False, **kwargs)
```

Bases: `object`

Support for writing a netCDF4 compliant file from a flopy model

Parameters

- **output_filename** (*str or PathLike*) - Path of the .nc file to write
- **model** (*flopy model instance*) -
- **time_values** (*the entries for the time dimension*) - if not None, the constructor will initialize the file. If None, the perlen array of ModflowDis will be used
- **z_positive** (*str ('up' or 'down')*) - Positive direction of vertical coordinates written to NetCDF file. (default 'down')
- **verbose** (*if True, stdout is verbose. If str, then a log file*) - is written to the verbose file
- **prj** (*str, optional, default None*) - PROJ4 string
- **logger** (*Logger, optional, default None*) - Logging object for custom logging configuration
- **forgive** (*what to do if a duplicate variable name is being created. If*) - True, then the newly requested var is skipped. If False, then an exception is raised.
- ****kwargs** (*keyword arguments*) -

modelgrid

[flopy.discretization.Grid instance] user supplied model grid which will be used in lieu of the model object modelgrid for netcdf production

Notes

This class relies heavily on the grid and modeltime objects, including these attributes: `lenuni`, `itmuni`, `start_datetime`, and `proj4`. Make sure these attributes have meaningful values.

add_global_attributes(attr_dict)

add global attribute to an initialized file

Parameters

attr_dict (*dict(attribute name, attribute value)*) -

Return type

None

Raises

- **Exception** of `self.nc` is None (`initialize_file()` -
- has not been called) -

add_sciencebase_metadata(id, check=True)

Add metadata from ScienceBase using the `flopy.export.metadata.acdd` class.

Returns

metadata

Return type

flopy.export.metadata.acdd object

append(other, suffix='_1')

copy(output_filename)

create_group_variable(group, name, attributes, precision_str, dimensions=('time',))

Create a new group variable in the netcdf object

Parameters

- **name** (*str*) - the name of the variable
- **attributes** (*dict*) - attributes to add to the new variable
- **precision_str** (*str*) - netcdf-compliant string. e.g. f4
- **dimensions** (*tuple*) - which dimensions the variable applies to
default : ("time", "layer", "x", "y")
- **group** (*str*) - which netcdf group the variable goes in default
: None which creates the variable in root

Return type

nc variable

Raises

- **AssertionError** if precision_str not right -
- **AssertionError** if variable name already in netcdf object -
- **AssertionError** if one of more dimensions do not exist -

create_variable(name, attributes, precision_str='f4', dimensions=('time', 'layer'), group=None)

Create a new variable in the netcdf object

Parameters

- **name** (*str*) - the name of the variable
- **attributes** (*dict*) - attributes to add to the new variable
- **precision_str** (*str*) - netcdf-compliant string. e.g. f4
- **dimensions** (*tuple*) - which dimensions the variable applies to
default : ("time", "layer", "x", "y")
- **group** (*str*) - which netcdf group the variable goes in default
: None which creates the variable in root

Return type

nc variable

Raises

- **AssertionError** if precision_str not right -
- **AssertionError** if variable name already in netcdf object -
- **AssertionError** if one of more dimensions do not exist -

difference(*other*, *minuend*='self', *mask_zero_diff*=True, *onlydiff*=True)

make a new NetCDF instance that is the difference with another netcdf file

Parameters

- **other** (*either an str filename of a netcdf file or*) - a netCDF4 instance
- **minuend** (*((optional) the order of the difference operation.*) - Default is self (e.g. self - other). Can be "self" or "other"
- **mask_zero_diff** (*bool flag to mask differences that are zero. If*) - True, positions in the difference array that are zero will be set to self.fillvalue
- **only_diff** (*bool flag to only add non-zero diffs to output file*) -

Return type

net NetCDF instance

Notes

assumes the current NetCDF instance has been populated. The variable names and dimensions between the two files must match exactly. The name of the new .nc file is <self.output_filename>.diff.nc. The masks from both self and other are carried through to the new instance

classmethod **empty_like**(*other*, *output_filename*=None, *verbose*=None, *logger*=None)

get_longnames_from_docstrings(*outfile*='longnames.py')

This is experimental.

Scrape Flopy module docstrings and return docstrings for parameters included in the list of variables added to NetCdf object. Create a dictionary of longnames keyed by the NetCdf variable names; make each longname from the first sentence of the docstring for that parameter.

One major limitation is that variables from mflists often aren't described in the docstrings.

initialize_file(*time_values*=None)

initialize the netcdf instance, including global attributes, dimensions, and grid information

Parameters

- time_values** (*list of times to use as time dimension*) - entries. If none, then use the times in self.model.dis.perlen and self.start_datetime

initialize_geometry()

initialize the geometric information needed for the netcdf file

initialize_group(*group*='timeseries', *dimensions*=('time',), *attributes*=None, *dimension_data*=None)

Method to initialize a new group within a netcdf file. This group can have independent dimensions from the global dimensions

Parameters:

name
[str] name of the netcdf group

dimensions
[tuple] data dimension names for group

dimension_shape
[tuple] tuple of data dimension lengths

attributes
[dict] nested dictionary of {dimension : {attributes}} for each netcdf group dimension

dimension_data
[dict] dictionary of {dimension : [data]} for each netcdf group dimension

property nc_crs

static normalize_name(name)

write()
write the nc object to disk

classmethod zeros_like(other, output_filename=None, verbose=None, logger=None)

flopy.export.shapefile_utils module

Module for exporting and importing flopy model attributes

enforce_10ch_limit(names)

Enforce 10 character limit for fieldnames. Add suffix for duplicate names starting at 0.

Parameters

names (*list of strings*) -

Returns

list of unique strings of len <= 10.

Return type

list

get_pyshp_field_dtypes(code)

Returns a numpy dtype for a pyshp field type.

get_pyshp_field_info(dtypename)

Get pyshp dtype information for a given numpy dtype.

model_attributes_to_shapefile(path: *str* | *PathLike*, ml, package_names=None, array_dict=None, verbose=False, **kwargs)

Wrapper function for writing a shapefile of model data. If package_names is not None, then search through the requested packages looking for arrays that can be added to the shapefile as attributes

Parameters

- **path** (*str* or *PathLike*) - path to write the shapefile to

- **ml** (*flopy.mbase*) - model instance
- **package_names** (*list of package names (e.g. ["dis", "lpf"])*) - Packages to export data arrays to shapefile. (default is None)
- **array_dict** (*dict of {name:2D array} pairs*) - Additional 2D arrays to add as attributes to the shapefile. (default is None)
- **verbose** (*bool, optional, default False*) - whether to print verbose output
- ****kwargs** (*keyword arguments*) -

modelgrid

[fp.modflow.Grid object] if modelgrid is supplied, user supplied modelgrid is used in lieu of the modelgrid attached to the modflow model object

crs

[pyproj.CRS, optional if *prjfile* is specified] Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg "EPSG:26916") or a WKT string.

prjfile

[str or pathlike, optional if *crs* is specified] ESRI-style projection file with well-known text defining the CRS for the model grid (must be projected; geographic CRS are not supported).

Return type

None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> flopy.utils.model_attributes_to_shapefile('model.shp', m)
```

recarray2shp(*recarray*, *geoms*, *shpname*: *str* | *PathLike* = 'recarray.shp', *mg*=None, *crs*=None, *prjfile*=None, *epsg*=None, *prj*: *str* | *PathLike* | None = None, *verbose*=False, ***kwargs*)

Write a numpy record array to a shapefile, using a corresponding list of geometries. Method supports list of flopy geometry objects, flopy Collection object, shapely Collection object, and geojson Geometry Collection objects

Parameters

- **recarray** (*np.recarray*) - Numpy record array with attribute information that will go in the shapefile
- **geoms** (*list of flopy.utils.geometry, shapely geometry collection,)* -
 flopy geometry collection, shapefile.Shapes,
 list of shapefile.Shape objects, or geojson geometry
 collection

The number of geometries in geoms must equal the number of records in recarray.

- **shpname** (*str* or *PathLike*, default "recarray.shp") - Path for the output shapefile
- **crs** (pyproj.CRS, optional if *prjfile* is specified) - Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg "EPSG:26916") or a WKT string.
- **prjfile** (str or pathlike, optional if *crs* is specified) - ESRI-style projection file with well-known text defining the CRS for the model grid (must be projected; geographic CRS are not supported).

Notes

Uses pyshp.

shape_attr_name(name, length=6, keep_layer=False)

Function for to format an array name to a maximum of 10 characters to conform with ESRI shapefile maximum attribute name length

Parameters

- **name** (*str*) - data array name
- **length** (*int*) - maximum length of string to return. Value passed to function is overridden and set to 10 if keep_layer=True. (default is 6)
- **keep_layer** (*bool*) - Boolean that determines if layer number in name should be retained. (default is False)

Return type

str

Examples

```
>>> import flopy
>>> name = flopy.utils.shape_attr_name('averylongstring')
>>> name
>>> 'averyl'
```

shp2recarray(shpname: *str* | *PathLike*)

Read a shapefile into a numpy recarray.

Parameters

shpname (*str* or *PathLike*) - ESRI Shapefile path

Return type

`np.recarray`

write_grid_shapefile(path: *str* | *PathLike*, mg, array_dict, nan_val=nan, crs=None, prjfile=None, epsg=None, prj: *str* | *PathLike* | *None* = None, verbose=False)

Method to write a shapefile of gridded input data

Parameters

- **path** (*str* or *PathLike*) - shapefile file path
- **mg** (*flopy.discretization.Grid* object) - flopy model grid
- **array_dict** (*dict*) - dictionary of model input arrays
- **nan_val** (*float*) - value to fill nans
- **crs** (*pyproj.CRS*, optional if *prjfile* is specified) - Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by *pyproj.CRS.from_user_input()*, such as an authority string (eg “EPSG:26916”) or a WKT string.
- **prjfile** (*str* or *pathlike*, optional if *crs* is specified) - ESRI-style projection file with well-known text defining the CRS for the model grid (must be projected; geographic CRS are not supported).

Return type

None

write_gridlines_shapefile(*filename: str | PathLike, mg*)

Write a polyline shapefile of the grid lines - a lightweight alternative to polygons.

Parameters

- **filename** (*str* or *PathLike*) - path of the shapefile to write
- **mg** (*model grid*) -

Return type

None

write_prj(*shpname, modelgrid=None, crs=None, epsg=None, prj=None, prjfile=None, wkt_string=None*)

flopy.export.unitsformat module

unit format strings for netCDF variables.

flopy.export.utils module

array2d_export(*f: str | PathLike, u2d, fmt=None, verbose=False, **kwargs*)

export helper for Util2d instances

Parameters

- **f** (*str* or *PathLike*) - filename or existing export instance type (NetCdf only for now)
- **u2d** (*Util2d* instance) -
- **fmt** (*str*) - output format flag. ‘vtk’ will export to vtk
- **verbose** (*bool*) - whether to print verbose output

- ****kwargs** (keyword arguments) - min_valid : minimum valid value max_valid : maximum valid value modelgrid : flopy.discretization.Grid

model grid instance which will supercede the
flopy.model.modelgrid

if fmt is set to 'vtk', parameters of vtk.export_array

array3d_export(f: *str* | *PathLike*, u3d, fmt=None, **kwargs)

export helper for Transient2d instances

Parameters

- **f** (*str* or *PathLike*) - filename or existing export instance type (NetCdf only for now)
- **u3d** (*Util3d* instance) -
- **fmt** (*str*) - output format flag. 'vtk' will export to vtk
- ****kwargs** (keyword arguments) - min_valid : minimum valid value max_valid : maximum valid value modelgrid : flopy.discretization.Grid

model grid instance which will supercede the
flopy.model.modelgrid

if fmt is set to 'vtk', parameters of vtk.export_array

contour_array(modelgrid, ax, a, **kwargs)

Create a QuadMesh plot of the specified array using pcolormesh

Parameters

- **modelgrid** (*flopy.discretization.Grid* object) - modelgrid object
- **ax** (*matplotlib.axes.Axes*) - ax to add the contours
- **a** (*np.ndarray*) - array to contour

Returns

contour_set

Return type

ContourSet

ensemble_helper(inputs_filename: *str* | *PathLike*, outputs_filename: *str* | *PathLike*, models, add_reals=True, **kwargs)

Helper to export an ensemble of model instances. Assumes all models have same dis and reference information, only difference is properties and boundary conditions. Assumes model.nam.split('_')[-1] is the realization suffix to use in the netcdf variable names

export_array(modelgrid, filename: *str* | *PathLike*, a, nodata=-9999, fieldname='value', verbose=False, **kwargs)

Write a numpy array to Arc Ascii grid or shapefile with the model reference.

Parameters

- **modelgrid** (*flopy.discretization.StructuredGrid* object) - model grid
- **filename** (*str* or *PathLike*) - Path of output file. Export format is determined by file extention. '.asc' Arc Ascii grid '.tif' GeoTIFF (requires rasterio package) '.shp' Shapefile

- **a** (*2D numpy.ndarray*) - Array to export
- **nodata** (*scalar*) - Value to assign to np.nan entries (default -9999)
- **fieldname** (*str*) - Attribute field name for array values (shapefile export only). (default 'values')
- **verbose** (*bool, optional, default False*) - whether to show verbose output
- **kwargs** - keyword arguments to np.savetxt (ascii) rasterio.open (GeoTIFF) or flopy.export.shapefile_utils.write_grid_shapefile

Notes

Rotated grids will be either be unrotated prior to export, using `scipy.ndimage.rotate` (Arc Ascii format) or rotation will be included in their transform property (GeoTiff format). In either case the pixels will be displayed in the (unrotated) projected geographic coordinate system, so the pixels will no longer align exactly with the model grid (as displayed from a shapefile, for example). A key difference between Arc Ascii and GeoTiff (besides disk usage) is that the unrotated Arc Ascii will have a different grid size, whereas the GeoTiff will have the same number of rows and pixels as the original.

```
export_array_contours(modelgrid, filename: str | PathLike, a, fieldname='level',
                      interval=None, levels=None, maxlevels=1000, **kwargs)
```

Contour an array using matplotlib; write shapefile of contours.

Parameters

- **modelgrid** (*flopy.discretization.Grid object*) - model grid object
- **filename** (*str or PathLike*) - Path of output file with '.shp' extention.
- **a** (*2D numpy array*) - Array to contour
- **fieldname** (*str*) - gis field name
- **interval** (*float*) - interval to calculate levels from
- **levels** (*list*) - list of contour levels
- **maxlevels** (*int*) - maximum number of contour levels
- ****kwargs** (*keyword arguments to flopy.export.shapefile_utils.reccarray2shp*) -

```
export_contourf(filename, contours, fieldname='level', verbose=False, **kwargs)
```

Write matplotlib filled contours to shapefile.

Parameters

- **filename** (*str or PathLike*) - name of output shapefile (e.g. myshp.shp)
- **contours** (*matplotlib.contour.QuadContourSet or list of them*) - (object returned by matplotlib.pyplot.contourf)
- **fieldname** (*str*) - Name of shapefile attribute field to contain the contour level. The fieldname column in the attribute table will

contain the lower end of the range represented by the polygon.
Default is 'level'.

- **verbose** (*bool*, optional, default *False*) - whether to show verbose output
- ****kwargs** (keyword arguments to *flopy.export.shapefile_utils.reccarray2shp*) -

Return type

None

Examples

```
>>> import flopy
>>> import matplotlib.pyplot as plt
>>> from flopy.export.utils import export_contourf
>>> a = np.random.random((10, 10))
>>> cs = plt.contourf(a)
>>> export_contourf('myfilledcontours.shp', cs)
```

export_contours(filename: *str* | *PathLike*, contours, fieldname='level', verbose=False, **kwargs)

Convert matplotlib contour plot object to shapefile.

Parameters

- **filename** (*str* or *PathLike*) - path of output shapefile
- **contours** (*matplotlib.contour.QuadContourSet* or *list* of them) - (object returned by *matplotlib.pyplot.contour*)
- **fieldname** (*str*) - gis attribute table field name
- **verbose** (*bool*, optional, default *False*) - whether to show verbose output
- ****kwargs** (key-word arguments to *flopy.export.shapefile_utils.reccarray2shp*) -

Returns

df

Return type

dataframe of shapefile contents

generic_array_export(f: *str* | *PathLike*, array, var_name='generic_array', dimensions=('time', 'layer', 'y', 'x'), precision_str='f4', units='unitless', **kwargs)

Method to export a generic array to NetCdf

Parameters

- **f** (*str* or *PathLike*) - filename or existing export instance type (NetCdf only for now)
- **array** (*np.ndarray*) -
- **var_name** (*str*) - variable name
- **dimensions** (*tuple*) - netcdf dimensions

- **precision_str** (*str*) - binary precision string, default “f4”
- **units** (*string*) - units of array data
- ****kwargs** (*keyword arguments*) -

model

[flopy.modflow.mbase] flopy model object

mflist_export(*f*: *str* | *PathLike* | *NetCdf*, *mfl*, ****kwargs**)

export helper for MfList instances

Parameters

- **f** (*str* or *PathLike* or *NetCdf*) - file path or existing export instance type (NetCdf only for now)
- **mfl** (*MfList* instance) -
- ****kwargs** (*keyword arguments*) -

modelgrid

[flopy.discretization.Grid] model grid instance which will supercede the flopy.model.modelgrid

crs

[pyproj.CRS, optional if *prjfile* is specified] Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg “EPSG:26916”) or a WKT string.

prjfile

[*str* or *pathlike*, optional if *crs* is specified] ESRI-style projection file with well-known text defining the CRS for the model grid (must be projected; geographic CRS are not supported).

model_export(*f*: *str* | *PathLike* | *NetCdf* | *dict*, *ml*, *fmt*=None, ****kwargs**)

Method to export a model to a shapefile or netcdf file

Parameters

- **f** (*str* or *PathLike* or *NetCdf* or *dict*) - file path (“nc” for netcdf or “.shp” for shapefile) or NetCDF object or dictionary
- **ml** (*flopy.modflow.mbase.ModelInterface* object) - flopy model object
- **fmt** (*str*) - output format flag. ‘vtk’ will export to vtk
- ****kwargs** (*keyword arguments*) -

modelgrid: `flopy.discretization.Grid`

user supplied modelgrid object which will supercede the built in modelgrid object

crs

[*pyproj.CRS*, optional if *prjfile* is specified] Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg “EPSG:26916”) or a WKT string.

prjfile

[str or pathlike, optional if crs is specified] ESRI-style projection file with well-known text defining the CRS for the model grid (must be projected; geographic CRS are not supported). if fmt is set to 'vtk', parameters of `vtk.export_model`

output_helper(f: *str* | *PathLike* | *NetCdf* | *dict*, ml, oudic, verbose=False, **kwargs)

Export model outputs using the model spatial reference info.

Parameters

- **f** (*str* or *PathLike* or *NetCdf* or *dict*) - filepath to write output to (must have .shp or .nc extension), NetCDF object, or dictionary
- **ml** (*flopy.mbase.ModelInterface* derived type) -
- **oudic** (*dict*) - output_filename, flopy datafile/cellbudgetfile instance
- **verbose** (*bool*) - whether to show verbose output
- ****kwargs** (*keyword arguments*) -

modelgrid

[*flopy.discretization.Grid*] user supplied model grid instance that will be used for export in lieu of the models model grid instance

mflay

[int] zero based model layer which can be used in shapefile exporting

kper

[int] zero based stress period which can be used for shapefile exporting

Return type

None

Note:

casts down double precision to single precision for netCDF files

package_export(f: *str* | *PathLike* | *NetCdf* | *dict*, pak, fmt=None, verbose=False, **kwargs)

Method to export a package to shapefile or netcdf

Parameters

- **f** (*str* or *PathLike* or *NetCdf* or *dict*) - output file path (extension .shp for shapefile or .nc for netcdf) or NetCDF object or dictionary
- **pak** (*flopy.pakbase.Package* object) - package to export
- **fmt** (*str*) - output format flag. 'vtk' will export to vtk
- **kwargs** (****) -

modelgrid: flopy.discretization.Grid

user supplied modelgrid object which will supercede the built in modelgrid object

crs
 [pyproj.CRS, optional if *prjfile* is specified] Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg “EPSG:26916”) or a WKT string.

prjfile
 [str or pathlike, optional if *crs* is specified] ESRI-style projection file with well-known text defining the CRS for the model grid (must be projected; geographic CRS are not supported).

if *fmt* is set to ‘vtk’, parameters of `vtk.export_package`

Returns
f

Return type
 NetCdf object or None

transient2d_export(*f*: *str* | *PathLike*, *t2d*, *fmt*=None, ***kwargs*)

export helper for Transient2d instances

Parameters

- **f** (*str* or *PathLike*) - filename or existing export instance type (NetCdf only for now)
- **t2d** (*Transient2d* instance) -
- **fmt** (*str*) - output format flag. ‘vtk’ will export to vtk
- ****kwargs** (*keyword arguments*) - *min_valid* : minimum valid value *max_valid* : maximum valid value *modelgrid* :

flopy.modelgrid instance which will supercede the *flopy.model.modelgrid*

if *fmt* is set to ‘vtk’, parameters of `vtk.export_transient`

flopy.export.vtk module

The `vtk` module provides functionality for exporting model inputs and outputs to VTK.

class Pvd

Bases: `object`

Simple class to build a Paraview Data File (PVD)

add_timevalue(*file*, *timevalue*)

Method to add a Dataset record to the pvd file

Parameters

- **file** (*os.PathLike* or *str*) - vtu file name
- **timevalue** (*float*) - time step value in model time

write(f)

Method to write a pvd file from the PVD object.

Parameters

f (*os.PathLike* or *str*) - PVD file name

class Vtk(*model=None, modelgrid=None, vertical_exageration=1, binary=True, xml=False, pvd=False, shared_points=False, smooth=False, point_scalars=False*)

Bases: *object*

Class that builds VTK objects and exports models to VTK files

Parameters

- **model** (*flopy.ModelInterface* object) - any flopy model object, example *flopy.modflow.Modflow()* object
- **modelgrid** (*flopy.discretization.Grid* object) - any flopy modelgrid object, example. *VertexGrid*
- **vertical_exageration** (*float*) - floating point value to scale vertical exageration of the vtk points default is 1.
- **binary** (*bool*) - flag that indicates if Vtk will write a binary or text file. Binary is preferred as paraview has a bug (8/4/2021) where it cannot represent NaN values from ASCII (non xml) files. In this case no-data values are set to 1e+30.
- **xml** (*bool*) - flag to write xml based VTK files
- **pvd** (*bool*) - boolean flag to write a paraview pvd file for transient data. This file maps vtk files to a model time.
- **shared_points** (*bool*) - boolean flag to share points in grid polyhedron construction. Default is False, as paraview has a bug (8/4/2021) where some polyhedrons will not properly close when using shared points. If *shared_points* is True file size will be slightly smaller.
- **smooth** (*bool*) - boolean flag to interpolate vertex elevations based on shared cell elevations. Default is False.
- **point_scalars** (*bool*) - boolean flag to write interpolated data at each point based "shared vertices".

add_array(*array, name, masked_values=None, dtype=None*)

Method to set an array to the vtk grid

Parameters

- **array** (*np.ndarray, list*) - list of array values to set to the *vtk_array*
- **name** (*str*) - array name for vtk
- **masked_values** (*list, None*) - list of values to set equal to nan
- **dtype** (*vtk datatype object*) - method to supply and force a vtk datatype

add_cell_budget(*cbc, text=None, kstpker=None, masked_values=None*)

Method to add cell budget data to vtk

Parameters

- **cbc** (*flopy.utils.CellBudget object*) - flopy binary CellBudget object
- **text** (*str or None*) - The text identifier for the record. Examples include 'RIVER LEAKAGE', 'STORAGE', 'FLOW RIGHT FACE', etc. If text=None all compatible records are exported
- **kstpker** (*tuple, list of tuples, None*) - tuple or list of tuples of kstpker, if kstpker=None all records are selected
- **masked_values** (*list, None*) - list of values to set equal to nan

add_heads(hds, kstpker=None, masked_values=None)

Method to add head data to a vtk file

Parameters

- **hds** (*flopy.utils.LayerFile object*) - Binary or Formatted HeadFile type object
- **kstpker** (*tuple, list of tuples, None*) - tuple or list of tuples of kstpker, if kstpker=None all records are selected
- **masked_values** (*list, None*) - list of values to set equal to nan

add_model(model, selpaklist=None, masked_values=None)

Method to add all array and transient list data from a modflow model to a timeseries of vtk files

Parameters

- **model** (*fp.modflow.ModelInterface*) - any flopy model object
- **selpaklist** (*list, None*) - optional parameter where the user can supply a list of packages to export.
- **masked_values** (*list, None*) - list of values to set equal to nan

add_package(pkg, masked_values=None)

Method to set all arrays within a package to VTK arrays

Parameters

- **pkg** (*flopy.pakbase.Package object*) - flopy package object, example ModflowWel
- **masked_values** (*list, None*) - list of values to set equal to nan

add_pathline_points(pathlines, timeseries=False)

Method to add Modpath output from a pathline or timeseries file to the grid. Colors will be representative of totim.

Parameters

- **pathlines** (*np.recarray or list*) - pathlines accepts a numpy recarray of a particle pathline or a list of numpy recarrays associated with pathlines

- **timeseries** (*bool*) - method to plot data as a series of vtk timeseries files for animation or as a single static vtk file. Default is false

add_transient_array(*d*, *name=None*, *masked_values=None*)

Method to add transient array data to the vtk object

Parameters

- **d** (*dict*) - dictionary of array2d, array3d data or numpy array data
- **name** (*str*, *None*) - parameter name, required when user provides a dictionary of numpy arrays
- **masked_values** (*list*, *None*) - list of values to set equal to nan

Return type

None

add_transient_list(*mflist*, *masked_values=None*)

Method to add transient list data to a vtk object

Parameters

- **mflist** (*flopy.utils.MfList* object) -
- **masked_values** (*list*, *None*) - list of values to set equal to nan

add_transient_vector(*d*, *name*, *masked_values=None*)

Method to add transient vector data to vtk

Parameters

- **d** (*dict*) - dictionary of vectors
- **name** (*str*) - name of vector to be displayed in vtk
- **masked_values** (*list*, *None*) - list of values to set equal to nan

add_vector(*vector*, *name*, *masked_values=None*)

Method to add vector data to vtk

Parameters

- **vector** (*array*) - array of dimension (3, nnodes)
- **name** (*str*) - name of the vector to be displayed in vtk
- **masked_values** (*list*, *None*) - list of values to set equal to nan

to_pyvista()

Convert VTK object to PyVista meshes. If the VTK object contains 0 or multiple meshes a list of meshes is returned. Otherwise the one mesh is returned alone. PyVista must be installed for this method.

Returns

PyVista mesh or list of meshes

Return type

pyvista.DataSet or list of pyvista.DataSet

write(f: str | PathLike, kper=None)

Method to write a vtk file from the VTK object

Parameters

- **f** (str or PathLike) - vtk file name
- **kpers** (int, list, tuple) - stress period or list of stress periods to write to vtk. This parameter only applies to transient package data.

export_array(model, array, output_folder: str | PathLike, name, nanval=-1e+20, array2d=False, smooth=False, point_scalars=False, vtk_grid_type='auto', true2d=False, binary=True)

Export array to vtk

Deprecated since version 3.3.5: Use `Vtk.add_array()`

Parameters

- **model** (flopy model instance) - the flopy model instance
- **array** (flopy array) - flopy 2d or 3d array
- **output_folder** (str or PathLike) - output folder to write the data
- **name** (str) - name of array
- **nanval** (scalar) - no data value, default value is -1e20
- **array2d** (bool) - true if the array is 2d and represents the first layer, default is False
- **smooth** (bool) - if True, will create smooth layer elevations, default is False
- **point_scalars** (bool) - if True, will also output array values at cell vertices, default is False; note this automatically sets smooth to True
- **vtk_grid_type** (str) - Specific vtk_grid_type or 'auto' (default). Possible specific values are 'ImageData', 'RectilinearGrid', and 'UnstructuredGrid'. If 'auto', the grid type is automatically determined. Namely:
 - A regular grid (in all three directions) will be saved as an 'ImageData'.
 - A rectilinear (in all three directions), non-regular grid will be saved as a 'RectilinearGrid'.
 - Other grids will be saved as 'UnstructuredGrid'.
- **true2d** (bool) - If True, the model is expected to be 2d (1 layer, 1 row or 1 column) and the data will be exported as true 2d data, default is False.
- **binary** (bool) - if True the output file will be binary, default is False

```
export_cbc(model, cbcfile, ofolder, precision='single', verbose=False, nanval=-1e+20,
           kstpiper=None, text=None, smooth=False, point_scalars=False,
           vtk_grid_type='auto', true2d=False, binary=True)
```

Exports cell by cell file to vtk

Deprecated since version 3.3.5: Use [Vtk.add_cell_budget\(\)](#)

Parameters

- **model** (*flopy model instance*) - the flopy model instance
- **cbcfile** (*str*) - the cell by cell file
- **otfolder** (*str*) - output folder to write the data
- **precision** (*str*) - Precision of data in the cell by cell file: 'single' or 'double'. Default is 'single'.
- **verbose** (*bool*) - If True, write information to the screen. Default is False.
- **nanval** (*scalar*) - no data value
- **kstpiper** (*tuple of ints or list of tuple of ints*) - A tuple containing the time step and stress period (kstp, kper). The kstp and kper values are zero based.
- **text** (*str or list of str*) - The text identifier for the record. Examples include 'RIVER LEAKAGE', 'STORAGE', 'FLOW RIGHT FACE', etc.
- **smooth** (*bool*) - if True, will create smooth layer elevations, default is False
- **point_scalars** (*bool*) - if True, will also output array values at cell vertices, default is False; note this automatically sets smooth to True
- **vtk_grid_type** (*str*) - Specific vtk_grid_type or 'auto' (default). Possible specific values are 'ImageData', 'RectilinearGrid', and 'UnstructuredGrid'. If 'auto', the grid type is automatically determined. Namely:
 - A regular grid (in all three directions) will be saved as an 'ImageData'.
 - A rectilinear (in all three directions), non-regular grid will be saved as a 'RectilinearGrid'.
 - Other grids will be saved as 'UnstructuredGrid'.
- **true2d** (*bool*) - If True, the model is expected to be 2d (1 layer, 1 row or 1 column) and the data will be exported as true 2d data, default is False.
- **binary** (*bool*) - if True the output file will be binary, default is False

```
export_heads(model, hdsfile, ofolder, text='head', precision='auto', verbose=False,
              nanval=-1e+20, kstpiper=None, smooth=False, point_scalars=False,
              vtk_grid_type='auto', true2d=False, binary=True)
```

Exports binary head file to vtk

Deprecated since version 3.3.5: Use [Vtk.add_heads\(\)](#)

Parameters

- **model** (*MFModel*) - the flopy model instance
- **hdsfile** (*str*, *HeadFile* object) - binary head file path or object
- **otfolder** (*str*) - output folder to write the data
- **text** (*string*) - Name of the text string in the head file. Default is 'head'.
- **precision** (*str*) - Precision of data in the head file: 'auto', 'single' or 'double'. Default is 'auto'.
- **verbose** (*bool*) - If True, write information to the screen. Default is False.
- **nanval** (*scalar*) - no data value, default value is -1e20
- **kstp** (*tuple of ints or list of tuple of ints*) - A tuple containing the time step and stress period (kstp, kper). The kstp and kper values are zero based.
- **smooth** (*bool*) - if True, will create smooth layer elevations, default is False
- **point_scalars** (*bool*) - if True, will also output array values at cell vertices, default is False; note this automatically sets smooth to True
- **vtk_grid_type** (*str*) - Specific vtk_grid_type or 'auto' (default). Possible specific values are 'ImageData', 'RectilinearGrid', and 'UnstructuredGrid'. If 'auto', the grid type is automatically determined. Namely:
 - A regular grid (in all three directions) will be saved as an 'ImageData'.
 - A rectilinear (in all three directions), non-regular grid will be saved as a 'RectilinearGrid'.
 - Other grids will be saved as 'UnstructuredGrid'.
- **true2d** (*bool*) - If True, the model is expected to be 2d (1 layer, 1 row or 1 column) and the data will be exported as true 2d data, default is False.
- **binary** (*bool*) - if True the output file will be binary, default is False

```
export_model(model, ofolder: str | PathLike, package_names=None, nanval=-1e+20,
             smooth=False, point_scalars=False, vtk_grid_type='auto', true2d=False,
             binary=True, kpers=None)
```

Export model to vtk

Deprecated since version 3.3.5: Use `Vtk.add_model()`

Parameters

- **model** (*flopy model instance*) - flopy model
- **otfolder** (*str* or *PathLike*) - output folder
- **package_names** (*list*) - list of package names to be exported
- **nanval** (*scalar*) - no data value, default value is -1e20

- **array2d** (*bool*) - True if array is 2d, default is False
- **smooth** (*bool*) - if True, will create smooth layer elevations, default is False
- **point_scalars** (*bool*) - if True, will also output array values at cell vertices, default is False; note this automatically sets smooth to True
- **vtk_grid_type** (*str*) - Specific `vtk_grid_type` or 'auto' (default). Possible specific values are 'ImageData', 'RectilinearGrid', and 'UnstructuredGrid'. If 'auto', the grid type is automatically determined. Namely:
 - A regular grid (in all three directions) will be saved as an 'ImageData'.
 - A rectilinear (in all three directions), non-regular grid will be saved as a 'RectilinearGrid'.
 - Other grids will be saved as 'UnstructuredGrid'.
- **true2d** (*bool*) - If True, the model is expected to be 2d (1 layer, 1 row or 1 column) and the data will be exported as true 2d data, default is False.
- **binary** (*bool*) - if True the output file will be binary, default is False
- **kpers** (*iterable of int*) - Stress periods to export. If None (default), all stress periods will be exported.

```
export_package(pak_model, pak_name, ofolder: str | PathLike, vtkobj=None,
              nanval=-1e+20, smooth=False, point_scalars=False, vtk_grid_type='auto',
              true2d=False, binary=True, kpers=None)
```

Export package to vtk

Deprecated since version 3.3.5: Use `Vtk.add_package()`

Parameters

- **pak_model** (*flopy model instance*) - the model of the package
- **pak_name** (*str*) - the name of the package
- **ofolder** (*str or PathLike*) - output folder to write the data
- **vtkobj** (*VTK instance*) - a vtk object (allows export_package to be called from export_model)
- **nanval** (*scalar*) - no data value, default value is -1e20
- **smooth** (*bool*) - if True, will create smooth layer elevations, default is False
- **point_scalars** (*bool*) - if True, will also output array values at cell vertices, default is False; note this automatically sets smooth to True
- **vtk_grid_type** (*str*) - Specific `vtk_grid_type` or 'auto' (default). Possible specific values are 'ImageData', 'RectilinearGrid', and 'UnstructuredGrid'. If 'auto', the grid type is automatically determined. Namely:

- A regular grid (in all three directions) will be saved as an 'ImageData'.
- A rectilinear (in all three directions), non-regular grid will be saved as a 'RectilinearGrid'.
- Other grids will be saved as 'UnstructuredGrid'.
- **true2d** (*bool*) - If True, the model is expected to be 2d (1 layer, 1 row or 1 column) and the data will be exported as true 2d data, default is False.
- **binary** (*bool*) - if True the output file will be binary, default is False
- **kpers** (*iterable of int*) - Stress periods to export. If None (default), all stress periods will be exported.

```
export_transient(model, array, output_folder: str | PathLike, name, nanval=-1e+20,  
array2d=False, smooth=False, point_scalars=False, vtk_grid_type='auto',  
true2d=False, binary=True, kpers=None)
```

Export transient arrays and lists to vtk

Deprecated since version 3.3.5: Use `Vtk.add_transient_array()` or `Vtk.add_transient_list()`

Parameters

- **model** (*MFModel*) - the flopy model instance
- **array** (*Transient instance*) - flopy transient array
- **output_folder** (*str or PathLike*) - output folder to write the data
- **name** (*str*) - name of array
- **nanval** (*scalar*) - no data value, default value is -1e20
- **array2d** (*bool*) - True if array is 2d, default is False
- **smooth** (*bool*) - if True, will create smooth layer elevations, default is False
- **point_scalars** (*bool*) - if True, will also output array values at cell vertices, default is False; note this automatically sets smooth to True
- **vtk_grid_type** (*str*) - Specific vtk_grid_type or 'auto' (default). Possible specific values are 'ImageData', 'RectilinearGrid', and 'UnstructuredGrid'. If 'auto', the grid type is automatically determined. Namely:
 - A regular grid (in all three directions) will be saved as an 'ImageData'.
 - A rectilinear (in all three directions), non-regular grid will be saved as a 'RectilinearGrid'.
 - Other grids will be saved as 'UnstructuredGrid'.
- **true2d** (*bool*) - If True, the model is expected to be 2d (1 layer, 1 row or 1 column) and the data will be exported as true 2d data, default is False.

