flopy Documentation

Release 3.3.4

Aug 07, 2021

Contents

Documentation for version 3.3.4

Return to the Github FloPy website where the code resides.

Contents:

CHAPTER 1

What is FloPy

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.

CHAPTER 2

FloPy Installation

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

2.1 conda Installation

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

2.2 pip Installation

To install FloPy type:

```
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
```

CHAPTER 3

FloPy Resources

CHAPTER 4

FloPy 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
- Davíd Brakenhoff
- and others

CHAPTER 5

How to Cite

- Groundwater Paper
- Software Citation

CHAPTER 6

Tutorial

Contents:

6.1 Tutorials

6.1.1 MODFLOW 6 Tutorials

Contents:

MODFLOW 6 Tutorial 1: Unconfined Steady-State Flow Model

This tutorial demonstrates use of FloPy to develop a simple MODFLOW 6 model.

Getting Started

[1]:
```
import os
import numpy as np
import matplotlib.pyplot as plt
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 `Nlay`, 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`.

[2]:
```
name = "tutorial01_mf6"
h1 = 100
Nlay = 10
N = 101
```

(continues on next page)

(continued from previous page)

```
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 the FloPy simulation object

[3]:
```
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws="."
)
```

Create the Flopy `TDIS` object

[4]:
```
tdis = flopy.mf6.ModflowTdis(
    sim, pname="tdis", time_units="DAYS", nper=1, perioddata=[(1.0, 1, 1.0)]
)
```

Create the Flopy `IMS` Package object

[5]:
```
ims = flopy.mf6.ModflowIms(
    sim,
    pname="ims",
    complexity="SIMPLE",
    linear_acceleration="BICGSTAB",
)
```

Create the Flopy groundwater flow (gwf) model object

[6]:
```
model_nam_file = "{}.nam".format(name)
gwf = flopy.mf6.ModflowGwf(
    sim,
    modelname=name,
    model_nam_file=model_nam_file,
    save_flows=True,
```

(continues on next page)

(continued from previous page)

```
    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.

[7]:
```
bot = np.linspace(-H / Nlay, -H, Nlay)
delrow = delcol = L / (N - 1)
dis = flopy.mf6.ModflowGwfdis(
    gwf,
    nlay=Nlay,
    nrow=N,
    ncol=N,
    delr=delrow,
    delc=delcol,
    top=0.0,
    botm=bot,
)
```

Create the initial conditions (IC) Package

[8]:
```
start = h1 * np.ones((Nlay, N, N))
ic = flopy.mf6.ModflowGwfic(gwf, pname="ic", strt=start)
```

Create the node property flow (NPF) Package

[9]:
```
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!

[10]:
```
chd_rec = []
layer = 0
for row_col in range(0, N):
    chd_rec.append(((layer, row_col, 0), h1))
```

(continues on next page)

(continued from previous page)

```
    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.

[11]:
```
iper = 0
ra = chd.stress_period_data.get_data(key=iper)
ra
```

[11]:
```
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.),
```

(continues on next page)

(continued from previous page)

```
((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.),
((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.),
```

(continues on next page)

(continued from previous page)

```
           ((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.),
           ((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.

[12]:
```
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.

[13]:
```
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.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

[14]:
```
sim.write_simulation()
```

```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing ims 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.

[15]:
```
success, buff = sim.run_simulation()
if not success:
    raise Exception("MODFLOW 6 did not terminate normally.")
```

```
FloPy is using the following executable to run the model: /home/runner/.local/bin/mf6
                                  MODFLOW 6
                U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
                            VERSION 6.2.2 07/30/2021

   MODFLOW 6 compiled Jul 31 2021 14:22:53 with IFORT compiler (ver. 19.10.3)

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): 2021/08/06 23:25:03

 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): 2021/08/06 23:25:04
 Elapsed run time:  1.472 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.

[16]:
```
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.)
```

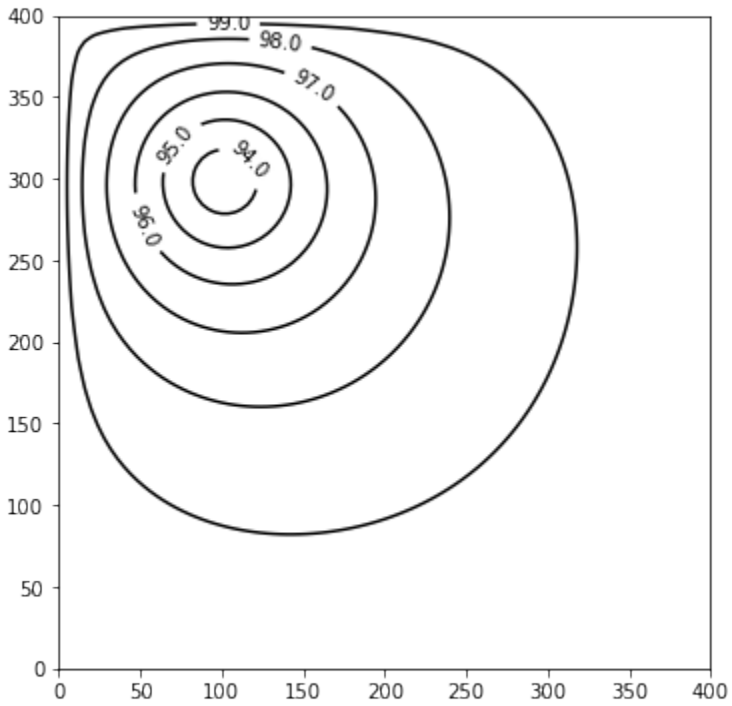
Plot a Map of Layer 1

[17]:
```
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
```

(continues on next page)

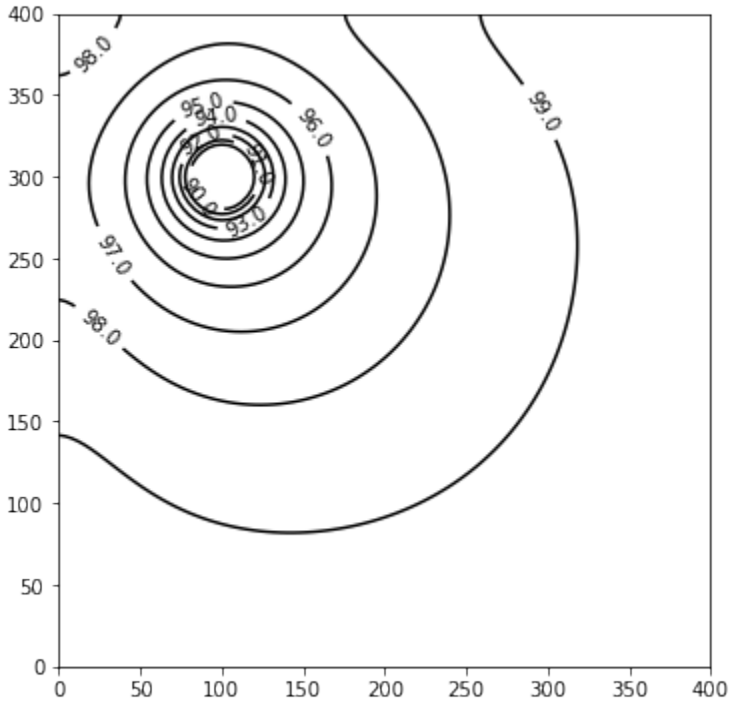
(continued from previous page)

```
c = ax.contour(x, y, h[0], contour_intervals, colors="black")
plt.clabel(c, fmt="%2.1f");
```



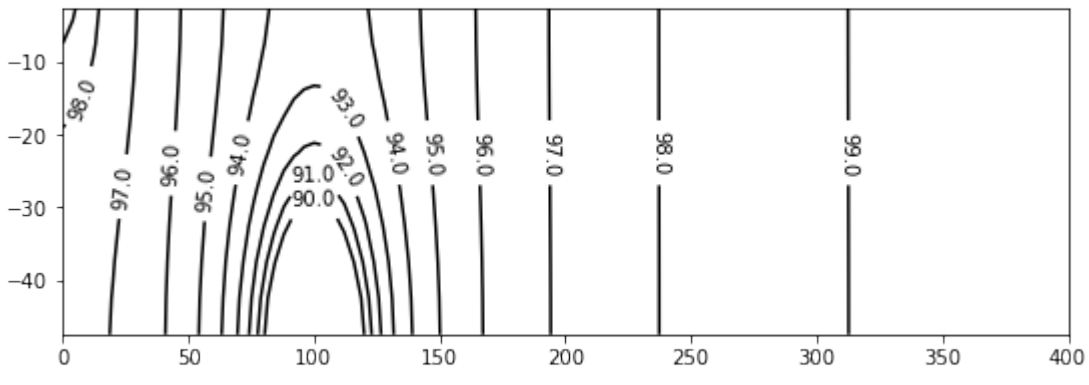
Plot a Map of Layer 10

[18]:
```
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");
```



Plot a Cross-section along row 25

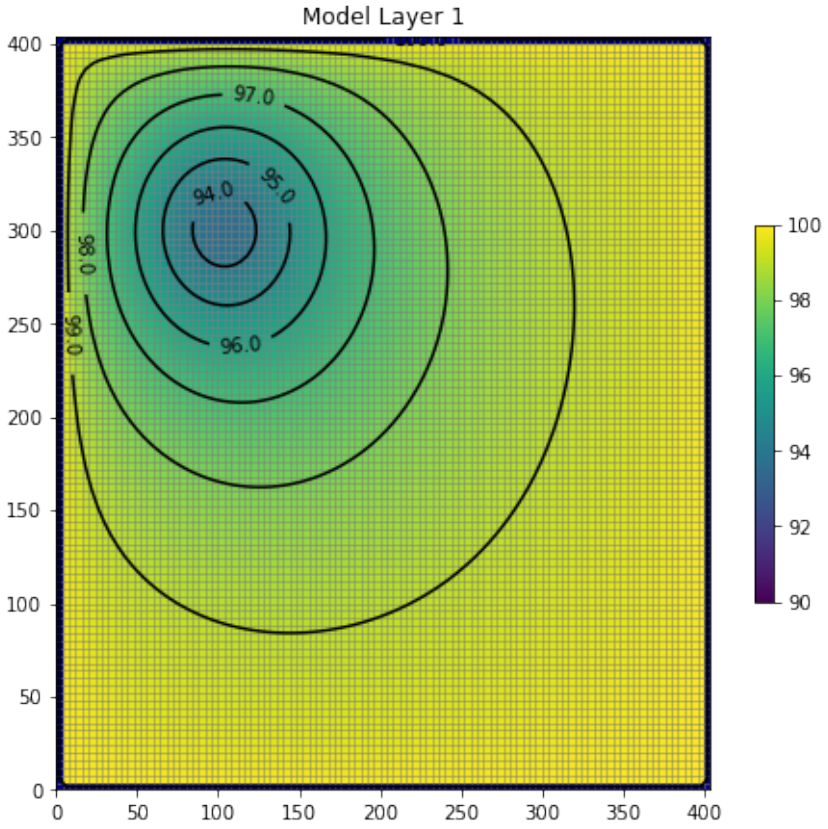
[19]:
```
z = np.linspace(-H / Nlay / 2, -H + H / Nlay / 2, Nlay)
fig = plt.figure(figsize=(9, 3))
ax = fig.add_subplot(1, 1, 1, aspect="auto")
c = ax.contour(x, z, h[:, int(N / 4), :], contour_intervals, colors="black")
plt.clabel(c, fmt="%1.1f");
```

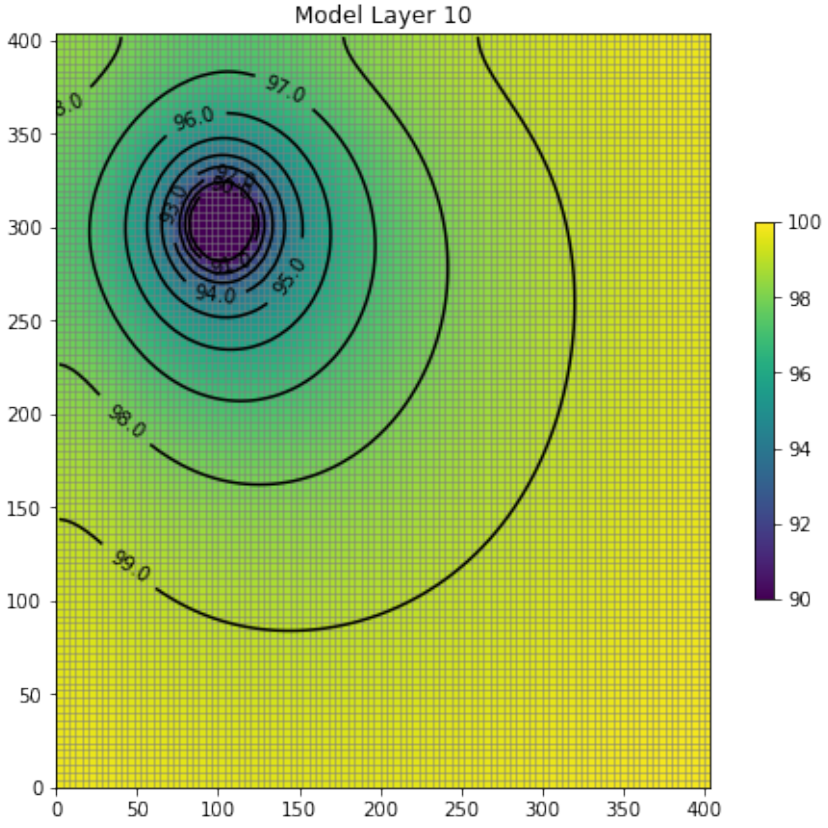


Use the FloPy `PlotMapView()` capabilities for MODFLOW 6

Plot a Map of heads in Layers 1 and 10

[20]:
```
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("Model Layer {}".format(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);
```

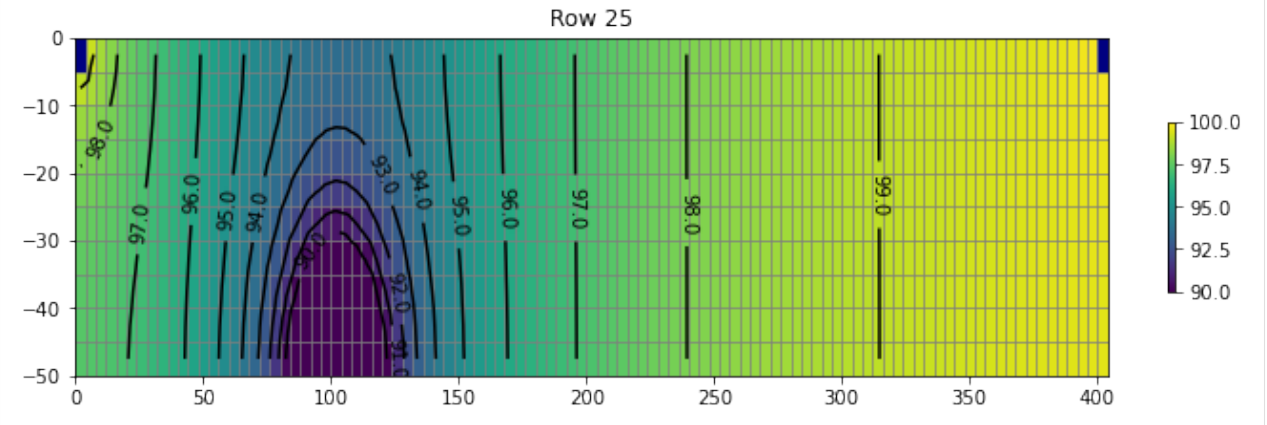

Model Layer 1
97.0
95.0
94.0
98.0
96.0
99.0
100
98
96
94
92
90
0
50
100
150
200
250
300
350
400


Model Layer 10
97.0
96.0
95.0
94.0
98.0
99.0
100
98
96
94
92
90
0
50
100
150
200
250
300
350
400