- **binary** (*bool*) - if True the output file will be binary, default is False
- **kpers** (*iterable of int*) - Stress periods to export. If None (default), all stress periods will be exported.

8.3.4 PEST Utilities

Contents:

`flopy.pest.params` module

```
class Params(mfpackage, partype, parname, startvalue, lbound, ubound, span,
             transform='log')
```

Bases: `object`

Class to define parameters that will be estimated using PEST.

Parameters

- **mfpackage** (*str*) - The Modflow package type to associated with this parameter. 'LPF' is one package that is working now.
- **partype** (*str*) - The parameter type, such as 'hk'. This must be a valid attribute in the mfpackage.
- **parname** (*str*) - The parameter name, such as 'HK_1'.
- **startvalue** (*float*) - The starting value for the parameter.
- **lbound** (*float*) - The lower bound for the parameter.
- **ubound** (*float*) - The upper bound for the parameter.
- **span** (*dict*) - The span over which the parameter applies. The span depends on the type of array that the parameter applies to. For 3d arrays, span should have either 'idx' or 'layers' keys. span['layers'] should be a list of layer to for which parname will be applied as a multiplier. idx is a tuple, which contains the indices to which this parameter applies. For example, if the parameter applies to a part of a 3D MODFLOW array, then idx can be a tuple of layer, row, and column indices (e.g. (karray, iarray, jarray)). This idx variable could also be a 3D bool array. It is ultimately used to assign parameter to the array using `arr[idx] = parname`. For transient 2d arrays, span must include a 'kpers' key such that span['kpers'] is a list of stress period to which parname will be applied as a multiplier.
- **transform** (*Parameter transformation type.*) -

```
zonearray2params(mfpackage, partype, parzones, lbound, ubound, parvals, transform,
                 zonearray)
```

Helper function to create a list of flopy parameters from a zone array and list of parameter zone numbers.

The parameter name is set equal to the parameter type and the parameter zone value, separated by an underscore.

flopy.pest.templatewriter module

```
class TemplateWriter(model, plist)
```

Bases: `object`

Class for writing PEST template files.

Parameters

- **model** (*flopy.modflow object*) - flopy model object.
- **plist** (*list*) - list of parameter objects of type `flopy.pest.params.Params`.

```
write_template()
```

Write the template files for all model files that have arrays that have been parameterized.

flopy.pest.tplarray module

```
class Transient2dTpl(transient2d)
```

Bases: `object`

```
add_parameter(p)
```

Store the parameters in a list for later substitution

```
get_kper_entry(kper)
```

```
class Util2dTpl(chararray, name, multiplier, indexed_param)
```

Bases: `object`

Class to define a two-dimensional template array for use with parameter estimation.

Parameters

- **chararray** (*A Numpy ndarray of dtype 'str'.*) -
- **name** (*The parameter type. This will be written to the control record*) - as a comment.
- **indexed_param** (*bool*) - A flag to indicated whether or not the array contains parameter names within the array itself.

```
get_file_entry()
```

Convert the array into a string.

Returns

`file_entry`

Return type

`str`

```
class Util3dTpl(u3d)
```

Bases: `object`

Class to define a three-dimensional template array for use with parameter estimation.

Parameters

u3d (*Util3d object*) -

```
add_parameter(p)
```

Fill the chararray with the parameter name.

Parameters

p (`flopy.pest.params.Params`) - Parameter. Must have `.idx` and `.name` attributes

get_template_array(*pakarray*)

Convert the package array into the appropriate template array

8.3.5 Discretization Utilities

Contents:

`flopy.discretization.grid` module

class `CachedData`(*data*)

Bases: `object`

property `data`

property `data_nocopy`

update_data(*data*)

class `Grid`(*grid_type=None, top=None, botm=None, idomain=None, lenuni=None, crs=None, epsg=None, proj4=None, prj=None, prjfile=None, xoff=0.0, yoff=0.0, angrot=0.0*)

Bases: `object`

Base class for a structured or unstructured model grid

Parameters

- **grid_type** (*enumeration*) - type of model grid ('structured', 'vertex', 'unstructured')
- **top** (*float* or *ndarray*) - top elevations of cells in topmost layer
- **botm** (*float* or *ndarray*) - bottom elevations of all cells
- **idomain** (*int* or *ndarray*) - ibound/idomain value for each cell
- **lenuni** (*int* or *ndarray*) - model length units
- **crs** (`pyproj.CRS`, optional if *prjfile* is specified) - Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg "EPSG:26916") or a WKT string.
- **prjfile** (*str* or *pathlike*, optional if *crs* is specified) - ESRI-style projection file with well-known text defining the CRS for the model grid (must be projected; geographic CRS are not supported).
- **xoff** (*float*) - x coordinate of the origin point (lower left corner of model grid) in the spatial reference coordinate system
- **yoff** (*float*) - y coordinate of the origin point (lower left corner of model grid) in the spatial reference coordinate system

- **angrot** (*float*) - rotation angle of model grid, as it is rotated around the origin point

grid_type
 type of model grid ('structured', 'vertex', 'unstructured')

Type
 enumeration

top
 top elevations of cells in topmost layer

Type
float or ndarray

botm
 bottom elevations of all cells

Type
float or ndarray

idomain
 ibound/idomain value for each cell

Type
int or ndarray

crs
 Coordinate reference system (CRS) for the model grid

Type
 pyproj.CRS

lenuni
 modflow lenuni parameter

Type
int

xoffset
 x coordinate of the origin point in the spatial reference coordinate system

Type
float

yoffset
 y coordinate of the origin point in the spatial reference coordinate system

Type
float

angrot
 rotation angle of model grid, as it is rotated around the origin point

Type
float

angrot_radians
 rotation angle of model grid, in radians

Type
float

xgrid

returns numpy meshgrid of x edges in reference frame defined by point_type

Type

ndarray

ygrid

returns numpy meshgrid of y edges in reference frame defined by point_type

Type

ndarray

zgrid

returns numpy meshgrid of z edges in reference frame defined by point_type

Type

ndarray

xcenters

returns x coordinate of cell centers

Type

ndarray

ycenters

returns y coordinate of cell centers

Type

ndarray

zcenters

returns z coordinate of cell centers

Type

ndarray

xyzgrid

returns the location of grid edges of all model cells. if the model grid contains spatial reference information, the grid edges are in the coordinate system provided by the spatial reference information. returns a list of three ndarrays for the x, y, and z coordinates

Type

[ndarray, ndarray, ndarray]

xyzcellcenters

returns the cell centers of all model cells in the model grid. if the model grid contains spatial reference information, the cell centers are in the coordinate system provided by the spatial reference information. otherwise the cell centers are based on a 0,0 location for the upper left corner of the model grid. returns a list of three ndarrays for the x, y, and z coordinates

Type

[ndarray, ndarray, ndarray]

get_coords(x, y)

transform point or array of points x, y from model coordinates to spatial coordinates

grid_lines : (point_type=PointType.spatialxyz) : list

returns the model grid lines in a list. each line is returned as a list containing two tuples in the format [(x1,y1), (x2,y2)] where x1,y1 and x2,y2 are the endpoints of the line.

xyvertices : (point_type) : ndarray

1D array of x and y coordinates of cell vertices for whole grid (single layer) in C-style (row-major) order (same as np.ravel())

intersect(x, y, local)

returns the row and column of the grid that the x, y point is in

Notes

Examples

property angrot

property angrot_radians

attrs_from_namfile_header(namefile)

property botm

property cell_thickness

Get the cell thickness for a structured, vertex, or unstructured grid.

Returns

thick

Return type

calculated thickness

cross_section_adjust_indicies(k, cbcnt)

Method to get adjusted indicies by layer and confining bed for PlotCrossSection plotting

Parameters

- **k** (*int*) - zero based layer number
- **cbcnt** (*int*) - confining bed counter

Returns

- **tuple** ((*int*, *int*, *int*) (*adjusted layer*, *nodeskip layer*, *node*)
- *adjustment value based on number of confining beds and the layer*)

cross_section_lay_ncpl_ncb(ncb)

Get PlotCrossSection compatible layers, ncpl, and ncb variables

Parameters

ncb (*int*) - number of confining beds

Returns

tuple

Return type

(*int*, *int*, *int*) layers, ncpl, ncb

cross_section_nodskip(*nlay*, *xypts*)

Get a nodskip list for PlotCrossSection. This is a correction for UnstructuredGridPlotting

Parameters

- **nlay** (*int*) - nlay is nlay + ncb
- **xypts** (*dict*) - dictionary of node number and xyvertices of a cross-section

Returns

list

Return type

n-dimensional list of nodes to not plot for each layer

cross_section_set_contour_arrays(*plotarray*, *xcenters*, *head*, *elev*, *projpts*)

Method to set countour array centers for rare instances where matplotlib contouring is preferred over trimesh plotting

Parameters

- **plotarray** (*np.ndarray*) - array of data for contouring
- **xcenters** (*np.ndarray*) - xcenters array
- **zcenters** (*np.ndarray*) - zcenters array
- **head** (*np.ndarray*) - head array to adjust cell centers location
- **elev** (*np.ndarray*) - cell elevation array
- **projpts** (*dict*) - dictionary of projected cross sectional vertices

Returns

- **tuple** ((*np.ndarray*, *np.ndarray*, *np.ndarray*, *bool*))
- *plotarray*, *xcenter* array, *ycenter* array, and a *boolean flag*
- *for contouring*

property `cross_section_vertices`

property `crs`

property `epsg`

property `extent`

classmethod `from_binary_grid_file`(*file_path*, *verbose=False*)

get_coords(*x*, *y*)

Given x and y array-like values, apply rotation, scale and offset, to convert them from model coordinates to real-world coordinates.

get_lni(*nodes*)

Get the layer index and within-layer node index (both 0-based) for the given nodes

Parameters

nodes (*node numbers (array-like)*) -

Return type

`list` of tuples (layer index, node index)

`get_local_coords(x, y)`

Given x and y array-like values, apply rotation, scale and offset, to convert them from real-world coordinates to model coordinates.

`get_number_plottable_layers(a)`

`get_plottable_layer_array(plotarray, layer)`

`get_plottable_layer_shape(layer=None)`

Determine the shape that is required in order to plot a 2d array for this grid. For a regular MODFLOW grid, this is (nrow, ncol). For a vertex grid, this is (ncpl,) and for an unstructured grid this is (ncpl[layer],).

Parameters

layer (`int`) - Has no effect unless grid changes by layer

Returns

shape - required shape of array to plot for a layer

Return type

`tuple`

`get_xcellcenters_for_layer(layer)`

`get_xvertices_for_layer(layer)`

`get_ycellcenters_for_layer(layer)`

`get_yvertices_for_layer(layer)`

property `grid_lines`

property `grid_type`

property `idomain`

`intersect(x, y, local=False, forgive=False)`

property `is_complete`

property `is_valid`

property `iverts`

property `laycbd`

property `lenuni`

`load_coord_info(namefile=None, reffile='usgs.model.reference')`

Attempts to load spatial reference information from the following files (in order): 1) usgs.model.reference 2) NAM file (header comment) 3) defaults

property `map_polygons`

property `ncpl`

neighbors(*node=None*, ***kwargs*)

Method to get nearest neighbors of a cell

Parameters

- **node** (*int*) - model grid node number
- **kwargs** (****) -

method

[*str*] "rook" for shared edge neighbors and "queen" for shared vertex neighbors

reset

[*bool*] flag to reset the neighbor calculation

Returns

list or *dict* - neighbors

Return type

list of cell node numbers or *dict* of all cells and

property *nlay*

property *nnodes*

property *nvert*

property *prj*

property *prjfile*

property *proj4*

read_usgs_model_reference_file(*reffile='usgs.model.reference'*)

read spatial reference info from the usgs.model.reference file <https://water.usgs.gov/ogw/policy/gw-model/modelers-setup.html>

remove_confining_beds(*array*)

Method to remove confining bed layers from an array

Parameters

array (*np.ndarray*) - array to remove quasi3d confining bed data from. Shape of axis 0 should be (*self.lay* + *ncb*) to remove beds

Return type

np.ndarray

saturated_thick(*array*, *mask=None*)

DEPRECATED method. *saturated_thick* will be removed in version 3.3.9

Get the saturated thickness for a structured, vertex, or unstructured grid. If the optional array is passed then thickness is returned relative to array values (saturated thickness). Returned values ranges from zero to cell thickness if optional array is passed.

Parameters

- **array** (*ndarray*) - array of elevations that will be used to adjust the cell thickness
- **mask** (*float*, *list*, *tuple*, *ndarray*) - array values to replace with a nan value.

Returns**thick****Return type**

calculated saturated thickness

saturated_thickness(array, mask=None)

Get the saturated thickness for a structured, vertex, or unstructured grid. If the optional array is passed then thickness is returned relative to array values (saturated thickness). Returned values ranges from zero to cell thickness if optional array is passed.

Parameters

- **array** (ndarray) - array of elevations that will be used to adjust the cell thickness
- **mask** (float, list, tuple, ndarray) - array values to replace with a nan value.

Returns**thickness****Return type**

calculated saturated thickness

set_coord_info(xoff=None, yoff=None, angrot=None, crs=None, prjfile=None, epsg=None, proj4=None, merge_coord_info=True)**property shape****property size****property thick**

DEPRECATED method. thick will be removed in version 3.3.9

Get the cell thickness for a structured, vertex, or unstructured grid.

Returns**thick****Return type**

calculated thickness

property top**property top_botm****property units****property verts****write_shapefile**(filename='grid.shp', epsg=None, prj=None)

Write a shapefile of the grid with just the row and column attributes.

property xcellcenters**property xoffset****property xvertices**


```

property xyzcellcenters
property xyzextent
property xyzvertices
property ycellcenters
property yoffset
property yvertices
property zcellcenters
property zvertices

```

flopy.discretization.modeltime module

```

class ModelTime(period_data=None, time_units='days', start_datetime=None,
                steady_state=None)
    Bases: object
    Class for MODFLOW simulation time
    Parameters
        • stress_periods (pandas dataframe) - headings are: perlen, nstp,
          tsmult
        • temporal_reference (TemporalReference) - contains start time and
          time units information
    property nper
    property nstp
    property perlen
    property start_datetime
    property steady_state
    property time_units
    property totim
    property tslen
    property tsmult

```

flopy.discretization.structuredgrid module

```

class StructuredGrid(delc=None, delr=None, top=None, botm=None, idomain=None,
                    lenuni=None, crs=None, epsg=None, proj4=None, prj=None,
                    prjfile=None, xoff=0.0, yoff=0.0, angrot=0.0, nlay=None, nrow=None,
                    ncol=None, laycbd=None)
    Bases: Grid
    class for a structured model grid

```

Parameters

- **delr** (*float* or *ndarray*) - column spacing along a row.
- **delc** (*float* or *ndarray*) - row spacing along a column.
- **top** (*float* or *ndarray*) - top elevations of cells in topmost layer
- **botm** (*float* or *ndarray*) - bottom elevations of all cells
- **idomain** (*int* or *ndarray*) - ibound/idomain value for each cell
- **lenuni** (*int* or *ndarray*) - model length units
- **crs** (*pyproj.CRS*, optional if *prjfile* is specified) - Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg "EPSG:26916") or a WKT string.
- **prjfile** (*str* or *pathlike*, optional if *crs* is specified) - ESRI-style projection file with well-known text defining the CRS for the model grid (must be projected; geographic CRS are not supported).
- **xoff** (*float*) - x coordinate of the origin point (lower left corner of model grid) in the spatial reference coordinate system
- **yoff** (*float*) - y coordinate of the origin point (lower left corner of model grid) in the spatial reference coordinate system
- **angrot** (*float*) - rotation angle of model grid, as it is rotated around the origin point
- **Properties** -
- ----- -
- **nlay** - returns the number of model layers
- **nrow** - returns the number of model rows
- **ncol** - returns the number of model columns
- **delc** - returns the delc array
- **delr** - returns the delr array
- **xyedges** - returns x-location points for the edges of the model grid and y-location points for the edges of the model grid

get_cell_vertices(i, j)

returns vertices for a single cell at row, column i, j.

array_at_faces(a, direction, withnan=True)

Computes values at the center of cell faces using linear interpolation.

Parameters

- **a** (*ndarray*) - Values at cell centers, shape (nlay, row, ncol).
- **direction** (*str*, possible values are 'x', 'y' and 'z') - Direction in which values will be interpolated at cell faces.
- **withnan** (*bool*) - If True (default), the result value will be set to NaN where the cell face sits between inactive cells. If False, not.

Returns

afaces - Array values interpolated at cell vertices, shape as input extended by 1 along the specified direction.

Return type

ndarray

array_at_verts(a)

Interpolate array values at cell vertices.

Parameters

a (ndarray) - Array values. Allowed shapes are: (nlay, nrow, ncol), (nlay, nrow, ncol+1), (nlay, nrow+1, ncol) and (nlay+1, nrow, ncol). * When the shape is (nlay, nrow, ncol), input values are considered at cell centers, and output values are computed by trilinear interpolation. * When the shape is extended in one direction, input values are considered at the center of cell faces in this direction, and output values are computed by bilinear interpolation in planes defined by these cell faces.

Returns

averts - Array values interpolated at cell vertices, shape (nlay+1, nrow+1, ncol+1).

Return type

ndarray

Notes

- Output values are smooth (continuous) even if top elevations or

bottom elevations are not constant across layers (i.e., in this case, vertices of neighboring cells are implicitly merged). * NaN values are assigned in accordance with inactive cells defined by idomain.

array_at_verts_basic(a)

Computes values at cell vertices using neighbor averaging.

Parameters

a (ndarray) - Array values at cell centers.

Returns

averts - Array values at cell vertices, shape (a.shape[0]+1, a.shape[1]+1, a.shape[2]+1). NaN values are assigned in accordance with inactive cells defined by idomain.

Return type

ndarray

property cross_section_vertices

Get a set of xvertices and yvertices ordered by node for plotting cross sections

Returns

xverts, yverts

Return type

(np.ndarray, np.ndarray)

`property delc`

`property delr`

`property delz`

`property extent`

classmethod `from_binary_grid_file(file_path, verbose=False)`

Instantiate a StructuredGrid model grid from a MODFLOW 6 binary grid (*.grb) file.

Parameters

- **file_path** (*str*) - file path for the MODFLOW 6 binary grid file
- **verbose** (*bool*) - Write information to standard output. Default is False.

Returns

return

Return type

StructuredGrid

classmethod `from_gridspec(file_path: str | PathLike, lenuni=0)`

Instantiate a StructuredGrid from grid specification file.

Parameters

- **file_path** (*str* or *PathLike*) - Path to the grid specification file
- **lenuni** (*int*) - Length unit code

Return type

A StructuredGrid

get_cell_vertices(*args, **kwargs)

Method to get a set of cell vertices for a single cell

used in the Shapefile export utilities and plotting code

Parameters

- **node** - (int) node number
- **i** - (int) cell row number
- **j** - (int) cell column number

Returns --- list of x,y cell vertices

get_lrc(nodes)

Get layer, row, column from a list of zero-based MODFLOW node numbers.

Parameters

nodes (*int*, *list* or *array_like*) - Zero-based node number

Returns

list of tuples containing the layer (k), row (i), and column (j) for each node in the input list

Return type`list`**Examples**

```
>>> import flopy
>>> sg = flopy.discretization.StructuredGrid(nlay=20, nrow=30, ncol=40)
>>> sg.get_lrc(100)
[(0, 2, 20)]
>>> sg.get_lrc([100, 1000, 10_000])
[(0, 2, 20), (0, 25, 0), (8, 10, 0)]
```

get_node(lrc_list)

Get node number from a list of zero-based MODFLOW layer, row, column tuples.

Parameters

lrc_list (*tuple of int or list of tuple of int*) - Zero-based layer, row, column tuples

Returns

list of MODFLOW nodes for each layer (k), row (i), and column (j) tuple in the input list

Return type`list`**Examples**

```
>>> import flopy
>>> sg = flopy.discretization.StructuredGrid(nlay=20, nrow=30, ncol=40)
>>> sg.get_node((0, 2, 20))
[100]
>>> sg.get_node([(0, 2, 20), (0, 25, 0), (8, 10, 0)])
[100, 1000, 10000]
```

get_number_plottable_layers(a)

Calculate and return the number of 2d plottable arrays that can be obtained from the array passed (a)

Parameters

a (*ndarray*) - array to check for plottable layers

Returns

nplottable - number of plottable layers

Return type`int`**get_plottable_layer_array(a, layer)****property grid_lines**

Get the grid lines as a list

intersect(x, y, z=None, local=False, forgive=False)

Get the row and column of a point with coordinates x and y

When the point is on the edge of two cells, the cell with the lowest row or column is returned.

Parameters

- **x** (*float*) - The x-coordinate of the requested point
- **y** (*float*) - The y-coordinate of the requested point
- **z** (*float*) - Optional z-coordinate of the requested point (will return layer, row, column) if supplied
- **local** (*bool (optional)*) - If True, x and y are in local coordinates (defaults to False)
- **forgive** (*bool (optional)*) - Forgive x,y arguments that fall outside the model grid and return NaNs instead (defaults to False - will throw exception)

Returns

- **row** (*int*) - The row number
- **col** (*int*) - The column number

property is_complete

property is_rectilinear

Test whether the grid is rectilinear (it is always so in the x and y directions, but not necessarily in the z direction).

property is_regular

Test if the grid spacing is regular and equal in x, y and z directions.

property is_regular_x

Test whether the grid spacing is regular in the x direction.

property is_regular_xy

Test if the grid spacing is regular and equal in x and y directions.

property is_regular_xz

Test if the grid spacing is regular and equal in x and z directions.

property is_regular_y

Test whether the grid spacing is regular in the y direction.

property is_regular_yz

Test if the grid spacing is regular and equal in y and z directions.

property is_regular_z

Test if the grid spacing is regular in z direction.

property is_valid

property iverts

property map_polygons

Get a list of matplotlib Polygon patches for plotting

Return type

`list` of Polygon objects

property ncol**property ncpl****neighbors(*args, **kwargs)**

Method to get nearest neighbors for a cell

Parameters

- ***args** - lay (int), row (int), column (int) or node (int)
- ****kwargs** -
 - k**
[int] layer number
 - i**
[int] row number
 - j**
[int] column number
 - as_node**
[bool] flag to return neighbors as node numbers
 - method**
[str] "rook" for shared edge neighbors (default) "queen" for shared vertex neighbors (for flow accumulation calculations)
 - reset**
[bool] flag to re-calculate neighbors, default is False

Return type

`list` of neighboring cells

property nlay**property nnodes****property nrow****property nvert****plot(**kwargs)**

Plot the grid lines.

Parameters

kwargs (*ax, colors. The remaining kwargs are passed into the*) - the LineCollection constructor.

Returns

`lc`

Return type

`matplotlib.collections.LineCollection`

property shape

property top_botm

property top_botm_withnan

Same as top_botm array but with NaN where idomain==0 both above and below a cell.

property verts

property xycenters

Return a list of two numpy one-dimensional float arrays for center x and y coordinates in model space - not offset or rotated.

property xyedges

one with the cell edge x coordinate (size = ncol+1) and the other with the cell edge y coordinate (size = nrow+1) in model space - not offset or rotated.

Type

Return a list of two 1D numpy arrays

property xyzcellcenters

two two-dimensional arrays for center x and y coordinates, and one three-dimensional array for center z coordinates. Coordinates are given in real-world coordinates.

Type

Return a list of three numpy float arrays

property xyzvertices

Method to get all grid vertices in a layer

Returns

[] 2D array

property zedges

Return zedges for (column, row)==(0, 0).

property zverts_smooth

Get a unique z of cell vertices using bilinear interpolation of top and bottom elevation layers.

Returns

zverts - z of cell vertices. NaN values are assigned in accordance with inactive cells defined by idomain.

Return type

ndarray, shape (nlay+1, nrow+1, ncol+1)

array_at_faces_1d(a, delta)

Interpolate array at cell faces of a 1d grid using linear interpolation.

Parameters

- **a** (1d ndarray) - Values at cell centers.
- **delta** (1d ndarray) - Grid steps.

Returns

afaces - Array values interpolated at cell faces, shape as input extended by 1.

Return type

1d ndarray

array_at_verts_basic2d(a)

Computes values at cell vertices on 2d array using neighbor averaging.

Parameters**a** (ndarray) - Array values at cell centers, could be a slice in any orientation.**Returns****averts** - Array values at cell vertices, shape (a.shape[0]+1, a.shape[1]+1).**Return type**

ndarray

flopy.discretization.unstructuredgrid module

```
class UnstructuredGrid(vertices=None, iverts=None, xcenters=None, ycenters=None,
                      top=None, botm=None, idomain=None, lenuni=None, ncpl=None,
                      crs=None, epsg=None, proj4=None, prj=None, prjfile=None, xoff=0.0,
                      yoff=0.0, angrot=0.0, iac=None, ja=None)
```

Bases: *Grid*

Class for an unstructured model grid

Parameters

- **vertices** (*list*) - list of vertices that make up the grid. Each vertex consists of three entries [iv, xv, yv] which are the vertex number, which should be zero-based, and the x and y vertex coordinates.
- **iverts** (*list*) - list of vertex numbers that comprise each cell. This list must be of size nodes, if the grid_varies_by_nodes argument is true, or it must be of size ncpl[0] if the same 2d spatial grid is used for each layer.
- **xcenters** (*list or ndarray*) - list of x center coordinates for all cells in the grid if the grid varies by layer or for all cells in a layer if the same grid is used for all layers
- **ycenters** (*list or ndarray*) - list of y center coordinates for all cells in the grid if the grid varies by layer or for all cells in a layer if the same grid is used for all layers
- **top** (*list or ndarray*) - top elevations for all cells in the grid.
- **botm** (*list or ndarray*) - bottom elevations for all cells in the grid.
- **idomain** (*int or ndarray*) - ibound/idomain value for each cell
- **lenuni** (*int or ndarray*) - model length units
- **ncpl** (ndarray) - one dimensional array of size nlay with the number of cells in each layer. This can also be passed in as a tuple or list as long as it can be set using ncpl = np.array(ncpl, dtype=int). The sum of ncpl must be equal to the number of cells in the grid. ncpl is optional and if it is not passed in, then

it is is set using `ncpl = np.array([len(iverts)], dtype=int)`, which means that all cells in the grid are contained in a single plottable layer. If the model grid defined in `verts` and `iverts` applies for all model layers, then the length of `iverts` can be equal to `ncpl[0]` and there is no need to repeat all of the vertex information for cells in layers

- **crs** (`pyproj.CRS`, optional if `prjfile` is specified) - Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg “EPSG:26916”) or a WKT string.
- **prjfile** (`str` or `pathlike`, optional if `crs` is specified) - ESRI-style projection file with well-known text defining the CRS for the model grid (must be projected; geographic CRS are not supported). beneath the top layer.
- **xoff** (`float`) - x coordinate of the origin point (lower left corner of model grid) in the spatial reference coordinate system
- **yoff** (`float`) - y coordinate of the origin point (lower left corner of model grid) in the spatial reference coordinate system
- **angrot** (`float`) - rotation angle of model grid, as it is rotated around the origin point
- **iac** (`list` or `ndarray`) - optional number of connections per node array
- **ja** (`list` or `ndarray`) - optional jagged connection array
- **Properties** -
- ----- -
- **vertices** - returns list of vertices that make up the grid
- **cell2d** - returns list of cells and their vertices

get_cell_vertices(*cellid*)

returns vertices for a single cell at `cellid`.

Notes

This class handles spatial representation of unstructured grids. It is based on the concept of being able to support multiple model layers that may have a different number of cells in each layer. The array `ncpl` is of size `nlay` and its sum must equal `nodes`. If the length of `iverts` is equal to `ncpl[0]` and the number of cells per layer is the same for each layer, then it is assumed that the grid does not vary by layer. In this case, the `xcenters` and `ycenters` arrays must also be of size `ncpl[0]`. This makes it possible to efficiently store spatial grid information for multiple layers.

If the spatial grid is different for each model layer, then the `grid_varies_by_layer` flag will automatically be set to `false`, and `iverts` must be of size `nodes`. The arrays for `xcenters` and `ycenters` must also be of size `nodes`.

cross_section_adjust_indicies(*k*, *cbcnt*)

Method to get adjusted indicies by layer and confining bed for PlotCrossSection plotting

Parameters

- **k** (*int*) - zero based model layer
- **cbcnt** (*int*) - confining bed counter

Returns

- **tuple** (*(int, int, int)*) (*adjusted layer, nodeskip layer, node*)
- *adjustment value based on number of confining beds and the layer*)

cross_section_lay_ncpl_ncb(*ncb*)

Get PlotCrossSection compatible layers, ncpl, and ncb variables

Parameters

- ncb** (*int*) - number of confining beds

Returns

tuple

Return type

(*int, int, int*) layers, ncpl, ncb

cross_section_nodeskip(*nlay*, *xypts*)

Get a nodeskip list for PlotCrossSection. This is a correction for UnstructuredGridPlotting

Parameters

- **nlay** (*int*) - nlay is nlay + ncb
- **xypts** (*dict*) - dictionary of node number and xyvertices of a cross-section

Returns

list

Return type

n-dimensional list of nodes to not plot for each layer

cross_section_set_contour_arrays(*plotarray*, *xcenters*, *head*, *elev*, *projpts*)

Method to set countour array centers for rare instances where matplotlib contouring is preferred over trimesh plotting

Parameters

- **plotarray** (*np.ndarray*) - array of data for contouring
- **xcenters** (*np.ndarray*) - xcenters array
- **head** (*np.ndarray*) - head array to adjust cell centers location
- **elev** (*np.ndarray*) - cell elevation array
- **projpts** (*dict*) - dictionary of projected cross sectional vertices

Returns

- **tuple** ((*np.ndarray*, *np.ndarray*, *np.ndarray*, *bool*))
- *plotarray*, *xcenter* array, *ycenter* array, and a *boolean flag*
- *for contouring*

property cross_section_vertices

Method to get vertices for cross-sectional plotting

Return type

xvertices, *yvertices*

property extent

classmethod from_argus_export(*file_path*, *nlay=1*)

Create a new UnstructuredGrid from an Argus One Trimesh file

Parameters

- **file_path** (*Path-like*) - Path to trimesh file
- **nlay** (*int*) - Number of layers to create

Return type

An UnstructuredGrid

classmethod from_binary_grid_file(*file_path*, *verbose=False*)

Instantiate a UnstructuredGrid model grid from a MODFLOW 6 binary grid (*.grb) file.

Parameters

- **file_path** (*str*) - file path for the MODFLOW 6 binary grid file
- **verbose** (*bool*) - Write information to standard output. Default is False.

Returns

return

Return type

UnstructuredGrid

classmethod from_gridspec(*file_path: str | PathLike*)

Create an UnstructuredGrid from a grid specification file.

Parameters

file_path (*str* or *PathLike*) - Path to the grid specification file

Return type

An UnstructuredGrid

get_cell_vertices(*cellid*)

Method to get a set of cell vertices for a single cell used in the Shapefile export utilities

Parameters

cellid - (int) cellid number

Returns --- list of x,y cell vertices

get_layer_node_range(*layer*)

get_number_plottable_layers(a)

Calculate and return the number of 2d plottable arrays that can be obtained from the array passed (a)

Parameters

a (*ndarray*) - array to check for plottable layers

Returns

nploortable - number of plottable layers

Return type

int

get_plottable_layer_array(a, layer)**get_plottable_layer_shape(layer=None)**

Determine the shape that is required in order to plot in 2d for this grid.

Parameters

layer (*int*) - Has no effect unless grid changes by layer

Returns

shape - required shape of array to plot for a layer

Return type

tuple

get_xcellcenters_for_layer(layer)**get_xvertices_for_layer(layer)****get_ycellcenters_for_layer(layer)****get_yvertices_for_layer(layer)****property grid_lines**

Creates a series of grid line vertices for drawing a model grid line collection. If the grid varies by layer, then return a dictionary with keys equal to layers and values equal to grid lines. Otherwise, just return the grid lines

Returns

grid lines or dictionary of lines by layer

Return type

dict

property grid_varies_by_layer**property iac****intersect(x, y, z=None, local=False, forgive=False)**

Get the CELL2D number of a point with coordinates x and y

When the point is on the edge of two cells, the cell with the lowest CELL2D number is returned.

Parameters

- **x** (*float*) - The x-coordinate of the requested point
- **y** (*float*) - The y-coordinate of the requested point

- **z** (*float*, *None*) - optional, z-coordiante of the requested point
- **local** (*bool* (*optional*)) - If True, x and y are in local coordinates (defaults to False)
- **forgive** (*bool* (*optional*)) - Forgive x,y arguments that fall outside the model grid and return NaNs instead (defaults to False - will throw exception)

Returns

icell2d - The CELL2D number

Return type

int

property is_complete

property is_valid

property iverts

property ja

property map_polygons

Property to get Matplotlib polygon objects for the modelgrid

Return type

list or *dict* of matplotlib.collections.Polygon

property ncpl

static ncpl_from_ihc(ihc, iac)

Use the ihc and iac arrays to calculate the number of cells per layer array (ncpl) assuming that the plottable layer number is stored in the diagonal position of the ihc array.

Parameters

- **ihc** (*ndarray*) - horizontal indicator array. If the plottable layer number is stored in the diagonal position, then this will be used to create the returned ncpl array. plottable layer numbers must increase monotonically and be consecutive with node number
- **iac** (*ndarray*) - array of size nodes that has the number of connections for a cell, plus one for the cell itself

Returns

ncpl - number of cells per plottable layer

Return type

ndarray

property nlay

property nnodes

property nvert

`plot(**kwargs)`

Plot the grid lines.

Parameters

kwargs (*ax, colors*. The remaining *kwargs* are passed into the) - the LineCollection constructor.

Returns

lc

Return type

`matplotlib.collections.LineCollection`

`set_ncpl(ncpl)`

property `shape`

property `top_botm`

property `verts`

property `xyzcellcenters`

Method to get cell centers and set to grid

property `xyzvertices`

Method to get model grid verticies

Returns

list of dimension `ncpl` by `nvertices`

`flopy.discretization.vertexgrid` module

```
class VertexGrid(vertices=None, cell2d=None, top=None, botm=None, idomain=None,
                 lenuni=None, crs=None, epsg=None, proj4=None, prj=None, prjfile=None,
                 xoff=0.0, yoff=0.0, angrot=0.0, nlay=None, ncpl=None, cell1d=None)
```

Bases: `Grid`

class for a vertex model grid

Parameters

- **vertices** - list of vertices that make up the grid
- **cell2d** - list of cells and their vertices
- **top** (*list* or *ndarray*) - top elevations for all cells in the grid.
- **botm** (*list* or *ndarray*) - bottom elevations for all cells in the grid.
- **idomain** (*int* or *ndarray*) - ibound/idomain value for each cell
- **lenuni** (*int* or *ndarray*) - model length units
- **crs** (`pyproj.CRS`, optional if *prjfile* is specified) - Coordinate reference system (CRS) for the model grid (must be projected; geographic CRS are not supported). The value can be anything accepted by `pyproj.CRS.from_user_input()`, such as an authority string (eg "EPSG:26916") or a WKT string.

- **prjfile** (str or pathlike, optional if crs is specified) - ESRI-style projection file with well-known text defining the CRS for the model grid (must be projected; geographic CRS are not supported).
- **xoff** (*float*) - x coordinate of the origin point (lower left corner of model grid) in the spatial reference coordinate system
- **yoff** (*float*) - y coordinate of the origin point (lower left corner of model grid) in the spatial reference coordinate system
- **angrot** (*float*) - rotation angle of model grid, as it is rotated around the origin point
- **Properties** -
- ----- -
- **vertices** - returns list of vertices that make up the grid
- **cell2d** - returns list of cells and their vertices

get_cell_vertices(cellid)

returns vertices for a single cell at cellid.

property cell1d

property cell2d

property extent

classmethod from_binary_grid_file(file_path, verbose=False)

Instantiate a VertexGrid model grid from a MODFLOW 6 binary grid (*.grb) file.

Parameters

- **file_path** (*str*) - file path for the MODFLOW 6 binary grid file
- **verbose** (*bool*) - Write information to standard output. Default is False.

Returns

return

Return type

VertexGrid

get_cell_vertices(cellid)

Method to get a set of cell vertices for a single cell
used in the Shapefile export utilities

Parameters

cellid - (int) cellid number

Returns --- list of x,y cell vertices

get_number_plottable_layers(a)

Calculate and return the number of 2d plottable arrays that can be obtained from the array passed (a)

Parameters

a (*ndarray*) - array to check for plottable layers

Returns

nplottable - number of plottable layers

Return type

`int`

get_plottable_layer_array(a, layer)

get_xcellcenters_for_layer(layer)

get_xvertices_for_layer(layer)

get_ycellcenters_for_layer(layer)

get_yvertices_for_layer(layer)

property grid_lines

Creates a series of grid line vertices for drawing a model grid line collection

Returns

grid line vertices

Return type

`list`

intersect(x, y, z=None, local=False, forgive=False)

Get the CELL2D number of a point with coordinates x and y

When the point is on the edge of two cells, the cell with the lowest CELL2D number is returned.

Parameters

- **x** (*float*) - The x-coordinate of the requested point
- **y** (*float*) - The y-coordinate of the requested point
- **z** (*float, None*) - optional, z-coordiante of the requested point will return (lay, icell2d)
- **local** (*bool (optional)*) - If True, x and y are in local coordinates (defaults to False)
- **forgive** (*bool (optional)*) - Forgive x,y arguments that fall outside the model grid and return NaNs instead (defaults to False - will throw exception)

Returns

icell2d - The CELL2D number

Return type

`int`

property is_complete

property is_valid

property iverts

property map_polygons

Get a list of matplotlib Polygon patches for plotting

Return type

`list` of Polygon objects

property ncpl

property nlay

property nnodes

property nvert

plot(**kwargs)

Plot the grid lines.

Parameters

kwargs (ax, colors. The remaining kwargs are passed into the) - the LineCollection constructor.

Returns

lc

Return type

matplotlib.collections.LineCollection

property shape

property top_botm

property verts

property xyzcellcenters

Method to get cell centers and set to grid

property xyzvertices

Method to get all grid vertices in a layer, arranged per cell

Returns

list of size sum(nvertices per cell)

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

f

- `flopy.discretization.grid`, 2015
- `flopy.discretization.modeltime`, 2023
- `flopy.discretization.structuredgrid`, 2023
- `flopy.discretization.unstructuredgrid`, 2031
- `flopy.discretization.vertexgrid`, 2037
- `flopy.export.longnames`, 1990
- `flopy.export.metadata`, 1990
- `flopy.export.netcdf`, 1991
- `flopy.export.shapefile_utils`, 1995
- `flopy.export.unitsformat`, 1998
- `flopy.export.utils`, 1998
- `flopy.export.vtk`, 2004
- `flopy.mbase`, 1605
- `flopy.mf6.data.mfdataarray`, 1578
- `flopy.mf6.data.mfdatalist`, 1587
- `flopy.mf6.data.mfdatascalar`, 1597
- `flopy.mf6.mbase`, 1279
- `flopy.mf6.mfmodel`, 1286
- `flopy.mf6.mfpackage`, 1294
- `flopy.mf6.modflow.mfgnc`, 1313
- `flopy.mf6.modflow.mfgwf`, 1334
- `flopy.mf6.modflow.mfgwfapi`, 1337
- `flopy.mf6.modflow.mfgwfbuy`, 1339
- `flopy.mf6.modflow.mfgwfchd`, 1341
- `flopy.mf6.modflow.mfgwfcsb`, 1345
- `flopy.mf6.modflow.mfgwfdis`, 1353
- `flopy.mf6.modflow.mfgwfdisu`, 1355
- `flopy.mf6.modflow.mfgwfdisv`, 1361
- `flopy.mf6.modflow.mfgwfdrn`, 1364
- `flopy.mf6.modflow.mfgwfevt`, 1368
- `flopy.mf6.modflow.mfgwfevta`, 1373
- `flopy.mf6.modflow.mfgwfgfb`, 1376
- `flopy.mf6.modflow.mfgwfgnc`, 1380
- `flopy.mf6.modflow.mfgwfgwf`, 1382
- `flopy.mf6.modflow.mfgwfgwt`, 1388
- `flopy.mf6.modflow.mfgwfhfb`, 1389
- `flopy.mf6.modflow.mfgwfgic`, 1390
- `flopy.mf6.modflow.mfgwflak`, 1391
- `flopy.mf6.modflow.mfgwfmaw`, 1403
- `flopy.mf6.modflow.mfgwfmvr`, 1414
- `flopy.mf6.modflow.mfgwfnam`, 1418
- `flopy.mf6.modflow.mfgwfnpf`, 1420
- `flopy.mf6.modflow.mfgwfoc`, 1427
- `flopy.mf6.modflow.mfgwfrch`, 1431
- `flopy.mf6.modflow.mfgwfrcha`, 1435
- `flopy.mf6.modflow.mfgwfriv`, 1438
- `flopy.mf6.modflow.mfgwfsfr`, 1442
- `flopy.mf6.modflow.mfgwfsto`, 1454
- `flopy.mf6.modflow.mfgwfuzf`, 1456
- `flopy.mf6.modflow.mfgwfvsc`, 1464
- `flopy.mf6.modflow.mfgwfwel`, 1467
- `flopy.mf6.modflow.mfgwt`, 1336
- `flopy.mf6.modflow.mfgwtadv`, 1472
- `flopy.mf6.modflow.mfgwtapi`, 1472
- `flopy.mf6.modflow.mfgwtcnc`, 1474
- `flopy.mf6.modflow.mfgwtdis`, 1477
- `flopy.mf6.modflow.mfgwtdisu`, 1480
- `flopy.mf6.modflow.mfgwtdisv`, 1485
- `flopy.mf6.modflow.mfgwt dsp`, 1488
- `flopy.mf6.modflow.mfgwtfmi`, 1490
- `flopy.mf6.modflow.mfgwtgt`, 1492
- `flopy.mf6.modflow.mfgwtic`, 1497
- `flopy.mf6.modflow.mfgwtist`, 1497
- `flopy.mf6.modflow.mfgwtlkt`, 1502
- `flopy.mf6.modflow.mfgwtmst`, 1508
- `flopy.mf6.modflow.mfgwtmvt`, 1510
- `flopy.mf6.modflow.mfgwtmwt`, 1512
- `flopy.mf6.modflow.mfgwt nam`, 1518
- `flopy.mf6.modflow.mfgwtoc`, 1519
- `flopy.mf6.modflow.mfgwtsft`, 1524
- `flopy.mf6.modflow.mfgwtsrc`, 1530
- `flopy.mf6.modflow.mfgwtssm`, 1533
- `flopy.mf6.modflow.mfgwtuzt`, 1535
- `flopy.mf6.modflow.mfims`, 1316
- `flopy.mf6.modflow.mfmvr`, 1326
- `flopy.mf6.modflow.mfnam`, 1330
- `flopy.mf6.modflow.mfsimulation`, 1301
- `flopy.mf6.modflow.mftdis`, 1332
- `flopy.mf6.modflow.mfutlats`, 1542
- `flopy.mf6.modflow.mfutllaktab`, 1544
- `flopy.mf6.modflow.mfutlobs`, 1546
- `flopy.mf6.modflow.mfutlsfrtab`, 1548
- `flopy.mf6.modflow.mfutlspc`, 1549

[flopy.mf6.modflow.mfutlspca](#), 1551
[flopy.mf6.modflow.mfutltas](#), 1552
[flopy.mf6.modflow.mfutlts](#), 1554
[flopy.mf6.modflow.mfutltvk](#), 1556
[flopy.mf6.modflow.mfutltvs](#), 1559
[flopy.mf6.utils.binaryfile_utils](#), 1562
[flopy.mf6.utils.binarygrid_util](#), 1563
[flopy.mf6.utils.createpackages](#), 1602
[flopy.mf6.utils.generate_classes](#), 1604
[flopy.mf6.utils.lakpak_utils](#), 1577
[flopy.mf6.utils.mfobservation](#), 1568
[flopy.mf6.utils.output_util](#), 1570
[flopy.mf6.utils.postprocessing](#), 1570
[flopy.mf6.utils.reference](#), 1571
[flopy.modflow.mf](#), 1617
[flopy.modflow.mfaddoutsidefile](#), 1619
[flopy.modflow.mfag](#), 1620
[flopy.modflow.mfbas](#), 1622
[flopy.modflow.mfbcf](#), 1624
[flopy.modflow.mfbct](#), 1627
[flopy.modflow.mfchd](#), 1627
[flopy.modflow.mfde4](#), 1629
[flopy.modflow.mfdis](#), 1632
[flopy.modflow.mfdrn](#), 1638
[flopy.modflow.mfdrt](#), 1640
[flopy.modflow.mfevt](#), 1643
[flopy.modflow.mffhb](#), 1645
[flopy.modflow.mfflwob](#), 1648
[flopy.modflow.mfgage](#), 1650
[flopy.modflow.mfghb](#), 1652
[flopy.modflow.mfgmg](#), 1654
[flopy.modflow.mfhfb](#), 1658
[flopy.modflow.mfhob](#), 1660
[flopy.modflow.mfhvd](#), 1664
[flopy.modflow.mflak](#), 1666
[flopy.modflow.mflmt](#), 1672
[flopy.modflow.mflpf](#), 1674
[flopy.modflow.mfmlt](#), 1677
[flopy.modflow.mfmnw1](#), 1679
[flopy.modflow.mfmnw2](#), 1680
[flopy.modflow.mfmnwi](#), 1693
[flopy.modflow.mfnwt](#), 1694
[flopy.modflow.mfoc](#), 1699
[flopy.modflow.mfpar](#), 1704
[flopy.modflow.mfparbc](#), 1706
[flopy.modflow.mfpbc](#), 1708
[flopy.modflow.mfpbcg](#), 1708
[flopy.modflow.mfpbcgn](#), 1710
[flopy.modflow.mfpks](#), 1714
[flopy.modflow.mfpval](#), 1716
[flopy.modflow.mfrch](#), 1717
[flopy.modflow.mfriv](#), 1721
[flopy.modflow.mfsfr2](#), 1724
[flopy.modflow.mfsip](#), 1734
[flopy.modflow.mfsor](#), 1736
[flopy.modflow.mfstr](#), 1737
[flopy.modflow.mfsub](#), 1742
[flopy.modflow.mfswi2](#), 1747
[flopy.modflow.mfswr1](#), 1751
[flopy.modflow.mfswt](#), 1752
[flopy.modflow.mfupw](#), 1757
[flopy.modflow.mfuzf1](#), 1760
[flopy.modflow.mfwel](#), 1766
[flopy.modflow.mfzon](#), 1769
[flopy.modpath.mp6](#), 1829
[flopy.modpath.mp6bas](#), 1831
[flopy.modpath.mp6sim](#), 1833
[flopy.modpath.mp7](#), 1815
[flopy.modpath.mp7bas](#), 1817
[flopy.modpath.mp7particledata](#), 1818
[flopy.modpath.mp7particlegroup](#), 1823
[flopy.modpath.mp7sim](#), 1825
[flopy.mt3d.mt](#), 1771
[flopy.mt3d.mtadv](#), 1774
[flopy.mt3d.mtbtn](#), 1777
[flopy.mt3d.mtcts](#), 1781
[flopy.mt3d.mtdsp](#), 1784
[flopy.mt3d.mtgcg](#), 1787
[flopy.mt3d.mtlkt](#), 1788
[flopy.mt3d.mtphc](#), 1791
[flopy.mt3d.mtrct](#), 1791
[flopy.mt3d.mtsft](#), 1795
[flopy.mt3d.mtssm](#), 1800
[flopy.mt3d.mttob](#), 1803
[flopy.mt3d.mtuzt](#), 1803
[flopy.pakbase](#), 1614
[flopy.pest.params](#), 2013
[flopy.pest.templatewriter](#), 2014
[flopy.pest.tplarray](#), 2014
[flopy.plot.crosssection](#), 1968
[flopy.plot.map](#), 1975
[flopy.plot.plotutil](#), 1980
[flopy.plot.styles](#), 1986
[flopy.seawat.swt](#), 1806
[flopy.seawat.swtvdf](#), 1808
[flopy.seawat.swtvsc](#), 1812
[flopy.utils.binaryfile](#), 1835
[flopy.utils.check](#), 1844
[flopy.utils.compare](#), 1847
[flopy.utils.crs](#), 1851
[flopy.utils.cvfdutil](#), 1853
[flopy.utils.datafile](#), 1855
[flopy.utils.datautil](#), 1858
[flopy.utils.flopy_io](#), 1862
[flopy.utils.formattedfile](#), 1865
[flopy.utils.geometry](#), 1867
[flopy.utils.geospatial_utils](#), 1871
[flopy.utils.get_modflow](#), 1873

`flopy.utils.gridgen`, 1874
`flopy.utils.gridintersect`, 1883
`flopy.utils.gridutil`, 1888
`flopy.utils.lgrutil`, 1889
`flopy.utils.mflistfile`, 1891
`flopy.utils.mfreadnam`, 1897
`flopy.utils.modpathfile`, 1900
`flopy.utils.mtlistfile`, 1908
`flopy.utils.observationfile`, 1909
`flopy.utils.optionblock`, 1913
`flopy.utils.parse_version`, 1915
`flopy.utils.postprocessing`, 1916
`flopy.utils.rasters`, 1919
`flopy.utils.reccarray_utils`, 1923
`flopy.utils.reference`, 1924
`flopy.utils.sfroutputfile`, 1925
`flopy.utils.swroutputfile`, 1926
`flopy.utils.triangle`, 1931
`flopy.utils.util_array`, 1935
`flopy.utils.util_list`, 1950
`flopy.utils.utils_def`, 1955
`flopy.utils.utl_import`, 1957
`flopy.utils.voronoi`, 1958
`flopy.utils.zonbud`, 1960

Symbols

`_float_characters` (*MFSimulationData* attribute), 1311
`_float_precision` (*MFSimulationData* attribute), 1310

A

`acdd` (class in *flopy.export.metadata*), 1990
`add_active_domain()` (*Gridgen* method), 1875
`add_annotation()` (*styles* class method), 1986
`add_array()` (*Vtk* method), 2005
`add_cell_budget()` (*Vtk* method), 2005
`add_data_item()` (*MFBBlockHeader* method), 1296
`add_dataset()` (*MFBBlock* method), 1295
`add_existing_package()` (*BaseModel* method), 1605
`add_ext_file()` (*MFFileMgmt* method), 1280
`add_external()` (*BaseModel* method), 1606
`add_file()` (*FileData* method), 1611
`add_global_attributes()` (*NetCDF* method), 1992
`add_heads()` (*Vtk* method), 2006
`add_hole()` (*Triangle* method), 1931
`add_model()` (*Vtk* method), 2006
`add_one()` (*MFSScalar* method), 1597
`add_one()` (*MFSScalarTransient* method), 1599
`add_output()` (*BaseModel* method), 1606
`add_output_file()` (*BaseModel* method), 1606
`add_package()` (*BaseModel* method), 1606
`add_package()` (*Vtk* method), 2006
`add_package()` (*ZoneBudget6* method), 1964
`add_parameter()` (*Transient2dTpl* method), 2014
`add_parameter()` (*Util3dTpl* method), 2014
`add_pathline_points()` (*Vtk* method), 2006
`add_polygon()` (*Triangle* method), 1931
`add_pop_key_list()` (*BaseModel* method), 1606
`add_record()` (*MfList* method), 1951
`add_record()` (*ModflowChd* method), 1628
`add_record()` (*ModflowDrn* method), 1639

`add_record()` (*ModflowDrt* method), 1642
`add_record()` (*ModflowGhb* method), 1653
`add_record()` (*ModflowRiv* method), 1722
`add_record()` (*ModflowWel* method), 1768
`add_refinement_features()` (*Gridgen* method), 1875
`add_region()` (*Triangle* method), 1931
`add_sciencebase_metadata()` (*NetCDF* method), 1992
`add_text()` (*styles* class method), 1987
`add_timevalue()` (*Pvd* method), 2004
`add_to_dtype()` (*Package* static method), 1614
`add_transient_array()` (*Vtk* method), 2007
`add_transient_key()` (*MFSScalarTransient* method), 1599
`add_transient_key()` (*MFTTransientArray* method), 1583
`add_transient_key()` (*MFTTransientList* method), 1592
`add_transient_list()` (*Vtk* method), 2007
`add_transient_vector()` (*Vtk* method), 2007
`add_var()` (in module *flopy.mf6.utils.createpackages*), 1603
`add_vector()` (*Vtk* method), 2007
`advanced_package_bc_helper()` (in module *flopy.plot.plotutil*), 1982
`afrcsv_filerecord` (*ModflowGwfwel* attribute), 1470
`alh` (*ModflowGwtdsp* attribute), 1490
`all()` (*Util2d* method), 1943
`alv` (*ModflowGwtdsp* attribute), 1490
`angldegx` (*ModflowGwfdisu* attribute), 1359
`angldegx` (*ModflowGwtdisu* attribute), 1484
`angle1` (*ModflowGwfnpf* attribute), 1425
`angle2` (*ModflowGwfnpf* attribute), 1425
`angle3` (*ModflowGwfnpf* attribute), 1425
`angrot` (*Grid* attribute), 2016
`angrot` (*Grid* property), 2018
`angrot` (*MfGrdFile* property), 1563
`angrot_radians` (*Grid* attribute), 2016

angrot_radians (*Grid* property), 2018
 append() (*MfList* method), 1951
 append() (*NetCdf* method), 1993
 append_data() (*MfList* method), 1587
 append_list_as_record() (*MfList* method), 1587
 append_list_as_record() (*MfTransientList* method), 1592
 append_package() (*GncPackages* method), 1313
 append_package() (*GwfgncPackages* method), 1380
 append_package() (*GwfmvrPackages* method), 1414
 append_package() (*GwtmvtPackages* method), 1510
 append_package() (*MvrPackages* method), 1330
 append_package() (*UtlatsPackages* method), 1544
 append_package() (*UtltasPackages* method), 1553, 1554
 append_package() (*UtltsPackages* method), 1556
 append_package() (*UtltvkPackages* method), 1558, 1559
 append_package() (*Utltvspackages* method), 1561
 append_passed() (*check* method), 1845
 arctan2() (*UnstructuredPlotUtilities* static method), 1981
 area (*ModflowGwfdisu* attribute), 1359
 area (*ModflowGwtdisu* attribute), 1484
 area_of_polygon() (in module *flopy.utils.cvfdutil*), 1853
 array (*MfList* property), 1951
 array (*Transient2d* property), 1938
 array (*Transient3d* property), 1941
 array (*Util2d* attribute), 1942
 array (*Util2d* property), 1943
 array (*Util3d* attribute), 1948
 array (*Util3d* property), 1948
 array2d_export() (in module *flopy.export.utils*), 1998
 array2string() (*Util2d* static method), 1943
 array3d_export() (in module *flopy.export.utils*), 1999
 array_at_faces() (*StructuredGrid* method), 2024
 array_at_faces_1d() (in module *flopy.discretization.structuredgrid*), 2030
 array_at_verts() (*StructuredGrid* method), 2025
 array_at_verts_basic() (*StructuredGrid* method), 2025
 array_at_verts_basic2d() (in module *flopy.discretization.structuredgrid*), 2031
 array_comp() (*PyListUtil* method), 1861
 array_free_format (*ArrayFormat* property), 1937
 ArrayFormat (class in *flopy.utils.util_array*), 1935
 ArrayIndexIter (class in *flopy.utils.datautil*), 1858
 assign_layers() (*ModflowSfr2* method), 1729
 ath1 (*ModflowGwtdsp* attribute), 1490
 ath2 (*ModflowGwtdsp* attribute), 1490
 ats_filerecord (*ModflowTdis* attribute), 1333
 attribs_from_namfile_header() (*Grid* method), 2018
 attribs_from_namfile_header() (in module *flopy.utils.mfreadnam*), 1898
 attribute_by_kper() (*MfList* method), 1951
 attribute_dict (*StructuredSpatialReference* property), 1573
 atv (*ModflowGwtdsp* attribute), 1490
 aux (*ModflowGwfevta* attribute), 1375
 aux (*ModflowGwfrcha* attribute), 1437
 auxiliary (*ModflowGwfchd* attribute), 1343
 auxiliary (*ModflowGwfdrn* attribute), 1366
 auxiliary (*ModflowGwfevt* attribute), 1371
 auxiliary (*ModflowGwfevta* attribute), 1375
 auxiliary (*ModflowGwfgfb* attribute), 1378
 auxiliary (*ModflowGwfgwf* attribute), 1386
 auxiliary (*ModflowGwflak* attribute), 1401
 auxiliary (*ModflowGwfmaw* attribute), 1412
 auxiliary (*ModflowGwfrch* attribute), 1433
 auxiliary (*ModflowGwfrcha* attribute), 1437
 auxiliary (*ModflowGwfriv* attribute), 1440
 auxiliary (*ModflowGwfsfr* attribute), 1451
 auxiliary (*ModflowGwfufz* attribute), 1462
 auxiliary (*ModflowGwfwel* attribute), 1470
 auxiliary (*ModflowGwtcnc* attribute), 1476
 auxiliary (*ModflowGwtgwt* attribute), 1495
 auxiliary (*ModflowGwtlkt* attribute), 1506
 auxiliary (*ModflowGwtmwt* attribute), 1516
 auxiliary (*ModflowGwtsft* attribute), 1528
 auxiliary (*ModflowGwtsrc* attribute), 1532
 auxiliary (*ModflowGwtuzt* attribute), 1539

B

backup_existing_dfn() (in module *flopy.mf6.utils.generate_classes*), 1604
 backward (*trackDir* attribute), 1829
 bands (*Raster* property), 1920
 base_version (*LegacyVersion* property), 1915

base_version (Version property), 1916
 BaseModel (class in *flopy.mbase*), 1605
 bc_stage_names (check attribute), 1845
 binary (ArrayFormat attribute), 1936
 binary (ArrayFormat property), 1937
 binary (MfList property), 1951
 BinaryHeader (class in *flopy.utils.binaryfile*), 1835
 BinaryLayerFile (class in *flopy.utils.binaryfile*), 1835
 binaryread() (in module *flopy.utils.binaryfile*), 1843
 binaryread_struct() (in module *flopy.utils.binaryfile*), 1843
 block_headers (MFBBlock attribute), 1294
 blocks (MFPackage attribute), 1297
 bot (MfGrdFile property), 1563
 bot (ModflowGwfdisu attribute), 1359
 bot (ModflowGwtdisu attribute), 1484
 botm (Grid attribute), 2016
 botm (Grid property), 2018
 botm (ModflowGwfdis attribute), 1355
 botm (ModflowGwfdisv attribute), 1363
 botm (ModflowGwtdis attribute), 1479
 botm (ModflowGwtdisv attribute), 1487
 bounds (acdd property), 1990
 bounds (Collection property), 1867
 bounds (LineString property), 1868
 bounds (Point property), 1869
 bounds (Polygon property), 1869
 bounds (Raster property), 1920
 budget_filerecord (ModflowGwflak attribute), 1401
 budget_filerecord (ModflowGwfmaw attribute), 1412
 budget_filerecord (ModflowGwfmvr attribute), 1417
 budget_filerecord (ModflowGwfoc attribute), 1429
 budget_filerecord (ModflowGwfsfr attribute), 1451
 budget_filerecord (ModflowGwfuzf attribute), 1462
 budget_filerecord (ModflowGwtist attribute), 1500
 budget_filerecord (ModflowGwtlkt attribute), 1506
 budget_filerecord (ModflowGwtmvt attribute), 1511
 budget_filerecord (ModflowGwtmwt attribute), 1516
 budget_filerecord (ModflowGwtoc attribute), 1522
 budget_filerecord (ModflowGwtsft attribute), 1528
 budget_filerecord (ModflowGwtuzt attribute), 1540
 budget_filerecord (ModflowMvr attribute), 1328
 BudgetIndexError, 1836
 budgetOpt (class in *flopy.modpath.mp7sim*), 1828
 build() (Gridgen method), 1875
 build() (Triangle method), 1932
 build_2d_instances() (Util3d method), 1948
 build_child_package() (MFPackage method), 1298
 build_child_packages_container() (MFPackage method), 1298
 build_dfn_string() (in module *flopy.mf6.utils.createpackages*), 1603
 build_doc_string() (in module *flopy.mf6.utils.createpackages*), 1603
 build_header_variables() (MFBBlockHeader method), 1296
 build_init_string() (in module *flopy.mf6.utils.createpackages*), 1603
 attribute), 1528
 budget_filerecord (ModflowGwtuzt attribute), 1540
 budget_filerecord (ModflowMvr attribute), 1328
 budgetcsv_filerecord (ModflowGwflak attribute), 1401
 budgetcsv_filerecord (ModflowGwfmaw attribute), 1412
 budgetcsv_filerecord (ModflowGwfmvr attribute), 1417
 budgetcsv_filerecord (ModflowGwfoc attribute), 1429
 budgetcsv_filerecord (ModflowGwfsfr attribute), 1451
 budgetcsv_filerecord (ModflowGwfuzf attribute), 1462
 budgetcsv_filerecord (ModflowGwtist attribute), 1500
 budgetcsv_filerecord (ModflowGwtlkt attribute), 1506
 budgetcsv_filerecord (ModflowGwtmvt attribute), 1511
 budgetcsv_filerecord (ModflowGwtmwt attribute), 1516
 budgetcsv_filerecord (ModflowGwtoc attribute), 1522
 budgetcsv_filerecord (ModflowGwtsft attribute), 1528
 budgetcsv_filerecord (ModflowGwtuzt attribute), 1540
 budgetcsv_filerecord (ModflowMvr attribute), 1328
 BudgetIndexError, 1836
 budgetOpt (class in *flopy.modpath.mp7sim*), 1828
 build() (Gridgen method), 1875
 build() (Triangle method), 1932
 build_2d_instances() (Util3d method), 1948
 build_child_package() (MFPackage method), 1298
 build_child_packages_container() (MFPackage method), 1298
 build_dfn_string() (in module *flopy.mf6.utils.createpackages*), 1603
 build_doc_string() (in module *flopy.mf6.utils.createpackages*), 1603
 build_header_variables() (MFBBlockHeader method), 1296
 build_init_string() (in module *flopy.mf6.utils.createpackages*), 1603

[build_list\(\)](#) (*MultiList* method), 1859
[build_mfdata\(\)](#) (*MFPackage* method), 1298
[build_model_init_vars\(\)](#) (in module *flopy.mf6.utils.createpackages*), 1603
[build_model_load\(\)](#) (in module *flopy.mf6.utils.createpackages*), 1603
[build_transient_sequence\(\)](#) (*Transient2d* method), 1938
[build_transient_sequence\(\)](#) (*Transient3d* method), 1941
[bulk_density](#) (*ModflowGwtist* attribute), 1500
[bulk_density](#) (*ModflowGwtmst* attribute), 1509
[by_node_variables](#) (*Mnw* attribute), 1686