Use the FloPy `PlotCrossSection()` capabilities for MODFLOW 6

Plot a cross-section of heads along row 25

[21]:
```
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

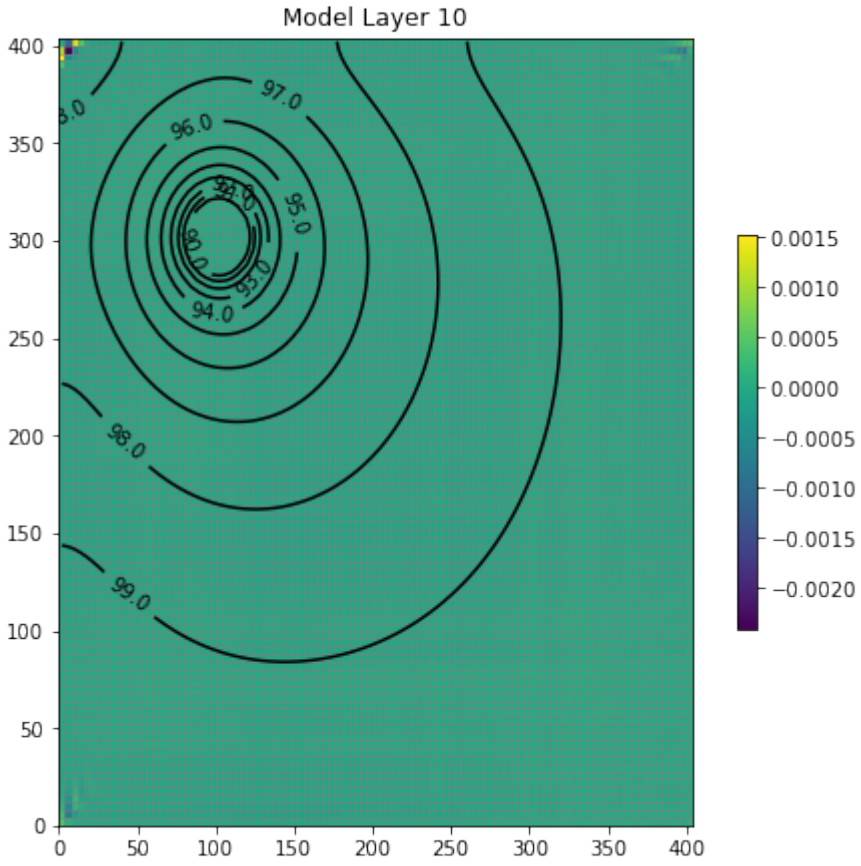
[22]:
```
flowja = gwf.oc.output.budget().get_data(
    text="FLOW-JA-FACE", kstpkper=(0, 0)
)[0]
```

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.

[23]:
```
grb_file = "{}.dis.grb".format(name)
residual = flopy.mf6.utils.get_residuals(flowja, grb_file=grb_file)
```

Plot a Map of the flow error in Layer 10

[24]:
```
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);
```



[ ]:

6.1.2 MODFLOW 6 FloPy Use Tutorials

Contents:

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
import flopy
```

[2]:
```
# set up simulation and basic packages
name = "tutorial02_mf6"
sim = flopy.mf6.MFSimulation(sim_name=name, sim_ws=".")
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 = name + ".bud"
head_file = name + ".hds"
flopy.mf6.ModflowGwfoc(
    gwf,
    budget_filerecord=budget_file,
    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.

[3]:
```
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.

[4]:
```
sim.simulation_data.indent_string = "    "
```

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

[5]:
```
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.

[6]:
```
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.

[7]:
```
print(sim)
```

```
sim_name = tutorial02_mf6
sim_path = /home/runner/work/flopy/flopy/.docs/.working/.
exe_name = mf6.exe

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

package_name = mfsim.nam
filename = mfsim.nam
package_type = nam
model_or_simulation_package = simulation
simulation_name = tutorial02_mf6


####################
Package tutorial02_mf6.tdis
####################

package_name = tutorial02_mf6.tdis
filename = tutorial02_mf6.tdis
package_type = tdis
model_or_simulation_package = simulation
simulation_name = tutorial02_mf6


####################
Package ims_-1
```

(continues on next page)

(continued from previous page)

```
####################

package_name = ims_-1
filename = tutorial02_mf6.ims
package_type = ims
model_or_simulation_package = simulation
simulation_name = tutorial02_mf6


@@@@@@@@@@@@@@@@@@@@
Model tutorial02_mf6
@@@@@@@@@@@@@@@@@@@@

name = tutorial02_mf6
model_type = gwf6
version = mf6
model_relative_path = .

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

package_name = dis
filename = tutorial02_mf6.dis
package_type = dis
model_or_simulation_package = model
model_name = tutorial02_mf6


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

package_name = ic
filename = tutorial02_mf6.ic
package_type = ic
model_or_simulation_package = model
model_name = tutorial02_mf6


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

package_name = npf
filename = tutorial02_mf6.npf
package_type = npf
model_or_simulation_package = model
model_name = tutorial02_mf6


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

package_name = chd_0
filename = tutorial02_mf6.chd
```

(continues on next page)

(continued from previous page)

```
package_type = chd
model_or_simulation_package = model
model_name = tutorial02_mf6


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

package_name = oc
filename = tutorial02_mf6.oc
package_type = oc
model_or_simulation_package = model
model_name = tutorial02_mf6
```

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

[8]:
```
tdis = sim.tdis
print(tdis)
```

```
package_name = tutorial02_mf6.tdis
filename = tutorial02_mf6.tdis
package_type = tdis
model_or_simulation_package = simulation
simulation_name = tutorial02_mf6

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

[9]:
```
ims = sim.get_package("ims_-1")
print(ims)
```

```
package_name = ims_-1
filename = tutorial02_mf6.ims
```

(continues on next page)

(continued from previous page)

```
package_type = ims
model_or_simulation_package = simulation
simulation_name = tutorial02_mf6
```

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

[10]:
```
ims = sim.get_package("ims")
print(ims)
```

```
package_name = ims_-1
filename = tutorial02_mf6.ims
package_type = ims
model_or_simulation_package = simulation
simulation_name = tutorial02_mf6
```

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.

[11]:
```
nam = sim.get_package("nam")
print(nam)
```

```
package_name = mfsim.nam
filename = mfsim.nam
package_type = nam
model_or_simulation_package = simulation
simulation_name = tutorial02_mf6

Block timing
--------------------
tdis6
{internal}
(tutorial02_mf6.tdis)


Block models
--------------------
models
{internal}
([('gwf6', 'tutorial02_mf6.nam', 'tutorial02_mf6')])


Block solutiongroup
--------------------
solutiongroup
{internal}
([('ims6', 'tutorial02_mf6.ims', 'tutorial02_mf6')])
```

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

[12]:
```
print(sim.model_names)
```

```
odict_keys(['tutorial02_mf6'])
```

`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

[13]:
```
model_names = list(sim.model_names)
for mname in model_names:
    print(mname)
```

```
tutorial02_mf6
```

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.

[14]:
```
model_names = list(sim.model_names)
for mname in model_names:
    m = sim.get_model(mname)
    print(m.name, m.model_type)
```

```
tutorial02_mf6 gwf6
```

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

[15]:
```
gwf = sim.get_model("tutorial02_mf6")
```

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

[16]:
```
print(gwf)
```

```
name = tutorial02_mf6
model_type = gwf6
version = mf6
model_relative_path = .

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

package_name = dis
filename = tutorial02_mf6.dis
package_type = dis
model_or_simulation_package = model
model_name = tutorial02_mf6


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

package_name = ic
filename = tutorial02_mf6.ic
package_type = ic
model_or_simulation_package = model
model_name = tutorial02_mf6
```

(continues on next page)

(continued from previous page)

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

package_name = npf
filename = tutorial02_mf6.npf
package_type = npf
model_or_simulation_package = model
model_name = tutorial02_mf6


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

package_name = chd_0
filename = tutorial02_mf6.chd
package_type = chd
model_or_simulation_package = model
model_name = tutorial02_mf6


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

package_name = oc
filename = tutorial02_mf6.oc
package_type = oc
model_or_simulation_package = model
model_name = tutorial02_mf6
```

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 obtained as

[17]:
```
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

[18]:
```
dis = gwf.get_package("dis")
print(dis)
```

```
package_name = dis
filename = tutorial02_mf6.dis
package_type = dis
model_or_simulation_package = model
model_name = tutorial02_mf6
```

(continues on next page)

(continued from previous page)

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

nrow
{internal}
(4)

ncol
{internal}
(5)


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

[19]:
```
print(type(dis))
```

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

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
package.obs.filename	str	Name of observations file to create. The default is packagename + '.obs'.
package.obs.continuous	dict	A dictionary that has file names as keys and a list of observations as the dictionary values.
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.

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

[1]:
```
# package import
import os
import numpy as np
import flopy
```

[2]:
```
# set up where simulation workspace will be stored
workspace = os.path.join("data", "mf6_working_with_data")
name = "example_1"
if not os.path.exists(workspace):
    os.makedirs(workspace)
```

[3]:
```
# 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_nam_file = "{}.nam".format(name)
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(-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],
)
```

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.

[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
```

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.

[5]:
```
# 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.

[6]:
```
# build ghb package passing obs dictionary to package constructor
ghb = flopy.mf6.modflow.mfgwfghb.ModflowGwfghb(
    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.

[7]:
```
ghb.obs.print_input = True
```

[8]:
```
# 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.

[9]:
```
# 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.

[10]:
```
# 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.

[11]:
```
# initialize obs package
ghb.obs.initialize(
    filename="child_pkgs_test.ghb.obs",
    digits=9,
    print_input=True,
    continuous=ghb_obs,
)
```

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
package.ts.filename	str	Name of time series file to create. The default is packagename + ".ts".
package.ts.timeseries	recarray	Array containing the time series information
package.ts.time_series_namerecord	str or list	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.
package.ts.interpolation_methodrecord	float	Scale factor to multiply the time series data column. Can only be used if there is one time series data column.
package.ts.sfacrecord_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.sfacrecord	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
import numpy as np
import flopy
```

[2]:
```
# set up where simulation workspace will be stored
workspace = os.path.join("data", "mf6_working_with_data")
name = "example_1"
if not os.path.exists(workspace):
    os.makedirs(workspace)
```

[3]:
```
# 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_nam_file = "{}.nam".format(name)
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(-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,
```

(continues on next page)

(continued from previous page)

```
    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 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.

[4]:
```
# 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.

[5]:
```
# 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))
```

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

[6]:
```
# build ghb package
ghb = flopy.mf6.modflow.mfgwfghb.ModflowGwfghb(
    gwf,
    print_input=True,
    print_flows=True,
    save_flows=True,
    boundnames=True,
    timeseries=ts_data,
    pname="ghb",
    maxbound=30,
    stress_period_data=ghb_spd_ts,
)
```

```
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".
```

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

[7]:
```
# set required time series attributes
ghb.ts.time_series_namerecord = "tides"
```

[8]:
```
# 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.

[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), "wl", 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.

[10]:
```
# 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

[11]:
```
# 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.

[12]:
```
# 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.

[13]:
```
# append additional time series
ghb.ts.append_package(
    filename="wls.ts",
    timeseries=ts_data2,
    time_series_namerecord="wl",
    interpolation_methodrecord="stepwise",
    sfacrecord=1.2,
)
# append additional time series
ghb.ts.append_package(
    filename="wls2.ts",
    timeseries=ts_data3,
    time_series_namerecord="wl2",
    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.

[14]:
```
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
```

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
package.tas.filename	str	Name of time series file to create. The default is packagename + ".tas".
package.tas.tas_array	{double:[double]}	Array containing the time array series information for specific times.
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.

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

[1]:
```
# package import
import os
import numpy as np
import flopy
```

[2]:
```
# set up where simulation workspace will be stored
workspace = os.path.join("data", "mf6_working_with_data")
name = "example_1"
if not os.path.exists(workspace):
    os.makedirs(workspace)
```

[3]:
```
# 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_nam_file = "{}.nam".format(name)
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(-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.

[4]:
```
tas = {0.0: 0.000002, 200.0: 0.0000001}
```

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

structed.

[5]:
```
rcha = flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha(
    gwf, timearrayseries=tas, recharge="TIMEARRAYSERIES rcharray_1"
)
```

```
WARNING: Time array series name rcharray_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.

[6]:
```
# finish defining the time array series properties
rcha.tas.time_series_namerecord = "rcharray_1"
rcha.tas.interpolation_methodrecord = "LINEAR"
```

The simulation is then written to files and run.

[7]:
```
sim.write_simulation()
sim.run_simulation()
```

```
writing simulation...
  writing simulation name file...
  writing simulation tdis package...
  writing ims package ims...
  writing model example_1...
    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/mf6
                                 MODFLOW 6
               U.S. GEOLOGICAL SURVEY MODULAR HYDROLOGIC MODEL
                            VERSION 6.2.2 07/30/2021

   MODFLOW 6 compiled Jul 31 2021 14:22:53 with IFORT compiler (ver. 19.10.3)

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): 2021/08/06 23:25:32

 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
```

(continues on next page)

(continued from previous page)

```
    Solving:  Stress period:     2    Time step:     3
    Solving:  Stress period:     2    Time step:     4
    Solving:  Stress period:     2    Time step:     5
    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:     4    Time step:     1

 Run end date and time (yyyy/mm/dd hh:mm:ss): 2021/08/06 23:25:33
 Elapsed run time:  0.238 Seconds

 Normal termination of simulation.
```

[7]:
```
(True, [])
```

[8]:
```
# 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.

[9]:
```
# 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 recharray_1
rcha = flopy.mf6.modflow.mfgwfrcha.ModflowGwfrcha(
    gwf, recharge="TIMEARRAYSERIES rcharray_1"
)
```

```
WARNING: Time array series name rcharray_1 not found in any time series file
```

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

[10]:
```
tas = {0.0: 0.000002, 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.

[11]:
```
# initialize the time array series
rcha.tas.initialize(
    filename="method2.tas",
    tas_array=tas,
    time_series_namerecord="rcharray_1",
    interpolation_methodrecord="LINEAR",
)
```

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 (`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 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
import numpy as np
import flopy
```

[2]:
```
# set up where simulation workspace will be stored
workspace = os.path.join("data", "mf6_working_with_data")
name = "example_1"
if not os.path.exists(workspace):
    os.makedirs(workspace)
```

[3]:
```
# create the flopy simulation object
sim = flopy.mf6.MFSimulation(
    sim_name=name, exe_name="mf6", version="mf6", sim_ws=workspace
)
```

[4]:
```
# create the flopy groundwater flow (gwf) model object
model_nam_file = "{}.nam".format(name)
gwf = flopy.mf6.ModflowGwf(sim, modelname=name, model_nam_file=model_nam_file)
# create the flopy iterative model solver (ims) package object
```

(continues on next page)

(continued from previous page)

```
# (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.

[5]:
```
# 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.

[6]:
```
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

[7]:
```
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.

[8]:
```
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.

[9]:
```
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.

[10]:
```
# create the node property flow package with xt3doptions as single
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 0x7f049b75c3a0>
<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7f049b75c3d0>
```

Next, the rhs option is turned off by setting xt3doptions to the string "xt3d".

[11]:
```
npf.xt3doptions = "xt3d"
```

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

[12]:
```
npf.xt3doptions = ""
```

Retrieving MODFLOW Scalar Data

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

Retrieval Method	Description
get_data	Returns scalar value
repr/str	Returns string with a header describing how data is stored (internal, external) with a string representation of the data on the next line
get_file_entry	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.

[13]:
```
print(ims.complexity.get_data())
```
```
simple
```

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

[14]:
```
print(npf.xt3doptions.get_data())
```
```
[]
```

The complexity data is then printed with repr

[15]:
```
print(repr(ims.complexity))
```
```
{internal}
('simple')
```

The xt3doptions data is printed with repr

[16]:
```
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.

[17]:
```
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.