C

[CachedData](#) (class in *flopy.discretization.grid*), 2015
[calc_conc\(\)](#) (*SwiConcentration* method), 1981
[cell1d](#) (*VertexGrid* property), 2038
[cell2d](#) (*MfGrdFile* property), 1564
[cell2d](#) (*ModflowGwfdisu* attribute), 1359
[cell2d](#) (*ModflowGwfdisv* attribute), 1363
[cell2d](#) (*ModflowGwtdisu* attribute), 1484
[cell2d](#) (*ModflowGwtdisv* attribute), 1487
[cell2d](#) (*VertexGrid* property), 2038
[cell_thickness](#) (*Grid* property), 2018
[CellBudgetFile](#) (class in *flopy.utils.binaryfile*), 1836
[cellcenters](#) (*MfGrdFile* property), 1564
[CellDataType](#) (class in *flopy.modpath.mp7particledata*), 1818
[centered_specific_discharge\(\)](#) (*PlotUtilities* static method), 1980
[centroid_of_polygon\(\)](#) (in module *flopy.utils.cvfduutil*), 1853
[cg_ske_cr](#) (*ModflowGwfsub* attribute), 1351
[cg_theta](#) (*ModflowGwfsub* attribute), 1351
[change_model_name\(\)](#) (*ZoneBudget6* method), 1964
[change_model_ws\(\)](#) (*BaseModel* method), 1607
[change_model_ws\(\)](#) (*Seawat* method), 1807
[change_model_ws\(\)](#) (*ZoneBudget6* method), 1965
[check](#) (class in *flopy.modflow.mfsfr2*), 1733
[check](#) (class in *flopy.utils.check*), 1844
[check\(\)](#) (*BaseModel* method), 1607
[check\(\)](#) (*MFMModel* method), 1287
[check\(\)](#) (*MFPackage* method), 1298
[check\(\)](#) (*MFSimulation* method), 1302

[check\(\)](#) (*Mnw* method), 1686
[check\(\)](#) (*ModelInterface* method), 1612
[check\(\)](#) (*ModflowBas* method), 1623
[check\(\)](#) (*ModflowDis* method), 1633
[check\(\)](#) (*ModflowMnw2* method), 1690
[check\(\)](#) (*ModflowMnwi* method), 1694
[check\(\)](#) (*ModflowOc* method), 1701
[check\(\)](#) (*ModflowRch* method), 1719
[check\(\)](#) (*ModflowRiv* method), 1722
[check\(\)](#) (*ModflowSfr2* method), 1729
[check\(\)](#) (*Modpath6Sim* method), 1833
[check\(\)](#) (*PackageInterface* method), 1616
[check_kij\(\)](#) (*MfList* method), 1951
[checklayerthickness\(\)](#) (*ModflowDis* method), 1634
[child](#) (*Lgr* property), 1889
[cim](#) (*ModflowGwtist* attribute), 1500
[cim_filerecord](#) (*ModflowGwtist* attribute), 1500
[cimprintrecord](#) (*ModflowGwtist* attribute), 1500
[cl12](#) (*ModflowGwfdisu* attribute), 1359
[cl12](#) (*ModflowGwtdisu* attribute), 1484
[clean\(\)](#) (*Triangle* method), 1932
[clean_class_string\(\)](#) (in module *flopy.mf6.utils.createpackages*), 1603
[clean_filename\(\)](#) (in module *flopy.utils.datautil*), 1862
[clean_name\(\)](#) (in module *flopy.utils.datautil*), 1862
[clean_numeric\(\)](#) (*PyListUtil* static method), 1861
[close\(\)](#) (*CellBudgetFile* method), 1836
[close\(\)](#) (*FileIter* method), 1859
[close\(\)](#) (*FormattedLayerFile* method), 1866
[close\(\)](#) (*LayerFile* method), 1855
[cnstnt_str](#) (*Util2d* property), 1944
[Collection](#) (class in *flopy.utils.geometry*), 1867
[combined](#) (*simType* attribute), 1828
[comment](#) (*MFBBlockHeader* attribute), 1296
[compaction_coarse_filerecord](#) (*ModflowGwfsub* attribute), 1351
[compaction_elastic_filerecord](#) (*ModflowGwfsub* attribute), 1351
[compaction_filerecord](#) (*ModflowGwfsub* attribute), 1351
[compaction_inelastic_filerecord](#) (*ModflowGwfsub* attribute), 1351
[compaction_interbed_filerecord](#) (*ModflowGwfsub* attribute), 1351
[compare\(\)](#) (in module *flopy.utils.compare*), 1847

[compare_budget\(\)](#) (in module [flopy.utils.compare](#)), 1847
[compare_concentrations\(\)](#) (in module [flopy.utils.compare](#)), 1848
[compare_heads\(\)](#) (in module [flopy.utils.compare](#)), 1849
[compare_stages\(\)](#) (in module [flopy.utils.compare](#)), 1850
[compare_swrbudget\(\)](#) (in module [flopy.utils.compare](#)), 1850
[concentration](#) ([ModflowUtlspca](#) attribute), 1551
[concentration_filerecord](#) ([ModflowGwtlkt](#) attribute), 1506
[concentration_filerecord](#) ([ModflowGwtmwt](#) attribute), 1516
[concentration_filerecord](#) ([ModflowGwtoc](#) attribute), 1522
[concentration_filerecord](#) ([ModflowGwtsft](#) attribute), 1528
[concentration_filerecord](#) ([ModflowGwtuzt](#) attribute), 1540
[concentrationprintrecord](#) ([ModflowGwtoc](#) attribute), 1522
[connect_to_dict\(\)](#) ([MFBBlockHeader](#) method), 1296
[connectiondata](#) ([ModflowGwflak](#) attribute), 1401
[connectiondata](#) ([ModflowGwfmaw](#) attribute), 1412
[connectiondata](#) ([ModflowGwfsfr](#) attribute), 1451
[consistent_delim](#) ([PyListUtil](#) attribute), 1861
[const](#) ([ModflowSfr2](#) property), 1730
[ConstIter](#) (class in [flopy.utils.datautil](#)), 1858
[continuous](#) ([ModflowUtllobs](#) attribute), 1547
[contour_array\(\)](#) (in module [flopy.export.utils](#)), 1999
[contour_array\(\)](#) ([PlotCrossSection](#) method), 1969
[contour_array\(\)](#) ([PlotMapView](#) method), 1975
[copy\(\)](#) ([NetCdf](#) method), 1993
[copy\(\)](#) ([ZoneBudget](#) method), 1961
[copy_all](#) ([ExtFileAction](#) attribute), 1279
[copy_files\(\)](#) ([MFFileMgmt](#) method), 1280
[copy_none](#) ([ExtFileAction](#) attribute), 1279
[copy_relative_paths](#) ([ExtFileAction](#) attribute), 1279
[create\(\)](#) ([BinaryHeader](#) static method), 1835
[create_basic_init\(\)](#) (in module [flopy.mf6.utils.createpackages](#)), 1603
[create_empty_reccarray\(\)](#) (in module [flopy.utils.reccarray_utils](#)), 1923
[create_group_variable\(\)](#) ([NetCdf](#) method), 1993
[create_init_var\(\)](#) (in module [flopy.mf6.utils.createpackages](#)), 1604
[create_mp7\(\)](#) ([Modpath7](#) class method), 1816
[create_mpsim\(\)](#) ([Modpath6](#) method), 1830
[create_package_dimensions\(\)](#) ([MFPackage](#) method), 1298
[create_package_init_var\(\)](#) (in module [flopy.mf6.utils.createpackages](#)), 1604
[create_packages\(\)](#) (in module [flopy.mf6.utils.createpackages](#)), 1604
[create_property\(\)](#) (in module [flopy.mf6.utils.createpackages](#)), 1604
[create_variable\(\)](#) ([NetCdf](#) method), 1993
[creator](#) ([acdd](#) property), 1990
[creator_url](#) ([acdd](#) property), 1990
[crop\(\)](#) ([Raster](#) method), 1920
[cross_section_adjust_indicies\(\)](#) ([Grid](#) method), 2018
[cross_section_adjust_indicies\(\)](#) ([UnstructuredGrid](#) method), 2032
[cross_section_lay_ncpl_ncb\(\)](#) ([Grid](#) method), 2018
[cross_section_lay_ncpl_ncb\(\)](#) ([UnstructuredGrid](#) method), 2033
[cross_section_nodeskip\(\)](#) ([Grid](#) method), 2019
[cross_section_nodeskip\(\)](#) ([UnstructuredGrid](#) method), 2033
[cross_section_set_contour_arrays\(\)](#) ([Grid](#) method), 2019
[cross_section_set_contour_arrays\(\)](#) ([UnstructuredGrid](#) method), 2033
[cross_section_vertices](#) ([Grid](#) property), 2019
[cross_section_vertices](#) ([StructuredGrid](#) property), 2025
[cross_section_vertices](#) ([UnstructuredGrid](#) property), 2034
[crosssections](#) ([ModflowGwfsfr](#) attribute), 1452
[crs](#) ([Grid](#) attribute), 2016
[crs](#) ([Grid](#) property), 2019
[csv_inner_output_filerecord](#) ([ModflowIms](#) attribute), 1324
[csv_names](#) ([MF6Output](#) property), 1570
[csv_outer_output_filerecord](#) ([ModflowIms](#)

attribute), 1324
 csv_output_filerecord (ModflowIms
 attribute), 1324
 CsvFile (class in flopy.utils.observationfile), 1909
 cvfd_to_patch_collection() (in module
 flopy.plot.plotutil), 1982

D

data (CachedData property), 2015
 data (MFArray property), 1578
 data (MfList property), 1951
 data (MFScalar property), 1597
 data (MFTransientList property), 1592
 data_factory() (MFBlock method), 1295
 data_items (MFBlockHeader attribute), 1296
 data_list (MFPackage property), 1298
 data_list (Package property), 1614
 data_list (PackageInterface property), 1617
 data_nocopy (CachedData property), 2015
 data_type (MFArray property), 1578
 data_type (MfList property), 1587
 data_type (MfList property), 1951
 data_type (MFScalar property), 1597
 data_type (MFScalarTransient property),
 1599
 data_type (MFTransientArray property), 1583
 data_type (MFTransientList property), 1593
 data_type (Transient2d property), 1938
 data_type (Transient3d property), 1941
 data_type (Util2d property), 1944
 data_type (Util3d property), 1948
 dataframe_to_netcdf_fmt() (in module
 flopy.utils.zonbud), 1967
 dataset_5 (ModflowSfr2 property), 1730
 datasets (MFBlock attribute), 1294
 datasets_keyword (MFBlock attribute), 1294
 DatumUtil (class in flopy.utils.datautil),
 1858
 deactivate_ibound_above() (ModflowSfr2
 method), 1730
 decay (ModflowGwtist attribute), 1500
 decay (ModflowGwtmst attribute), 1509
 decay_sorbed (ModflowGwtist attribute),
 1500
 decay_sorbed (ModflowGwtmst attribute),
 1510
 decimal (ArrayFormat attribute), 1936
 decimal (ArrayFormat property), 1937
 decode_fortran_descriptor() (ArrayFormat
 static method), 1937
 default_value (ModflowSfr2 attribute), 1730
 defaults (TemporalReference attribute),
 1924

delc (MfGrdFile property), 1564
 delc (ModflowGwfdis attribute), 1355
 delc (ModflowGwtdis attribute), 1479
 delc (StructuredGrid property), 2025
 delete_files() (in module
 flopy.mf6.utils.generate_classes),
 1604
 delete_mf6_classes() (in module
 flopy.mf6.utils.generate_classes),
 1604
 delete_output_files() (MFSimulation
 method), 1303
 delimiter_list (PyListUtil attribute), 1861
 delimiter_used (PyListUtil attribute), 1861
 delr (MfGrdFile property), 1564
 delr (ModflowGwfdis attribute), 1355
 delr (ModflowGwtdis attribute), 1479
 delr (StructuredGrid property), 2026
 delz (StructuredGrid property), 2026
 density_filerecord (ModflowGwfbuy
 attribute), 1341
 depth (ModflowGwfevt attribute), 1375
 dev (LegacyVersion property), 1915
 dev (Version property), 1916
 df (MfList property), 1951
 df (ModflowSfr2 property), 1730
 df (SfrFile property), 1925
 dfn (ModflowGnc attribute), 1315
 dfn (ModflowGwfapi attribute), 1338
 dfn (ModflowGwfbuy attribute), 1341
 dfn (ModflowGwfchd attribute), 1343
 dfn (ModflowGwfcsb attribute), 1351
 dfn (ModflowGwfdis attribute), 1355
 dfn (ModflowGwfdisu attribute), 1359
 dfn (ModflowGwfdisv attribute), 1363
 dfn (ModflowGwfdrn attribute), 1366
 dfn (ModflowGwfevt attribute), 1371
 dfn (ModflowGwfevt attribute), 1375
 dfn (ModflowGwfgfb attribute), 1378
 dfn (ModflowGwfgnc attribute), 1382
 dfn (ModflowGwfgwf attribute), 1386
 dfn (ModflowGwfgwt attribute), 1388
 dfn (ModflowGwfhfb attribute), 1390
 dfn (ModflowGwfic attribute), 1391
 dfn (ModflowGwflak attribute), 1401
 dfn (ModflowGwfmaw attribute), 1412
 dfn (ModflowGwfmvr attribute), 1417
 dfn (ModflowGwfnam attribute), 1419
 dfn (ModflowGwfnpf attribute), 1425
 dfn (ModflowGwfoc attribute), 1429
 dfn (ModflowGwfrch attribute), 1433
 dfn (ModflowGwfrcha attribute), 1437
 dfn (ModflowGwfriv attribute), 1440
 dfn (ModflowGwfsfr attribute), 1452

Index 2053

`dfn_file_name` (*ModflowGwtdisu* attribute), 1484
`dfn_file_name` (*ModflowGwtdisv* attribute), 1488
`dfn_file_name` (*ModflowGwtdsp* attribute), 1490
`dfn_file_name` (*ModflowGwtfmi* attribute), 1491
`dfn_file_name` (*ModflowGwtgwt* attribute), 1496
`dfn_file_name` (*ModflowGwtic* attribute), 1497
`dfn_file_name` (*ModflowGwtist* attribute), 1501
`dfn_file_name` (*ModflowGwtlkt* attribute), 1508
`dfn_file_name` (*ModflowGwtmst* attribute), 1510
`dfn_file_name` (*ModflowGwtmvt* attribute), 1512
`dfn_file_name` (*ModflowGwtmwt* attribute), 1518
`dfn_file_name` (*ModflowGwtnam* attribute), 1519
`dfn_file_name` (*ModflowGwtoc* attribute), 1523
`dfn_file_name` (*ModflowGwtsft* attribute), 1530
`dfn_file_name` (*ModflowGwtsrc* attribute), 1533
`dfn_file_name` (*ModflowGwtssm* attribute), 1535
`dfn_file_name` (*ModflowGwtuzt* attribute), 1542
`dfn_file_name` (*ModflowIms* attribute), 1326
`dfn_file_name` (*ModflowMvr* attribute), 1329
`dfn_file_name` (*ModflowNam* attribute), 1332
`dfn_file_name` (*ModflowTdis* attribute), 1334
`dfn_file_name` (*ModflowUtlats* attribute), 1544
`dfn_file_name` (*ModflowUtlaktab* attribute), 1545
`dfn_file_name` (*ModflowUtlobs* attribute), 1547
`dfn_file_name` (*ModflowUtlsfrtab* attribute), 1549
`dfn_file_name` (*ModflowUtlspc* attribute), 1550
`dfn_file_name` (*ModflowUtlspca* attribute), 1552
`dfn_file_name` (*ModflowUtltas* attribute), 1553
`dfn_file_name` (*ModflowUtlts* attribute), 1555

`dfn_file_name` (*ModflowUtltvk* attribute), 1558
`dfn_file_name` (*ModflowUtltvsv* attribute), 1561
`diffc` (*ModflowGwtdsp* attribute), 1490
`difference()` (*NetCdf* method), 1993
`dimensions` (*MFPackage* attribute), 1298
`distcoef` (*ModflowGwtist* attribute), 1501
`distcoef` (*ModflowGwtmst* attribute), 1510
`diversions` (*ModflowGwfsfr* attribute), 1454
`download_dfn()` (in module `flopy.mf6.utils.generate_classes`), 1604
`drop()` (*MfList* method), 1951
`dtype` (*MfArray* property), 1578
`dtype` (*MfList* property), 1587
`dtype` (*MfList* property), 1951
`dtype` (*MfScalar* property), 1597
`dtype` (*MfTransientList* property), 1593
`dtype` (*OptionBlock* attribute), 1913
`dtype` (*Transient2d* property), 1938
`dtype` (*Transient3d* property), 1941
`dtype` (*Util2d* property), 1944
`dtype` (*Util3d* property), 1948
`dtypes` (*SfrFile* attribute), 1925

E

`elements()` (*MultiList* method), 1860
`elevations()` (*check* method), 1733
`empty_like()` (*NetCdf* class method), 1994
`enabled` (*MFBBlock* attribute), 1294
`endpoint` (*simType* attribute), 1828
`EndpointFile` (class in `flopy.utils.modpathfile`), 1900
`enforce_10ch_limit()` (in module `flopy.export.shapefile_utils`), 1995
`ensemble_helper()` (in module `flopy.export.utils`), 1999
`epoch` (*LegacyVersion* property), 1915
`epoch` (*Version* property), 1916
`epsg` (*Grid* property), 2019
`eval_bud_diff()` (in module `flopy.utils.compare`), 1851
`exchange_files` (*MFSimulation* property), 1303
`exchangedata` (*ModflowGwfgwf* attribute), 1387
`exchangedata` (*ModflowGwtgwt* attribute), 1496
`exchanges` (*ModflowNam* attribute), 1332
`exe_name` (*MfModel* attribute), 1287
`exename` (*BaseModel* property), 1607
`exename` (*MfModel* property), 1287
`exename` (*ModelInterface* property), 1612

export() (*BaseModel* method), 1607
 export() (*Gridgen* method), 1875
 export() (*MfList* method), 1951
 export() (*MfModel* method), 1287
 export() (*MFPackage* method), 1298
 export() (*ModelInterface* method), 1612
 export() (*ModflowMnw2* method), 1690
 export() (*ModflowSfr2* method), 1730
 export() (*Package* method), 1614
 export() (*PackageInterface* method), 1617
 export() (*Transient2d* method), 1938
 export() (*Util2d* method), 1944
 export() (*Util3d* method), 1948
 export() (*ZoneBudget* method), 1961
 export() (*ZoneBudget6* method), 1965
 export_array() (in module *flopy.export.utils*), 1999
 export_array() (in module *flopy.export.vtk*), 2008
 export_array_contours() (in module *flopy.export.utils*), 2000
 export_cbc() (in module *flopy.export.vtk*), 2008
 export_contourf() (in module *flopy.export.utils*), 2000
 export_contours() (in module *flopy.export.utils*), 2001
 export_heads() (in module *flopy.export.vtk*), 2009
 export_linkages() (*ModflowSfr2* method), 1730
 export_model() (in module *flopy.export.vtk*), 2010
 export_outlets() (*ModflowSfr2* method), 1730
 export_package() (in module *flopy.export.vtk*), 2011
 export_transient() (in module *flopy.export.vtk*), 2012
 export_transient_variable() (*ModflowSfr2* method), 1730
 extend (*stopOpt* attribute), 1829
 extent (*Grid* property), 2019
 extent (*PlotMapView* property), 1975
 extent (*StructuredGrid* property), 2026
 extent (*UnstructuredGrid* property), 2034
 extent (*VertexGrid* property), 2038
 external_formatting (*MFSimulationData* attribute), 1310
 ExtFileAction (class in *flopy.mf6.mfbase*), 1279

F

FaceDataType (class in *flopy.modpath.mp7particledata*), 1819
 features_to_shapefile() (in module *flopy.utils.gridgen*), 1882
 fields_view() (in module *flopy.utils.check*), 1846
 FileData (class in *flopy.mbase*), 1611
 FileDataEntry (class in *flopy.mbase*), 1611
 filehandle (*NamData* attribute), 1897
 fileinput (*ModflowGwtssm* attribute), 1535
 FileIter (class in *flopy.utils.datautil*), 1859
 filename (*MFPackage* property), 1299
 filename (*NamData* attribute), 1897
 filename (*Util2d* property), 1944
 filetype (*NamData* attribute), 1897
 filter_modpath_by_travel_time() (in module *flopy.plot.plotutil*), 1982
 filter_query_result() (*GridIntersect* method), 1884
 find_in_path() (*SimulationDict* method), 1312
 find_keyword() (in module *flopy.utils.datautil*), 1862
 find_path() (in module *flopy.modflow.mfsfr2*), 1733
 find_position_in_array() (*ModflowGridIndices* static method), 1886
 first_index() (*MultiList* method), 1860
 first_item() (*MultiList* method), 1860
 first_item() (*PyListUtil* static method), 1861
 float() (*ArrayFormat* class method), 1937
 FLOAT32 (*Raster* attribute), 1920
 FLOAT64 (*Raster* attribute), 1920
 float_characters (*MFSimulationData* property), 1311
 float_precision (*MFSimulationData* property), 1311
 flopy.discretization.grid module, 2015
 flopy.discretization.modeltime module, 2023
 flopy.discretization.structuredgrid module, 2023
 flopy.discretization.unstructuredgrid module, 2031
 flopy.discretization.vertexgrid module, 2037
 flopy.export.longnames module, 1990
 flopy.export.metadata module, 1990
 flopy.export.netcdf

- module, [1991](#)
- flopy.export.shapefile_utils
 - module, [1995](#)
- flopy.export.unitsformat
 - module, [1998](#)
- flopy.export.utils
 - module, [1998](#)
- flopy.export.vtk
 - module, [2004](#)
- flopy.mbase
 - module, [1605](#)
- flopy.mf6.data.mfdataarray
 - module, [1578](#)
- flopy.mf6.data.mfdataalist
 - module, [1587](#)
- flopy.mf6.data.mfdatascalar
 - module, [1597](#)
- flopy.mf6.mfbase
 - module, [1279](#)
- flopy.mf6.mfmodel
 - module, [1286](#)
- flopy.mf6.mfpackage
 - module, [1294](#)
- flopy.mf6.modflow.mfgnc
 - module, [1313](#)
- flopy.mf6.modflow.mfgwf
 - module, [1334](#)
- flopy.mf6.modflow.mfgwfapi
 - module, [1337](#)
- flopy.mf6.modflow.mfgwfbuy
 - module, [1339](#)
- flopy.mf6.modflow.mfgwfchd
 - module, [1341](#)
- flopy.mf6.modflow.mfgwfcsb
 - module, [1345](#)
- flopy.mf6.modflow.mfgwfdis
 - module, [1353](#)
- flopy.mf6.modflow.mfgwfdisu
 - module, [1355](#)
- flopy.mf6.modflow.mfgwfdisv
 - module, [1361](#)
- flopy.mf6.modflow.mfgwfdrn
 - module, [1364](#)
- flopy.mf6.modflow.mfgwfevt
 - module, [1368](#)
- flopy.mf6.modflow.mfgwfevta
 - module, [1373](#)
- flopy.mf6.modflow.mfgwfgfb
 - module, [1376](#)
- flopy.mf6.modflow.mfgwfgnc
 - module, [1380](#)
- flopy.mf6.modflow.mfgwfgwf
 - module, [1382](#)
- flopy.mf6.modflow.mfgwfgwt

- module, [1388](#)
- flopy.mf6.modflow.mfgwfhib
 - module, [1389](#)
- flopy.mf6.modflow.mfgwfic
 - module, [1390](#)
- flopy.mf6.modflow.mfgwflak
 - module, [1391](#)
- flopy.mf6.modflow.mfgwfmau
 - module, [1403](#)
- flopy.mf6.modflow.mfgwfmvr
 - module, [1414](#)
- flopy.mf6.modflow.mfgwfnam
 - module, [1418](#)
- flopy.mf6.modflow.mfgwfnpf
 - module, [1420](#)
- flopy.mf6.modflow.mfgwfoc
 - module, [1427](#)
- flopy.mf6.modflow.mfgwfrch
 - module, [1431](#)
- flopy.mf6.modflow.mfgwfrcha
 - module, [1435](#)
- flopy.mf6.modflow.mfgwfriv
 - module, [1438](#)
- flopy.mf6.modflow.mfgwfsfr
 - module, [1442](#)
- flopy.mf6.modflow.mfgwfsto
 - module, [1454](#)
- flopy.mf6.modflow.mfgwfuzf
 - module, [1456](#)
- flopy.mf6.modflow.mfgwfvsc
 - module, [1464](#)
- flopy.mf6.modflow.mfgwfwel
 - module, [1467](#)
- flopy.mf6.modflow.mfgwt
 - module, [1336](#)
- flopy.mf6.modflow.mfgwtadv
 - module, [1472](#)
- flopy.mf6.modflow.mfgwtapi
 - module, [1472](#)
- flopy.mf6.modflow.mfgwtcnc
 - module, [1474](#)
- flopy.mf6.modflow.mfgwtdis
 - module, [1477](#)
- flopy.mf6.modflow.mfgwtdisu
 - module, [1480](#)
- flopy.mf6.modflow.mfgwtdisv
 - module, [1485](#)
- flopy.mf6.modflow.mfgwt dsp
 - module, [1488](#)
- flopy.mf6.modflow.mfgwt fmi
 - module, [1490](#)
- flopy.mf6.modflow.mfgwt gwt
 - module, [1492](#)
- flopy.mf6.modflow.mfgwtic

module, 1497
 flopy.mf6.modflow.mfgwtist
 module, 1497
 flopy.mf6.modflow.mfgwtlkt
 module, 1502
 flopy.mf6.modflow.mfgwtmst
 module, 1508
 flopy.mf6.modflow.mfgwtmvt
 module, 1510
 flopy.mf6.modflow.mfgwtmwt
 module, 1512
 flopy.mf6.modflow.mfgwtmam
 module, 1518
 flopy.mf6.modflow.mfgwtoc
 module, 1519
 flopy.mf6.modflow.mfgwtsft
 module, 1524
 flopy.mf6.modflow.mfgwtsrc
 module, 1530
 flopy.mf6.modflow.mfgwtssm
 module, 1533
 flopy.mf6.modflow.mfgwtuzt
 module, 1535
 flopy.mf6.modflow.mfims
 module, 1316
 flopy.mf6.modflow.mfmvr
 module, 1326
 flopy.mf6.modflow.mfnam
 module, 1330
 flopy.mf6.modflow.mfsimulation
 module, 1301
 flopy.mf6.modflow.mftdis
 module, 1332
 flopy.mf6.modflow.mfutlats
 module, 1542
 flopy.mf6.modflow.mfutllaktab
 module, 1544
 flopy.mf6.modflow.mfutlobs
 module, 1546
 flopy.mf6.modflow.mfutlsfrtab
 module, 1548
 flopy.mf6.modflow.mfutlspc
 module, 1549
 flopy.mf6.modflow.mfutlspca
 module, 1551
 flopy.mf6.modflow.mfutltas
 module, 1552
 flopy.mf6.modflow.mfutlts
 module, 1554
 flopy.mf6.modflow.mfutltvk
 module, 1556
 flopy.mf6.modflow.mfutltvs
 module, 1559
 flopy.mf6.utils.binaryfile_utils

module, 1562
 flopy.mf6.utils.binarygrid_util
 module, 1563
 flopy.mf6.utils.createpackages
 module, 1602
 flopy.mf6.utils.generate_classes
 module, 1604
 flopy.mf6.utils.lakpak_utils
 module, 1577
 flopy.mf6.utils.mfobservation
 module, 1568
 flopy.mf6.utils.output_util
 module, 1570
 flopy.mf6.utils.postprocessing
 module, 1570
 flopy.mf6.utils.reference
 module, 1571
 flopy.modflow.mf
 module, 1617
 flopy.modflow.mfaddoutsidefile
 module, 1619
 flopy.modflow.mfag
 module, 1620
 flopy.modflow.mfbas
 module, 1622
 flopy.modflow.mfbcf
 module, 1624
 flopy.modflow.mfbct
 module, 1627
 flopy.modflow.mfchd
 module, 1627
 flopy.modflow.mfde4
 module, 1629
 flopy.modflow.mfdis
 module, 1632
 flopy.modflow.mfdrn
 module, 1638
 flopy.modflow.mfdrt
 module, 1640
 flopy.modflow.mfevt
 module, 1643
 flopy.modflow.mffhb
 module, 1645
 flopy.modflow.mfflwob
 module, 1648
 flopy.modflow.mfgage
 module, 1650
 flopy.modflow.mfghb
 module, 1652
 flopy.modflow.mfgmg
 module, 1654
 flopy.modflow.mfhfb
 module, 1658
 flopy.modflow.mfhob

module, 1660	module, 1752
flopy.modflow.mfhyd	flopy.modflow.mfupw
module, 1664	module, 1757
flopy.modflow.mflak	flopy.modflow.mfuzf1
module, 1666	module, 1760
flopy.modflow.mflmt	flopy.modflow.mfwel
module, 1672	module, 1766
flopy.modflow.mflpf	flopy.modflow.mfzon
module, 1674	module, 1769
flopy.modflow.mfm1t	flopy.modpath.mp6
module, 1677	module, 1829
flopy.modflow.mfmnw1	flopy.modpath.mp6bas
module, 1679	module, 1831
flopy.modflow.mfmnw2	flopy.modpath.mp6sim
module, 1680	module, 1833
flopy.modflow.mfmnwi	flopy.modpath.mp7
module, 1693	module, 1815
flopy.modflow.mfnwt	flopy.modpath.mp7bas
module, 1694	module, 1817
flopy.modflow.mfoc	flopy.modpath.mp7particledata
module, 1699	module, 1818
flopy.modflow.mfpar	flopy.modpath.mp7particlegroup
module, 1704	module, 1823
flopy.modflow.mfparbc	flopy.modpath.mp7sim
module, 1706	module, 1825
flopy.modflow.mfpbc	flopy.mt3d.mt
module, 1708	module, 1771
flopy.modflow.mfpccg	flopy.mt3d.mtadv
module, 1708	module, 1774
flopy.modflow.mfpccgn	flopy.mt3d.mtbtn
module, 1710	module, 1777
flopy.modflow.mfpks	flopy.mt3d.mtcts
module, 1714	module, 1781
flopy.modflow.mfpval	flopy.mt3d.mtdsp
module, 1716	module, 1784
flopy.modflow.mfrch	flopy.mt3d.mtgccg
module, 1717	module, 1787
flopy.modflow.mfriv	flopy.mt3d.mtlkt
module, 1721	module, 1788
flopy.modflow.mfsfr2	flopy.mt3d.mtphc
module, 1724	module, 1791
flopy.modflow.mfsip	flopy.mt3d.mtrct
module, 1734	module, 1791
flopy.modflow.mfsor	flopy.mt3d.mtsft
module, 1736	module, 1795
flopy.modflow.mfstr	flopy.mt3d.mtssm
module, 1737	module, 1800
flopy.modflow.mfsub	flopy.mt3d.mttob
module, 1742	module, 1803
flopy.modflow.mfswi2	flopy.mt3d.mtu2t
module, 1747	module, 1803
flopy.modflow.mfswr1	flopy.pakbase
module, 1751	module, 1614
flopy.modflow.mfswt	flopy.pest.params

- module, 2013
- flopy.pest.templatewriter
 - module, 2014
- flopy.pest.tplarray
 - module, 2014
- flopy.plot.crossection
 - module, 1968
- flopy.plot.map
 - module, 1975
- flopy.plot.plotutil
 - module, 1980
- flopy.plot.styles
 - module, 1986
- flopy.seawat.swt
 - module, 1806
- flopy.seawat.swtvdf
 - module, 1808
- flopy.seawat.swtvsc
 - module, 1812
- flopy.utils.binaryfile
 - module, 1835
- flopy.utils.check
 - module, 1844
- flopy.utils.compare
 - module, 1847
- flopy.utils.crs
 - module, 1851
- flopy.utils.cvfdutil
 - module, 1853
- flopy.utils.datafile
 - module, 1855
- flopy.utils.datautil
 - module, 1858
- flopy.utils.flopy_io
 - module, 1862
- flopy.utils.formattedfile
 - module, 1865
- flopy.utils.geometry
 - module, 1867
- flopy.utils.geospatial_utils
 - module, 1871
- flopy.utils.get_modflow
 - module, 1873
- flopy.utils.gridgen
 - module, 1874
- flopy.utils.gridintersect
 - module, 1883
- flopy.utils.gridutil
 - module, 1888
- flopy.utils.lgrutil
 - module, 1889
- flopy.utils.mflistfile
 - module, 1891
- flopy.utils.mfreadnam
 - module, 1897
- flopy.utils.modpathfile
 - module, 1900
- flopy.utils.mtlistfile
 - module, 1908
- flopy.utils.observationfile
 - module, 1909
- flopy.utils.optionblock
 - module, 1913
- flopy.utils.parse_version
 - module, 1915
- flopy.utils.postprocessing
 - module, 1916
- flopy.utils.rasters
 - module, 1919
- flopy.utils.reccarray_utils
 - module, 1923
- flopy.utils.reference
 - module, 1924
- flopy.utils.sfroutputfile
 - module, 1925
- flopy.utils.swroutputfile
 - module, 1926
- flopy.utils.triangle
 - module, 1931
- flopy.utils.util_array
 - module, 1935
- flopy.utils.util_list
 - module, 1950
- flopy.utils.utils_def
 - module, 1955
- flopy.utils.utl_import
 - module, 1957
- flopy.utils.voronoi
 - module, 1958
- flopy.utils.zonbud
 - module, 1960
- flopy_geometry (*GeoSpatialCollection* property), 1872
- flopy_geometry (*GeoSpatialUtil* property), 1872
- FlopyBinaryData (*class in flopy.utils.utils_def*), 1955
- FlopyException, 1279
- flux_to_wel() (*in module flopy.utils.flopy_io*), 1862
- fmt_string (*MfList* property), 1951
- for_nans() (*check method*), 1733
- format (*ArrayFormat* attribute), 1936
- format (*ArrayFormat* property), 1937
- format (*Util2d* attribute), 1943
- format (*Util2d* property), 1944
- format_var_list() (*in module flopy.mf6.utils.createpackages*),

1604
 FormattedHeader (class in
 flopy.utils.formattedfile), 1866
 FormattedHeadFile (class in
 flopy.utils.formattedfile), 1865
 FormattedLayerFile (class in
 flopy.utils.formattedfile), 1866
 fortran (ArrayFormat attribute), 1935
 fortran (ArrayFormat property), 1937
 forward (trackDir attribute), 1829
 free (ArrayFormat attribute), 1936
 free (ArrayFormat property), 1937
 from_4d() (MfList class method), 1951
 from_4d() (Transient2d class method), 1938
 from_argus_export() (UnstructuredGrid class
 method), 2034
 from_binary_grid_file() (Grid class
 method), 2019
 from_binary_grid_file() (StructuredGrid
 class method), 2026
 from_binary_grid_file() (UnstructuredGrid
 class method), 2034
 from_binary_grid_file() (VertexGrid class
 method), 2038
 from_geojson() (Shape static method), 1870
 from_gridspec() (StructuredGrid class
 method), 2026
 from_gridspec() (StructuredSpatialReference
 class method), 1573
 from_gridspec() (UnstructuredGrid class
 method), 2034
 from_namfile_header()
 (StructuredSpatialReference class
 method), 1573
 from_namfile_header()
 (VertexSpatialReference class
 method), 1576
 from_package() (Mt3dSsm method), 1802
 ftype() (ModflowFlwob method), 1649

G

generate_classes() (in module
 flopy.mf6.utils.generate_classes),
 1604
 generator_type() (in module
 flopy.mf6.utils.createpackages),
 1604
 generic_array_export() (in module
 flopy.export.utils), 2001
 geojson (GeoSpatialCollection property),
 1872
 geojson (GeoSpatialUtil property), 1873
 geojson (Shape property), 1870
 geospatial_bounds (acdd property), 1990

geospatial_bounds_vertical_crs (acdd
 property), 1990
 GeoSpatialCollection (class in
 flopy.utils.geospatial_utils), 1871
 GeoSpatialUtil (class in
 flopy.utils.geospatial_utils), 1872
 get() (ModflowParBc method), 1707
 get_active() (check method), 1845
 get_active() (mf6check method), 1846
 get_alldata() (EndpointFile method), 1900
 get_alldata() (LayerFile method), 1855
 get_alldata() (PathlineFile method), 1903
 get_alldata() (TimeseriesFile method), 1906
 get_allnode_data() (ModflowMnw2 method),
 1690
 get_angldegx() (Gridgen method), 1876
 get_area() (Gridgen method), 1876
 get_array() (Raster method), 1920
 get_attribute_array() (Triangle method),
 1932
 get_authority_crs() (in module
 flopy.utils.crs), 1851
 get_bot() (Gridgen method), 1876
 get_boundary_marker_array() (Triangle
 method), 1932
 get_budget() (ListBudget method), 1891
 get_budget() (ZoneBudget method), 1961
 get_budget() (ZoneBudget6 method), 1965
 get_budgetunit() (ModflowOc method), 1701
 get_cell2d() (Triangle method), 1932
 get_cell_edge_length() (Triangle method),
 1932
 get_cell_vertices() (StructuredGrid
 method), 2024, 2026
 get_cell_vertices() (UnstructuredGrid
 method), 2032, 2034
 get_cell_vertices() (VertexGrid method),
 2038
 get_cell_volumes() (ModflowDis method),
 1634
 get_cellxy() (Gridgen method), 1876
 get_center() (Gridgen method), 1876
 get_cl12() (Gridgen method), 1877
 get_comment() (MFBBlockHeader method), 1296
 get_connectivity() (SrwFile method), 1927
 get_constant_cr() (Util2d method), 1944
 get_coords() (Grid method), 2017, 2019
 get_crs() (in module *flopy.utils.crs*), 1852
 get_cumulative() (ListBudget method), 1892
 get_data() (CellBudgetFile method), 1836
 get_data() (EndpointFile method), 1901
 get_data() (LayerFile method), 1856
 get_data() (ListBudget method), 1892
 get_data() (MFArray method), 1578

`get_data()` (*MFList method*), 1587
`get_data()` (*MFMultipleList method*), 1592
`get_data()` (*MFScalar method*), 1597
`get_data()` (*MFScalarTransient method*), 1599
`get_data()` (*MFTransientArray method*), 1583
`get_data()` (*MFTransientList method*), 1593
`get_data()` (*Observations method*), 1569
`get_data()` (*ObsFiles method*), 1910
`get_data()` (*PathlineFile method*), 1903
`get_data()` (*SwrFile method*), 1927
`get_data()` (*TimeseriesFile method*), 1906
`get_databytes()` (*BinaryLayerFile method*), 1835
`get_databytes()` (*HeadUFile method*), 1841
`get_dataframe()` (*MFList method*), 1951
`get_dataframe()` (*Observations method*), 1569
`get_dataframe()` (*ObsFiles method*), 1911
`get_dataframe()` (*SfrFile method*), 1925
`get_dataframes()` (*ListBudget method*), 1893
`get_dataframes()` (*ZoneBudget method*), 1962
`get_dataframes()` (*ZoneBudget6 method*), 1965
`get_default_CTS_dtype()` (*Mt3dCts static method*), 1783
`get_default_dtype()` (*ModflowAg static method*), 1620
`get_default_dtype()` (*ModflowChd static method*), 1628
`get_default_dtype()` (*ModflowDrn static method*), 1639
`get_default_dtype()` (*ModflowDrt static method*), 1642
`get_default_dtype()` (*ModflowFhb static method*), 1647
`get_default_dtype()` (*ModflowGage static method*), 1651
`get_default_dtype()` (*ModflowGhb static method*), 1653
`get_default_dtype()` (*ModflowHfb static method*), 1659
`get_default_dtype()` (*ModflowHyd static method*), 1665
`get_default_dtype()` (*ModflowMnw1 static method*), 1680
`get_default_dtype()` (*ModflowRiv static method*), 1723
`get_default_dtype()` (*ModflowStr static method*), 1741
`get_default_dtype()` (*ModflowWel static method*), 1768
`get_default_dtype()` (*Mt3dLkt static method*), 1790
`get_default_dtype()` (*Mt3dSft static method*), 1798
`get_default_dtype()` (*Mt3dSsm static method*), 1802
`get_default_node_dtype()` (*ModflowMnw2 static method*), 1691
`get_default_numpy_fmt()` (*ArrayFormat static method*), 1937
`get_default_reach_dtype()` (*ModflowSfr2 static method*), 1731
`get_default_segment_dtype()` (*ModflowSfr2 static method*), 1731
`get_default_spd_dtype()` (*Mnw static method*), 1686
`get_default_spd_dtype()` (*ModflowMnw2 static method*), 1691
`get_delr_delc()` (*Lgr method*), 1889
`get_destination_endpoint_data()` (*EndpointFile method*), 1901
`get_destination_pathline_data()` (*PathlineFile method*), 1904
`get_destination_timeseries_data()` (*TimeseriesFile method*), 1907
`get_dis()` (*in module flopy.utils.utils_def*), 1955
`get_disu()` (*Gridgen method*), 1877
`get_disu5_gridprops()` (*VoronoiGrid method*), 1958
`get_disu6_gridprops()` (*VoronoiGrid method*), 1958
`get_disu_kwargs()` (*in module flopy.utils.gridutil*), 1888
`get_disv_gridprops()` (*in module flopy.utils.cvfduutil*), 1853
`get_disv_gridprops()` (*VoronoiGrid method*), 1958
`get_dtype()` (*Header method*), 1855
`get_dtypes()` (*StartingLocationsFile static method*), 1834
`get_edge_cells()` (*Triangle method*), 1933
`get_empty()` (*MFList method*), 1952
`get_empty()` (*ModflowAg static method*), 1621
`get_empty()` (*ModflowChd static method*), 1628
`get_empty()` (*ModflowDrn static method*), 1639
`get_empty()` (*ModflowDrt static method*), 1642
`get_empty()` (*ModflowFhb static method*), 1647
`get_empty()` (*ModflowGage static method*), 1651
`get_empty()` (*ModflowGhb static method*), 1653
`get_empty()` (*ModflowHfb static method*), 1659
`get_empty()` (*ModflowHyd static method*),

[1665](#)
[get_empty\(\) \(ModflowRiv static method\), 1723](#)
[get_empty\(\) \(ModflowStr static method\), 1741](#)
[get_empty\(\) \(ModflowWel static method\), 1768](#)
[get_empty_node_data\(\) \(ModflowMnw2 static method\), 1691](#)
[get_empty_reach_data\(\) \(ModflowSfr2 static method\), 1731](#)
[get_empty_segment_data\(\) \(ModflowSfr2 static method\), 1731](#)
[get_empty_starting_locations_data\(\) \(StartingLocationsFile static method\), 1834](#)
[get_empty_stress_period_data\(\) \(Mnw static method\), 1687](#)
[get_empty_stress_period_data\(\) \(ModflowMnw1 static method\), 1680](#)
[get_empty_stress_period_data\(\) \(ModflowMnw2 static method\), 1691](#)
[get_entries_from_namefile\(\) \(in module flopy.utils.mfreadnam\), 1898](#)
[get_exchange_data\(\) \(Lgr method\), 1889](#)
[get_exchange_file\(\) \(MFSimulation method\), 1303](#)
[get_ext_dict_attr\(\) \(BaseModel method\), 1608](#)
[get_extended_budget\(\) \(in module flopy.utils.postprocessing\), 1916](#)
[get_extent\(\) \(PlotCrossSection method\), 1969](#)
[get_extent\(\) \(StructuredSpatialReference method\), 1573](#)
[get_extent\(\) \(VertexSpatialReference method\), 1576](#)
[get_external_cr\(\) \(Util2d method\), 1944](#)
[get_fahl\(\) \(Gridgen method\), 1877](#)
[get_file\(\) \(MFSimulation method\), 1303](#)
[get_file_entry\(\) \(MFArray method\), 1578](#)
[get_file_entry\(\) \(MFList method\), 1587](#)
[get_file_entry\(\) \(MFScalar method\), 1597](#)
[get_file_entry\(\) \(MFScalarTransient method\), 1599](#)
[get_file_entry\(\) \(MFTransientArray method\), 1583](#)
[get_file_entry\(\) \(MFTransientList method\), 1593](#)
[get_file_entry\(\) \(Util2d method\), 1944](#)
[get_file_entry\(\) \(Util2dTpl method\), 2014](#)
[get_file_entry\(\) \(Util3d method\), 1948](#)
[get_file_path\(\) \(MFPackage method\), 1299](#)
[get_filename\(\) \(MfList method\), 1952](#)
[get_filenames\(\) \(MfList method\), 1952](#)
[get_final_totim\(\) \(ModflowDis method\), 1634](#)
[get_fldr\(\) \(Gridgen method\), 1877](#)
[get_gnc_file\(\) \(MFSimulation method\), 1303](#)
[get_gradients\(\) \(in module flopy.utils.postprocessing\), 1917](#)
[get_grid_line_collection\(\) \(PlotCrossSection method\), 1969](#)
[get_grid_lines\(\) \(StructuredSpatialReference method\), 1574](#)
[get_grid_patch_collection\(\) \(PlotCrossSection method\), 1969](#)
[get_grid_type\(\) \(MFModel method\), 1288](#)
[get_gridprops_dis6\(\) \(SimpleRegularGrid method\), 1891](#)
[get_gridprops_disu5\(\) \(Gridgen method\), 1878](#)
[get_gridprops_disu6\(\) \(Gridgen method\), 1878](#)
[get_gridprops_disv\(\) \(Gridgen method\), 1878](#)
[get_gridprops_unstructuredgrid\(\) \(Gridgen method\), 1878](#)
[get_gridprops_unstructuredgrid\(\) \(VoronoiGrid method\), 1958](#)
[get_gridprops_vertexgrid\(\) \(Gridgen method\), 1878](#)
[get_gridprops_vertexgrid\(\) \(VoronoiGrid method\), 1959](#)
[get_headfile_precision\(\) \(in module flopy.utils.binaryfile\), 1843](#)
[get_hwva\(\) \(Gridgen method\), 1879](#)
[get_ia_from_iac\(\) \(in module flopy.utils.gridgen\), 1883](#)
[get_iac\(\) \(Gridgen method\), 1879](#)
[get_idomain\(\) \(Lgr method\), 1889](#)
[get_ifrefm\(\) \(Modflow method\), 1618](#)
[get_ifrefm\(\) \(Seawat method\), 1807](#)
[get_ihc\(\) \(Gridgen method\), 1879](#)
[get_ims_package\(\) \(MFModel method\), 1288](#)
[get_ims_package\(\) \(MFSimulation method\), 1304](#)
[get_incremental\(\) \(ListBudget method\), 1893](#)
[get_indices\(\) \(CellBudgetFile method\), 1837](#)
[get_indices\(\) \(MfList method\), 1952](#)
[get_input_files\(\) \(in module flopy.utils.mfreadnam\), 1898](#)
[get_internal_cr\(\) \(Util2d method\), 1944](#)
[get_isym\(\) \(in module flopy.utils.gridgen\), 1883](#)
[get_item2_names\(\) \(Mnw static method\), 1687](#)
[get_itmp\(\) \(MfList method\), 1952](#)
[get_ivc\(\) \(Gridgen method\), 1879](#)
[get_ja\(\) \(Gridgen method\), 1880](#)
[get_kper_entry\(\) \(Transient2d method\), 1939](#)

get_kper_entry() (*Transient2dTpl method*),
 2014
 get_kper_entry() (*Transient3d method*), 1941
 get_kstp_kper_toffset() (*ModflowDis
 method*), 1634
 get_kstpkper() (*CellBudgetFile method*),
 1837
 get_kstpkper() (*LayerFile method*), 1856
 get_kstpkper() (*ListBudget method*), 1894
 get_kswrkstp_kper() (*SwrFile method*), 1928
 get_lak_connections() (in module
 flopymf6utils.lakpakutils), 1577
 get_layer() (in module
 flopymodflow.mfdis), 1637
 get_layer() (*ModflowDis method*), 1635
 get_layer_node_range() (*UnstructuredGrid
 method*), 2034
 get_lni() (*Grid method*), 2019
 get_lni() (in module flopyutils.gridutil),
 1888
 get_local_coords() (*Grid method*), 2020
 get_longnames_from_docstrings() (*NetCDF
 method*), 1994
 get_lower_left() (*Lgr method*), 1889
 get_lrc() (*ModflowDis method*), 1635
 get_lrc() (*StructuredGrid method*), 2026
 get_maxid() (*EndpointFile method*), 1902
 get_maxid() (*PathlineFile method*), 1904
 get_maxid() (*TimeseriesFile method*), 1907
 get_maxtime() (*EndpointFile method*), 1902
 get_maxtime() (*PathlineFile method*), 1905
 get_maxtime() (*TimeseriesFile method*), 1907
 get_maxtraveltime() (*EndpointFile method*),
 1902
 get_mf6_blockdata() (in module
 flopymf6utils.mfreadnam), 1898
 get_mf6_files() (in module
 flopymf6utils.mfreadnam), 1898
 get_mf6_ftypes() (in module
 flopymf6utils.mfreadnam), 1898
 get_mf6_mshape() (in module
 flopymf6utils.mfreadnam), 1899
 get_mf6_nper() (in module
 flopymf6utils.mfreadnam), 1899
 get_model() (*MFSimulation method*), 1304
 get_model_path() (*MFFileMgmt method*), 1280
 get_model_runtime() (*ListBudget method*),
 1894
 get_model_shape() (*ZoneBudget method*), 1962
 get_module_val() (*PackageContainer static
 method*), 1281
 get_mvr_file() (*MFSimulation method*), 1304
 get_mvt_file() (*MFSimulation method*), 1304
 get_name_file_entries() (*BaseModel method*),
 1608
 get_names() (*Header method*), 1855
 get_neighbors() (*check method*), 1845
 get_next_line() (in module
 flopymf6utils.flopymf6io), 1862
 get_nlay() (*Gridgen method*), 1880
 get_nnodes() (*Mnw static method*), 1687
 get_nobs() (*Observations method*), 1569
 get_nobs() (*ObsFiles method*), 1912
 get_nod_rearray() (*Gridgen method*), 1880
 get_node() (*ModflowDis method*), 1635
 get_node() (*StructuredGrid method*), 2027
 get_node_coordinates() (*ModflowDis method*),
 1635
 get_nodelay() (*Gridgen method*), 1880
 get_nodes() (*Gridgen method*), 1880
 get_nrecords() (*CellBudgetFile method*),
 1837
 get_nrecords() (*Observations method*), 1569
 get_nrecords() (*SwrFile method*), 1928
 get_nrow_ncol_nlay_nper() (*Modflow method*),
 1618
 get_nrow_ncol_nlay_nper() (*Mt3dms method*),
 1772
 get_nrow_ncol_nlay_nper() (*Seawat method*),
 1807
 get_nstrm() (*SfrFile static method*), 1925
 get_ntimes() (*Observations method*), 1569
 get_ntimes() (*ObsFiles method*), 1912
 get_ntimes() (*SwrFile method*), 1928
 get_number_plottable_layers() (*Grid
 method*), 2020
 get_number_plottable_layers()
 (*StructuredGrid method*), 2027
 get_number_plottable_layers()
 (*UnstructuredGrid method*), 2034
 get_number_plottable_layers() (*VertexGrid
 method*), 2038
 get_obs_data() (*Observations method*), 1569
 get_obsnames() (*ObsFiles method*), 1912
 get_ocoutput_units() (*Modflow0c static
 method*), 1702
 get_open_file_object() (in module
 flopymf6utils.flopymf6io), 1955
 get_openclose_cr() (*Util2d method*), 1944
 get_outlets() (*ModflowSfr2 method*), 1731
 get_output() (*BaseModel method*), 1608
 get_output_attribute() (*BaseModel method*),
 1608
 get_package() (*BaseModel method*), 1608
 get_package() (*PackageContainer method*),
 1282
 get_package_list() (*ModelInterface method*),
 1612

`get_pak_vals_shape()` (in module `flopy.utils.utils_def`), 1955
`get_parent_connections()` (*Lgr method*), 1890
`get_parent_indices()` (*Lgr method*), 1890
`get_patch()` (*Polygon method*), 1869
`get_patch_collection()` (*VoronoiGrid method*), 1959
`get_plottable_layer_array()` (*Grid method*), 2020
`get_plottable_layer_array()` (*StructuredGrid method*), 2027
`get_plottable_layer_array()` (*UnstructuredGrid method*), 2035
`get_plottable_layer_array()` (*VertexGrid method*), 2039
`get_plottable_layer_shape()` (*Grid method*), 2020
`get_plottable_layer_shape()` (*UnstructuredGrid method*), 2035
`get_polygon_area()` (in module `flopy.utils.geometry`), 1870
`get_polygon_centroid()` (in module `flopy.utils.geometry`), 1870
`get_position()` (*CellBudgetFile method*), 1837
`get_pyshp_field_dtypes()` (in module `flopy.export.shapefile_utils`), 1995
`get_pyshp_field_info()` (in module `flopy.export.shapefile_utils`), 1995
`get_rc_from_node_coordinates()` (*ModflowDis method*), 1635
`get_record()` (*CellBudgetFile method*), 1838
`get_record()` (*MFArray method*), 1579
`get_record()` (*MFList method*), 1588
`get_record()` (*MFTransientArray method*), 1583
`get_record()` (*MFTransientList method*), 1593
`get_record_names()` (*ListBudget method*), 1894
`get_record_names()` (*SwrFile method*), 1928
`get_record_names()` (*ZoneBudget method*), 1962
`get_reduced_pumping()` (in module `flopy.utils.observationfile`), 1913
`get_reduced_pumping()` (*ListBudget method*), 1895
`get_replicated_parent_array()` (*Lgr method*), 1890
`get_residual()` (*CellBudgetFile method*), 1838
`get_residuals()` (in module `flopy.mf6.utils.postprocessing`), 1570
`get_results()` (*SfrFile method*), 1925
`get_sciencebase_metadata()` (*acdd method*), 1990
`get_sciencebase_xml_metadata()` (*acdd method*), 1990
`get_selection()` (in module `flopy.utils.observationfile`), 1913
`get_shape()` (*Lgr method*), 1890
`get_shapefile_crs()` (in module `flopy.utils.crs`), 1852
`get_sim_name()` (in module `flopy.utils.mfreadnam`), 1899
`get_sim_path()` (*MFFileMgmt method*), 1280
`get_slopes()` (*ModflowSfr2 method*), 1731
`get_solution_package()` (*MFSimulation method*), 1304
`get_sorted_vertices()` (in module `flopy.utils.voronoi`), 1959
`get_specific_discharge()` (in module `flopy.utils.postprocessing`), 1917
`get_steadystate_list()` (*MFModel method*), 1288
`get_structured_faceflows()` (in module `flopy.mf6.utils.postprocessing`), 1571
`get_template_array()` (in module `flopy.pest.tplarray`), 2015
`get_times()` (*CellBudgetFile method*), 1839
`get_times()` (*LayerFile method*), 1856
`get_times()` (*ListBudget method*), 1895
`get_times()` (*Observations method*), 1570
`get_times()` (*ObsFiles method*), 1912
`get_times()` (*SfrFile method*), 1926
`get_times()` (*SwrFile method*), 1928
`get_top()` (*Gridgen method*), 1881
`get_top_botm()` (*Lgr method*), 1890
`get_total_size()` (*MultiList method*), 1860
`get_totim()` (*ModflowDis method*), 1636
`get_totim_from_kper_toffset()` (*ModflowDis method*), 1636
`get_transient_key()` (*MFBlockHeader method*), 1296
`get_transmissivities()` (in module `flopy.utils.postprocessing`), 1918
`get_ts()` (*BinaryLayerFile method*), 1835
`get_ts()` (*CellBudgetFile method*), 1839
`get_ts()` (*FormattedLayerFile method*), 1867
`get_ts()` (*HeadUFile method*), 1842
`get_ts()` (*SwrFile method*), 1929
`get_ts_sp()` (in module `flopy.utils.flopy_io`), 1862
`get_tslens()` (*ListBudget method*), 1895
`get_unique_package_names()` (*CellBudgetFile method*), 1839
`get_unique_record_names()` (*CellBudgetFile*

`method`), 1839
`get_unitnumber_from_ext_unit_dict()` (in module `flopy.utils.utils_def`), 1956
`get_updated_path()` (`MFFileMgmt` method), 1280
`get_upsegs()` (`ModflowSfr2` method), 1731
`get_url_text()` (in module `flopy.utils.flopy_io`), 1862
`get_util2d_shape_for_layer()` (in module `flopy.utils.utils_def`), 1956
`get_valid_faces()` (in module `flopy.utils.voronoi`), 1959
`get_value()` (`Util2d` method), 1944
`get_value()` (`Util3d` method), 1948
`get_values()` (`Header` method), 1855
`get_variable_by_stress_period()` (`ModflowSfr2` method), 1732
`get_version()` (in module `flopy.utils.utl_import`), 1957
`get_vertices()` (`Gridgen` method), 1881
`get_vertices()` (`StructuredSpatialReference` method), 1574
`get_vertices()` (`Triangle` method), 1933
`get_verts_iverts()` (`Gridgen` method), 1881
`get_volumetric_budget()` (`ZoneBudget` method), 1963
`get_volumetric_budget()` (`ZoneBudget6` method), 1966
`get_water_table()` (in module `flopy.utils.postprocessing`), 1919
`get_xcellcenters_for_layer()` (`Grid` method), 2020
`get_xcellcenters_for_layer()` (`UnstructuredGrid` method), 2035
`get_xcellcenters_for_layer()` (`VertexGrid` method), 2039
`get_xcenter_array()` (`StructuredSpatialReference` method), 1574
`get_xcyc()` (`Triangle` method), 1933
`get_xedge_array()` (`StructuredSpatialReference` method), 1574
`get_xvertices_for_layer()` (`Grid` method), 2020
`get_xvertices_for_layer()` (`UnstructuredGrid` method), 2035
`get_xvertices_for_layer()` (`VertexGrid` method), 2039
`get_ycellcenters_for_layer()` (`Grid` method), 2020
`get_ycellcenters_for_layer()` (`UnstructuredGrid` method), 2035
`get_ycellcenters_for_layer()` (`VertexGrid` method), 2039
`get_ycenter_array()` (`StructuredSpatialReference` method), 1574
`get_yedge_array()` (`StructuredSpatialReference` method), 1574
`get_yvertices_for_layer()` (`Grid` method), 2020
`get_yvertices_for_layer()` (`UnstructuredGrid` method), 2035
`get_yvertices_for_layer()` (`VertexGrid` method), 2039
`get_zero_2d()` (`Transient2d` method), 1939
`get_zero_3d()` (`Transient3d` method), 1942
`getbotm()` (`ModflowDis` method), 1636
`getfiletypeunit()` (in module `flopy.utils.mfreadnam`), 1899
`getkeys()` (`MFObservationRequester` static method), 1568
`getkeys()` (`MFOutputRequester` static method), 1563
`getmf()` (`Modpath6` method), 1831
`getsim()` (`Modpath6` method), 1831
`gettop()` (`ModflowDis` method), 1636
`gnc_filerecord` (`ModflowGwfgwf` attribute), 1388
`gncdata` (`ModflowGnc` attribute), 1316
`gncdata` (`ModflowGwfgnc` attribute), 1382
`GncPackages` (class in `flopy.mf6.modflow.mfgnc`), 1313
`graph` (`ModflowSfr2` property), 1732
`graph_legend()` (`styles` class method), 1987
`graph_legend_title()` (`styles` class method), 1988
`Grid` (class in `flopy.discretization.grid`), 2015
`grid_lines` (`Grid` property), 2020
`grid_lines` (`StructuredGrid` property), 2027
`grid_lines` (`UnstructuredGrid` property), 2035
`grid_lines` (`VertexGrid` property), 2039
`grid_type` (`Grid` attribute), 2016
`grid_type` (`Grid` property), 2020
`grid_type` (`MfGrdFile` property), 1564
`grid_varies_by_layer` (`UnstructuredGrid` property), 2035
`gridarray_to_flopyusg_gridarray()` (`Gridgen` static method), 1881
`Gridgen` (class in `flopy.utils.gridgen`), 1874
`GridIntersect` (class in `flopy.utils.gridintersect`), 1883
`gridlist_to_disv_gridprops()` (in module `flopy.utils.cvfduutil`), 1853
`gridlist_to_verts()` (in module

`flopy.utils.cvfduutil`), 1853
`GwfgncPackages` (class in
`flopy.mf6.modflow.mfgwfgnc`), 1380
`GwfmvrPackages` (class in
`flopy.mf6.modflow.mfgwfmvr`), 1414
`GwtmvtPackages` (class in
`flopy.mf6.modflow.mfgwtmvt`), 1510

H

`has_data()` (*MFArrary* method), 1579
`has_data()` (*MFList* method), 1588
`has_data()` (*MFSscalar* method), 1598
`has_data()` (*MFSscalarTransient* method), 1600
`has_data()` (*MFTransientArray* method), 1583
`has_data()` (*MFTransientList* method), 1593
`has_one_item()` (*PyListUtil* static method), 1861
`has_package()` (*BaseModel* method), 1608
`has_stress_period_data` (*PackageInterface* property), 1617
`has_z` (*LineString* attribute), 1868
`has_z` (*Point* attribute), 1869
`hdry` (*BaseModel* property), 1608
`hdry` (*MFModel* property), 1288
`hdry` (*ModelInterface* property), 1612
`hdry` (*Modpath7* property), 1817
`head_filerecord` (*ModflowGwfmaw* attribute), 1414
`head_filerecord` (*ModflowGwfoc* attribute), 1430
`Header` (class in *flopy.utils.datafile*), 1855
`header_exists()` (*MFBBlock* method), 1295
`HeadFile` (class in *flopy.utils.binaryfile*), 1840
`heading` (*ModflowBas* attribute), 1622
`heading` (*ModflowDis* attribute), 1633
`heading` (*Modpath6Bas* attribute), 1832
`heading` (*Modpath6Sim* attribute), 1833
`heading()` (*styles* class method), 1988
`HeadObservation` (class in
`flopy.modflow.mfhob`), 1660
`headprintrecord` (*ModflowGwfoc* attribute), 1430
`HeadUFile` (class in
`flopy.utils.binaryfile`), 1841
`histogram()` (*Raster* method), 1920
`hnoflo` (*BaseModel* property), 1608
`hnoflo` (*MFModel* property), 1288
`hnoflo` (*ModelInterface* property), 1612
`hnoflo` (*Modpath7* property), 1817
`how` (*Util2d* attribute), 1942
`how` (*Util2d* property), 1944
`hwva` (*ModflowGwfdisu* attribute), 1360

`hwva` (*ModflowGwtdisu* attribute), 1485
`HydmodObs` (class in
`flopy.utils.observationfile`), 1910

I

`ia` (*MfGrdFile* property), 1564
`iac` (*ModflowGwfdisu* attribute), 1360
`iac` (*ModflowGwtdisu* attribute), 1485
`iac` (*UnstructuredGrid* property), 2035
`iavert` (*MfGrdFile* property), 1565
`icelltype` (*ModflowGwfnpf* attribute), 1427
`iconvert` (*ModflowGwfsto* attribute), 1456
`idomain` (*Grid* attribute), 2016
`idomain` (*Grid* property), 2020
`idomain` (*MfGrdFile* property), 1565
`idomain` (*ModflowGwfdisu* attribute), 1355
`idomain` (*ModflowGwfdisu* attribute), 1360
`idomain` (*ModflowGwfdisu* attribute), 1364
`idomain` (*ModflowGwtdisu* attribute), 1479
`idomain` (*ModflowGwtdisu* attribute), 1485
`idomain` (*ModflowGwtdisu* attribute), 1488
`ievt` (*ModflowGwfevt* attribute), 1375
`ifrefm` (*ModflowBas* attribute), 1623
`ifrefm` (*ModflowBas* property), 1623
`ignore_shapely2_stree_warning()` (in
module *flopy.utils.gridintersect*), 1888
`ignore_shapely_warnings_for_object_array()`
(in module *flopy.utils.gridintersect*), 1888
`ihc` (*ModflowGwfdisu* attribute), 1360
`ihc` (*ModflowGwtdisu* attribute), 1485
`import_optional_dependency()` (in module
flopy.utils.utl_import), 1957
`in_shape()` (*MultiList* method), 1860
`inc_shape_idx()` (*MultiList* method), 1860
`increment_dimension()` (*MultiList* method), 1860
`indent_string` (*MFSimulationData* attribute), 1310
`indexes()` (*MultiList* method), 1860
`init_package()` (*MFChildPackages* method), 1297
`initialize()` (*GncPackages* method), 1313
`initialize()` (*GwfgncPackages* method), 1380
`initialize()` (*GwfmvrPackages* method), 1414
`initialize()` (*GwtmvtPackages* method), 1510
`initialize()` (*MvrPackages* method), 1329, 1330
`initialize()` (*UtlatsPackages* method), 1544
`initialize()` (*UtlatsPackages* method), 1547
`initialize()` (*UtlatsPackages* method), 1553, 1554
`initialize()` (*UtltsPackages* method), 1556