[18]:
```
print(npf.xt3doptions.get_file_entry())
```

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 (`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 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 a 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
import numpy as np
import flopy
```

[2]:
```
# set up where simulation workspace will be stored
workspace = os.path.join("data", "mf6_working_with_data")
name = "example_1"
if not os.path.exists(workspace):
    os.makedirs(workspace)
```

[3]:
```
# 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_nam_file = "{}.nam".format(name)
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,
```

(continues on next page)

(continued from previous page)

```
    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.

[4]:
```
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 0x7f6d8f78b520>
<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7f6d8f78b550>
```

`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.

[5]:
```
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.

[6]:
```
stress_period_data = [((1, 10, 10), 100.0), ((1, 10, 11), 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,
```

(continues on next page)

(continued from previous page)

```
    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

[7]:
```
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.

[8]:
```
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
get_data	Returns recarray
array	Return recarray
repr/str	Returns string with storage information followed by recarray's repr/str
get_file_entry	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.

[9]:
```
print(npf.rewet_record.get_data())
```
```
[('WETFCT', 1.1, 'IWETIT', 0, 'IHDWET', 1)]
```

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

[10]:
```
print(npf.rewet_record.array)
```
```
[('WETFCT', 1.1, 'IWETIT', 0, 'IHDWET', 1)]
```

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

[11]:
```
print(repr(npf.rewet_record))
```
```
{internal}
(rec.array([('WETFCT', 1.1, 'IWETIT', 0, 'IHDWET', 1)],
          dtype=[('wetfct_label', 'O'), ('wetfct', '<f8'), ('iwetit_label', 'O'), (
↪'iwetit', '<i8'), ('ihdwet_label', 'O'), ('ihdwet', '<i8')]))
```

Using str prints a similar string representation of rewet_record.

[12]:
```
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.

[13]:
```
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
get_data	Returns dictionary of recarrays
array	Return single recarray for all stress periods
repr/str	Returns string with storage information followed by recarray repr/str for each recarray
get_file_entry(key)	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.

[14]:
```
print(wel.stress_period_data.get_data())
```
```
{0: rec.array([((2, 3, 1), -25.)],
          dtype=[('cellid', 'O'), ('q', '<f8')])}
```

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

[15]:
```
print(wel.stress_period_data.array)
```

```
[rec.array([((2, 3, 1), -25.)],
           dtype=[('cellid', 'O'), ('q', '<f8')]), None]
```

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

[16]:
```
print(repr(wel.stress_period_data))
```

```
{internal}
(rec.array([((2, 3, 1), -25.)],
           dtype=[('cellid', 'O'), ('q', '<f8')]))
```

str produces a similar string representation of stress period data.

[17]:
```
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.

[18]:
```
print(wel.stress_period_data.get_file_entry(0))
```

```
  3 4 2     -25.00000000
```

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 a 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
import numpy as np
import flopy
```

[2]:
```
# set up where simulation workspace will be stored
workspace = os.path.join("data", "mf6_working_with_data")
name = "example_1"
if not os.path.exists(workspace):
    os.makedirs(workspace)
# 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_nam_file = "{}.nam".format(name)
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")
```

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

[3]:
```
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.

[4]:
```
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.

[5]:
```
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.

[6]:
```
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.

[7]:
```
k33 = 1.0
```

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

[8]:
```
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 0x7f2fcfaea280>
<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7f2fcfaea2b0>
```

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 `MFArray` object, such as `npf.k22` can be converted to a layered array as follows.

[9]:
```
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.

[10]:
```
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.

[11]:
```
print(type(npf.k22))
print(npf.k22)
```

```
<class 'flopy.mf6.data.mfdataarray.MFArray'>
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]
 [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:

[12]:
```
print(npf.k22.get_file_entry())
```

```
  k22  LAYERED
    INTERNAL  FACTOR  1.0
```

(continues on next page)

(continued from previous page)

```
          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
↪139.00000000
        140.00000000     141.00000000     142.00000000     143.00000000     144.
↪00000000     145.00000000     146.00000000     147.00000000     148.00000000
↪149.00000000
        150.00000000     151.00000000     152.00000000     153.00000000     154.
↪00000000     155.00000000     156.00000000     157.00000000     158.00000000
↪159.00000000
        160.00000000     161.00000000     162.00000000     163.00000000     164.
↪00000000     165.00000000     166.00000000     167.00000000     168.00000000
↪169.00000000
        170.00000000     171.00000000     172.00000000     173.00000000     174.
↪00000000     175.00000000     176.00000000     177.00000000     178.00000000
↪179.00000000
        180.00000000     181.00000000     182.00000000     183.00000000     184.
↪00000000     185.00000000     186.00000000     187.00000000     188.00000000
↪189.00000000
```

(continues on next page)

(continued from previous page)

```
        190.00000000     191.00000000     192.00000000     193.00000000     194.
↪00000000     195.00000000     196.00000000     197.00000000     198.00000000
↪199.00000000
   INTERNAL  FACTOR  1.0
        200.00000000     201.00000000     202.00000000     203.00000000     204.
↪00000000     205.00000000     206.00000000     207.00000000     208.00000000
↪209.00000000
        210.00000000     211.00000000     212.00000000     213.00000000     214.
↪00000000     215.00000000     216.00000000     217.00000000     218.00000000
↪219.00000000
        220.00000000     221.00000000     222.00000000     223.00000000     224.
↪00000000     225.00000000     226.00000000     227.00000000     228.00000000
↪229.00000000
        230.00000000     231.00000000     232.00000000     233.00000000     234.
↪00000000     235.00000000     236.00000000     237.00000000     238.00000000
↪239.00000000
        240.00000000     241.00000000     242.00000000     243.00000000     244.
↪00000000     245.00000000     246.00000000     247.00000000     248.00000000
↪249.00000000
        250.00000000     251.00000000     252.00000000     253.00000000     254.
↪00000000     255.00000000     256.00000000     257.00000000     258.00000000
↪259.00000000
        260.00000000     261.00000000     262.00000000     263.00000000     264.
↪00000000     265.00000000     266.00000000     267.00000000     268.00000000
↪269.00000000
        270.00000000     271.00000000     272.00000000     273.00000000     274.
↪00000000     275.00000000     276.00000000     277.00000000     278.00000000
↪279.00000000
        280.00000000     281.00000000     282.00000000     283.00000000     284.
↪00000000     285.00000000     286.00000000     287.00000000     288.00000000
↪289.00000000
        290.00000000     291.00000000     292.00000000     293.00000000     294.
↪00000000     295.00000000     296.00000000     297.00000000     298.00000000
↪299.00000000
```

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

[13]:
```
npf.k22.set_data([1, a[2], 200])
print(npf.k22.get_file_entry())
```

```
  k22  LAYERED
    CONSTANT       1.00000000
    INTERNAL  FACTOR  1.0
        200.00000000     201.00000000     202.00000000     203.00000000     204.
↪00000000     205.00000000     206.00000000     207.00000000     208.00000000
↪209.00000000
        210.00000000     211.00000000     212.00000000     213.00000000     214.
↪00000000     215.00000000     216.00000000     217.00000000     218.00000000
↪219.00000000
        220.00000000     221.00000000     222.00000000     223.00000000     224.
↪00000000     225.00000000     226.00000000     227.00000000     228.00000000
↪229.00000000
        230.00000000     231.00000000     232.00000000     233.00000000     234.
↪00000000     235.00000000     236.00000000     237.00000000     238.00000000
↪239.00000000
```

(continues on next page)

(continued from previous page)

```
        240.00000000      241.00000000      242.00000000      243.00000000      244.
↪00000000      245.00000000      246.00000000      247.00000000      248.00000000      ␣
↪249.00000000
        250.00000000      251.00000000      252.00000000      253.00000000      254.
↪00000000      255.00000000      256.00000000      257.00000000      258.00000000      ␣
↪259.00000000
        260.00000000      261.00000000      262.00000000      263.00000000      264.
↪00000000      265.00000000      266.00000000      267.00000000      268.00000000      ␣
↪269.00000000
        270.00000000      271.00000000      272.00000000      273.00000000      274.
↪00000000      275.00000000      276.00000000      277.00000000      278.00000000      ␣
↪279.00000000
        280.00000000      281.00000000      282.00000000      283.00000000      284.
↪00000000      285.00000000      286.00000000      287.00000000      288.00000000      ␣
↪289.00000000
        290.00000000      291.00000000      292.00000000      293.00000000      294.
↪00000000      295.00000000      296.00000000      297.00000000      298.00000000      ␣
↪299.00000000
    CONSTANT      200.00000000
```

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

[14]:
```
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.
↪00000000      100.00000000      100.00000000      100.00000000      100.00000000      ␣
↪100.00000000
        100.00000000      100.00000000      100.00000000      100.00000000      100.
↪00000000      100.00000000      100.00000000      100.00000000      100.00000000      ␣
↪100.00000000
        100.00000000      100.00000000      100.00000000      100.00000000      100.
↪00000000      100.00000000      100.00000000      100.00000000      100.00000000      ␣
↪100.00000000
        100.00000000      100.00000000      100.00000000      100.00000000      100.
↪00000000      100.00000000      100.00000000      100.00000000      100.00000000      ␣
↪100.00000000
        100.00000000      100.00000000      100.00000000      100.00000000      100.
↪00000000      100.00000000      100.00000000      100.00000000      100.00000000      ␣
↪100.00000000
        100.00000000      100.00000000      100.00000000      100.00000000      100.
↪00000000      100.00000000      100.00000000      100.00000000      100.00000000      ␣
↪100.00000000
        100.00000000      100.00000000      100.00000000      100.00000000      100.
↪00000000      100.00000000      100.00000000      100.00000000      100.00000000      ␣
↪100.00000000
        100.00000000      100.00000000      100.00000000      100.00000000      100.
↪00000000      100.00000000      100.00000000      100.00000000      100.00000000      ␣
↪100.00000000
        100.00000000      100.00000000      100.00000000      100.00000000      100.
↪00000000      100.00000000      100.00000000      100.00000000      100.00000000      ␣
↪100.00000000
```

(continues on next page)

(continued from previous page)

```
        100.00000000      100.00000000      100.00000000      100.00000000      100.
↪00000000      100.00000000      100.00000000      100.00000000      100.00000000     ␣
↪100.00000000
   CONSTANT       50.00000000
   INTERNAL  FACTOR  1.0  IPRN  14
        30.00000000       30.00000000       30.00000000       30.00000000       30.
↪00000000       30.00000000       30.00000000       30.00000000       30.00000000      ␣
↪30.00000000
        30.00000000       30.00000000       30.00000000       30.00000000       30.
↪00000000       30.00000000       30.00000000       30.00000000       30.00000000      ␣
↪30.00000000
        30.00000000       30.00000000       30.00000000       30.00000000       30.
↪00000000       30.00000000       30.00000000       30.00000000       30.00000000      ␣
↪30.00000000
        30.00000000       30.00000000       30.00000000       30.00000000       30.
↪00000000       30.00000000       30.00000000       30.00000000       30.00000000      ␣
↪30.00000000
        30.00000000       30.00000000       30.00000000       30.00000000       30.
↪00000000       30.00000000       30.00000000       30.00000000       30.00000000      ␣
↪30.00000000
        30.00000000       30.00000000       30.00000000       30.00000000       30.
↪00000000       30.00000000       30.00000000       30.00000000       30.00000000      ␣
↪30.00000000
        30.00000000       30.00000000       30.00000000       30.00000000       30.
↪00000000       30.00000000       30.00000000       30.00000000       30.00000000      ␣
↪30.00000000
        30.00000000       30.00000000       30.00000000       30.00000000       30.
↪00000000       30.00000000       30.00000000       30.00000000       30.00000000      ␣
↪30.00000000
        30.00000000       30.00000000       30.00000000       30.00000000       30.
↪00000000       30.00000000       30.00000000       30.00000000       30.00000000      ␣
↪30.00000000
        30.00000000       30.00000000       30.00000000       30.00000000       30.
↪00000000       30.00000000       30.00000000       30.00000000       30.00000000      ␣
↪30.00000000
```

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

[15]:
```
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.]
```

(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.]]

 [[ 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

[16]:
```
print(npf.k22.array)
```

```
[[[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.]
  [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.

[17]:
```
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.

[18]:
```
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
get_data	Returns ndarray of data without multiplier applied, unless the apply_mult parameter is set to True
array	Returns ndarray of data with multiplier applied
[]	Returns a particular layer of data if data is layered, otherwise is an array index
repr/str	Returns string with storage information followed by ndarray repr/str
get_file_entry(layer)	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.

[19]:
```
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.]
```

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

[20]:
```
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.

[21]:
```
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.]
 [10. 10. 10. 10. 10. 10. 10. 10. 10. 10.]]
```

repr gives a string representation of the data.

[22]:
```
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.

[23]:
```
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.]
```

(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.]])
Layer_2{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_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.]
 [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.

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

```
  k  LAYERED
    INTERNAL  FACTOR  1.0
          10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
          10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
          10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
          10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
          10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
          10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
          10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
```

(continues on next page)

(continued from previous page)

```
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
    INTERNAL  FACTOR  1.0
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
    INTERNAL  FACTOR  1.0
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
        10.00000000       10.00000000       10.00000000       10.00000000       10.
↪00000000       10.00000000       10.00000000       10.00000000       10.00000000     ␣
↪10.00000000
```

(continues on next page)

(continued from previous page)

```
         10.00000000      10.00000000      10.00000000      10.00000000      10.
↪00000000      10.00000000      10.00000000      10.00000000      10.00000000      ␣
↪10.00000000
         10.00000000      10.00000000      10.00000000      10.00000000      10.
↪00000000      10.00000000      10.00000000      10.00000000      10.00000000      ␣
↪10.00000000
         10.00000000      10.00000000      10.00000000      10.00000000      10.
↪00000000      10.00000000      10.00000000      10.00000000      10.00000000      ␣
↪10.00000000
         10.00000000      10.00000000      10.00000000      10.00000000      10.
↪00000000      10.00000000      10.00000000      10.00000000      10.00000000      ␣
↪10.00000000
```

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
get_data	Returns dictionary of ndarrays without multiplier applied. The dictionary key is the stress period as a zero-based integer.
array	Returns ndarray of data for all stress periods (stress period is an added dimension)
repr/str	Returns string with storage information followed by ndarray repr/str for all stress periods
get_file_entry(layer)	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).

```
[25]: 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],
       [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, 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]])}
```

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

```
[26]: 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.

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

```
{constant 0.03}
```

str gives a similar representation of the data.

```
[28]: 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).

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

```
  recharge
    CONSTANT       0.03000000
```

MODFLOW 6: Data Storage Information - How and Where to Store MODFLOW-6 Data

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.

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
import numpy as np
import flopy
```

[2]:
```
# set up where simulation workspace will be stored
workspace = os.path.join("data", "mf6_working_with_data")
name = "example_1"
if not os.path.exists(workspace):
    os.makedirs(workspace)
# 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_nam_file = "{}.nam".format(name)
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.

[3]:
```
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.

[4]:
```
npf = flopy.mf6.ModflowGwfnpf(
    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 0x7f18c460d580>
<flopy.mf6.data.mfstructure.MFDataItemStructure object at 0x7f18c460d5b0>
```

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.

[5]:
```
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_data` method.