[initialize\(\) \(UtlvtkPackages method\)](#), 1558, 1559
[initialize\(\) \(UtlvtvsPackages method\)](#), 1561
[initialize_file\(\) \(NetCdf method\)](#), 1994
[initialize_geometry\(\) \(NetCdf method\)](#), 1994
[initialize_group\(\) \(NetCdf method\)](#), 1994
[input_keys\(\) \(SimulationDict method\)](#), 1312
[inspect_cells\(\) \(MFModel method\)](#), 1288
[inspect_cells\(\) \(MFPackage method\)](#), 1299
[INT16 \(Raster attribute\)](#), 1920
[INT32 \(Raster attribute\)](#), 1920
[INT64 \(Raster attribute\)](#), 1920
[INT8 \(Raster attribute\)](#), 1920
[integer\(\) \(ArrayFormat class method\)](#), 1937
[internal_formatting \(MFSimulationData attribute\)](#), 1310
[interpolate\(\) \(StructuredSpatialReference method\)](#), 1574
[interpolation_methodrecord \(ModflowUtlts attribute\)](#), 1553
[interpolation_methodrecord \(ModflowUtlts attribute\)](#), 1555
[interpolation_methodrecord_single \(ModflowUtlts attribute\)](#), 1555
[intersect\(\) \(Grid method\)](#), 2018, 2020
[intersect\(\) \(Gridgen method\)](#), 1881
[intersect\(\) \(GridIntersect method\)](#), 1884
[intersect\(\) \(StructuredGrid method\)](#), 2027
[intersect\(\) \(UnstructuredGrid method\)](#), 2035
[intersect\(\) \(VertexGrid method\)](#), 2039
[intersect_modpath_with_crosssection\(\) \(in module \[flopy.plot.plotutil\]\(#\)\)](#), 1983
[intersects\(\) \(GridIntersect method\)](#), 1885
[InvalidVersion](#), 1915
[irch \(ModflowGwfrcha attribute\)](#), 1437
[irregular_shape_patch\(\) \(UnstructuredPlotUtilities static method\)](#), 1982
[is_allowed\(\) \(MFBlock method\)](#), 1295
[is_basic_type\(\) \(DatumUtil static method\)](#), 1858
[is_clockwise\(\) \(in module \[flopy.utils.geometry\]\(#\)\)](#), 1870
[is_complete \(Grid property\)](#), 2020
[is_complete \(StructuredGrid property\)](#), 2028
[is_complete \(UnstructuredGrid property\)](#), 2036
[is_complete \(VertexGrid property\)](#), 2039
[is_devrelease \(LegacyVersion property\)](#), 1915
[is_devrelease \(Version property\)](#), 1916
[is_empty\(\) \(MFBlock method\)](#), 1295
[is_empty_list\(\) \(PyListUtil static method\)](#), 1861
[is_float\(\) \(DatumUtil static method\)](#), 1859
[is_float\(\) \(in module \[flopy.utils.formattedfile\]\(#\)\)](#), 1867
[is_int\(\) \(DatumUtil static method\)](#), 1859
[is_int\(\) \(in module \[flopy.utils.formattedfile\]\(#\)\)](#), 1867
[is_iterable\(\) \(PyListUtil static method\)](#), 1861
[is_postrelease \(LegacyVersion property\)](#), 1915
[is_postrelease \(Version property\)](#), 1916
[is_prerelease \(LegacyVersion property\)](#), 1915
[is_prerelease \(Version property\)](#), 1916
[is_rectilinear \(StructuredGrid property\)](#), 2028
[is_regular \(StructuredGrid property\)](#), 2028
[is_regular_x \(StructuredGrid property\)](#), 2028
[is_regular_xy \(StructuredGrid property\)](#), 2028
[is_regular_xz \(StructuredGrid property\)](#), 2028
[is_regular_y \(StructuredGrid property\)](#), 2028
[is_regular_yz \(StructuredGrid property\)](#), 2028
[is_regular_z \(StructuredGrid property\)](#), 2028
[is_same_header\(\) \(MFBlockHeader method\)](#), 1297
[is_symmetrical\(\) \(in module \[flopy.utils.gridgen\]\(#\)\)](#), 1883
[is_valid \(Grid property\)](#), 2020
[is_valid \(StructuredGrid property\)](#), 2028
[is_valid \(UnstructuredGrid property\)](#), 2036
[is_valid \(VertexGrid property\)](#), 2039
[is_valid\(\) \(MFBlock method\)](#), 1295
[is_valid\(\) \(MFModel method\)](#), 1289
[is_valid\(\) \(MFPackage method\)](#), 1299
[is_valid\(\) \(MFSimulation method\)](#), 1304
[isabs\(\) \(MFFilePath method\)](#), 1281
[isBetween\(\) \(in module \[flopy.utils.cvfdutil\]\(#\)\)](#), 1854
[isfloat\(\) \(OptionUtil static method\)](#), 1915
[isint\(\) \(OptionUtil static method\)](#), 1915
[isvalid\(\) \(check method\)](#), 1845
[isvalid\(\) \(ListBudget method\)](#), 1896
[isvalid\(\) \(OptionUtil static method\)](#), 1915
[items \(Logger attribute\)](#), 1991
[ItmpError](#), 1680
[itmuni_text \(TemporalReference attribute\)](#), 1924
[itmuni_values \(TemporalReference](#)

attribute), 1924
 itype_dict() (Mt3dSsm static method), 1802
 iverts (Grid property), 2020
 iverts (MfGrdFile property), 1565
 iverts (StructuredGrid property), 2028
 iverts (UnstructuredGrid property), 2036
 iverts (VertexGrid property), 2039

J

ja (MfGrdFile property), 1565
 ja (ModflowGwfdisu attribute), 1360
 ja (ModflowGwtDisu attribute), 1485
 ja (UnstructuredGrid property), 2036
 javert (MfGrdFile property), 1565
 join_struct_arrays() (in module
 flopys.utils.binaryfile), 1843

K

k (ModflowGwfnfp attribute), 1427
 k22 (ModflowGwfnfp attribute), 1427
 k33 (ModflowGwfnfp attribute), 1427
 keys() (SimulationDict method), 1312
 kij_from_nn0() (ModflowGridIndices static
 method), 1887
 kij_from_nodenum() (ModflowGridIndices
 static method), 1887
 kijnames (EndpointFile attribute), 1902
 kijnames (PathlineFile attribute), 1905
 kijnames (TimeseriesFile attribute), 1907

L

label_cells() (Triangle method), 1933
 label_vertices() (Triangle method), 1933
 lakeperioddata (ModflowGwtlkt attribute),
 1508
 laycbd (BaseModel property), 1608
 laycbd (Grid property), 2020
 laycbd (MFModel property), 1289
 laycbd (ModelInterface property), 1612
 LayerFile (class in flopys.utils.datafile),
 1855
 laytyp (BaseModel property), 1608
 laytyp (MFModel property), 1289
 laytyp (ModelInterface property), 1612
 laytyp (Modpath7 property), 1817
 lazy_io (MFSimulationData property), 1311
 LegacyVersion (class in
 flopys.utils.parse_version), 1915
 len_const (ModflowSfr2 attribute), 1732
 lenuni (Grid attribute), 2016
 lenuni (Grid property), 2020
 level1_arraylist() (Package method), 1614
 Lgr (class in flopys.utils.lgrutil), 1889

line_intersect_grid()
 (UnstructuredPlotUtilities static
 method), 1982
 line_num (PyListUtil attribute), 1861
 line_parse() (in module
 flopys.utils.flopys_io), 1862
 line_strip() (in module
 flopys.utils.flopys_io), 1862
 LineString (class in flopys.utils.geometry),
 1868
 list_files() (in module
 flopys.mf6.utils.generate_classes),
 1604
 list_records() (CellBudgetFile method),
 1840
 list_records() (LayerFile method), 1856
 list_records() (Observations method), 1570
 list_unique_packages() (CellBudgetFile
 method), 1840
 list_unique_records() (CellBudgetFile
 method), 1840
 ListBudget (class in
 flopys.utils.mflistfile), 1891
 load() (MFArray method), 1579
 load() (MFBlock method), 1295
 load() (MFList method), 1588
 load() (MFPackage method), 1299
 load() (MFScalar method), 1598
 load() (MFScalarTransient method), 1600
 load() (MFSimulation class method), 1305
 load() (MFTransientArray method), 1584
 load() (MFTransientList method), 1593
 load() (Modflow class method), 1618
 load() (ModflowAg class method), 1621
 load() (ModflowBas class method), 1623
 load() (ModflowBcf class method), 1626
 load() (ModflowChd class method), 1628
 load() (ModflowDe4 class method), 1631
 load() (ModflowDis class method), 1637
 load() (ModflowDrn class method), 1639
 load() (ModflowDrt class method), 1642
 load() (ModflowEvt class method), 1644
 load() (ModflowFhb class method), 1647
 load() (ModflowFlwob class method), 1649
 load() (ModflowGage class method), 1651
 load() (ModflowGhb class method), 1653
 load() (ModflowGmg class method), 1657
 load() (ModflowGwf class method), 1336
 load() (ModflowGwt class method), 1337
 load() (ModflowHfb class method), 1660
 load() (ModflowHob class method), 1663
 load() (ModflowHyd class method), 1665
 load() (ModflowLak class method), 1671
 load() (ModflowLmt class method), 1673

load() (*ModflowLpf* class method), 1676
 load() (*ModflowMlt* class method), 1678
 load() (*ModflowMnw1* class method), 1680
 load() (*ModflowMnw2* class method), 1692
 load() (*ModflowMnwi* class method), 1694
 load() (*ModflowNwt* class method), 1698
 load() (*ModflowOc* class method), 1702
 load() (*ModflowPar* static method), 1704
 load() (*ModflowParBc* class method), 1707
 load() (*ModflowPcg* class method), 1710
 load() (*ModflowPcgn* class method), 1713
 load() (*ModflowPks* class method), 1715
 load() (*ModflowPval* class method), 1717
 load() (*ModflowRch* class method), 1720
 load() (*ModflowRiv* class method), 1723
 load() (*ModflowSfr2* class method), 1732
 load() (*ModflowSip* class method), 1735
 load() (*ModflowSor* class method), 1736
 load() (*ModflowStr* class method), 1741
 load() (*ModflowSub* class method), 1746
 load() (*ModflowSwi2* class method), 1750
 load() (*ModflowSwr1* class method), 1751
 load() (*ModflowSwt* class method), 1756
 load() (*ModflowUpw* class method), 1759
 load() (*ModflowUzf1* class method), 1765
 load() (*ModflowWel* class method), 1768
 load() (*ModflowZon* class method), 1770
 load() (*Mt3dAdv* class method), 1776
 load() (*Mt3dBtn* class method), 1780
 load() (*Mt3dCts* class method), 1783
 load() (*Mt3dDsp* class method), 1786
 load() (*Mt3dGcg* class method), 1788
 load() (*Mt3dLkt* class method), 1790
 load() (*Mt3dms* class method), 1772
 load() (*Mt3dRct* class method), 1794
 load() (*Mt3dSft* class method), 1798
 load() (*Mt3dSsm* class method), 1802
 load() (*Mt3dUzt* class method), 1805
 load() (*Package* static method), 1615
 load() (*Raster* static method), 1921
 load() (*Seawat* class method), 1807
 load() (*SeawatVdf* class method), 1811
 load() (*SeawatVsc* class method), 1814
 load() (*Util2d* class method), 1944
 load() (*Util3d* class method), 1948
 load() (*ZoneBudget6* static method), 1966
 load() (*ZoneFile6* static method), 1967
 load_base() (*MFModel* class method), 1289
 load_bin() (*Util2d* static method), 1944
 load_block() (*Util2d* static method), 1945
 load_coord_info() (*Grid* method), 2020
 load_mas() (*Mt3dms* static method), 1773
 load_obs() (*Mt3dms* static method), 1773

load_options() (*OptionBlock* class method), 1913
 load_package() (*MFModel* method), 1290
 load_package() (*MFSimulation* method), 1306
 load_results() (*BaseModel* method), 1609
 load_results() (*Modflow* method), 1619
 load_results() (*Mt3dms* method), 1773
 load_txt() (*Util2d* static method), 1945
 loadarray() (*ModflowParBc* static method), 1707
 loadtxt() (in module *flopy.utils.flopy_io*), 1863
 local (*LegacyVersion* property), 1915
 local (*Version* property), 1916
 log() (*Logger* method), 1991
 Logger (class in *flopy.export.netcdf*), 1991
 LRCParticleData (class in *flopy.modpath.mp7particledata*), 1820

M

major (*Version* property), 1916
 make_layered() (*MFArray* method), 1580
 make_mnw_objects() (*ModflowMnw2* method), 1692
 make_node_data() (*Mnw* method), 1687
 make_node_data() (*ModflowMnw2* method), 1692
 make_stress_period_data() (*ModflowMnw2* method), 1692
 map_polygons (*Grid* property), 2020
 map_polygons (*StructuredGrid* property), 2028
 map_polygons (*UnstructuredGrid* property), 2036
 map_polygons (*VertexGrid* property), 2039
 masked4d_array_to_kper_dict() (*Transient2d* static method), 1939
 masked4D_arrays_to_stress_period_data() (*MfList* static method), 1952
 masked_4D_arrays (*MfList* property), 1952
 masked_4D_arrays (*MFTransientList* property), 1594
 masked_4D_arrays_itr() (*MfList* method), 1952
 masked_4D_arrays_itr() (*MFTransientList* method), 1594
 match_array_cells() (*MFModel* method), 1291
 max_columns_of_data (*MFSimulationData* attribute), 1310
 max_columns_of_data (*MFSimulationData* property), 1311
 max_multi_dim_list_size() (*PyListUtil* static method), 1861

max_tuple_abs_size() (in module *flopy.utils.datautil*), 1862
 mcomp (Mt3dms property), 1773
 mcomp (Seawat property), 1808
 methods() (MF6Output method), 1570
 mf (Modpath6 property), 1831
 mf6check (class in *flopy.utils.check*), 1846
 Mf6ListBudget (class in *flopy.utils.mflistfile*), 1896
 Mf6Obs (class in *flopy.utils.observationfile*), 1910
 MF6Output (class in *flopy.mf6.utils.output_util*), 1570
 mfaddoutsidefile (class in *flopy.modflow.mfaddoutsidefile*), 1619
 MFArray (class in *flopy.mf6.data.mfdataarray*), 1578
 MFBlock (class in *flopy.mf6.mfpackage*), 1294
 MFBlockHeader (class in *flopy.mf6.mfpackage*), 1296
 MFChildPackages (class in *flopy.mf6.mfpackage*), 1297
 mfdata (MFSimulationData attribute), 1311
 MFDataException, 1279
 MFFileMgmt (class in *flopy.mf6.mfbase*), 1280
 MFFilePath (class in *flopy.mf6.mfbase*), 1281
 MfGrdFile (class in *flopy.mf6.utils.binarygrid_util*), 1563
 MFInvalidTransientBlockHeaderException, 1281
 MFList (class in *flopy.mf6.data.mfdatalist*), 1587
 MfList (class in *flopy.utils.util_list*), 1950
 mflist_export() (in module *flopy.export.utils*), 2002
 MfListBudget (class in *flopy.utils.mflistfile*), 1896
 MFModel (class in *flopy.mf6.mfmodel*), 1286
 MFMultipleList (class in *flopy.mf6.data.mfdatalist*), 1591
 MFObservation (class in *flopy.mf6.utils.mfobservation*), 1568
 MFObservationRequester (class in *flopy.mf6.utils.mfobservation*), 1568
 MFOutput (class in *flopy.mf6.utils.binaryfile_utils*), 1562
 MFOutputRequester (class in *flopy.mf6.utils.binaryfile_utils*), 1562
 MFPackage (class in *flopy.mf6.mfpackage*), 1297
 mfpath (MFSimulationData attribute), 1311
 mfrcsv_filerecord (ModflowGwfma attribute), 1414
 MFScalar (class in *flopy.mf6.data.mfdatascalar*), 1597
 MFScalarTransient (class in *flopy.mf6.data.mfdatascalar*), 1599
 MFSimulation (class in *flopy.mf6.modflow.mfsimulation*), 1301
 MFSimulationData (class in *flopy.mf6.modflow.mfsimulation*), 1310
 MFTransientArray (class in *flopy.mf6.data.mfdataarray*), 1582
 MFTransientList (class in *flopy.mf6.data.mfdatalist*), 1592
 MfusgListBudget (class in *flopy.utils.mflistfile*), 1896
 mg (MfList property), 1952
 micro (Version property), 1916
 minor (Version property), 1916
 Mnw (class in *flopy.modflow.mfmnw2*), 1680
 model (MfList property), 1952
 model (PackageContainerType attribute), 1286
 model (Transient2d property), 1939
 model (Transient3d property), 1942
 model (Util2d property), 1945
 model (Util3d property), 1948
 model_attributes_to_shapefile() (in module *flopy.export.shapefile_utils*), 1995
 model_dict (MFSimulation property), 1306
 model_dimensions (MFSimulationData attribute), 1311
 model_export() (in module *flopy.export.utils*), 2002
 model_factory() (PackageContainer static method), 1282
 model_file_path (Util2d property), 1945
 model_level (PackageLevel attribute), 1603
 model_names (MFSimulation property), 1306
 model_relative_path (MFFileMgmt attribute), 1280
 model_time_units (TemporalReference property), 1924
 model_type (ModflowGwf attribute), 1336
 model_type (ModflowGwt attribute), 1337
 model_ws (BaseModel property), 1609

model_ws (MFModel property), 1291
 model_ws (ModelInterface property), 1612
 modeldiscrit (MFModel property), 1291
 modelgrid (BaseModel property), 1609
 modelgrid (MfGrdFile property), 1566
 modelgrid (MFModel property), 1291
 modelgrid (ModelInterface property), 1612
 modelgrid (Modflow property), 1619
 modelgrid (Mt3dms property), 1773
 modelgrid (Seawat property), 1808
 modelgrid (SimpleRegularGrid property), 1891
 ModelInterface (class in flopy.mbase), 1611
 models (ModflowNam attribute), 1332
 models_by_type (PackageContainer attribute), 1282
 modeltime (BaseModel property), 1609
 ModelTime (class in flopy.discretization.modeltime), 2023
 modeltime (MFModel property), 1291
 modeltime (Modflow property), 1619
 modeltime (Mt3dms property), 1773
 modeltime (Seawat property), 1808
 Modflow (class in flopy.modflow.mf), 1617
 modflow_models (PackageContainer attribute), 1282
 modflow_packages (PackageContainer attribute), 1282
 ModflowAg (class in flopy.modflow.mfag), 1620
 ModflowBas (class in flopy.modflow.mfbas), 1622
 ModflowBcf (class in flopy.modflow.mfbcf), 1624
 ModflowBct (class in flopy.modflow.mfbct), 1627
 ModflowChd (class in flopy.modflow.mfchd), 1627
 ModflowDe4 (class in flopy.modflow.mfde4), 1629
 ModflowDis (class in flopy.modflow.mfdis), 1632
 ModflowDrn (class in flopy.modflow.mfdrn), 1638
 ModflowDrt (class in flopy.modflow.mfdrt), 1640
 ModflowEvt (class in flopy.modflow.mfevt), 1643
 ModflowFhb (class in flopy.modflow.mffhb), 1645
 ModflowFlwob (class in flopy.modflow.mfflwob), 1648
 ModflowGage (class in flopy.modflow.mfgage), 1650
 ModflowGhb (class in flopy.modflow.mfghb), 1652
 ModflowGlobal (class in flopy.modflow.mf), 1619
 ModflowGmg (class in flopy.modflow.mfgmg), 1654
 ModflowGnc (class in flopy.mf6.modflow.mfgnc), 1313
 ModflowGridIndices (class in flopy.utils.gridintersect), 1886
 ModflowGwf (class in flopy.mf6.modflow.mfgwf), 1334
 ModflowGwfapi (class in flopy.mf6.modflow.mfgwfapi), 1337
 ModflowGwfbuy (class in flopy.mf6.modflow.mfgwfbuy), 1339
 ModflowGwfchd (class in flopy.mf6.modflow.mfgwfchd), 1341
 ModflowGwfcsb (class in flopy.mf6.modflow.mfgwfcsb), 1345
 ModflowGwfdis (class in flopy.mf6.modflow.mfgwfdis), 1353
 ModflowGwfdisu (class in flopy.mf6.modflow.mfgwfdisu), 1355
 ModflowGwfdisv (class in flopy.mf6.modflow.mfgwfdisv), 1361
 ModflowGwfdrn (class in flopy.mf6.modflow.mfgwfdrn), 1364
 ModflowGwfevt (class in flopy.mf6.modflow.mfgwfevt), 1368
 ModflowGwfevta (class in flopy.mf6.modflow.mfgwfevta), 1373
 ModflowGwfgfb (class in flopy.mf6.modflow.mfgwfgfb), 1376
 ModflowGwfgnc (class in flopy.mf6.modflow.mfgwfgnc), 1380
 ModflowGwfgwf (class in flopy.mf6.modflow.mfgwfgwf), 1382
 ModflowGwfgwt (class in flopy.mf6.modflow.mfgwfgwt), 1388
 ModflowGwfhfb (class in flopy.mf6.modflow.mfgwfhfb), 1389
 ModflowGwfic (class in flopy.mf6.modflow.mfgwfic), 1390
 ModflowGwflak (class in flopy.mf6.modflow.mfgwflak), 1391
 ModflowGwfmaw (class in flopy.mf6.modflow.mfgwfmaw), 1403
 ModflowGwfmvr (class in flopy.mf6.modflow.mfgwfmvr), 1414
 ModflowGwfnam (class in flopy.mf6.modflow.mfgwfnam), 1418
 ModflowGwfnpf (class in

[flopymf6.modflow.mfgwfnf](#)), 1420
 ModflowGwfoc (class in [flopymf6.modflow.mfgwfoc](#)), 1427
 ModflowGwfrch (class in [flopymf6.modflow.mfgwfrch](#)), 1431
 ModflowGwfrcha (class in [flopymf6.modflow.mfgwfrcha](#)), 1435
 ModflowGwfriv (class in [flopymf6.modflow.mfgwfriv](#)), 1438
 ModflowGwfsfr (class in [flopymf6.modflow.mfgwfsfr](#)), 1442
 ModflowGwfsto (class in [flopymf6.modflow.mfgwfsto](#)), 1454
 ModflowGwfuzf (class in [flopymf6.modflow.mfgwfuzf](#)), 1456
 ModflowGwfvsc (class in [flopymf6.modflow.mfgwfvsc](#)), 1464
 ModflowGwfwel (class in [flopymf6.modflow.mfgwfwel](#)), 1467
 ModflowGwt (class in [flopymf6.modflow.mfgwt](#)), 1336
 ModflowGwtadv (class in [flopymf6.modflow.mfgwtadv](#)), 1472
 ModflowGwtapi (class in [flopymf6.modflow.mfgwtapi](#)), 1472
 ModflowGwtcnc (class in [flopymf6.modflow.mfgwtcnc](#)), 1474
 ModflowGwtdis (class in [flopymf6.modflow.mfgwtdis](#)), 1477
 ModflowGwtdisu (class in [flopymf6.modflow.mfgwtdisu](#)), 1480
 ModflowGwtdivs (class in [flopymf6.modflow.mfgwtdivs](#)), 1485
 ModflowGwt dsp (class in [flopymf6.modflow.mfgwt dsp](#)), 1488
 ModflowGwtfmi (class in [flopymf6.modflow.mfgwtfmi](#)), 1490
 ModflowGwtgwt (class in [flopymf6.modflow.mfgwtgwt](#)), 1492
 ModflowGwtic (class in [flopymf6.modflow.mfgwtic](#)), 1497
 ModflowGwtist (class in [flopymf6.modflow.mfgwtist](#)), 1497
 ModflowGwtlkt (class in [flopymf6.modflow.mfgwtlkt](#)), 1502
 ModflowGwtmst (class in [flopymf6.modflow.mfgwtmst](#)), 1508
 ModflowGwtmvt (class in [flopymf6.modflow.mfgwtmvt](#)), 1511
 ModflowGwtmw (class in [flopymf6.modflow.mfgwtmw](#)), 1512
 ModflowGwt nam (class in [flopymf6.modflow.mfgwt nam](#)), 1518
 ModflowGwtoc (class in

[flopymf6.modflow.mfgwtoc](#)), 1519
 ModflowGwtsft (class in [flopymf6.modflow.mfgwtsft](#)), 1524
 ModflowGwtsrc (class in [flopymf6.modflow.mfgwtsrc](#)), 1530
 ModflowGwtssm (class in [flopymf6.modflow.mfgwtssm](#)), 1533
 ModflowGwtuzt (class in [flopymf6.modflow.mfgwtuzt](#)), 1535
 ModflowHfb (class in [flopymodflow.mfhfb](#)), 1658
 ModflowHob (class in [flopymodflow.mfhob](#)), 1662
 ModflowHyd (class in [flopymodflow.mfhyd](#)), 1664
 ModflowIms (class in [flopymf6.modflow.mfims](#)), 1316
 ModflowLak (class in [flopymodflow.mflak](#)), 1666
 ModflowList (class in [flopymodflow.mf](#)), 1619
 ModflowLmt (class in [flopymodflow.mflmt](#)), 1672
 ModflowLpf (class in [flopymodflow.mflpf](#)), 1674
 ModflowMlt (class in [flopymodflow.mfm lt](#)), 1677
 ModflowMnw1 (class in [flopymodflow.mfmnw1](#)), 1679
 ModflowMnw2 (class in [flopymodflow.mfmnw2](#)), 1688
 ModflowMnwi (class in [flopymodflow.mfmnwi](#)), 1693
 ModflowMvr (class in [flopymf6.modflow.mfmvr](#)), 1326
 ModflowNam (class in [flopymf6.modflow.mfnam](#)), 1330
 ModflowNwt (class in [flopymodflow.mfnwt](#)), 1694
 ModflowOc (class in [flopymodflow.mfoc](#)), 1699
 ModflowPar (class in [flopymodflow.mfpar](#)), 1704
 ModflowParBc (class in [flopymodflow.mfparbc](#)), 1706
 ModflowPbc (class in [flopymodflow.mfpbc](#)), 1708
 ModflowPcg (class in [flopymodflow.mfp cg](#)), 1708
 ModflowPcgn (class in [flopymodflow.mfp cgn](#)), 1710
 ModflowPks (class in [flopymodflow.mfpks](#)), 1714
 ModflowPval (class in

`flopy.modflow.mfpval`), 1716
`ModflowRch` (class in `flopy.modflow.mfrch`), 1717
`ModflowRiv` (class in `flopy.modflow.mfriv`), 1721
`ModflowSfr2` (class in `flopy.modflow.mfsfr2`), 1724
`ModflowSip` (class in `flopy.modflow.mfsip`), 1734
`ModflowSor` (class in `flopy.modflow.mfsor`), 1736
`ModflowStr` (class in `flopy.modflow.mfstr`), 1737
`ModflowSub` (class in `flopy.modflow.mfsub`), 1742
`ModflowSwi2` (class in `flopy.modflow.mfswi2`), 1747
`ModflowSwr1` (class in `flopy.modflow.mfswr1`), 1751
`ModflowSwt` (class in `flopy.modflow.mfswt`), 1752
`ModflowTdis` (class in `flopy.mf6.modflow.mftdis`), 1332
`ModflowUpw` (class in `flopy.modflow.mfupw`), 1757
`ModflowUtlats` (class in `flopy.mf6.modflow.mfutlats`), 1542
`ModflowUtlaktab` (class in `flopy.mf6.modflow.mfutllaktab`), 1544
`ModflowUtllobs` (class in `flopy.mf6.modflow.mfutllobs`), 1546
`ModflowUtlstrtab` (class in `flopy.mf6.modflow.mfutlstrtab`), 1548
`ModflowUtlspc` (class in `flopy.mf6.modflow.mfutlspc`), 1549
`ModflowUtlspca` (class in `flopy.mf6.modflow.mfutlspca`), 1551
`ModflowUtltas` (class in `flopy.mf6.modflow.mfutltas`), 1552
`ModflowUtlts` (class in `flopy.mf6.modflow.mfutlts`), 1554
`ModflowUtltvk` (class in `flopy.mf6.modflow.mfutltvk`), 1556
`ModflowUtltvts` (class in `flopy.mf6.modflow.mfutltvts`), 1559
`ModflowUzf1` (class in `flopy.modflow.mfuzf1`), 1760
`ModflowWel` (class in `flopy.modflow.mfwel`), 1766
`ModflowZon` (class in `flopy.modflow.mfzon`), 1769
`Modpath6` (class in `flopy.modpath.mp6`), 1829

`Modpath6Bas` (class in `flopy.modpath.mp6bas`), 1831
`Modpath6List` (class in `flopy.modpath.mp6`), 1831
`Modpath6Sim` (class in `flopy.modpath.mp6sim`), 1833
`Modpath7` (class in `flopy.modpath.mp7`), 1815
`Modpath7Bas` (class in `flopy.modpath.mp7bas`), 1817
`Modpath7List` (class in `flopy.modpath.mp7`), 1817
`Modpath7Sim` (class in `flopy.modpath.mp7sim`), 1825
module
 `flopy.discretization.grid`, 2015
 `flopy.discretization.modeltime`, 2023
 `flopy.discretization.structuredgrid`, 2023
 `flopy.discretization.unstructuredgrid`, 2031
 `flopy.discretization.vertexgrid`, 2037
 `flopy.export.longnames`, 1990
 `flopy.export.metadata`, 1990
 `flopy.export.netcdf`, 1991
 `flopy.export.shapefile_utils`, 1995
 `flopy.export.unitsformat`, 1998
 `flopy.export.utils`, 1998
 `flopy.export.vtk`, 2004
 `flopy.mbase`, 1605
 `flopy.mf6.data.mfdataarray`, 1578
 `flopy.mf6.data.mfdatalist`, 1587
 `flopy.mf6.data.mfdatascalar`, 1597
 `flopy.mf6.mfbase`, 1279
 `flopy.mf6.mfmodel`, 1286
 `flopy.mf6.mfpackage`, 1294
 `flopy.mf6.modflow.mfgnc`, 1313
 `flopy.mf6.modflow.mfgwf`, 1334
 `flopy.mf6.modflow.mfgwfapi`, 1337
 `flopy.mf6.modflow.mfgwfbuy`, 1339
 `flopy.mf6.modflow.mfgwfchd`, 1341
 `flopy.mf6.modflow.mfgwfcsub`, 1345
 `flopy.mf6.modflow.mfgwfdis`, 1353
 `flopy.mf6.modflow.mfgwfdisu`, 1355
 `flopy.mf6.modflow.mfgwfdisv`, 1361
 `flopy.mf6.modflow.mfgwfdrn`, 1364
 `flopy.mf6.modflow.mfgwfevt`, 1368
 `flopy.mf6.modflow.mfgwfevta`, 1373
 `flopy.mf6.modflow.mfgwfgfb`, 1376
 `flopy.mf6.modflow.mfgwfgnc`, 1380
 `flopy.mf6.modflow.mfgwfgwf`, 1382
 `flopy.mf6.modflow.mfgwfgwt`, 1388
 `flopy.mf6.modflow.mfgwfhib`, 1389
 `flopy.mf6.modflow.mfgwfic`, 1390
 `flopy.mf6.modflow.mfgwflak`, 1391