[6]:
```
dis.botm.set_data([a0, a1, a2])
print(dis.botm.get_file_entry())
```

```
  botm  LAYERED
    INTERNAL  FACTOR  0.5  IPRN  1
           1.00000000       1.00000000       1.00000000       1.00000000       1.
↪00000000       1.00000000       1.00000000       1.00000000       1.00000000
↪1.00000000
```

(continues on next page)

(continued from previous page)

```
          1.00000000       1.00000000       1.00000000       1.00000000       1.
↪00000000       1.00000000       1.00000000       1.00000000       1.00000000     ␣
↪1.00000000
          1.00000000       1.00000000       1.00000000       1.00000000       1.
↪00000000       1.00000000       1.00000000       1.00000000       1.00000000     ␣
↪1.00000000
          1.00000000       1.00000000       1.00000000       1.00000000       1.
↪00000000       1.00000000       1.00000000       1.00000000       1.00000000     ␣
↪1.00000000
          1.00000000       1.00000000       1.00000000       1.00000000       1.
↪00000000       1.00000000       1.00000000       1.00000000       1.00000000     ␣
↪1.00000000
          1.00000000       1.00000000       1.00000000       1.00000000       1.
↪00000000       1.00000000       1.00000000       1.00000000       1.00000000     ␣
↪1.00000000
          1.00000000       1.00000000       1.00000000       1.00000000       1.
↪00000000       1.00000000       1.00000000       1.00000000       1.00000000     ␣
↪1.00000000
          1.00000000       1.00000000       1.00000000       1.00000000       1.
↪00000000       1.00000000       1.00000000       1.00000000       1.00000000     ␣
↪1.00000000
          1.00000000       1.00000000       1.00000000       1.00000000       1.
↪00000000       1.00000000       1.00000000       1.00000000       1.00000000     ␣
↪1.00000000
          1.00000000       1.00000000       1.00000000       1.00000000       1.
↪00000000       1.00000000       1.00000000       1.00000000       1.00000000     ␣
↪1.00000000
    CONSTANT    -100.00000000
    OPEN/CLOSE  'dis.botm.3.txt'  FACTOR  2.0  IPRN  1
```

Note that we could have also specified `botm` this way as part of the original `flopy.mf6.ModflowGwfdis` constructor:

[7]:
```
a0 = {"factor": 0.5, "iprn": 1, "data": np.ones((10, 10))}
a1 = -100
a2 = {
    "filename": "dis.botm.3.bin",
    "factor": 2.0,
    "iprn": 1,
    "data": -100 * np.ones((4, 5)),
    "binary": True,
}
botm = [a0, a1, a2]
flopy.mf6.ModflowGwfdis(gwf, nlay=3, nrow=10, ncol=10, botm=botm)
```

```
WARNING: Package with type dis already exists. Replacing existing package.
```

[7]:
```
package_name = dis
filename = example_1.dis
package_type = dis
model_or_simulation_package = model
model_name = example_1

Block dimensions
--------------------
nlay
```

(continues on next page)

(continued from previous page)

```
{internal}
(3)

nrow
{internal}
(10)

ncol
{internal}
(10)


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

delc
{constant 1.0}

top
{constant 1.0}

botm
Layer_1{internal, factor 0.5, iprn 1}
(array([[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., 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., 1., 1., 1., 1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1., 1., 1., 1., 1., 1.]]))
Layer_2{constant -100}
```

6.1.3 MODFLOW 6 Output Tutorials

Contents:

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]:
```
# ## Package import
import flopy
import os
import platform
import numpy as np
```

[2]:
```
# ## Load a simple demonstration model
exe_name = "mf6"
if platform.system().lower() == "windows":
    exe_name += ".exe"
ws = os.path.abspath(os.path.dirname(""))
if os.path.split(ws)[-1] == "modflow6output":
    sim_ws = os.path.join(ws, "..", "..", "data", "mf6", "test001e_UZF_3lay")
else:
    sim_ws = os.path.join(
        ws, "..", "..", "examples", "data", "mf6", "test001e_UZF_3lay"
    )
# 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
sim_ws = os.path.join("..", "..", "..", "autotest", "temp")
sim.set_sim_path(sim_ws)
sim.write_simulation(silent=True)
sim.run_simulation(silent=True)
```

[2]:
```
(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
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.

[3]:
```
ml = sim.get_model("gwf_1")
```

[4]:
```
bud = ml.output.budget()
bud.get_data(idx=0, full3D=True)
```

[4]:
```
[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., -0.]]])]
```

[5]:
```
hds = ml.output.head()
hds.get_data()
```

[5]:
```
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.        ]]])
```

[6]:
```
bud = ml.oc.output.budget()
bud.get_data(idx=0, full3D=True)
```

[6]:
```
[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., -0.]]])]
```

[7]:
```
hds = ml.oc.output.head()
hds.get_data()
```

[7]:
```
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, package convergence output, and observation data.

[8]:
```
uzf = ml.uzf
uzf_bud = uzf.output.budget()
uzf_bud.get_data(idx=0)
```

[8]:
```
[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.),
            (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
↪')])]
```

[9]:
```
uzf_conv = uzf.output.package_convergence()
if uzf_conv is not None:
    uzf_conv.data[0:10]
```

[10]:
```
uzf_obs = uzf.output.obs()
uzf_obs.data[0:10]
```

[10]:
```
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_DPTH80', '<f8'), ('ID3_DPTH240', '<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

[11]:
```
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()', 'package_convergence()', 'obs()']
Model object:  ['zonebudget()', 'budget()', 'head()']
```

(continues on next page)

(continued from previous page)

```
OC package:  ['zonebudget()', 'budget()', '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.

[12]:
```
output = ml.obs[0].output
obs_names = output.obs_names
output.obs(f=obs_names[0]).data[0:10]
```

[12]:
```
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_DPTH80', '<f8'), ('ID3_DPTH240', '<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

[13]:
```
zarr = np.ones(ml.modelgrid.shape, dtype=int)
for i in range(1, 4):
    zarr[i - 1] *= i
```

[14]:
```
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/
↪zbud6
                                 ZONEBUDGET Version 6
                               U.S. GEOLOGICAL SURVEY
                              VERSION 6.2.2 07/30/2021
...

Normal Termination
```

[14]:
```
(True, [])
```

[15]:
```
df = zonbud.get_dataframes(net=True)
df = df.reset_index()
df
```

[15]:
```
      totim    name  ZONE_1         ZONE_2         ZONE_3
0       2.0  ZONE_0     0.0  0.000000e+00  0.000000e+00
```

(continues on next page)

(continued from previous page)

```
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  1.705303e-09  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  1.136868e-09
359  2560.0   ZONE_3     0.0 -1.136868e-09  0.000000e+00

[360 rows x 5 columns]
```

6.1.4 MODFLOW Tutorials

Contents:

MODFLOW Tutorial 1: Confined Steady-State Flow Model

This tutorial demonstrates use of FloPy to develop a simple MODFLOW-2005 model.

Getting Started

If FloPy has been properly installed, then it can be imported as follows:

[1]:
```
import numpy as np
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 our flopy model object.

[2]:
```
modelname = "tutorial1_mf"
mf = flopy.modflow.Modflow(modelname, exe_name="mf2005")
```

Next, let's proceed by defining our model domain and creating a MODFLOW grid to span the domain.

[3]:
```
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:

[4]:
```
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:

[5]:
```
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
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:

[6]:
```
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:

[7]:
```
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:

[8]:
```
pcg = flopy.modflow.ModflowPcg(mf)
```

Writing the MODFLOW Data Files

The MODFLOW input data files are written by simply issuing the following:

[9]:
```
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:

[10]:
```
success, buff = mf.run_model()
if not success:
    raise Exception("MODFLOW did not terminate normally.")
```

```
FloPy is using the following executable to run the model: /home/runner/.local/bin/
↪mf2005

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

 Using NAME file: tutorial1_mf.nam
 Run start date and time (yyyy/mm/dd hh:mm:ss): 2021/08/06 23:25:00

 Solving:  Stress period:     1    Time step:     1    Ground-Water Flow Eqn.
 Run end date and time (yyyy/mm/dd hh:mm:ss): 2021/08/06 23:25:00
 Elapsed run time:  0.001 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:

[11]:
```
import matplotlib.pyplot as plt
import flopy.utils.binaryfile as bf
```

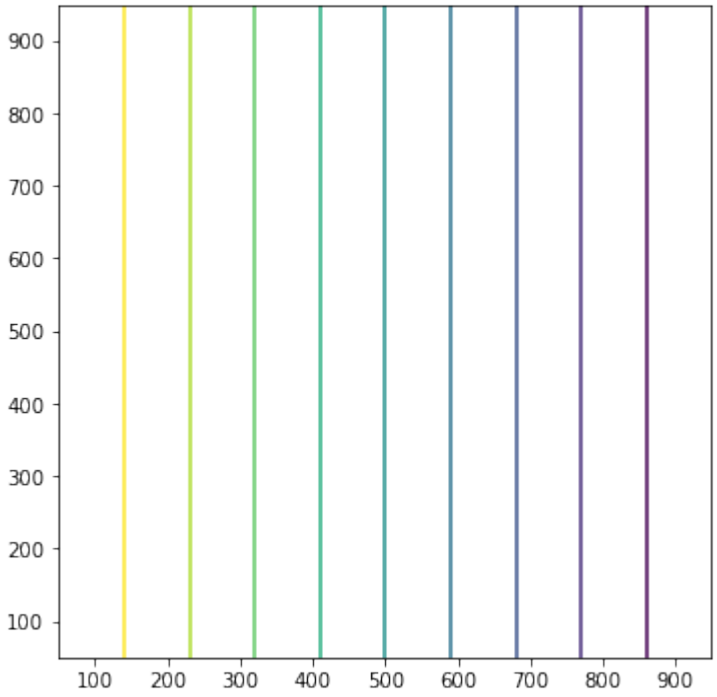
Extract the heads

[12]:
```
hds = bf.HeadFile(modelname + ".hds")
head = hds.get_data(totim=1.0)
```

Contour the heads

[13]:
```
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)
```

[13]:
```
<matplotlib.contour.QuadContourSet at 0x7f7fc6e4e070>
```



Flopy also has some pre-canned plotting capabilities can 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:

[14]:
```
# Extract the heads
hds = bf.HeadFile(modelname + ".hds")
times = hds.get_times()
head = hds.get_data(totim=times[-1])
```

Extract the cell-by-cell flows

[15]:
```
cbb = bf.CellBudgetFile(modelname + ".cbc")
kstpkper_list = cbb.get_kstpkper()
frf = cbb.get_data(text="FLOW RIGHT FACE", totim=times[-1])[0]
fff = cbb.get_data(text="FLOW FRONT FACE", totim=times[-1])[0]
```

Create the figure

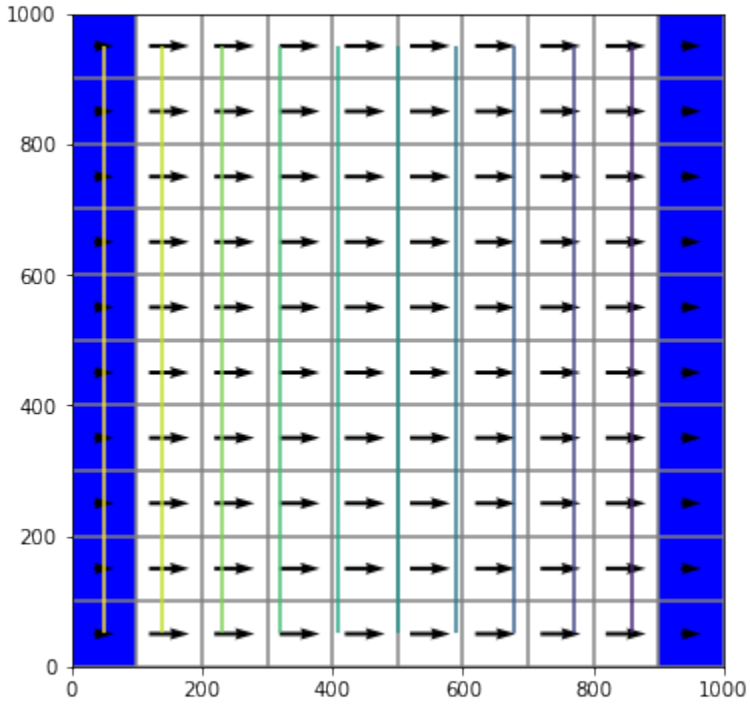
[16]:
```
fig = plt.figure(figsize=(6, 6))
ax = fig.add_subplot(1, 1, 1, aspect="equal")
```

(continues on next page)

(continued from previous page)

```
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_discharge(frf, fff, head=head)
```

```
/usr/share/miniconda/envs/flopy/lib/python3.8/site-packages/flopy/plot/map.py:819:␣
↪DeprecationWarning: plot_discharge() has been deprecated and will be replaced in␣
↪version 3.3.5. Use plot_vector() instead, which should follow after postprocessing.
↪get_specific_discharge()
  warnings.warn(
/usr/share/miniconda/envs/flopy/lib/python3.8/site-packages/flopy/plot/plotutil.py:
↪1630: DeprecationWarning: centered_specific_discharge() has been deprecated and␣
↪will be removed in version 3.3.5. Use postprocessing.get_specific_discharge()␣
↪instead.
  warnings.warn(
```



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.

Getting Started

As shown in the previous MODFLOW tutorial, import flopy.

[1]:
```
import numpy as np
import flopy
```

Creating the MODFLOW Model

Define the Model Extent, Grid Resolution, and Characteristics

Assign the model information

[2]:
```
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

[3]:
```
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:

[4]:
```
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:

[5]:
```
modelname = "tutorial2_mf"
mf = flopy.modflow.Modflow(modelname, exe_name="mf2005")
```

(continues on next page)

(continued from previous page)

```
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]`.

[6]:
```
stageleft = 10.0
stageright = 10.0
bound_sp1 = []
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.
```

[7]:
```
# 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.
```

[8]:
```
# 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}
```

[9]:
```
# 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()`.

[10]:
```
# 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()`.

[11]:
```
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.

[12]:
```
# Write the model input files
mf.write_input()
```

[13]:
```
# Run the model
success, mfoutput = mf.run_model(silent=True, pause=False)
if not success:
    raise Exception("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.

[14]:
```
# Imports
import matplotlib.pyplot as plt
import flopy.utils.binaryfile as bf
```

[15]:
```
# Create the headfile and budget file objects
headobj = bf.HeadFile(modelname + ".hds")
times = headobj.get_times()
cbb = bf.CellBudgetFile(modelname + ".cbc")
```

[16]:
```
# 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)
```

[17]:
```
# Well point for plotting
wpt = (450.0, 550.0)
```

Create a figure with maps for three times

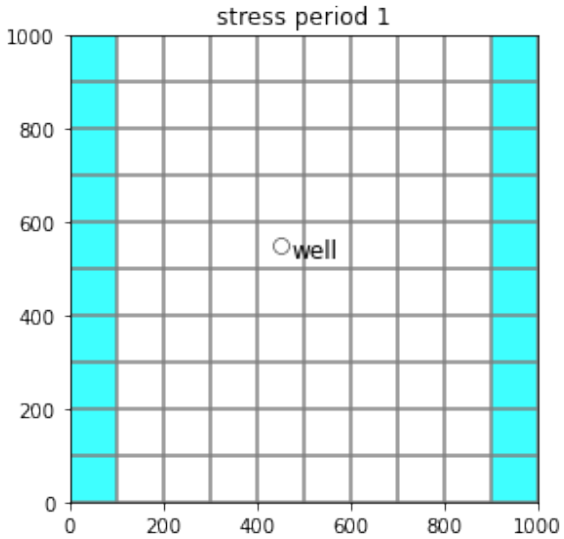
[18]:
```
# Make the plots
fig = plt.figure(figsize=(5, 15))
mytimes = [1.0, 101.0, 201.0]
for iplot, time in enumerate(mytimes):
    print("*****Processing time: ", time)
    head = headobj.get_data(totim=time)
    # Print statistics
    print("Head statistics")
    print("  min: ", head.min())
    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]
```

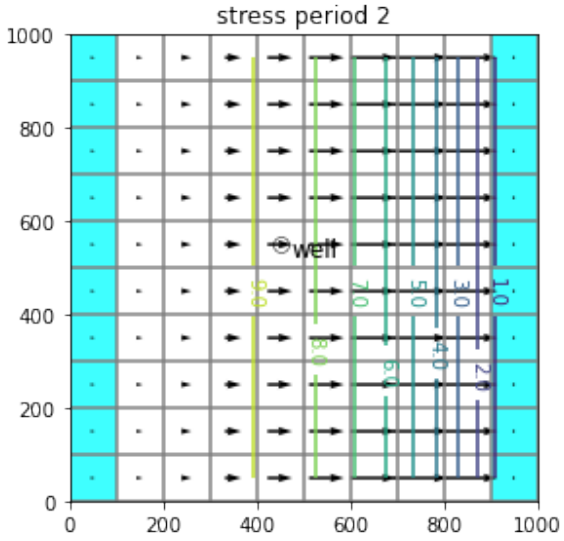
(continues on next page)

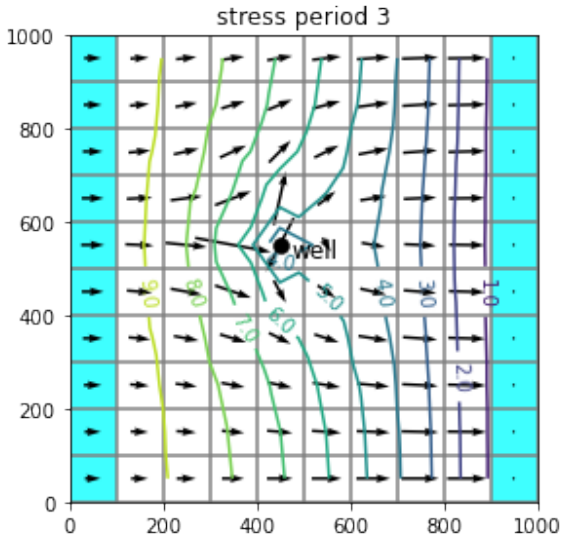
(continued from previous page)