[flopy.mf6.modflow.mfgwfmaw](#), 1403
[flopy.mf6.modflow.mfgwfmvr](#), 1414
[flopy.mf6.modflow.mfgwfnam](#), 1418
[flopy.mf6.modflow.mfgwfnpf](#), 1420
[flopy.mf6.modflow.mfgwfoc](#), 1427
[flopy.mf6.modflow.mfgwfrch](#), 1431
[flopy.mf6.modflow.mfgwfrcha](#), 1435
[flopy.mf6.modflow.mfgwfriv](#), 1438
[flopy.mf6.modflow.mfgwfsfr](#), 1442
[flopy.mf6.modflow.mfgwfsto](#), 1454
[flopy.mf6.modflow.mfgwfuzf](#), 1456
[flopy.mf6.modflow.mfgwfvsc](#), 1464
[flopy.mf6.modflow.mfgwfwel](#), 1467
[flopy.mf6.modflow.mfgwt](#), 1336
[flopy.mf6.modflow.mfgwtadv](#), 1472
[flopy.mf6.modflow.mfgwtapi](#), 1472
[flopy.mf6.modflow.mfgwtcnc](#), 1474
[flopy.mf6.modflow.mfgwtdis](#), 1477
[flopy.mf6.modflow.mfgwtdisu](#), 1480
[flopy.mf6.modflow.mfgwtdisv](#), 1485
[flopy.mf6.modflow.mfgwt dsp](#), 1488
[flopy.mf6.modflow.mfgwtfmi](#), 1490
[flopy.mf6.modflow.mfgwtgwt](#), 1492
[flopy.mf6.modflow.mfgwtic](#), 1497
[flopy.mf6.modflow.mfgwtist](#), 1497
[flopy.mf6.modflow.mfgwtlkt](#), 1502
[flopy.mf6.modflow.mfgwtmst](#), 1508
[flopy.mf6.modflow.mfgwtmvt](#), 1510
[flopy.mf6.modflow.mfgwtmw](#), 1512
[flopy.mf6.modflow.mfgwtmnam](#), 1518
[flopy.mf6.modflow.mfgwtoc](#), 1519
[flopy.mf6.modflow.mfgwtsft](#), 1524
[flopy.mf6.modflow.mfgwtsrc](#), 1530
[flopy.mf6.modflow.mfgwtssm](#), 1533
[flopy.mf6.modflow.mfgwtuzt](#), 1535
[flopy.mf6.modflow.mfims](#), 1316
[flopy.mf6.modflow.mfmvr](#), 1326
[flopy.mf6.modflow.mfnam](#), 1330
[flopy.mf6.modflow.mfsimulation](#), 1301
[flopy.mf6.modflow.mftdis](#), 1332
[flopy.mf6.modflow.mfutlats](#), 1542
[flopy.mf6.modflow.mfutllaktab](#), 1544
[flopy.mf6.modflow.mfutlobs](#), 1546
[flopy.mf6.modflow.mfutlsfrtab](#), 1548
[flopy.mf6.modflow.mfutlspc](#), 1549
[flopy.mf6.modflow.mfutlspca](#), 1551
[flopy.mf6.modflow.mfutltas](#), 1552
[flopy.mf6.modflow.mfutlts](#), 1554
[flopy.mf6.modflow.mfutltvk](#), 1556
[flopy.mf6.modflow.mfutltvs](#), 1559
[flopy.mf6.utils.binaryfile_utils](#), 1562
[flopy.mf6.utils.binarygrid_util](#), 1563
[flopy.mf6.utils.createpackages](#), 1602
[flopy.mf6.utils.generate_classes](#), 1604
[flopy.mf6.utils.lapak_utils](#), 1577
[flopy.mf6.utils.mfobservation](#), 1568
[flopy.mf6.utils.output_util](#), 1570
[flopy.mf6.utils.postprocessing](#), 1570
[flopy.mf6.utils.reference](#), 1571
[flopy.modflow.mf](#), 1617
[flopy.modflow.mfaddoutsidefile](#), 1619
[flopy.modflow.mfag](#), 1620
[flopy.modflow.mfbas](#), 1622
[flopy.modflow.mfbcf](#), 1624
[flopy.modflow.mfbct](#), 1627
[flopy.modflow.mfchd](#), 1627
[flopy.modflow.mfde4](#), 1629
[flopy.modflow.mfdis](#), 1632
[flopy.modflow.mfdrn](#), 1638
[flopy.modflow.mfdrt](#), 1640
[flopy.modflow.mfevt](#), 1643
[flopy.modflow.mffhb](#), 1645
[flopy.modflow.mfflwob](#), 1648
[flopy.modflow.mfgage](#), 1650
[flopy.modflow.mfghb](#), 1652
[flopy.modflow.mfgmg](#), 1654
[flopy.modflow.mfhfb](#), 1658
[flopy.modflow.mfhob](#), 1660
[flopy.modflow.mfhyd](#), 1664
[flopy.modflow.mflak](#), 1666
[flopy.modflow.mflmt](#), 1672
[flopy.modflow.mflpf](#), 1674
[flopy.modflow.mfm1t](#), 1677
[flopy.modflow.mfmnw1](#), 1679
[flopy.modflow.mfmnw2](#), 1680
[flopy.modflow.mfmnwi](#), 1693
[flopy.modflow.mfnwt](#), 1694
[flopy.modflow.mfoc](#), 1699
[flopy.modflow.mfpar](#), 1704
[flopy.modflow.mfparbc](#), 1706
[flopy.modflow.mfpbc](#), 1708
[flopy.modflow.mfpchg](#), 1708
[flopy.modflow.mfpchg](#), 1710
[flopy.modflow.mfpks](#), 1714
[flopy.modflow.mfpval](#), 1716
[flopy.modflow.mfrch](#), 1717
[flopy.modflow.mfriv](#), 1721
[flopy.modflow.mfsfr2](#), 1724
[flopy.modflow.mfsip](#), 1734
[flopy.modflow.mfsor](#), 1736
[flopy.modflow.mfstr](#), 1737
[flopy.modflow.mfsub](#), 1742
[flopy.modflow.mfswi2](#), 1747
[flopy.modflow.mfswr1](#), 1751
[flopy.modflow.mfswt](#), 1752
[flopy.modflow.mfupw](#), 1757
[flopy.modflow.mfuzf1](#), 1760
[flopy.modflow.mfwel](#), 1766

flopy.modflow.mfzon, 1769
 flopy.modpath.mp6, 1829
 flopy.modpath.mp6bas, 1831
 flopy.modpath.mp6sim, 1833
 flopy.modpath.mp7, 1815
 flopy.modpath.mp7bas, 1817
 flopy.modpath.mp7particledata, 1818
 flopy.modpath.mp7particlegroup, 1823
 flopy.modpath.mp7sim, 1825
 flopy.mt3d.mt, 1771
 flopy.mt3d.mtadv, 1774
 flopy.mt3d.mtbtn, 1777
 flopy.mt3d.mtcts, 1781
 flopy.mt3d.mtdsp, 1784
 flopy.mt3d.mtgcg, 1787
 flopy.mt3d.mtlkt, 1788
 flopy.mt3d.mtphc, 1791
 flopy.mt3d.mtrct, 1791
 flopy.mt3d.mtsft, 1795
 flopy.mt3d.mtssm, 1800
 flopy.mt3d.mttob, 1803
 flopy.mt3d.mtuzt, 1803
 flopy.pakbase, 1614
 flopy.pest.params, 2013
 flopy.pest.templatewriter, 2014
 flopy.pest.tplarray, 2014
 flopy.plot.crossection, 1968
 flopy.plot.map, 1975
 flopy.plot.plotutil, 1980
 flopy.plot.styles, 1986
 flopy.seawat.swt, 1806
 flopy.seawat.swtvdf, 1808
 flopy.seawat.swtvsc, 1812
 flopy.utils.binaryfile, 1835
 flopy.utils.check, 1844
 flopy.utils.compare, 1847
 flopy.utils.crs, 1851
 flopy.utils.cvfdutil, 1853
 flopy.utils.datafile, 1855
 flopy.utils.datautil, 1858
 flopy.utils.flopy_io, 1862
 flopy.utils.formattedfile, 1865
 flopy.utils.geometry, 1867
 flopy.utils.geospatial_utils, 1871
 flopy.utils.get_modflow, 1873
 flopy.utils.gridgen, 1874
 flopy.utils.gridintersect, 1883
 flopy.utils.gridutil, 1888
 flopy.utils.lgrutil, 1889
 flopy.utils.mflistfile, 1891
 flopy.utils.mfreadnam, 1897
 flopy.utils.modpathfile, 1900
 flopy.utils.mtlistfile, 1908
 flopy.utils.observationfile, 1909
 flopy.utils.optionblock, 1913
 flopy.utils.parse_version, 1915
 flopy.utils.postprocessing, 1916
 flopy.utils.rasters, 1919
 flopy.utils.reccarray_utils, 1923
 flopy.utils.reference, 1924
 flopy.utils.sfroutputfile, 1925
 flopy.utils.swroutputfile, 1926
 flopy.utils.triangle, 1931
 flopy.utils.util_array, 1935
 flopy.utils.util_list, 1950
 flopy.utils.utils_def, 1955
 flopy.utils.utl_import, 1957
 flopy.utils.voronoi, 1958
 flopy.utils.zonbud, 1960
 Mt3dAdv (class in flopy.mt3d.mtadv), 1774
 Mt3dBtn (class in flopy.mt3d.mtbtn), 1777
 Mt3dCts (class in flopy.mt3d.mtcts), 1781
 Mt3dDsp (class in flopy.mt3d.mtdsp), 1784
 Mt3dGcg (class in flopy.mt3d.mtgcg), 1787
 Mt3dList (class in flopy.mt3d.mt), 1771
 Mt3dLkt (class in flopy.mt3d.mtlkt), 1788
 Mt3dms (class in flopy.mt3d.mt), 1771
 Mt3dPhc (class in flopy.mt3d.mtphc), 1791
 Mt3dRct (class in flopy.mt3d.mtrct), 1791
 Mt3dSft (class in flopy.mt3d.mtsft), 1795
 Mt3dSsm (class in flopy.mt3d.mtssm), 1800
 Mt3dTob (class in flopy.mt3d.mttob), 1803
 Mt3dUzt (class in flopy.mt3d.mtuzt), 1803
 MtListBudget (class in
 flopy.utils.mtlistfile), 1908
 mult_function() (ModflowMlt static method),
 1678
 multi_line_strip() (in module
 flopy.utils.flopy_io), 1863
 MultiLineString (class in
 flopy.utils.geometry), 1868
 MultiList (class in flopy.utils.datautil),
 1859
 MultiListIter (class in
 flopy.utils.datautil), 1860
 MultiPoint (class in flopy.utils.geometry),
 1868
 MultiPolygon (class in
 flopy.utils.geometry), 1868
 mvr_filerecord (ModflowGwfgwf attribute),
 1388
 MvrPackages (class in
 flopy.mf6.modflow.mfmvr), 1329
 mvt_filerecord (ModflowGwtgwt attribute),
 1496
 mwtperioddata (ModflowGwtmwt attribute),
 1518
 mxact (MfList attribute), 1950

mxact (*MfList* property), 1952
 mxactc (*ModflowChd* attribute), 1628
 mxactr (*ModflowRiv* attribute), 1722
 mxactw (*ModflowWel* attribute), 1768

N

n_nested (*OptionBlock* attribute), 1914
 NamData (class in *flopy.utils.mfreadnam*), 1897
 name (*BaseModel* property), 1609
 name (*MFBlockHeader* attribute), 1296
 name (*MfList* property), 1952
 name (*MFModel* attribute), 1287
 name (*MFPackage* property), 1299
 name (*Package* property), 1615
 name (*PackageInterface* property), 1617
 name (*Transient2d* property), 1939
 name (*Transient3d* property), 1942
 name (*Util2d* property), 1945
 name (*Util3d* property), 1949
 name_file (*MFSimulation* attribute), 1302
 namefile (*BaseModel* property), 1609
 namefile (*MFModel* property), 1291
 namefile (*ModelInterface* property), 1612
 NameIter (class in *flopy.utils.datautil*), 1860
 nc_crs (*NetCdf* property), 1995
 ncells (*MfGrdFile* property), 1566
 ncells (*ZoneFile6* property), 1967
 ncol (*MfGrdFile* property), 1566
 ncol (*Modflow* property), 1619
 ncol (*Mt3dms* property), 1773
 ncol (*Seawat* property), 1808
 ncol (*StructuredGrid* property), 2029
 ncol (*StructuredSpatialReference* property), 1574
 ncomp (*Mt3dms* property), 1773
 ncomp (*Seawat* property), 1808
 ncpl (*Grid* property), 2020
 ncpl (*MfGrdFile* property), 1566
 ncpl (*Modflow* property), 1619
 ncpl (*StructuredGrid* property), 2029
 ncpl (*UnstructuredGrid* property), 2036
 ncpl (*VertexGrid* property), 2040
 ncpl (*VertexSpatialReference* property), 1576
 ncpl_from_ihc() (*UnstructuredGrid* static method), 2036
 ndarray_to_asciigrid() (in module *flopy.utils.gridgen*), 1883
 neighbors() (*Grid* method), 2020
 neighbors() (*StructuredGrid* method), 2029
 nested (*OptionBlock* attribute), 1914
 NetCdf (class in *flopy.export.netcdf*), 1991

new_simulation() (*MFArray* method), 1580
 new_simulation() (*MFList* method), 1588
 new_u2d() (in module *flopy.utils.util_array*), 1950
 next_ext_unit() (*BaseModel* method), 1609
 next_ext_unit() (*Modpath6* method), 1831
 next_item() (*PyListUtil* static method), 1861
 next_list() (*PyListUtil* static method), 1861
 next_unit() (*BaseModel* method), 1609
 nja (*MfGrdFile* property), 1566
 nlay (*Grid* property), 2021
 nlay (*MfGrdFile* property), 1567
 nlay (*Modflow* property), 1619
 nlay (*Mt3dms* property), 1773
 nlay (*Seawat* property), 1808
 nlay (*StructuredGrid* property), 2029
 nlay (*UnstructuredGrid* property), 2036
 nlay (*VertexGrid* property), 2040
 nn0_from_kij() (*ModflowGridIndices* static method), 1887
 nnodes (*Grid* property), 2021
 nnodes (*StructuredGrid* property), 2029
 nnodes (*UnstructuredGrid* property), 2036
 nnodes (*VertexGrid* property), 2040
 no (*budgetOpt* attribute), 1828
 no_ptcrecord (*ModflowIms* attribute), 1326
 nob (CsvFile property), 1909
 nodatavals (*Raster* property), 1921
 nodenumber_from_kij() (*ModflowGridIndices* static method), 1887
 NodeParticleData (class in *flopy.modpath.mp7particledata*), 1821
 nodes (*MfGrdFile* property), 1567
 normal (*VerbosityLevel* attribute), 1286
 normalize_name() (*NetCdf* static method), 1995
 nper (*MFModel* property), 1291
 nper (*ModelTime* property), 2023
 nper (*Modflow* property), 1619
 nper (*ModflowSfr2* property), 1732
 nper (*Mt3dms* property), 1773
 nper (*Seawat* property), 1808
 npl (*ArrayFormat* attribute), 1936
 npl (*ArrayFormat* property), 1937
 nrow (*MfGrdFile* property), 1567
 nrow (*Modflow* property), 1619
 nrow (*Mt3dms* property), 1773
 nrow (*Seawat* property), 1808
 nrow (*StructuredGrid* property), 2029
 nrow (*StructuredSpatialReference* property), 1574

nrow_ncol_nlay_nper (*Modflow* property), 1619
 nrow_ncol_nlay_nper (*Mt3dms* property), 1773
 nrow_ncol_nlay_nper (*Seawat* property), 1808
 nsfrpar (*ModflowSfr2* attribute), 1732
 nss (*ModflowSfr2* property), 1732
 nstp (*ModelTime* property), 2023
 nstrm (*ModflowSfr2* property), 1732
 nth_index() (*MultiList* method), 1860
 numbering() (*check* method), 1733
 numeric_chars (*PyListUtil* attribute), 1861
 numpy (*ArrayFormat* attribute), 1936
 numpy (*ArrayFormat* property), 1937
 nuzgag (*ModflowUzfl* attribute), 1765
 nuzgag (*ModflowUzfl* property), 1766
 nvert (*Grid* property), 2021
 nvert (*StructuredGrid* property), 2029
 nvert (*UnstructuredGrid* property), 2036
 nvert (*VertexGrid* property), 2040

O

obs_filerecord (*ModflowGwfapi* attribute), 1339
 obs_filerecord (*ModflowGwfchd* attribute), 1344
 obs_filerecord (*ModflowGwfcsb* attribute), 1353
 obs_filerecord (*ModflowGwfdn* attribute), 1367
 obs_filerecord (*ModflowGwfevt* attribute), 1372
 obs_filerecord (*ModflowGwfevt* attribute), 1375
 obs_filerecord (*ModflowGwfgbh* attribute), 1379
 obs_filerecord (*ModflowGwfgwf* attribute), 1388
 obs_filerecord (*ModflowGwflak* attribute), 1403
 obs_filerecord (*ModflowGwfma* attribute), 1414
 obs_filerecord (*ModflowGwfrch* attribute), 1434
 obs_filerecord (*ModflowGwfrcha* attribute), 1437
 obs_filerecord (*ModflowGwfriv* attribute), 1441
 obs_filerecord (*ModflowGwfsfr* attribute), 1454
 obs_filerecord (*ModflowGwfuzf* attribute), 1464
 obs_filerecord (*ModflowGfwel* attribute), 1471
 obs_filerecord (*ModflowGwtapi* attribute), 1474
 obs_filerecord (*ModflowGwtcnc* attribute), 1477
 obs_filerecord (*ModflowGwtgwt* attribute), 1496
 obs_filerecord (*ModflowGwtlkt* attribute), 1508
 obs_filerecord (*ModflowGwtmwt* attribute), 1518
 obs_filerecord (*ModflowGwtsft* attribute), 1530
 obs_filerecord (*ModflowGwtsrc* attribute), 1533
 obs_filerecord (*ModflowGwtuzt* attribute), 1542
 obs_names (*MF6Output* property), 1570
 observation_keys() (*SimulationDict* method), 1312
 Observations (class in *flopy.mf6.utils.mfobservation*), 1568
 ObsFiles (class in *flopy.utils.observationfile*), 1910
 obsnames (*CsvFile* property), 1909
 off (*onoffOpt* attribute), 1828
 on (*onoffOpt* attribute), 1828
 onoffOpt (class in *flopy.modpath.mp7sim*), 1828
 open_close_formatting (*MFSimulationData* attribute), 1310
 optional (*OptionBlock* attribute), 1914
 OptionBlock (class in *flopy.utils.optionblock*), 1913
 options (*ModflowBas* attribute), 1622
 OptionUtil (class in *flopy.utils.optionblock*), 1914
 outlets (*ModflowGwflak* attribute), 1403
 outlets (*ModflowSfr2* attribute), 1728
 output (*MFModel* property), 1291
 output (*MFPackage* property), 1300
 output_helper() (in module *flopy.export.utils*), 2003
 output_keys() (*SimulationDict* method), 1313
 outsegs (*ModflowSfr2* attribute), 1728
 overlapping_conductance() (*check* method), 1733

P

Package (class in *flopy.pakbase*), 1614
 package (*MFList* property), 1588
 package (*MfList* property), 1953
 package (*NamData* attribute), 1897

package (PackageContainerType attribute), 1286	package_abbrev (ModflowGwfrcha attribute), 1437
package_abbrev (GncPackages attribute), 1313	package_abbrev (ModflowGwfriv attribute), 1441
package_abbrev (GwfgncPackages attribute), 1380	package_abbrev (ModflowGwfsfr attribute), 1454
package_abbrev (GwfmvrPackages attribute), 1414	package_abbrev (ModflowGwfsto attribute), 1456
package_abbrev (GwtmvtPackages attribute), 1510	package_abbrev (ModflowGwfuzf attribute), 1464
package_abbrev (ModflowGnc attribute), 1316	package_abbrev (ModflowGwfvsc attribute), 1467
package_abbrev (ModflowGwfapi attribute), 1339	package_abbrev (ModflowGwfwel attribute), 1471
package_abbrev (ModflowGwfbuy attribute), 1341	package_abbrev (ModflowGwtadv attribute), 1472
package_abbrev (ModflowGwfchd attribute), 1344	package_abbrev (ModflowGwtapi attribute), 1474
package_abbrev (ModflowGwfcsb attribute), 1353	package_abbrev (ModflowGwtcnc attribute), 1477
package_abbrev (ModflowGwfdis attribute), 1355	package_abbrev (ModflowGwtdis attribute), 1479
package_abbrev (ModflowGwfdisu attribute), 1360	package_abbrev (ModflowGwtdisu attribute), 1485
package_abbrev (ModflowGwfdisv attribute), 1364	package_abbrev (ModflowGwtldisv attribute), 1488
package_abbrev (ModflowGwfdrn attribute), 1367	package_abbrev (ModflowGwtldsp attribute), 1490
package_abbrev (ModflowGwfevt attribute), 1372	package_abbrev (ModflowGwtfmi attribute), 1491
package_abbrev (ModflowGwfevta attribute), 1375	package_abbrev (ModflowGwtgwt attribute), 1496
package_abbrev (ModflowGwfgbh attribute), 1379	package_abbrev (ModflowGwtic attribute), 1497
package_abbrev (ModflowGwfgnc attribute), 1382	package_abbrev (ModflowGwtist attribute), 1501
package_abbrev (ModflowGwfgwf attribute), 1388	package_abbrev (ModflowGwtlkt attribute), 1508
package_abbrev (ModflowGwfgwt attribute), 1388	package_abbrev (ModflowGwtmst attribute), 1510
package_abbrev (ModflowGwfhhb attribute), 1390	package_abbrev (ModflowGwtmvt attribute), 1512
package_abbrev (ModflowGwfic attribute), 1391	package_abbrev (ModflowGwtmwt attribute), 1518
package_abbrev (ModflowGwflak attribute), 1403	package_abbrev (ModflowGwtmam attribute), 1519
package_abbrev (ModflowGwfmaw attribute), 1414	package_abbrev (ModflowGwtoc attribute), 1523
package_abbrev (ModflowGwfmvr attribute), 1417	package_abbrev (ModflowGwtsft attribute), 1530
package_abbrev (ModflowGwfnam attribute), 1419	package_abbrev (ModflowGwtsrc attribute), 1533
package_abbrev (ModflowGwfnpf attribute), 1427	package_abbrev (ModflowGwtssm attribute), 1535
package_abbrev (ModflowGwfoc attribute), 1430	package_abbrev (ModflowGwtuzt attribute), 1542
package_abbrev (ModflowGwfrch attribute), 1434	

- `package_abbr` (*ModflowIms* attribute), 1326
- `package_abbr` (*ModflowMvr* attribute), 1329
- `package_abbr` (*ModflowNam* attribute), 1332
- `package_abbr` (*ModflowTdis* attribute), 1334
- `package_abbr` (*ModflowUtlats* attribute), 1544
- `package_abbr` (*ModflowUtlaktab* attribute), 1545
- `package_abbr` (*ModflowUtlobs* attribute), 1547
- `package_abbr` (*ModflowUtlstftab* attribute), 1549
- `package_abbr` (*ModflowUtlspc* attribute), 1550
- `package_abbr` (*ModflowUtlspca* attribute), 1552
- `package_abbr` (*ModflowUtlts* attribute), 1553
- `package_abbr` (*ModflowUtlts* attribute), 1555
- `package_abbr` (*ModflowUtltvk* attribute), 1558
- `package_abbr` (*ModflowUtltv* attribute), 1561
- `package_abbr` (*MvrPackages* attribute), 1330
- `package_abbr` (*UtlatsPackages* attribute), 1544
- `package_abbr` (*UtlobsPackages* attribute), 1548
- `package_abbr` (*UtltsPackages* attribute), 1554
- `package_abbr` (*UtltsPackages* attribute), 1556
- `package_abbr` (*UtltvkPackages* attribute), 1559
- `package_abbr` (*Utltv* attribute), 1561
- `package_check_levels` (*check* attribute), 1846
- `package_convergence_filerecord` (*ModflowGwfcs* attribute), 1353
- `package_convergence_filerecord` (*ModflowGwflak* attribute), 1403
- `package_convergence_filerecord` (*ModflowGwfsfr* attribute), 1454
- `package_convergence_filerecord` (*ModflowGwfufz* attribute), 1464
- `package_dict` (*PackageContainer* property), 1284
- `package_export()` (in module *flopy.export.utils*), 2003
- `package_factory()` (*PackageContainer* static method), 1284
- `package_key_dict` (*PackageContainer* property), 1284
- `package_list()` (*PackageContainer* static method), 1284
- `package_name_dict` (*PackageContainer* attribute), 1281
- `package_names` (*PackageContainer* property), 1284
- `package_type` (*MFPackage* property), 1300
- `package_type` (*Package* property), 1615
- `package_type` (*PackageInterface* property), 1617
- `package_type_dict` (*PackageContainer* attribute), 1281
- PackageContainer* (class in *flopy.mf6.mfbase*), 1281
- PackageContainerType* (class in *flopy.mf6.mfbase*), 1286
- `packagedata` (*ModflowGwfbuy* attribute), 1341
- `packagedata` (*ModflowGwfcs* attribute), 1353
- `packagedata` (*ModflowGwflak* attribute), 1403
- `packagedata` (*ModflowGwfmaw* attribute), 1414
- `packagedata` (*ModflowGwfsfr* attribute), 1454
- `packagedata` (*ModflowGwfufz* attribute), 1464
- `packagedata` (*ModflowGwfvsc* attribute), 1467
- `packagedata` (*ModflowGwtfmi* attribute), 1491
- `packagedata` (*ModflowGwtlkt* attribute), 1508
- `packagedata` (*ModflowGwtmwt* attribute), 1518
- `packagedata` (*ModflowGwtsft* attribute), 1530
- `packagedata` (*ModflowGwtuzt* attribute), 1542
- PackageInterface* (class in *flopy.pakbase*), 1616
- PackageLevel* (class in *flopy.mf6.utils.createpackages*), 1603
- `packagelist` (*BaseModel* property), 1609
- `packagelist` (*MFMModel* property), 1291
- `packagelist` (*ModelInterface* property), 1612
- PackageLoadException*, 1612
- `packages` (*MFMModel* attribute), 1287
- `packages` (*ModflowGwfmvr* attribute), 1418
- `packages` (*ModflowGwfnam* attribute), 1419
- `packages` (*ModflowGwt* attribute), 1519
- `packages` (*ModflowMvr* attribute), 1329
- `packages_by_abbr` (*PackageContainer* attribute), 1284
- `parameter_bcfill()` (*ModflowParBc* static method), 1708
- `parameter_fill()` (*ModflowPar* static method), 1705
- Params* (class in *flopy.pest.params*), 2013
- `parent` (*Lgr* property), 1890
- `parent` (*MFPackage* property), 1300
- `parent` (*Package* property), 1615
- `parent` (*PackageInterface* property), 1617

parse() (in module *flopy.utils.parse_version*), 1916
 parse() (*MtListBudget* method), 1908
 parse_control_record() (*Util2d* static method), 1945
 parse_modpath_selection_options() (in module *flopy.plot.plotutil*), 1983
 parse_shapely_ix_result() (in module *flopy.utils.gridintersect*), 1888
 parse_value() (*Util2d* method), 1945
 parsenamefile() (in module *flopy.utils.mfreadnam*), 1899
 ParticleData (class in *flopy.modpath.mp7particledata*), 1821
 ParticleGroup (class in *flopy.modpath.mp7particlegroup*), 1823
 ParticleGroupLRCTemplate (class in *flopy.modpath.mp7particlegroup*), 1824
 ParticleGroupNodeTemplate (class in *flopy.modpath.mp7particlegroup*), 1824
 pass_through (*weakOpt* attribute), 1829
 patch (*Polygon* property), 1869
 path (*MFBBlock* attribute), 1294
 path (*MFPackage* attribute), 1297
 PathIter (class in *flopy.utils.datautil*), 1860
 pathline (*simType* attribute), 1828
 PathlineFile (class in *flopy.utils.modpathfile*), 1903
 paths (*ModflowSfr2* property), 1732
 perioddata (*ModflowGwflak* attribute), 1403
 perioddata (*ModflowGwfmaw* attribute), 1414
 perioddata (*ModflowGwfmvr* attribute), 1418
 perioddata (*ModflowGwfsfr* attribute), 1454
 perioddata (*ModflowGwfuzf* attribute), 1464
 perioddata (*ModflowMvr* attribute), 1329
 perioddata (*ModflowTdis* attribute), 1334
 perioddata (*ModflowUtlats* attribute), 1544
 perioddata (*ModflowUtlspc* attribute), 1550
 perioddata (*ModflowUtltvk* attribute), 1558
 perioddata (*ModflowUtltvsv* attribute), 1561
 perlen (*ModelTime* property), 2023
 plot() (*BaseModel* method), 1609
 plot() (*Collection* method), 1867
 plot() (*Gridgen* method), 1882
 plot() (*LayerFile* method), 1856
 plot() (*LineString* method), 1868
 plot() (*MFArray* method), 1580
 plot() (*MFList* method), 1589
 plot() (*MfList* method), 1953
 plot() (*MFModel* method), 1291
 plot() (*MFPackage* method), 1300
 plot() (*MFScalar* method), 1598
 plot() (*MFScalarTransient* method), 1600
 plot() (*MFSimulation* method), 1306
 plot() (*MFTransientArray* method), 1584
 plot() (*MFTransientList* method), 1594
 plot() (*Package* method), 1615
 plot() (*Point* method), 1869
 plot() (*Polygon* method), 1869
 plot() (*Raster* method), 1921
 plot() (*StructuredGrid* method), 2029
 plot() (*Transient2d* method), 1939
 plot() (*Triangle* method), 1934
 plot() (*UnstructuredGrid* method), 2036
 plot() (*Util2d* method), 1946
 plot() (*Util3d* method), 1949
 plot() (*VertexGrid* method), 2040
 plot() (*VoronoiGrid* method), 1959
 plot_array() (*PlotCrossSection* method), 1970
 plot_array() (*PlotMapView* method), 1975
 plot_bc() (*PlotCrossSection* method), 1970
 plot_bc() (*PlotMapView* method), 1976
 plot_boundary() (*Triangle* method), 1934
 plot_centroids() (*Triangle* method), 1935
 plot_cvfd() (in module *flopy.plot.plotutil*), 1983
 plot_endpoint() (*PlotCrossSection* method), 1971
 plot_endpoint() (*PlotMapView* method), 1976
 plot_fill_between() (*PlotCrossSection* method), 1971
 plot_grid() (*PlotCrossSection* method), 1971
 plot_grid() (*PlotMapView* method), 1977
 plot_ibound() (*PlotCrossSection* method), 1971
 plot_ibound() (*PlotMapView* method), 1977
 plot_inactive() (*PlotCrossSection* method), 1972
 plot_inactive() (*PlotMapView* method), 1977
 plot_linestring() (*GridIntersect* static method), 1885
 plot_path() (*ModflowSfr2* method), 1732
 plot_pathline() (*PlotCrossSection* method), 1972
 plot_pathline() (*PlotMapView* method), 1978
 plot_point() (*GridIntersect* static method), 1885
 plot_polygon() (*GridIntersect* static method), 1886
 plot_shapefile() (in module *flopy.plot.plotutil*), 1984
 plot_shapefile() (*PlotMapView* method), 1978

- `plot_shapes()` (*PlotMapView* method), 1978
 - `plot_surface()` (*PlotCrossSection* method), 1973
 - `plot_timeseries()` (*PlotCrossSection* method), 1973
 - `plot_timeseries()` (*PlotMapView* method), 1979
 - `plot_vector()` (*PlotCrossSection* method), 1973
 - `plot_vector()` (*PlotMapView* method), 1979
 - `plot_vertices()` (*Triangle* method), 1935
 - PlotCrossSection* (class in *flopy.plot.crosssection*), 1968
 - PlotMapView* (class in *flopy.plot.map*), 1975
 - plottable* (*MFArrary* property), 1581
 - plottable* (*MFList* property), 1590
 - plottable* (*MfList* property), 1954
 - plottable* (*MFPackage* property), 1300
 - plottable* (*MFScalar* property), 1598
 - plottable* (*MFScalarTransient* property), 1601
 - plottable* (*ModflowAg* property), 1621
 - plottable* (*Package* property), 1615
 - plottable* (*PackageInterface* property), 1617
 - plottable* (*Transient2d* property), 1940
 - plottable* (*Transient3d* property), 1942
 - plottable* (*Util2d* property), 1947
 - plottable* (*Util3d* property), 1950
 - PlotUtilities* (class in *flopy.plot.plotutil*), 1980
 - Point* (class in *flopy.utils.cvfdutil*), 1853
 - Point* (class in *flopy.utils.geometry*), 1869
 - `point_in_cell()` (in module *flopy.utils.voronoi*), 1959
 - `point_in_polygon()` (in module *flopy.utils.geometry*), 1871
 - points* (*GeoSpatialCollection* property), 1872
 - points* (*GeoSpatialUtil* property), 1873
 - Polygon* (class in *flopy.utils.geometry*), 1869
 - polygons* (*PlotCrossSection* property), 1974
 - `pop_item()` (in module *flopy.utils.flopy_io*), 1863
 - porosity* (*ModflowGwtist* attribute), 1501
 - porosity* (*ModflowGwtmst* attribute), 1510
 - post* (*LegacyVersion* property), 1915
 - post* (*Version* property), 1916
 - pre* (*LegacyVersion* property), 1915
 - pre* (*Version* property), 1916
 - `print_summary()` (*check* method), 1846
 - `printrecord` (*ModflowGwfoc* attribute), 1431
 - `printrecord` (*ModflowGwtoc* attribute), 1523
 - prj* (*Grid* property), 2021
 - prjfile* (*Grid* property), 2021
 - proj4* (*Grid* property), 2021
 - `project_point_onto_xc_line()` (in module *flopy.utils.geometry*), 1871
 - property_threshold_values* (*check* attribute), 1846
 - public* (*LegacyVersion* property), 1915
 - public* (*Version* property), 1916
 - Pvd* (class in *flopy.export.vtk*), 2004
 - py* (*ArrayFormat* attribute), 1935
 - py* (*ArrayFormat* property), 1937
 - PyListUtil* (class in *flopy.utils.datautil*), 1860
 - pyshp_parts* (*LineString* property), 1868
 - pyshp_parts* (*Point* property), 1869
 - pyshp_parts* (*Polygon* property), 1869
 - python_file_path* (*Util2d* property), 1947
- ## Q
- `query_grid()` (*GridIntersect* method), 1886
 - quiet* (*VerbosityLevel* attribute), 1286
 - quote_list* (*PyListUtil* attribute), 1861
 - quoted_filename* (*MFPackage* property), 1300
- ## R
- `ra_slice()` (in module *flopy.utils.reccarray_utils*), 1923
 - Raster* (class in *flopy.utils.rasters*), 1919
 - rate* (*ModflowGwfevt* attribute), 1376
 - rcloserecord* (*ModflowIms* attribute), 1326
 - reachperioddata* (*ModflowGwtsft* attribute), 1530
 - `readld()` (in module *flopy.utils.gridgen*), 1883
 - `readld()` (in module *flopy.utils.util_array*), 1950
 - `read_csv()` (*CsvFile* static method), 1909
 - `read_fixed_var()` (in module *flopy.utils.flopy_io*), 1863
 - `read_header()` (*FormattedHeader* method), 1866
 - `read_integer()` (*FlopyBinaryData* method), 1955
 - `read_output()` (*ZoneBudget* class method), 1963
 - `read_real()` (*FlopyBinaryData* method), 1955
 - `read_record()` (*FlopyBinaryData* method), 1955
 - `read_text()` (*FlopyBinaryData* method), 1955
 - `read_usgs_model_reference_file()` (*Grid* method), 2021
 - `read_zone_file()` (*ZoneBudget* class method), 1963
 - ReadAsArraysException*, 1286

recarray() (in module *flopy.utils.recarray_utils*), 1924
 recarray2shp() (in module *flopy.export.shapefile_utils*), 1996
 recharge (*ModflowGwfrcha* attribute), 1437
 record_summary (*budgetOpt* attribute), 1828
 references (*acdd* property), 1991
 register_exchange_file() (*MFSimulation* method), 1307
 register_ims_package() (*MFSimulation* method), 1307
 register_model() (*MFSimulation* method), 1307
 register_package() (*MFModel* method), 1292
 register_package() (*MFSimulation* method), 1307
 register_package() (*PackageContainer* method), 1286
 register_solution_package() (*MFSimulation* method), 1308
 release (*LegacyVersion* property), 1916
 release (*Version* property), 1916
 relpath_safe() (in module *flopy.utils.flopy_io*), 1864
 remove() (*MFPackage* method), 1300
 remove_confining_beds() (*Grid* method), 2021
 remove_edge_ticks() (*styles* class method), 1988
 remove_exchange_file() (*MFSimulation* method), 1308
 remove_external() (*BaseModel* method), 1610
 remove_model() (*MFSimulation* method), 1308
 remove_output() (*BaseModel* method), 1610
 remove_package() (*BaseModel* method), 1610
 remove_package() (*MFModel* method), 1292
 remove_package() (*MFSimulation* method), 1308
 remove_passed() (check method), 1846
 remove_transient_key() (*MFTransientArray* method), 1585
 remove_transient_key() (*MFTransientList* method), 1595
 rename_all_packages() (*MFModel* method), 1293
 rename_all_packages() (*MFSimulation* method), 1308
 rename_model_namefile() (*MFSimulation* method), 1308
 renumber_segments() (*ModflowSfr2* method), 1732
 repair_array_asymmetry() (in module *flopy.utils.gridgen*), 1883
 repair_outsegs() (*ModflowSfr2* method), 1732
 replace_dfn_files() (in module

flopy.mf6.utils.generate_classes), 1604
 reproject_modpath_to_crosssection() (in module *flopy.plot.plotutil*), 1985
 resample_to_grid() (*Raster* method), 1921
 reset() (*StructuredSpatialReference* method), 1574
 reset_budgetunit() (*Modflow0c* method), 1703
 reset_delimiter_used() (*PyListUtil* static method), 1861
 reset_reaches() (*ModflowSfr2* method), 1732
 resolve_exe() (in module *flopy.mbase*), 1612
 resolve_path() (*MFFileMgmt* method), 1280
 rewet_record (*ModflowGwfnpf* attribute), 1427
 riv_array_comp() (*PyListUtil* method), 1861
 rotate() (in module *flopy.utils.geometry*), 1871
 rotate() (*StructuredSpatialReference* static method), 1574
 rotate() (*VertexSpatialReference* static method), 1576
 routing() (check method), 1733
 run_all() (check method), 1733
 run_main() (in module *flopy.utils.get_modflow*), 1873
 run_model() (*BaseModel* method), 1610
 run_model() (in module *flopy.mbase*), 1613
 run_model() (*ZoneBudget6* method), 1966
 run_simulation() (*MFSimulation* method), 1308