```
    # Create a map for this time
    ax = fig.add_subplot(len(mytimes), 1, iplot + 1, aspect="equal")
    ax.set_title("stress period " + str(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,
        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
```


stress period 1
well
0
200
400
600
800
1000

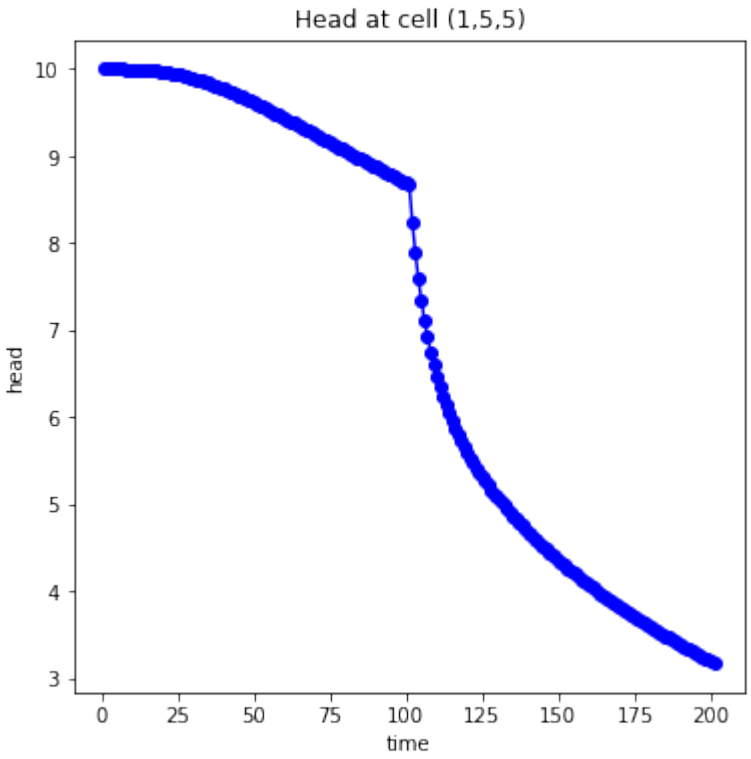

stress period 2
well
0
200
400
600
800
1000


stress period 3
well
0
200
400
600
800
1000

Create a hydrograph

[19]:
```
# 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 = "Head at cell ({0},{1},{2})".format(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-")
```

[19]:
```
[<matplotlib.lines.Line2D at 0x7fcdfb724460>]
```



6.1.5 SEAWAT Tutorials

Contents:

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.

Getting Started

[1]:
```
import numpy as np
import flopy
```

Input variables for the Henry Problem

[2]:
```
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

[3]:
```
modelname = "henry"
swt = flopy.seawat.Seawat(modelname, exe_name="swtv4")
print(swt.namefile)
```

```
henry.nam
```

save cell fluxes to unit 53

[4]:
```
ipakcb = 53
```

Add DIS package to the MODFLOW model

[5]:
```
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,
)
```

[6]:
```
# Variables for the BAS package
ibound = np.ones((nlay, nrow, ncol), dtype=np.int32)
ibound[:, :, -1] = -1
```

Add BAS package to the MODFLOW model

[7]:
```
bas = flopy.modflow.ModflowBas(swt, ibound, 0)
```

Add LPF package to the MODFLOW model

[8]:
```
lpf = flopy.modflow.ModflowLpf(swt, hk=hk, vka=hk, ipakcb=ipakcb)
```

Add PCG Package to the MODFLOW model

[9]:
```
pcg = flopy.modflow.ModflowPcg(swt, hclose=1.0e-8)
```

Add OC package to the MODFLOW model

[10]:
```
oc = flopy.modflow.ModflowOc(
    swt,
    stress_period_data={(0, 0): ["save head", "save budget"]},
    compact=True,
)
```

Create WEL and SSM data

[11]:
```
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

[12]:
```
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

[13]:
```
vdf = flopy.seawat.SeawatVdf(
    swt,
    iwtable=0,
    densemin=0,
    densemax=0,
    denseref=1000.0,
    denseslp=0.7143,
    firstdt=1e-3,
)
```

Write the input files

[14]:
```
swt.write_input()
```

Run the model

[15]:
```
success, buff = swt.run_model(silent=True, report=True)
if not success:
    raise Exception("SEAWAT did not terminate normally.")
```

Post-process the results

[16]:
```
import numpy as np
import flopy.utils.binaryfile as bf
```

Load the concentration data

[17]:
```
ucnobj = bf.UcnFile("MT3D001.UCN", model=swt)
times = ucnobj.get_times()
concentration = ucnobj.get_data(totim=times[-1])
```

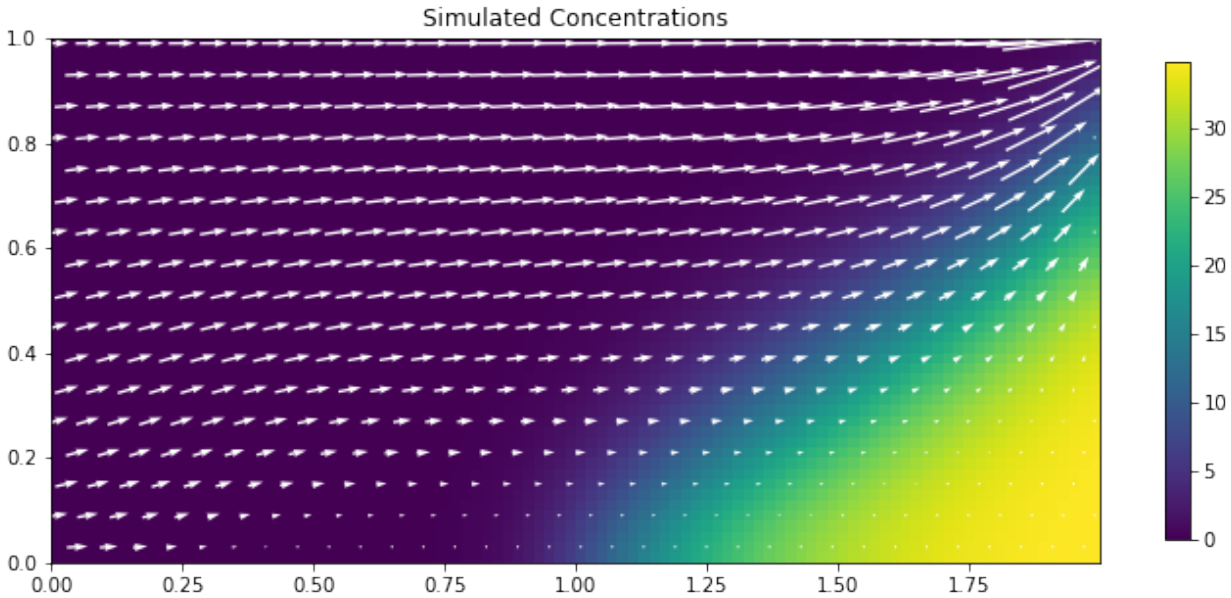
Load the cell-by-cell flow data

[18]:
```
cbbobj = bf.CellBudgetFile("henry.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

[19]:
```
import matplotlib.pyplot as plt
```

[20]:
```
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");
```

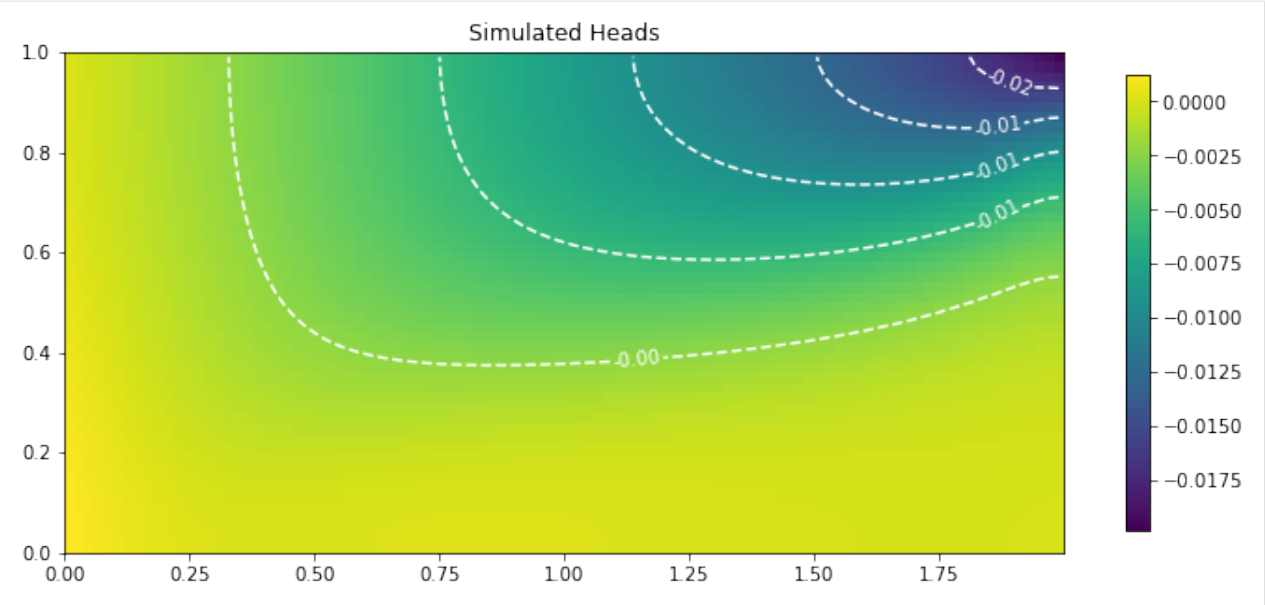


Load the head data

[21]:
```
headobj = bf.HeadFile("henry.hds")
times = headobj.get_times()
head = headobj.get_data(totim=times[-1])
```

Create a plot with heads

[22]:
```
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");
```



6.2 Jupyter Notebooks

A list of Jupyter Notebooks are available here. After the tutorials, the Jupyter Notebooks are the best resource for learning the underlying capabilities of FloPy.

CHAPTER 7

Code Description

Contents:

7.1 Flopy Code

7.1.1 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.

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

class ExtFileAction

Bases: enum.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*, *mfsim=None*)
Bases: object

Class containing MODFLOW path data

Parameters **path** (*str*) – Path on disk to the simulation

model_relative_path
Dictionary of relative paths to each model folder

Type OrderedDict

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*, *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*) – Full path or relative path from working directory to simulation folder

Examples

self.simulation_data.mfdata.set_sim_path('sim_folder')

static string_to_file_path(*fp_string*)
Interpret string as a file path. For internal FloPy use, not intended for end user.

strip_model_relative_path(*model_name*, *path*)
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(*error*)
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

package_key_dict
Dictionary of packages by package key

Type dictionary

static get_module(*package_file_path*)
Static method that returns the python module library. For internal FloPy use only, not intended for end users.

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 get_package_file_paths()
Static method that gets the paths of all the FloPy python package files. For internal FloPy use only, not intended for end users.

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

package_dict
Returns a copy of the package name dictionary.

static package_factory(*package_type*, *model_type*)
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

package_names
Returns a list of package names.

register_package(*package*)
Base method for registering a package. Should be overridden.

class PackageContainerType
Bases: enum.Enum

Determines whether a package container is a simulation, model, or package.

model = 2

package = 3

simulation = 1

exception ReadAsArraysException(*error*)
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
Bases: enum.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.exe'*, *add_to_simulation=True*, *structure=None*, *model_rel_path='.'*, *verbose=False*, ***kwargs*)
Bases: *flopy.mf6.mfbase.PackageContainer*, *flopy.mbase.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

Type OrderedDict(*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()
```

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

hdry
Dry cell value

hnoflo
No-flow cell value

is_valid()
Checks the validity of the model and all of its packages

Returns valid

Return type bool

laycbd
Quasi-3D confining bed. Not supported in MODFLOW-6.

Returns None

Return type None

laytyp
Layering type

classmethod load_base(*simulation*, *structure*, *modelname='NewModel'*, *model_nam_file='modflowtest.nam'*, *mtype='gwf'*, *version='mf6'*, *exe_name='mf6.exe'*, *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_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 relative to simulation working folder
- **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

model_ws

Model file path.

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

modelgrid

Model spatial discretization information.

Returns **model grid** – FloPy object containing spatial discretization information for the model.

Return type Grid subclass

modeltime

Model time discretization information.

Returns **modeltime** – FloPy object containing time discretization information for the simulation.

Return type *ModelTime*

namefile

Model namefile object.

nper

Number of stress periods.

Returns **nper** – Number of stress periods in the simulation.

Return type int

output

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

solver_tols
Returns the solver inner hclose and rclose values.

Returns **inner_hclose, rclose**

Return type float, float

verbose
Verbose setting for model operations (True/False)

version
Version of MODFLOW

write (*ext_file_action=<ExtFileAction.copy_relative_paths: 3>*)
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 OrderDict

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.

static data_factory(*sim_data*, *model_or_sim*, *structure*, *enable*, *path*, *dimensions*, *data=None*, *package=None*)
Creates the appropriate data child object derived from MFData.

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 of 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 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: 3>*)
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*)
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()
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*, *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.

class MFPackage (*model_or_sim*, *package_type*, *filename=None*, *pname=None*, *loading_package=False*, *parent_file=None*)
Bases: *flopy.mf6.mfbase.PackageContainer*, *flopy.pakbase.PackageInterface*

Provides an interface for the user to specify data to build a package.

Parameters

- **model_or_sim** (*MFModel of MFSimulation*) – The parent model or simulation containing this package
- **package_type** (*str*) – String defining the package type
- **filename** (*str*) – Filename of file where this package is stored
- **pname** (*str*) – Package name
- **loading_package** (*bool*) – Whether or not to add this package to the parent container's package list during initialization
- **parent_file** (MFPackage) – Parent package that contains this package

blocks
Dictionary of blocks contained in this package by block name

Type OrderedDict

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 (mfdatalist, mfdataarray, or mfdatascalar) 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 MFData 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

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

Returns

Return type None or Netcdf object

filename
Package's file name.

get_file_path()
Returns the package file's path.

Returns **file path**

Return type str

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

name
Name of package

output
Method to get output associated with a specific package

Returns

Return type MF6Output object

package_type
String describing type of package

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 file_name_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

plottable
If package is plottable

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: 3>*)
Writes the package to a file.

Parameters **ext_file_action** (ExtFileAction) – How to handle pathing of external data files.

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='mf6.exe'*, *sim_ws='.'*, *verbosity_level=1*, *continue_=None*, *nocheck=None*, *memory_print_option=None*, *write_headers=True*)

Bases: *flopy.mf6.mfbase.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*) – Relative path to MODFLOW 6 executable from the simulation working folder.
- **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.

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=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 **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.

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_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 the ims package with the specified *key*.

Parameters **key** (*str*) – ims package key

Returns **ims_package**

Return type *ModflowIms*

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*

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='mf6.exe'*, *sim_ws='.'*, *strict=True*, *verbosity_level=1*, *load_only=None*, *verify_data=False*, *write_headers=True*)
Load an existing model.

Parameters

- **sim_name** (*str*) – Name of the simulation.
- **version** (*str*) – MODFLOW version
- **exe_name** (*str*) – Relative path to MODFLOW executable from the simulation working folder
- **sim_ws** (*str*) – Path to simulation working folder
- **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

Returns **sim**

Return type MFSimulation object

Examples

```
>>> s = flopy.mf6.mfsimulation.load('my simulation')
```

load_package(*ftype*, *fname*, *pname*, *strict*, *ref_path*, *dict_package_name=None*, *parent_package=None*)
Load a package from a file.

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

model_dict
Return a dictionary of models associated with this simulation.

Returns **model dict** – dictionary of models

Return type dict

model_names
Return a list of model names associated with this simulation.

Returns **list**

Return type list of model names

plot(*model_list=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)*) – List of model names to plot, if none all models will be plotted
- **SelPackList** (*(list)*) – 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(*ims_file*, *model_list*)
Register an ims package with the simulation.

Parameters

ims_file [MFPackage] ims package to register

model_list [list of strings] list of models using the ims package to be registered

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)

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

run_simulation(*silent=None*, *pause=False*, *report=False*, *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)
- **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** – 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*)

set_sim_path(*path*)
Return a list of output data keys.

Parameters **path** (*str*) – Relative or absolute path to simulation root folder.

sim_package_list
List of all "simulation level" packages

write_simulation(*ext_file_action=<ExtFileAction.copy_relative_paths: 3>*, *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*, *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

mfpath
File path location information for the simulation

Type *MFFileMgmt*

model_dimensions
Dictionary containing discretization information for each model

Type OrderedDict

mfdata
Custom dictionary containing all model data for the simulation

Type *SimulationDict*

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: collections.OrderedDict

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 OrderedDict 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

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 ModflowGnc(*simulation*, *loading_package=False*, *print_input=None*, *print_flows=None*, *explicit=None*, *numgnc=None*, *numalphaj=None*, *gncdata=None*, *filename=None*, *pname=None*, *parent_file=None*)
Bases: *flopy.mf6.mfpackage.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 = [['block options', 'name print_input', 'type keyword', 'reader urword', 'optiona
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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 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 / 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 specificiation 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:math:^*6*; a value of 10:math:^*4* 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 $\Delta h \leq$ 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, mfutllaktab 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 = [['block options', 'name print_option', 'type string', 'reader urword', 'optiona

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(*simulation*, *loading_package=False*, *print_input=None*, *print_flows=None*, *modelnames=None*, *budget_filerecord=None*, *maxmvr=None*, *maxpackages=None*, *packages=None*, *perioddata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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

- **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 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.
- **maxmvr** (*integer*) –
  - maxmvr (integer) integer value specifying the maximum number of water mover entries that will 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, mfutllaktab package must have a mfgwflak package parent_file.

budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name print_input', 'type keyword', 'reader urword', 'optiona

dfn_file_name = 'gwf-mvr.dfn'

package_abbr = 'mvr'

packages = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

flopy.mf6.modflow.mfnam module

class ModflowNam(*simulation*, *loading_package=False*, *continue_=None*, *nocheck=None*, *memory_print_option=None*, *maxerrors=None*, *tdis6=None*, *models=None*, *exchanges=None*, *mxiter=None*, *solutiongroup=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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.
- **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, mfutllaktab package must have a mfgwflak package parent_file.

dfn = [['block options', 'name continue', 'type keyword', 'reader urword', 'optional t

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_filerecord=None*, *nper=1*, *perioddata=((1.0, 1, 1.0),)*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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_filerecord** (*[ats6_filename]*) –
  - ats6_filename (string) defines an adaptive time step (ATS) input file defining ATS controls. Records in the ATS file can be used to override the time step behavior for selected stress periods.
- **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 $\Delta t_1 = perlen \frac{tsmult-1}{tsmult^{nstp}-1}$.
- **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, mfutllaktab package must have a mfgwflak package parent_file.

ats_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name time_units', 'type string', 'reader urword', 'optional

dfn_file_name = 'sim-tdis.dfn'

package_abbr = 'tdis'

perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

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.exe'*, *model_rel_path='.'*, *list=None*, *print_input=None*, *print_flows=None*, *save_flows=None*, *newtonoptions=None*, *packages=None*, ***kwargs*)

Bases: *flopy.mf6.mfmodel.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.exe'*, *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.exe'*, *model_rel_path='.'*, *list=None*, *print_input=None*, *print_flows=None*, *save_flows=None*, *packages=None*, ***kwargs*)

Bases: *flopy.mf6.mfmodel.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.exe'*, *strict=True*, *model_rel_path='.'*, *load_only=None*)

model_type = 'gwt'

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 = [['block options', 'name boundnames', 'type keyword', 'shape', 'reader urword',

dfn_file_name = 'gwf-api.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfapi'

flopy.mf6.modflow.mfgwfbuy 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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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. The value must be one if concentrations are specified using the CONCENTRATION keyword in the PERIOD block below.
- **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, mfutllaktab package must have a mfgwflak package parent_file.

density_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name hhformulation_rhs', 'type keyword', 'reader urword', 'o

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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, mfutllaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo

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>

flopy.mf6.modflow.mfgwfcsub module

class ModflowGwfcsub(*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*, *fileout=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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfcsub defines a csub 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 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.
- **fileout** (*boolean*) –
  - fileout (boolean) keyword to specify that an output filename is expected next.
- **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, mfutllaktab 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 objec

compaction_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```
compaction_inelastic_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator obj
compaction_interbed_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator obje
dfn = [['block options', 'name boundnames', 'type keyword', 'shape', 'reader urword',
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 obje
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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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, mfutllaktab 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 = [['block options', 'name length_units', 'type string', 'reader urword', 'optiona
dfn_file_name = 'gwf-dis.dfn'
idomain = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
package_abr = '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*, *cell2d=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 specifed 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, ncvert, 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.
  - ncvert (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 = [['block options', 'name length_units', 'type string', 'reader urword', 'optiona

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.mfgwfdisv module

class ModflowGwfdisv(*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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfdisv defines a disv 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, ncvert, 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.
  - ncvert (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, mfutllaktab package must have a mfgwflak package parent_file.

botm = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

```
cell2d = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
dfn = [['block options', 'name length_units', 'type string', 'reader urword', 'optiona
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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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.

- **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 = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo

dfn_file_name = 'gwf-drn.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfdrn'

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 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, PXDP and PETM arrays must be specified NSEG - 1 times each, in order from the uppermost segment down. 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). 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). 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, mfutllaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name fixed_cell', 'type keyword', 'shape', 'reader urword',

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfevta defines a evta 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.
- **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 (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>

depth = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

dfn = [['block options', 'name readasarrays', 'type keyword', 'shape', 'reader urword'

dfn_file_name = 'gwf-evta.dfn'

ievt = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = '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.mfgwfghb module

class ModflowGwfghb(*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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfghb 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 = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo

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 ModflowGwfgnc(*model*, *loading_package=False*, *print_input=None*, *print_flows=None*, *explicit=None*, *numgnc=None*, *numalphaj=None*, *gncdata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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, mfutllaktab package must have a mfgwflak package parent_file.

dfn = [['block options', 'name print_input', 'type keyword', 'reader urword', 'optiona

dfn_file_name = 'gwf-gnc.dfn'

gncdata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfgnc'

flopy.mf6.modflow.mfgwfgwf module

class ModflowGwfgwf(*simulation*, *loading_package=False*, *exgtype=None*, *exgmnamea=None*, *exgmnameb=None*, *auxiliary=None*, *print_input=None*, *print_flows=None*, *save_flows=None*, *cell_averaging=None*, *cvoptions=None*, *newton=None*, *gnc_filerecord=None*, *mvr_filerecord=None*, *observations=None*, *nexg=None*, *exchangedata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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.

- **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.

- **gnc_filerecord** (*[gnc6_filename]*) –

  - gnc6_filename (string) is the file name for ghost node correction input file. Information for the ghost nodes are provided in the file provided with these keywords. The format for specifying the ghost nodes is the same as described for the GNC Package of the GWF Model. This includes specifying OPTIONS, DIMENSIONS, and GNCDATA blocks. The order of the ghost nodes must follow the same order as the order of the cells in the EXCHANGEDATA block. For the GNCDATA, noden and all of the nodej values are assumed to be located in model 1, and nodem is assumed to be in model 2.

- **mvr_filerecord** (*[mvr6_filename]*) –
  - mvr6_filename (string) is the file name of the water mover input file to apply to this exchange. Information for the water mover are provided in the file provided with these keywords. The format for specifying the water mover information is the same as described for the Water Mover (MVR) Package of the GWF Model, with two exceptions. First, in the PACKAGES block, the model name must be included as a separate string before each package. Second, the appropriate model name must be included before package name 1 and package name 2 in the BEGIN PERIOD block. This allows providers and receivers to be located in both models listed as part of this exchange.

- **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.

- **nexg** (*integer*) –
  - nexg (integer) keyword and integer value specifying the number of GWF-GWF exchanges.

- **exchangedata** (*[cellidm1, cellidm2, ihc, cl1, cl2, hwva, aux]*) –
  - 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.

- **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 = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo

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=None*, *exgmnamea=None*, *exgmnameb=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 = []

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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, mfutllaktab package must have a mfgwflak package parent_file.

dfn = [['block options', 'name print_input', 'type keyword', 'reader urword', 'optiona

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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, mfutllaktab package must have a mfgwflak package parent_file.

```
dfn = [['block griddata', 'name strt', 'type double precision', 'shape (nodes)', 'read
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*, *package_convergence_filerecord=None*, *timeseries=None*, *observations=None*, *mover=None*, *surfdep=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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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.
- **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.
- **time_conversion** (*double*) –
  - time_conversion (double) value that is used in converting outlet flow terms that use Manning's equation or gravitational acceleration to consistent 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 in converting outlet flow terms that use Manning's equation or gravitational acceleration to consistent 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*) –
  - noutlets (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 (double) 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, mfutllaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

connectiondata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo

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 obje

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*, *no_well_storage=None*, *flow_correction=None*, *flowing_wells=None*, *shutdown_theta=None*, *shutdown_kappa=None*, *timeseries=None*, *observations=None*, *mover=None*, *nmawwells=None*, *packagedata=None*, *connectiondata=None*, *perioddata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfmaw defines a maw 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 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.
- **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.

- **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, THIEM, 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, THIEM, 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, THIEM, SKIN, or COMPOSITE, SCRN_BOT can be any value is set to the bottom of the cell. If CONDEQN is MEAN, SCRN_BOT 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, mfutllaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```
budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
connectiondata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
dfn = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo
dfn_file_name = 'gwf-maw.dfn'
head_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
package_abbr = 'gwfmaw'
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 ModflowGwfmvr(*model*, *loading_package=False*, *print_input=None*, *print_flows=None*, *modelnames=None*, *budget_filerecord=None*, *maxmvr=None*, *maxpackages=None*, *packages=None*, *perioddata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfmvr defines a mvr package within a gwf6 model. This package can only be used to move water between packages within a single model. To move water between models use ModflowMvr.

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 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.
- **maxmvr** (*integer*) –

  - maxmvr (integer) integer value specifying the maximum number of water mover entries that will 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, mfutllaktab package must have a mfgwflak package parent_file.

budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name print_input', 'type keyword', 'reader urword', 'optiona

dfn_file_name = 'gwf-mvr.dfn'

package_abbr = 'gwfmvr'

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfnam defines a nam 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.
- **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 = [['block options', 'name list', 'type string', '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*, *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*, *icelltype=0*, *k=1.0*, *k22=None*, *k33=None*, *angle1=None*, *angle2=None*, *angle3=None*, *wetdry=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfnpf defines a npf 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 budget flow terms will be written to the file specified with "BUDGET SAVE FILE" in Output Control.
- **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.
- **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.

- **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 of icelltype indicates that saturated thickness will be computed as STRT-BOT and held constant.

- **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 counterclockwise 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 equaled 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 = [['block options', 'name save_flows', 'type keyword', 'reader urword', 'optional

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>

wetdry = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

flopy.mf6.modflow.mfgwfoc module

class ModflowGwfoc(*model*, *loading_package=False*, *budget_filerecord=None*, *head_filerecord=None*, *headprintrecord=None*, *saverecord=None*, *printrecord=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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.
- **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 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.

budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name budget_filerecord', 'type record budget fileout budgetf

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfrch defines a rch 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 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 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, mfutllaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name fixed_cell', 'type keyword', 'shape', 'reader urword',

dfn_file_name = 'gwf-rch.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfrch'

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfrcha defines a rcha 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.
- **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 = [['block options', 'name readasarrays', 'type keyword', 'shape', 'reader urword'
dfn_file_name = 'gwf-rcha.dfn'
irch = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>
obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
package_abbr = 'gwfrcha'
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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo

dfn_file_name = 'gwf-riv.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfriv'

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*, *package_convergence_filerecord=None*, *timeseries=None*, *observations=None*, *mover=None*, *maximum_picard_iterations=None*, *maximum_iterations=None*, *maximum_depth_change=None*, *unit_conversion=None*, *nreaches=None*, *packagedata=None*, *connectiondata=None*, *diversions=None*, *perioddata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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.
- **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) 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) 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_depth_change** (*double*) –
  - maximum_depth_change (double) value that defines the depth closure tolerance. By default, DMAXCHG is equal to 1×10^{-5}.
- **unit_conversion** (*double*) –
  - unit_conversion (double) value (or conversion factor) that is used in calculating stream depth for stream reach. 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.

- **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 top elevation of the reach streambed.
  - 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.

* **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 $\leq$ 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 ($Q_{DS}-$ 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 ($Q_{DS}-$ 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 simu-

lated 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.

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>
connectiondata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
dfn = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo
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 obje
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*, *iconvert=0*, *ss=1e-05*, *sy=0.15*, *steady_state=None*, *transient=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 specific storage is only calculated when a cell is under confined conditions (head greater than or equal to the top of the cell). This option is identical to the approach used to calculate storage changes under confined conditions in MODFLOW-2005.
- **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 = [['block options', 'name save_flows', 'type keyword', 'reader urword', 'optional

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>

flopy.mf6.modflow.mfgwfuzf module

class ModflowGwfuzf(*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*, *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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfuzf 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.
- **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.
- **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.
  - ivertcon (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 calculated 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 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), val-

ues 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, mfutllaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo

dfn_file_name = 'gwf-uzf.dfn'

obs_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

package_abbr = 'gwfuzf'

package_convergence_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator obje

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.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*, *timeseries=None*, *observations=None*, *mover=None*, *maxbound=None*, *stress_period_data=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwfwel defines a wel 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 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.