S

sample_point() (*Raster* method), 1922
 sample_polygon() (*Raster* method), 1922
 saturated_thick() (*Grid* method), 2021
 saturated_thickness() (*Grid* method), 2022
 saturated_thickness() (*PlotUtilities* static method), 1980
 save_array() (*PyListUtil* method), 1861
 save_array_diff() (*PyListUtil* method), 1861
 saverrecord (*ModflowGwfoc* attribute), 1431
 saverrecord (*ModflowGwtoc* attribute), 1523
 sci_note_lower_thres (*MFSimulationData* attribute), 1311
 sci_note_upper_thres (*MFSimulationData* attribute), 1311
 scrub_login() (in module *flopy.utils.flopy_io*), 1864
 search_data() (*MFList* method), 1590
 Seawat (class in *flopy.seawat.swt*), 1806
 SeawatList (class in *flopy.seawat.swt*), 1808

SeawatVdf (class in *flopy.seawat.swtvdf*), 1808
 SeawatVsc (class in *flopy.seawat.swtvsc*), 1812
 segment_face() (in module *flopy.utils.cvfduutil*), 1854
 segment_list (ModflowAg property), 1621
 set_all_data_external() (MFBBlock method), 1295
 set_all_data_external() (MFModel method), 1293
 set_all_data_external() (MFPackage method), 1300
 set_all_data_external() (MFSimulation method), 1309
 set_all_data_internal() (MFBBlock method), 1295
 set_all_data_internal() (MFModel method), 1293
 set_all_data_internal() (MFPackage method), 1301
 set_all_data_internal() (MFSimulation method), 1309
 set_budget_key() (ListBudget method), 1896
 set_budget_key() (Mf6ListBudget method), 1896
 set_budget_key() (MfListBudget method), 1896
 set_budget_key() (MfusgListBudget method), 1896
 set_budget_key() (SvrListBudget method), 1896
 set_budget_key() (SwtListBudget method), 1896
 set_coord_info() (Grid method), 2022
 set_data() (MFArrary method), 1581
 set_data() (MFList method), 1590
 set_data() (MFScalar method), 1599
 set_data() (MFScalarTransient method), 1601
 set_data() (MFTransientArray method), 1585
 set_data() (MFTransientList method), 1595
 set_dtype() (BinaryHeader static method), 1835
 set_float() (FlopyBinaryData method), 1955
 set_fmtin() (Util2d method), 1947
 set_font_type() (styles class method), 1989
 set_ifrefm() (Modflow method), 1619
 set_last_accessed_model_path() (MFFileMgmt method), 1280
 set_last_accessed_path() (MFFileMgmt method), 1280
 set_layered_data() (MFArrary method), 1581
 set_model_relative_path() (MFBBlock method), 1295
 set_model_relative_path() (MFModel method), 1293
 set_model_relative_path() (MFPackage method), 1301
 set_model_units() (BaseModel method), 1610
 set_model_units() (Modflow method), 1619
 set_mult() (ModflowPar method), 1705
 set_ncpl() (UnstructuredGrid method), 2037
 set_output_attribute() (BaseModel method), 1611
 set_outreaches() (ModflowSfr2 method), 1732
 set_pval() (ModflowPar method), 1706
 set_record() (MFArrary method), 1581
 set_record() (MFList method), 1590
 set_record() (MFTransientArray method), 1586
 set_record() (MFTransientList method), 1595
 set_sci_note_lower_thres() (MFSimulationData method), 1311
 set_sci_note_upper_thres() (MFSimulationData method), 1312
 set_sim_path() (MFFileMgmt method), 1281
 set_sim_path() (MFSimulation method), 1309
 set_spatialreference() (StructuredSpatialReference method), 1575
 set_spatialreference() (VertexSpatialReference method), 1576
 set_surface_interpolation() (Gridgen method), 1882
 set_values() (BinaryHeader method), 1835
 set_version() (BaseModel method), 1611
 set_zcentergrid() (PlotCrossSection method), 1974
 set_zone() (ModflowPar method), 1706
 set_zpts() (PlotCrossSection method), 1974
 setmodflowvars() (Mt3dBtn method), 1780
 sfacrecord (ModflowUtilts attribute), 1553
 sfacrecord (ModflowUtilts attribute), 1556
 sfacrecord_single (ModflowUtilts attribute), 1556
 SfrFile (class in *flopy.utils.sfroutputfile*), 1925
 sgm (ModflowGwfcs sub attribute), 1353
 sgs (ModflowGwfcs sub attribute), 1353
 Shape (class in *flopy.utils.geometry*), 1869
 shape (GeoSpatialCollection property), 1872
 shape (GeoSpatialUtil property), 1873
 shape (Grid property), 2022
 shape (MfGrdFile property), 1567
 shape (StructuredGrid property), 2029
 shape (UnstructuredGrid property), 2037
 shape (VertexGrid property), 2040

shape_attr_name() (in module *flopy.export.shapefile_utils*), 1997
 shapefile_extents() (in module *flopy.plot.plotutil*), 1985
 shapefile_get_vertices() (in module *flopy.plot.plotutil*), 1985
 shapefile_to_cvfd() (in module *flopy.utils.cvfdutil*), 1854
 shapefile_to_patch_collection() (in module *flopy.plot.plotutil*), 1986
 shapefile_to_xcyc() (in module *flopy.utils.cvfdutil*), 1854
 shapely (GeoSpatialCollection property), 1872
 shapely (GeoSpatialUtil property), 1873
 shapetype (GeoSpatialCollection property), 1872
 shapetype (GeoSpatialUtil property), 1873
 shapeType (LineString attribute), 1868
 shapeType (Point attribute), 1869
 shapeType (Polygon attribute), 1869
 shared_face() (in module *flopy.utils.cvfdutil*), 1854
 shp2recarray() (in module *flopy.export.shapefile_utils*), 1997
 sim (Modpath6 property), 1831
 sim_enum_error() (in module *flopy.modpath.mp7sim*), 1828
 sim_level (PackageLevel attribute), 1603
 sim_name (MFSimulation attribute), 1302
 sim_package_list (MFSimulation property), 1309
 sim_path (MFSimulation property), 1309
 simple_flag (OptionBlock attribute), 1914
 simple_float (OptionBlock attribute), 1914
 simple_int (OptionBlock attribute), 1914
 simple_str (OptionBlock attribute), 1914
 simple_tabfile (OptionBlock attribute), 1914
 SimpleRegularGrid (class in *flopy.utils.lgrutil*), 1890
 simType (class in *flopy.modpath.mp7sim*), 1828
 simulation (PackageContainerType attribute), 1286
 SimulationDict (class in *flopy.mf6.modflow.mfsimulation*), 1312
 single_line_options (OptionBlock property), 1914
 size (Grid property), 2022
 skipcomments() (in module *flopy.modflow.mfmrw1*), 1680
 slope() (check method), 1733
 solutiongroup (ModflowNam attribute), 1332
 solver_packages (check attribute), 1846
 solver_tols (MFModel property), 1293
 solver_tols (ModelInterface property), 1612
 solver_tols (Modflow property), 1619
 solver_tols (Mt3dms property), 1773
 sort_gridshapes() (GridIntersect static method), 1886
 sort_node_data() (Mnw static method), 1687
 sort_tuple() (in module *flopy.utils.zonbud*), 1968
 sort_vertices() (in module *flopy.utils.voronoi*), 1959
 sources (ModflowGwtssm attribute), 1535
 sp2 (ModflowGwtmst attribute), 1510
 SpatialReference (class in *flopy.mf6.utils.reference*), 1571
 specified (stopOpt attribute), 1829
 split_data_line() (PyListUtil static method), 1861
 sr (Mt3dms property), 1773
 ss (ModflowGwfsto attribute), 1456
 SsmPackage (class in *flopy.mt3d.mtssm*), 1803
 stage_filerecord (ModflowGwflak attribute), 1403
 stage_filerecord (ModflowGwfsfr attribute), 1454
 start_datetime (ModelTime property), 2023
 StartingLocationsFile (class in *flopy.modpath.mp6sim*), 1834
 steady_state (ModelTime property), 2023
 stop_at (weakOpt attribute), 1829
 stopOpt (class in *flopy.modpath.mp7sim*), 1828
 store_as_external_file() (MFArray method), 1582
 store_as_external_file() (MFList method), 1590
 store_as_external_file() (MFTransientArray method), 1586
 store_as_external_file() (MFTransientList method), 1596
 store_internal() (MFArray method), 1582
 store_internal() (MFList method), 1591
 store_internal() (MFTransientArray method), 1586
 store_internal() (MFTransientList method), 1596
 straincg_filerecord (ModflowGwfcsb attribute), 1353
 strainib_filerecord (ModflowGwfcsb attribute), 1353
 stress_period_data (ModflowGwfchd

attribute), 1344
 stress_period_data (ModflowGwfcsb attribute), 1353
 stress_period_data (ModflowGwfdrn attribute), 1367
 stress_period_data (ModflowGwfevt attribute), 1372
 stress_period_data (ModflowGwfgfb attribute), 1379
 stress_period_data (ModflowGwfhhb attribute), 1390
 stress_period_data (ModflowGwfrch attribute), 1434
 stress_period_data (ModflowGwfriv attribute), 1441
 stress_period_data (ModflowGwfwel attribute), 1471
 stress_period_data (ModflowGwtcnc attribute), 1477
 stress_period_data (ModflowGwtsrc attribute), 1533
 stress_period_data_values() (check method), 1846
 string (Util2d property), 1947
 strip_model_relative_path() (MFFileMgmt method), 1281
 strt (ModflowGwfic attribute), 1391
 strt (ModflowGwtic attribute), 1497
 StructException, 1286
 structure (MFBBlock attribute), 1294
 structure (MFPackage attribute), 1298
 StructuredGrid (class in *flopy.discretization.structuredgrid*), 2023
 StructuredSpatialReference (class in *flopy.mf6.utils.reference*), 1572
 styles (class in *flopy.plot.styles*), 1986
 sum() (Util2d method), 1947
 sum_flux_tuples() (in module *flopy.utils.zonbud*), 1968
 summarize() (check method), 1846
 summary (budgetOpt attribute), 1828
 supports_layered() (MFArray method), 1582
 surface (ModflowGwfevta attribute), 1376
 SwiConcentration (class in *flopy.plot.plotutil*), 1981
 SwrBudget (class in *flopy.utils.swroutputfile*), 1926
 SwrExchange (class in *flopy.utils.swroutputfile*), 1926
 SwrFile (class in *flopy.utils.swroutputfile*), 1927
 SwrFlow (class in *flopy.utils.swroutputfile*), 1929

SwrListBudget (class in *flopy.utils.mflistfile*), 1896
 SwrObs (class in *flopy.utils.observationfile*), 1912
 SwrStage (class in *flopy.utils.swroutputfile*), 1930
 SwrStructure (class in *flopy.utils.swroutputfile*), 1930
 SwtListBudget (class in *flopy.utils.mflistfile*), 1896
 sy (ModflowGwfsto attribute), 1456

T

table (ModflowUtlaktab attribute), 1545
 table (ModflowUtlstfrtab attribute), 1549
 tables (ModflowGwflak attribute), 1403
 tas_array (ModflowUtlstas attribute), 1553
 tas_filerecord (ModflowGwfevta attribute), 1376
 tas_filerecord (ModflowGwfrcha attribute), 1437
 tas_filerecord (ModflowUtlspca attribute), 1552
 TemplateWriter (class in *flopy.pest.templatewriter*), 2014
 TemporalReference (class in *flopy.utils.reference*), 1924
 thick (Grid property), 2022
 thin_cell_threshold (check attribute), 1846
 time_const (ModflowSfr2 attribute), 1732
 time_coverage (acdd property), 1991
 time_series_namerecord (ModflowUtlstas attribute), 1553
 time_series_namerecord (ModflowUtlstas attribute), 1556
 time_units (ModelTime property), 2023
 timeseries (ModflowUtlstas attribute), 1556
 timeseries (simType attribute), 1828
 TimeseriesFile (class in *flopy.utils.modpathfile*), 1905
 to_array() (MFList method), 1591
 to_array() (MfList method), 1954
 to_array() (MFTransientList method), 1596
 to_csv() (ZoneBudget method), 1964
 to_cvfd() (in module *flopy.utils.cvfdutil*), 1854
 to_pyvista() (Vtk method), 2007
 to_shapefile() (BaseModel method), 1611
 to_shapefile() (LayerFile method), 1858
 to_shapefile() (MfList method), 1954
 to_shapefile() (Package method), 1615
 top (Grid attribute), 2016
 top (Grid property), 2022
 top (MfGrdFile property), 1567

[top \(ModflowGwfdis attribute\)](#), 1355
[top \(ModflowGwfdisu attribute\)](#), 1361
[top \(ModflowGwfdisv attribute\)](#), 1364
[top \(ModflowGwtddis attribute\)](#), 1480
[top \(ModflowGwtddisu attribute\)](#), 1485
[top \(ModflowGwtddisv attribute\)](#), 1488
[top_botm \(Grid property\)](#), 2022
[top_botm \(StructuredGrid property\)](#), 2030
[top_botm \(UnstructuredGrid property\)](#), 2037
[top_botm \(VertexGrid property\)](#), 2040
[top_botm_withnan \(StructuredGrid property\)](#), 2030
[total \(stopOpt attribute\)](#), 1829
[totim \(ModelTime property\)](#), 2023
[totim_to_datetime\(\) \(in module \[flopy.utils.utils_def\]\(#\)\)](#), 1956
[trackDir \(class in \[flopy.modpath.mp7sim\]\(#\)\)](#), 1829
[transform\(\) \(in module \[flopy.utils.geometry\]\(#\)\)](#), 1871
[Transient2d \(class in \[flopy.utils.util_array\]\(#\)\)](#), 1937
[transient2d_export\(\) \(in module \[flopy.export.utils\]\(#\)\)](#), 2004
[Transient2dTpl \(class in \[flopy.pest.tplarray\]\(#\)\)](#), 2014
[Transient3d \(class in \[flopy.utils.util_array\]\(#\)\)](#), 1940
[transient_2ds \(Transient2d attribute\)](#), 1938
[transient_3ds \(Transient3d attribute\)](#), 1941
[tri2vor\(\) \(in module \[flopy.utils.voronoi\]\(#\)\)](#), 1959
[Triangle \(class in \[flopy.utils.triangle\]\(#\)\)](#), 1931
[try_float\(\) \(in module \[flopy.mf6.utils.mfobservation\]\(#\)\)](#), 1570
[ts_filerecord \(ModflowGwfchd attribute\)](#), 1344
[ts_filerecord \(ModflowGwfcsu attribute\)](#), 1353
[ts_filerecord \(ModflowGwfdrn attribute\)](#), 1367
[ts_filerecord \(ModflowGwfevt attribute\)](#), 1373
[ts_filerecord \(ModflowGwfgbh attribute\)](#), 1379
[ts_filerecord \(ModflowGwflak attribute\)](#), 1403
[ts_filerecord \(ModflowGwfmaw attribute\)](#), 1414
[ts_filerecord \(ModflowGwfrch attribute\)](#), 1434
[ts_filerecord \(ModflowGwfriv attribute\)](#),

[1441](#)
[ts_filerecord \(ModflowGwfsfr attribute\)](#), 1454
[ts_filerecord \(ModflowGwfufz attribute\)](#), 1464
[ts_filerecord \(ModflowGwfwel attribute\)](#), 1471
[ts_filerecord \(ModflowGwtcnc attribute\)](#), 1477
[ts_filerecord \(ModflowGwtlkt attribute\)](#), 1508
[ts_filerecord \(ModflowGwtmwt attribute\)](#), 1518
[ts_filerecord \(ModflowGwtsft attribute\)](#), 1530
[ts_filerecord \(ModflowGwtsrc attribute\)](#), 1533
[ts_filerecord \(ModflowGwtuzt attribute\)](#), 1542
[ts_filerecord \(ModflowUtlspc attribute\)](#), 1551
[ts_filerecord \(ModflowUtltvk attribute\)](#), 1558
[ts_filerecord \(ModflowUtltvsv attribute\)](#), 1561
[tslen \(ModelTime property\)](#), 2023
[tsmult \(ModelTime property\)](#), 2023
[tvk_filerecord \(ModflowGwfnpf attribute\)](#), 1427
[tvsv_filerecord \(ModflowGwfsto attribute\)](#), 1456
[type \(LineString attribute\)](#), 1868
[type \(Point attribute\)](#), 1869
[type \(Polygon attribute\)](#), 1869
[type_from_iterable\(\) \(in module \[flopy.utils.utils_def\]\(#\)\)](#), 1957

U

[UcnFile \(class in \[flopy.utils.binaryfile\]\(#\)\)](#), 1842
[ulstrd\(\) \(in module \[flopy.utils.flopy_io\]\(#\)\)](#), 1864
[uniform_flow_field\(\) \(in module \[flopy.utils.gridutil\]\(#\)\)](#), 1889
[unique\(\) \(Util2d method\)](#), 1947
[unique_file_name\(\) \(MFFileMgmt static method\)](#), 1281
[unitnumber \(Mt3dGcg attribute\)](#), 1788
[unitnumber \(Mt3dPhc attribute\)](#), 1791
[unitnumber \(SeawatVdf attribute\)](#), 1812
[unitnumber \(SeawatVsc attribute\)](#), 1815
[units \(Grid property\)](#), 2022
[UnstructuredGrid \(class in \[flopy.discretization.unstructuredgrid\]\(#\)\)](#),

2031
 UnstructuredPlotUtilities (class in *flopy.plot.plotutil*), 1981
 update_data() (*CachedData* method), 2015
 update_from_package() (*OptionBlock* method), 1914
 update_modelgrid() (*ModelInterface* method), 1612
 update_package_filename() (*MFModel* method), 1293
 update_package_filename() (*MFSimulation* method), 1309
 update_record() (*MFList* method), 1591
 update_record() (*MFTransientList* method), 1596
 USGSMap() (*styles* class method), 1986
 USGSPlot() (*styles* class method), 1986
 Util2d (class in *flopy.utils.util_array*), 1942
 Util2dTpl (class in *flopy.pest.tplarray*), 2014
 Util3d (class in *flopy.utils.util_array*), 1947
 Util3dTpl (class in *flopy.pest.tplarray*), 2014
 UtlatsPackages (class in *flopy.mf6.modflow.mfutlats*), 1544
 UtlobsPackages (class in *flopy.mf6.modflow.mfutlobs*), 1547
 UtltasPackages (class in *flopy.mf6.modflow.mfutltas*), 1553
 UtltsPackages (class in *flopy.mf6.modflow.mfutlts*), 1556
 UtltvkPackages (class in *flopy.mf6.modflow.mfutltvk*), 1558
 Utltvspackages (class in *flopy.mf6.modflow.mfutltvs*), 1561
 uzgag (*ModflowUzf1* property), 1766
 uztperioddata (*ModflowGwtuzt* attribute), 1542

V

values() (*check* method), 1846
 variable_strings (*MFBBlockHeader* attribute), 1296
 vars (*OptionBlock* attribute), 1914
 verbose (*BaseModel* property), 1611
 verbose (*MFModel* property), 1293
 verbose (*ModelInterface* property), 1612
 verbose (*VerbosityLevel* attribute), 1286
 VerbosityLevel (class in *flopy.mf6.mfbase*), 1286
 version (*BaseModel* property), 1611

Version (class in *flopy.utils.parse_version*), 1916
 version (*MFModel* property), 1293
 version (*ModelInterface* property), 1612
 VertexGrid (class in *flopy.discretization.vertexgrid*), 2037
 VertexSpatialReference (class in *flopy.mf6.utils.reference*), 1575
 vertical_datum (*acdd* property), 1991
 vertices (*ModflowGwfdisu* attribute), 1361
 vertices (*ModflowGwfdisv* attribute), 1364
 vertices (*ModflowGwtdisu* attribute), 1485
 vertices (*ModflowGwtdisv* attribute), 1488
 verts (*Grid* property), 2022
 verts (*MfGrdFile* property), 1567
 verts (*StructuredGrid* property), 2030
 verts (*UnstructuredGrid* property), 2037
 verts (*VertexGrid* property), 2040
 view_summary_array_fields() (*check* method), 1846
 viscosity_filerecord (*ModflowGwfvsc* attribute), 1467
 volfrac (*ModflowGwtist* attribute), 1501
 VoronoiGrid (class in *flopy.utils.voronoi*), 1958
 Vtk (class in *flopy.export.vtk*), 2005
 vtype (*MFList* property), 1955
 vtype (*Util2d* property), 1947

W

warn() (*Logger* method), 1991
 wc_filerecord (*ModflowGwfufz* attribute), 1464
 weakOpt (class in *flopy.modpath.mp7sim*), 1829
 webdoc() (*Package* method), 1616
 wetdry (*ModflowGwfnpf* attribute), 1427
 width (*ArrayFormat* attribute), 1936
 width (*ArrayFormat* property), 1937
 wrap_multidim_arrays (*MFSimulationData* attribute), 1310
 write() (*CellDataType* method), 1819
 write() (*FaceDataType* method), 1820
 write() (*LRCParticleData* method), 1821
 write() (*MFBBlock* method), 1296
 write() (*MFModel* method), 1293
 write() (*MFPackage* method), 1301
 write() (*NetCDF* method), 1995
 write() (*NodeParticleData* method), 1821
 write() (*ParticleData* method), 1823
 write() (*ParticleGroup* method), 1824
 write() (*ParticleGroupLRCTemplate* method), 1824

```

write() (ParticleGroupNodeTemplate method),
    1825
write() (Pvd method), 2004
write() (Raster method), 1922
write() (Vtk method), 2008
write_bin() (Util2d static method), 1947
write_budget() (in module
    flopy.utils.binaryfile), 1843
write_file() (mfaddoutsidefile method),
    1619
write_file() (ModflowAg method), 1621
write_file() (ModflowBas method), 1624
write_file() (ModflowBcf method), 1626
write_file() (ModflowBct method), 1627
write_file() (ModflowChd method), 1629
write_file() (ModflowDe4 method), 1631
write_file() (ModflowDis method), 1637
write_file() (ModflowDrn method), 1640
write_file() (ModflowDrt method), 1642
write_file() (ModflowEvt method), 1644
write_file() (ModflowFhb method), 1647
write_file() (ModflowFlwob method), 1650
write_file() (ModflowGage method), 1652
write_file() (ModflowGhb method), 1654
write_file() (ModflowGlobal method), 1619
write_file() (ModflowGmg method), 1658
write_file() (ModflowHfb method), 1660
write_file() (ModflowHob method), 1663
write_file() (ModflowHyd method), 1666
write_file() (ModflowLak method), 1671
write_file() (ModflowList method), 1619
write_file() (ModflowLmt method), 1673
write_file() (ModflowLpf method), 1677
write_file() (ModflowMlt method), 1678
write_file() (ModflowMnw1 method), 1680
write_file() (ModflowMnw2 method), 1693
write_file() (ModflowMnwi method), 1694
write_file() (ModflowNwt method), 1698
write_file() (ModflowOc method), 1704
write_file() (ModflowPbc method), 1708
write_file() (ModflowPcg method), 1710
write_file() (ModflowPcgn method), 1714
write_file() (ModflowPks method), 1716
write_file() (ModflowPval method), 1717
write_file() (ModflowRch method), 1720
write_file() (ModflowRiv method), 1723
write_file() (ModflowSfr2 method), 1733
write_file() (ModflowSip method), 1735
write_file() (ModflowSor method), 1737
write_file() (ModflowStr method), 1742
write_file() (ModflowSub method), 1746
write_file() (ModflowSwi2 method), 1750
write_file() (ModflowSwr1 method), 1752
write_file() (ModflowSwt method), 1757
write_file() (ModflowUpw method), 1760
write_file() (ModflowUzf1 method), 1766
write_file() (ModflowWel method), 1769
write_file() (ModflowZon method), 1770
write_file() (Modpath6Bas method), 1832
write_file() (Modpath6List method), 1831
write_file() (Modpath6Sim method), 1834
write_file() (Modpath7Bas method), 1818
write_file() (Modpath7List method), 1817
write_file() (Modpath7Sim method), 1828
write_file() (Mt3dAdv method), 1777
write_file() (Mt3dBtn method), 1780
write_file() (Mt3dDsp method), 1786
write_file() (Mt3dGcg method), 1788
write_file() (Mt3dList method), 1771
write_file() (Mt3dLkt method), 1791
write_file() (Mt3dPhc method), 1791
write_file() (Mt3dRct method), 1795
write_file() (Mt3dSft method), 1799
write_file() (Mt3dSsm method), 1802
write_file() (Mt3dTob method), 1803
write_file() (Mt3dUzt method), 1806
write_file() (Package method), 1616
write_file() (SeawatList method), 1808
write_file() (SeawatVdf method), 1812
write_file() (SeawatVsc method), 1815
write_file() (StartingLocationsFile
    method), 1834
write_fixed_var() (in module
    flopy.utils.flopy_io), 1865
write_footer() (MFBBlockHeader method), 1297
write_grid_shapefile() (in module
    flopy.export.shapefile_utils), 1997
write_gridlines_shapefile() (in module
    flopy.export.shapefile_utils), 1998
write_gridSpec() (StructuredSpatialReference
    method), 1575
write_head() (in module
    flopy.utils.binaryfile), 1843
write_header() (MFBBlockHeader method), 1297
write_headers (MFSimulationData attribute),
    1311
write_input() (BaseModel method), 1611
write_input() (ZoneBudget6 method), 1967
write_input() (ZoneFile6 method), 1967
write_name_file() (BaseModel method), 1611
write_name_file() (Modflow method), 1619
write_name_file() (Modpath6 method), 1831
write_name_file() (Modpath7 method), 1817
write_name_file() (Mt3dms method), 1773
write_name_file() (Seawat method), 1808
write_options() (OptionBlock method), 1914
write_prj() (in module
    flopy.export.shapefile_utils), 1998

```

write_shapefile() (*EndpointFile* method), 1902
 write_shapefile() (*Grid* method), 2022
 write_shapefile() (*PathlineFile* method), 1905
 write_shapefile() (*TimeseriesFile* method), 1907
 write_simulation() (*MFSimulation* method), 1309
 write_template() (*TemplateWriter* method), 2014
 write_transient() (*MfList* method), 1955
 write_txt() (*Util2d* static method), 1947
 write_zone_file() (*ZoneBudget* class method), 1964

X

x (*LineString* property), 1868
 x (*Point* property), 1869
 xarr (*VertexSpatialReference* property), 1576
 xcellcenters (*Grid* property), 2022
 xcenter (*StructuredSpatialReference* attribute), 1573
 xcenter (*StructuredSpatialReference* property), 1575
 xcenter (*VertexSpatialReference* attribute), 1576
 xcenter_array (*VertexSpatialReference* property), 1576
 xcentergrid (*StructuredSpatialReference* attribute), 1573
 xcentergrid (*StructuredSpatialReference* property), 1575
 xcentergrid (*VertexSpatialReference* attribute), 1576
 xcenters (*Grid* attribute), 2017
 xcenters (*Raster* property), 1923
 xdict (*VertexSpatialReference* property), 1577
 xedge (*StructuredSpatialReference* attribute), 1572
 xedge (*StructuredSpatialReference* property), 1575
 xedge (*VertexSpatialReference* attribute), 1575
 xgrid (*Grid* attribute), 2016
 xgrid (*StructuredSpatialReference* attribute), 1573
 xgrid (*StructuredSpatialReference* property), 1575
 xgrid (*VertexSpatialReference* attribute), 1575
 xlabel() (*styles* class method), 1989

xmlfile (*acdd* property), 1991
 xmlroot (*acdd* property), 1991
 xoffset (*Grid* attribute), 2016
 xoffset (*Grid* property), 2022
 xorigin (*MfGrdFile* property), 1567
 xvertices (*Grid* property), 2022
 xycenters (*StructuredGrid* property), 2030
 xydict (*VertexSpatialReference* property), 1577
 xyedges (*StructuredGrid* property), 2030
 xyzcellcenters (*Grid* attribute), 2017
 xyzcellcenters (*Grid* property), 2022
 xyzcellcenters (*StructuredGrid* property), 2030
 xyzcellcenters (*UnstructuredGrid* property), 2037
 xyzcellcenters (*VertexGrid* property), 2040
 xyzextent (*Grid* property), 2023
 xyzgrid (*Grid* attribute), 2017
 xyzvertices (*Grid* property), 2023
 xyzvertices (*StructuredGrid* property), 2030
 xyzvertices (*UnstructuredGrid* property), 2037
 xyzvertices (*VertexGrid* property), 2040

Y

y (*LineString* property), 1868
 y (*Point* property), 1869
 yarr (*VertexSpatialReference* property), 1577
 ycellcenters (*Grid* property), 2023
 ycenter (*StructuredSpatialReference* attribute), 1573
 ycenter (*StructuredSpatialReference* property), 1575
 ycenter (*VertexSpatialReference* attribute), 1576
 ycenter_array (*VertexSpatialReference* property), 1577
 ycentergrid (*StructuredSpatialReference* attribute), 1573
 ycentergrid (*StructuredSpatialReference* property), 1575
 ycentergrid (*VertexSpatialReference* attribute), 1576
 ycenters (*Grid* attribute), 2017
 ycenters (*Raster* property), 1923
 ydict (*VertexSpatialReference* property), 1577
 yedge (*StructuredSpatialReference* attribute), 1573
 yedge (*StructuredSpatialReference* property), 1575

yedge (*VertexSpatialReference* attribute),
1575
ygrid (*Grid* attribute), 2017
ygrid (*StructuredSpatialReference*
attribute), 1573
ygrid (*StructuredSpatialReference*
property), 1575
ygrid (*VertexSpatialReference* attribute),
1576
ylabel() (*styles* class method), 1989
yoffset (*Grid* attribute), 2016
yoffset (*Grid* property), 2023
yorigin (*MfGrdFile* property), 1568
yvertices (*Grid* property), 2023

Z

z (*LineString* property), 1868
z (*Point* property), 1869
ZBNetOutput (class in *flopy.utils.zonbud*),
1960
zcellcenters (*Grid* property), 2023
zcentroids (*ModflowDis* property), 1637
zdisplacement_filerecord (*ModflowGwfcsb*
attribute), 1353
zedges (*StructuredGrid* property), 2030
zeros_like() (*NetCdf* class method), 1995
zetaim (*ModflowGwtist* attribute), 1502
zgrid (*Grid* attribute), 2017
zonearray2params() (in module
flopy.pest.params), 2013
ZoneBudget (class in *flopy.utils.zonbud*),
1960
ZoneBudget6 (class in *flopy.utils.zonbud*),
1964
ZoneFile6 (class in *flopy.utils.zonbud*),
1967
zvertices (*Grid* property), 2023
zverts_smooth (*StructuredGrid* property),
2030