- **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.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo

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>

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 = [['block options', 'name scheme', 'type string', 'valid central upstream tvd', '

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 = [['block options', 'name boundnames', 'type keyword', 'shape', 'reader urword',

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo

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.mfgwtdis module

class ModflowGwtdis(*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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwtdis 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, mfutllaktab 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 = [['block options', 'name length_units', 'type string', 'reader urword', 'optiona

dfn_file_name = 'gwt-dis.dfn'

idomain = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

package_abbr = '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*, *cell2d=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwtdisu defines a disu 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*) –
  - 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 specifed 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, ncvert, 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.
- ncvert (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 = [['block options', 'name length_units', 'type string', 'reader urword', 'optiona

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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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*) –
  - 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, 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, ncvert, 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.
  - ncvert (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, mfutllaktab package must have a mfgwflak package parent_file.

```
botm = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

cell2d = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name length_units', 'type string', 'reader urword', 'optiona

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.mfgwtdsp 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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 dispsersivity 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, mfutllaktab 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 = [['block options', 'name xt3d_off', 'type keyword', 'shape', 'reader urword', 'o

dfn_file_name = 'gwt-dsp.dfn'

diffc = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

package_abbr = 'gwtdsp'

flopy.mf6.modflow.mfgwtfmi module

class ModflowGwtfmi(*model*, *loading_package=False*, *save_flows=None*, *flow_imbalance_correction=None*, *packagedata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 = [['block options', 'name save_flows', 'type keyword', 'reader urword', 'optional

dfn_file_name = 'gwt-fmi.dfn'

package_abbr = 'gwtfmi'

packagedata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

flopy.mf6.modflow.mfgwtic module

class ModflowGwtic(*model*, *loading_package=False*, *strt=0.0*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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, mfutllaktab package must have a mfgwflak package parent_file.

```
dfn = [['block griddata', 'name strt', 'type double precision', 'shape (nodes)', 'read
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*, *sorption=None*, *first_order_decay=None*, *zero_order_decay=None*, *cim_filerecord=None*, *cimprintrecord=None*, *cim=None*, *thetaim=None*, *zetaim=None*, *decay=None*, *decay_sorbed=None*, *bulk_density=None*, *distcoef=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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.
- **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.
- **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.
- **thetaim** (*[double]*) –
  - thetaim (double) porosity of the immobile domain specified as the volume of immobile pore space per total volume (dimensionless).
- **zetaim** (*[double]*) –
  - zetaim (double) mass transfer rate coefficient between the mobile and immobile domains, in dimensions of per time.
- **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 the aquifer in mass per length cubed. bulk_density will have no effect on simulation results unless the SORPTION keyword is specified in the options block.
- **distcoef** (*[double]*) –
  - distcoef (double) is the distribution coefficient for the equilibrium-controlled linear sorption isotherm in dimensions of length cubed per mass. distcoef will have no effect on simulation results unless the SORPTION keyword is specified in the options 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.

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 = [['block options', 'name save_flows', 'type keyword', 'reader urword', 'optional

dfn_file_name = 'gwt-ist.dfn'

distcoef = <flopy.mf6.data.mfdatautil.ArrayTemplateGenerator object>

package_abbr = 'gwtist'

thetaim = <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*, *timeseries=None*, *observations=None*, *packagedata=None*, *lakeperioddata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 "HEAD 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.
- **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, mfutllaktab package must have a mfgwflak package parent_file.

```
auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
concentration_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
dfn = [['block options', 'name flow_package_name', 'type string', 'shape', 'reader urw
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.mfgwtmst 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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 aquifer porosity.

- **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.
- **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, mfutllaktab 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 = [['block options', 'name save_flows', 'type keyword', 'reader urword', 'optional

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.mfgwtmvt module

class ModflowGwtmvt(*model*, *loading_package=False*, *print_input=None*, *print_flows=None*, *save_flows=None*, *budget_filerecord=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwtmvt defines a mvt 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_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.
- **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>

dfn = [['block options', 'name print_input', 'type keyword', 'reader urword', 'optiona

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*, *timeseries=None*, *observations=None*, *packagedata=None*, *mwtperioddata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 "HEAD 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.
- **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 of 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>

concentration_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```
dfn = [['block options', 'name flow_package_name', 'type string', 'shape', 'reader urw
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.mfgwtnam module

class ModflowGwtnam(*model*, *loading_package=False*, *list=None*, *print_input=None*, *print_flows=None*, *save_flows=None*, *packages=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwtnam 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 = [['block options', 'name list', 'type string', '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*, *concentration_filerecord=None*, *concentrationprintrecord=None*, *saverecord=None*, *printrecord=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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.
- **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 (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.

budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

concentration_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

concentrationprintrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name budget_filerecord', 'type record budget fileout budgetf

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*, *timeseries=None*, *observations=None*, *packagedata=None*, *reachperioddata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 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 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.
- **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, mfutllaktab package must have a mfgwflak package parent_file.

auxiliary = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

```
budget_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
concentration_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>
dfn = [['block options', 'name flow_package_name', 'type string', 'shape', 'reader urw
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*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 = [['block options', 'name auxiliary', 'type string', 'shape (naux)', 'reader urwo

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*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.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 concenration 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.

- **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 = [['block options', 'name print_flows', 'type keyword', 'reader urword', 'optiona

dfn_file_name = 'gwt-ssm.dfn'

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*, *timeseries=None*, *observations=None*, *packagedata=None*, *uztperioddata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowGwtuzt defines a uzt 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 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 "HEAD 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.
- **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>

concentration_filerecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name flow_package_name', 'type string', 'shape', 'reader urw

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>

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(*model*, *loading_package=False*, *maxats=1*, *perioddata=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowUtlats defines a ats 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.
- **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 multipled 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 of 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 = [['block dimensions', 'name maxats', 'type integer', 'reader urword', 'optional

dfn_file_name = 'utl-ats.dfn'

package_abbr = 'utlats'

perioddata = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

class UtlatsPackages(*model*, *parent*, *pkg_type*, *filerecord*, *package=None*, *package_class=None*)

Bases: *flopy.mf6.mfpackage.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.mfutllaktab module

class ModflowUtllaktab(*model*, *loading_package=False*, *nrow=None*, *ncol=None*, *table=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowUtllaktab defines a tab 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 = [['block dimensions', 'name nrow', 'type integer', 'reader urword', 'optional fa

dfn_file_name = 'utl-lak-tab.dfn'

package_abbr = 'utltab'

table = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

flopy.mf6.modflow.mfutlobs module

class ModflowUtlobs(*model*, *loading_package=False*, *digits=None*, *print_input=None*, *continuous=None*, *filename=None*, *pname=None*, *parent_file=None*)

Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowUtlobs defines a obs 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.

- **digits** (*integer*) –
  - digits (integer) Keyword and an integer digits specifier used for conversion of simulated values to text on output. The default is 5 digits. When simulated values are written to a file specified as file type DATA in the Name File, 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.
- **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, mfutllaktab package must have a mfgwflak package parent_file.

continuous = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

dfn = [['block options', 'name digits', 'type integer', 'shape', 'reader urword', 'opt

dfn_file_name = 'utl-obs.dfn'

package_abbr = 'utlobs'

class UtlobsPackages(*model*, *parent*, *pkg_type*, *filerecord*, *package=None*, *package_class=None*)

Bases: *flopy.mf6.mfpackage.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.mfutltas module

class ModflowUtltas(*model*, *loading_package=False*, *time_series_namerecord=None*, *interpolation_methodrecord=None*, *sfacrecord=None*, *tas_array=None*, *filename=None*, *pname=None*, *parent_file=None*)
Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowUtltas defines a tas 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.
- **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 = [['block attributes', 'name time_series_namerecord', 'type record name time_seri

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*, *parent*, *pkg_type*, *filerecord*, *package=None*, *package_class=None*)
Bases: *flopy.mf6.mfpackage.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 ModflowUtltas package to the container. See ModflowUtltas 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.mfutlts module

class ModflowUtlts(*model*, *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*, *parent_file=None*)
Bases: *flopy.mf6.mfpackage.MFPackage*

ModflowUtlts defines a ts 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.
- **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 (mfutl*). For example, mfutllaktab package must have a mfgwflak package parent_file.

dfn = [['block attributes', 'name time_series_namerecord', 'type record names time_ser

dfn_file_name = 'utl-ts.dfn'

interpolation_methodrecord = <flopy.mf6.data.mfdatautil.ListTemplateGenerator object>

interpolation_methodrecord_single = <flopy.mf6.data.mfdatautil.ListTemplateGenerator o

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*, *parent*, *pkg_type*, *filerecord*, *package=None*, *package_class=None*)

Bases: *flopy.mf6.mfpackage.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'

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** (*OrderedDictionary 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.
mfdict: OrderedDict local instance of the SimulationDict() object
path: pointer to the MFSimulationPath object
key: tuple user requested data key

MFOutputRequester.querybinarydata returns: Xarray object

```
>>> 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: *flopy.utils.utils_def.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')
```

angrot
Model grid rotation angle. None if not defined in the MODFLOW 6 grid file.

Returns **angrot**

Return type float

bot
Bottom of the model cells.

Returns **bot**

Return type ndarray of floats

cell2d
cell2d data for a DISV grid. None for DIS and DISU grids.

Returns **cell2d**

Return type list of lists

cellcenters
Cell centers (x,y).

Returns **cellcenters**

Return type ndarray of floats

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

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

grid_type
Grid type defined in the MODFLOW 6 grid file.

Returns **grid_type**

Return type str

ia
CRS row pointers for the model grid.

Returns **ia**

Return type ndarray of ints

iavert
CRS cell pointers for cell vertices.

Returns **iavert**

Return type ndarray of ints

idomain
IDOMAIN for the model grid. None if not defined in the MODFLOW 6 grid file.

Returns **idomain**

Return type ndarray of ints

iverts
Vertex numbers comprising each cell for every cell in model grid.

Returns **iverts**

Return type list of lists of ints

ja
CRS column pointers for the model grid.

Returns **ja**

Return type ndarray of ints

javert
CRS vertex numbers for the vertices comprising each cell.

Returns **javerts**

Return type ndarray of ints

modelgrid
Model grid object.

Returns **modelgrid**

Return type *StructuredGrid*, *VertexGrid*, *UnstructuredGrid*

ncells
Number of cells.

Returns **ncells**

Return type int

ncol
Number of columns. None for DISV and DISU grids.

Returns **ncol**

Return type int

ncpl
Number of cells per layer. None for DISU grids.

Returns ncpl

Return type int

nja
Number of non-zero entries in the CRS column pointer vector.

Returns nja

Return type int

nlay
Number of layers. None for DISU grids.

Returns nlay

Return type int

nodes
Number of nodes.

Returns nodes

Return type int

nrow
Number of rows. None for DISV and DISU grids.

Returns nrow

Return type int

shape
Shape of the model grid (tuple).

Returns shape

Return type tuple

spatialreference
Spatial reference for model grid.

Returns spatialreference

Return type *SpatialReference*

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

verts
x,y location of each vertex that defines the model grid.

Returns verts

Return type ndarray of floats

xorigin
x-origin of the model grid. None if not defined in the MODFLOW 6 grid file.

Returns xorigin

Return type float

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

mfdict: (dict) the sim.simulation_dict.mfdict object for the flopy project path: (object) the path object detailing model names and paths key: (tuple, stings) 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)?

fi = (sting) name of the observation binary output file
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

Returns

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*) –

csv_names

Method to get csv file names

Returns

Return type list

methods()

Method that returns a list of available method calls

Returns

Return type list

obs_names

Method to get obs file names

Returns

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*, *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.
- **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
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

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

ncol

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

xcenter

xcentergrid

xedge

xgrid

ycenter

ycentergrid

yedge

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)

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

xarr

xcenter_array

xdict

xydict

yarr

ycenter_array

ydict

flopy.mf6.utils.lakpak_utils module

get_lak_connections(*modelgrid*, *lake_map*, *idomain=None*, *bedleak=0.1*)

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.

Returns

- **idomain** (*ndarry*) – 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

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*)

Bases: `flopy.mf6.data.mfdata.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

data
Returns array data. Calls get_data with default parameters.

data_type
Type of data (DataType) stored in the array

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_num". If "layer_num" is None, returns all data.

Parameters **layer_num** (*int*) –

Returns **data** – Array data in an ndarray

Return type ndarray

get_file_entry(*layer=None*, *ext_file_action=<ExtFileAction.copy_relative_paths: 3>*)
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

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** (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*)

make_layered()
Changes the data to be stored by layer instead of as a single array.

new_simulation(*sim_data*)
Initialize MFArray 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 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

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 4) {'filename':filename, 'factor':fct, 'iprn':print, 'data':data} - dictionary defining external file information 5) {'data':data, 'factor':fct, 'iprn':print) - dictionary defining internal information. Data that is layered can also be set by defining a list with a length equal to the number of layers in the model. Each layer in the list contains the data as defined in the formats above:

[layer_1_val, [layer_2_array_vals], {'filename':file_with_layer_3_data, 'factor':fct, 'iprn':print}]

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 MFArray supports layered data

Parameters **layered_data** (*bool*) – Whether data is layered or not.

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 MFArray 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*)
Bases: *flopy.mf6.data.mfdataarray.MFArray*, flopy.mf6.data.mfdata.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 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

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

data_type
Type of data (DataType) stored in the array

get_data(*layer=None*, *apply_mult=True*, ***kwargs*)
Returns the data associated with stress period key *layer*. If *layer* is None, returns all data for time *layer*.

Parameters

- **layer** (*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: 3>*)
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

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** (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*)

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.

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*)

Bases: flopy.mf6.data.mfdata.MFMultiDimVar, flopy.datbase.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.

data_type

Type of data (DataType) stored in the list

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: 3>*)

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

has_data()

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.

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

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" with. 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
3) {'filename':filename, factor=fct, iprn=print_code, data=data} - dictionary defining the external file containing the datalist.

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

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*)
Bases: *flopy.mf6.data.mfdatalist.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*)

Bases: *flopy.mf6.data.mfdatalist.MFList*, flopy.mf6.data.mfdata.MFTransient, flopy.datbase.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.

data

Returns list data. Calls get_data with default parameters.

data_type

Type of data (DataType) stored in the list

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: 3>*)

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

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** (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

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 an recarray or list of tuples, 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.
- **key** (*int*) – Zero based stress period to assign data too. Does not apply if *data* is a dictionary.
- **autofill** (*bool*) – Automatically correct 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: flopy.mf6.data.mfdata.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

data
Returns the scalar data. Calls get_data with default parameters.

data_type
Type of data (DataType) stored in the scalar

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: 3>*)
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()
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** (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*)

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

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: *flopy.mf6.data.mfdatascalar.MFScalar*, flopy.mf6.data.mfdata.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

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: 3>*)

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** (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

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

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 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.

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

class PackageLevel
Bases: enum.Enum

An enumeration.

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*)

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*)

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_dfns(*flopy_dfn_path*)

delete_files(*files*, *pth*, *allow_failure=False*, *exclude=None*)

delete_mf6_classes()

download_dfn(*branch*, *new_dfn_pth*)

generate_classes(*branch='master'*, *dfnpath=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

- **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.
- **dfnpath** (*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. dfnpath will take precedence over branch if dfnpath 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*)

7.1.2 Previous Versions of MODFLOW

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='mf2k.exe'*, *model_ws=None*, *structured=True*, *verbose=False*, ***kwargs*)
Bases: *flopy.mbase.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, default "mf2k.exe"*) – Name of the modflow executable.
- **model_ws** (*str, optional*) – Path to the model workspace. Model files will be created in this directory. Default is None, in which case model_ws is assigned to 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), `proj4_str` for a PROJ string, and `start_datetime` for model start date (default "1-1-1970").

add_existing_package (*filename*, *ptype=None*, *copy_to_model_ws=True*)

Add an existing package to a model instance.

Parameters

- **filename** (*str*) – the name of the file to add as a package
- **ptype** (*optional*) – the 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*) – flag to copy the package file into the model_ws directory.

Returns

Return type None

add_external (*fname*, *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*) – filename of external array
- **unit** (*int*) – unit number of external array
- **binflag** (*boolean*) – binary or not. (default is False)

add_output (*fname*, *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=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*) – filename 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*) – boolean flag indicating if the output file is a binary file. Default is True

- **package** (*str*) – string that defines 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=None*, *reset_external=False*)
Change the model work space.

Parameters **new_pth** (*str*) – Location of new model workspace. If this path does not exist, it will be created. (default is None, which will be assigned to 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=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

Return type None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.check()
```

exename

export(*f*, ***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*) – 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

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=None*, *unit=None*)
Get an output file from the model by specifying either the file name or the unit number.

Parameters

- **fname** (*str*) – filename of output array
- **unit** (*int*) – unit number of output array

get_output_attribute(*fname=None*, *unit=None*, *attr=None*)
Get a attribute for an output file from the model by specifying either the file name or the unit number.

Parameters

- **fname** (*str*) – filename of output array
- **unit** (*int*) – 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

hdry

hnoflo

laycbd

laytyp

load_results()

model_ws

modelgrid

modeltime

name
Get model name

Returns **name** – name of model

Return type str

namefile

next_ext_unit()
Function to encapsulate next_ext_unit attribute

next_unit(*i=None*)

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=None*, *unit=None*)
Remove an external file from the model by specifying either the file name or the unit number.

Parameters

- **fname** (*str*) – filename of external array
- **unit** (*int*) – unit number of external array

remove_output(*fname=None*, *unit=None*)
Remove an output file from the model by specifying either the file name or the unit number.

Parameters

- **fname** (*str*) – filename of output array
- **unit** (*int*) – 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'*)
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, buff)*
- **success** (*boolean*)
- **buff** (*list of lines of stdout*)

set_model_units()
Every model needs its own set_model_units method

set_output_attribute(*fname=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*) – filename of output array
- **unit** (*int*) – unit number of output array

set_version(*version*)

to_shapefile(*filename*, *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** (*string*) – name 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)

Returns

Return type None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> m.to_shapefile('model.shp', SelPackList)
```

verbose

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

check(*f=None*, *verbose=True*, *level=1*)

exename

export(*f*, ***kwargs*)

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

hdry

hnoflo

laycbd

laytyp

model_ws

modelgrid

namefile

packagelist

solver_tols

update_modelgrid()

verbose

version

run_model(*exe_name*, *namefile*, *model_ws='./'*, *silent=False*, *pause=False*, *report=False*, *normal_msg='normal termination'*, *use_async=False*, *cargs=None*)

This function will run the model using subprocess.Popen. It communicates with the model's stdout asynchronously and reports progress to the screen with timestamps

Parameters

- **exe_name** (*str*) – Executable name (with path, if necessary) to run.
- **namefile** (*str*) – Namefile of model to run. The namefile must be the filename of the namefile without the path. Namefile can be None to allow programs that do not require a control file (name file) to be passed as a command line argument.
- **model_ws** (*str*) – Path to the location of the namefile. (default is the current working directory - './')
- **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 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** (*boolean*) – 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, buff)*
- **success** (*boolean*)
- **buff** (*list of lines of stdout*)

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*, *extra=''*, *filenames=None*, *allowDuplicates=False*)

Bases: *flopy.pakbase.PackageInterface*

Base package class from which most other packages are derived.

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.

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

Returns

Return type None or Netcdf object

level1_arraylist (*idx*, *v*, *name*, *txt*)

static load (*f*, *model*, *pak_type*, *ext_unit_dict=None*, ***kwargs*)
Default load method for standard boundary packages.

name

package_type

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()
```

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

Returns

Return type None

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.lpf.to_shapefile('test_hk.shp')
```

webdoc()

write_file(*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 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.
- **checktype** (check) – Checker type to be used. By default class check is used from check.py.

Returns

Return type None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.dis.check()
```

data_list

export(*f*, ***kwargs*)

has_stress_period_data

name

package_type

parent

plottable

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='mf2005.exe'*, *structured=True*, *listunit=2*, *model_ws='.'*, *external_path=None*, *verbose=False*, ***kwargs*)

Bases: *flopy.mbase.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, default "mf2005.exe"*) – 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.

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
```

get_ifrefm()

get_nrow_ncol_nlay_nper()

classmethod load(*f*, *version='mf2005'*, *exe_name='mf2005.exe'*, *verbose=False*, *model_ws='.'*, *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), "mfnwt", or "mfusg". Note that this can be modified on loading packages unique to different MODFLOW versions.
- **exe_name** (*str, default "mf2005.exe"*) – MODFLOW executable name.
- **verbose** (*bool, default False*) – Show messages that can be useful for debugging.
- **model_ws** (*str, 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" or "disu" 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.

Returns

Return type *flopy.modflow.mf.Modflow*

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('model.nam')
```

load_results(***kwargs*)

modelgrid

modeltime

ncol

ncpl

nlay

nper

nrow

nrow_ncol_nlay_nper

set_ifrefm(*value=True*)

set_model_units(*iunit0=None*)

Write the model name file.

solver_tols

write_name_file()
Write the model name file.

class ModflowGlobal(*model*, *extension='glo'*)
Bases: *flopy.pakbase.Package*

ModflowGlobal Package class

write_file()
Every Package needs its own write_file function

class ModflowList(*model*, *extension='list'*, *unitnumber=2*)
Bases: *flopy.pakbase.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: *flopy.pakbase.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: *flopy.pakbase.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"

Returns

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 pacakge to
- **nper** (*int*) – number of stress periods in model
- **ext_unit_dict** (*dict, optional*) –

Returns

Return type ModflowAg object

plottable

segment_list
Method to get a unique list of segments from irrdiversion

Returns

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: *flopy.pakbase.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 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.

Returns

Return type None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.bas6.check()
```

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)

Returns

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*)

Bases: *flopy.pakbase.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.

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow()
>>> bcf = flopy.modflow.ModflowBcf(ml)
```

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.parsenamefile*.

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.

Returns

Return type None

flopy.modflow.mfbct module

class ModflowBct(*model*, *itrnsp=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*, *sconc=0.0*, *extension='bct'*, *unitnumber=None*)
Bases: *flopy.pakbase.Package*

Block centered transport package class for MODFLOW-USG

write_file()
Write the package file.

Returns

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: *flopy.pakbase.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]
    ], ...
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.

Returns

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: *flopy.pakbase.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.

Returns

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*, *start_datetime=None*)

Bases: *flopy.pakbase.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
- **proj4_str** (*str*) – PROJ4 string that defines the projected coordinate system (e.g. '+proj=utm +zone=14 +datum=WGS84 +units=m +no_defs '). Can be an EPSG code (e.g. 'EPSG:32614'). Default is None.
- **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.

Returns

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.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** (*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)
```

sr

thickness
Return cell thicknesses.

Returns **thickness**

Return type array of floats (nlay, nrow, ncol)

write_file(*check=True*)
Write the package file.

Parameters **check** (*bool*) – Check package data for common errors. (default True)

Returns

Return type None

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.mfdisu module

mfdisu module. Contains the ModflowDisU class. Note that the user can access the ModflowDisU class as *flopy.modflow.ModflowDisU*.

class ModflowDisU(*model*, *nodes=2*, *nlay=1*, *njag=None*, *ivsd=0*, *nper=1*, *itmuni=4*, *lenuni=2*, *idsymrd=0*, *laycbd=0*, *nodelay=None*, *top=1*, *bot=0*, *area=1.0*, *iac=None*, *ja=None*, *ivc=None*, *cl1=None*, *cl2=None*, *cl12=None*, *fahl=None*, *perlen=1*, *nstp=1*, *tsmult=1*, *steady=True*, *extension='disu'*, *unitnumber=None*, *filenames=None*, *start_datetime=None*)

Bases: *flopy.pakbase.Package*

MODFLOW Unstructured Discretization Package Class.

Parameters

- **model** (*model object*) – The model object (of type `flopy.modflow.Modflow`) to which this package will be added.
- **nodes** (*int*) – Number of nodes in the model grid (default is 2).
- **nlay** (*int*) – Number of layers in the model grid (default is 1).
- **njag** (*int*) – Total number of connections of an unstructured grid. njag is used to dimension the sparse matrix in a compressed row storage format. For symmetric arrays, only the upper triangle of the matrix may be entered. For that case, the symmetric portion (minus the diagonal terms) is dimensioned as njags = (njag - nodes) / 2. (default is None).
- **ivsd** (*int*) – is the vertical sub-discretization index. For purposes of this flag, vertical sub-discretization is defined to occur when all layers are not a stacked representation of each other. If IVSD = 0 there is no sub-discretization of layers within the model domain. That is, grids are not nested in the vertical direction. However, one layer may have a different grid structure from the next due to different sub-gridding structures within each layer. If IVSD = 1 there could be sub-discretization of layers with vertically nested grids (as shown in Figure 5c in the MODFLOW-USG document) within the domain. For this case, the vertical connection index IVC is required to determine the vertical connections of every node. Otherwise, the vertical connections are internally computed and IVC is not read. If IVSD = -1 there is no vertical sub-discretization of layers, and further, the horizontal discretization of all layers is the same. For this case, the cell areas (AREA) are read only for one layer and are computed to be the same for all the stacked layers. A structured finite-difference grid is an example of this condition. (default is 0).
- **nper** (*int*) – Number of model stress periods (the default is 1).
- **itmuni** (*int*) – Time units, default is days (4)
- **lenuni** (*int*) – Length units, default is meters (2)
- **idsymrd** (*int*) – A flag indicating if the finite-volume connectivity information of an unstructured grid is input as a full matrix or as a symmetric matrix in the input file. If idsymrd is 0 the finite-volume connectivity information is provided for the full matrix of the porous matrix grid-block connections of an unstructured grid. The code internally stores only the symmetric portion of this information. This input structure (IDSYMRD=0) is easy to organize but contains unwanted information which is parsed out when the information is stored. If idsymrd is 1 then finite-volume connectivity information is provided only for the upper triangular portion of the porous matrix grid-block connections within the unstructured grid. This input structure (IDSYMRD=1) is compact but is slightly more complicated to organize. Only the non-zero upper triangular items of each row are read in sequence for all symmetric matrices. (default is 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)

- **nodelay** (*int or array of ints (nlay)*) – The number of cells in each layer. (the default is None, which means the number of cells in a layer is equal to nodes / nlay).
- **top** (*float or array of floats (nodes), optional*) – An array of the top elevation for every cell. For the 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)
- **bot** (*float or array of floats (nodes), optional*) – An array of the bottom elevation for each model cell (the default is 0.)
- **area** (*float or array of floats*) – Surface area for model cells. Area is for only one layer if IVSD = -1 to indicate that the grid is vertically stacked. Otherwise, area is required for each layer in the model grid. Note that there may be different number of nodes per layer (ndslay) for an unstructured grid. (default is 1.0)
- **iac** (*array of integers*) – is a vector indicating the number of connections plus 1 for each node. Note that the IAC array is only supplied for the GWF cells; the IAC array is internally expanded to include CLN or GNC nodes if they are present in a simulation. (default is None. iac must be provided).
- **ja** (*array of integers*) – is a list of cell number (n) followed by its connecting cell numbers (m) for each of the m cells connected to cell n. This list is sequentially provided for the first to the last GWF cell. Note that the cell and its connections are only supplied for the GWF cells and their connections to the other GWF cells. This connectivity is internally expanded if CLN or GNC nodes are present in a simulation. Also note that the JA list input may be chopped up to have every node number and its connectivity list 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 corrected by the code. (default is None. ja must be provided).
- **ivc** (*int or array of integers*) – is an index array indicating the direction between a node n and all its m connections. IVC = 0 if the connection between n and m is horizontal. IVC = 1 if the connecting node m is vertically oriented to node n. Note that if the CLN Process is active, the connection between two CLN cells has IVC = 2 and the connection between a CLN cell and a GWF cell has IVC = 3. (default is None. ivc must be provided if ivsd = 1)
- **cl1** (*float or array of floats*) – is the perpendicular length between the center of a node (node 1) and the interface between the node and its adjoining node (node 2). (default is None. cl1 and cl2 must be specified, or cl12 must be specified)
- **cl2** (*float or array of floats*) – is the perpendicular length between node 2 and the interface between nodes 1 and 2, and is at the symmetric location of CL1. (default is None. cl1 and cl2 must be specified, or cl12 must be specified)
- **cl12** (*float or array of floats*) – is the array containing CL1 and CL2 lengths, where CL1 is the perpendicular length between the center of a node (node 1) and the interface between the node and its adjoining node (node 2). CL2, which is the perpendicular length between node 2 and the interface between nodes 1 and 2 is at the symmetric location of CL1. The array CL12 reads both CL1 and CL2 in the upper and lower triangular portions of the matrix respectively. Note that the CL1 and CL2 arrays are only supplied for the GWF cell connections and are internally expanded if CLN or GNC nodes exist in a simulation. (default is None. cl1 and cl2 must be specified, or cl12 must be specified)

- **fahl** (*float or arry of floats*) – Area of the interface Anm between nodes n and m. (default is None. fahl must be specified.)
- **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).
- **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.

heading
Text string written to top of package input file.

Type str

Notes

Now works for multi-layer USG models since u3d was modified to handle multiple u2d instances of different size.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> disu = flopy.modflow.ModflowDisU(m)
```

checklayerthickness()
Check layer thickness.

get_cell_volumes()
Get an array of cell volumes.

Returns vol

Return type array of floats (nodes)

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** (*bool*) – Check package data for common errors. (default False; not setup yet)

Returns **dis** – ModflowDisU object.

Return type ModflowDisU object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> disu = flopy.modflow.ModflowDisU.load('test.disu', m)
```

ncpl

thickness
Return cell thicknesses.

Returns **thickness**

Return type array of floats (nodes,)

write_file()
Write the package file.

Returns

Return type None

zcentroids
Return an array of size nodes that contains the vertical cell center elevation.

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: *flopy.pakbase.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" in passed in options, the drain return package (DRT) activated, which expects a different (longer) dtype for stress_period_data

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow()
>>> lrcec = {0:[2, 3, 4, 10., 100.]}  #this drain will be applied to all
```

(continues on next page)

(continued from previous page)

```
>>>                                    #stress periods
>>> drn = flopy.modflow.ModflowDrn(ml, 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)

Returns

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: *flopy.pakbase.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],
    [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
>>> ml = 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(ml, 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)

Returns

Return type None

flopy.modflow.mfevt module

mfghb 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: *flopy.pakbase.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.

Returns

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, nfhbx1=0, nfhbx2=0, ifhbpt=0, bdtimecnstm=1.0, bdtime=[0.0], cnstm5=1.0, ds5=None, cnstm7=1.0, ds7=None, extension='fhb', unitnumber=None, filenames=None*)
Bases: *flopy.pakbase.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).
- **nfhbx1** (*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)
- **nfhbx2** (*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).
- **bdtimecnstm** (*float*) – A constant multiplier for data list bdtime. (default is 1.0)
- **bdtime** (*float or list of floats*) – Simulation time at which values of specified flow and (or) values of specified head will be read. nbdtim values are required. (default is 0.0)
- **cnstm5** (*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[nbdtime](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, flwra2, ...,␣
↪flwrat(nbdtime)],
    [lay, row, col, iaux, flwrat1, flwra2, ...,␣
↪flwrat(nbdtime)],
    [lay, row, col, iaux, flwrat1, flwra2, ...,␣
↪flwrat(nbdtime)],
    [lay, row, col, iaux, flwrat1, flwra2, ..., flwrat(nbdtime)]
]
```

- **cnstm7** (*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[nbdtime](float). There should be nhed 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(nbdtime)],
    [lay, row, col, iaux, sbhed1, sbhed2, ..., sbhed(nbdtime)],
    [lay, row, col, iaux, sbhed1, sbhed2, ..., sbhed(nbdtime)],
    [lay, row, col, iaux, sbhed1, sbhed2, ..., sbhed(nbdtime)]
]
```

- **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.

Returns

Return type None

flopy.modflow.mfflwob module

class ModflowFlwob(*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: *flopy.pakbase.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

Returns

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: *flopy.pakbase.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.parsenamefile*.

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.

Returns

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: *flopy.pakbase.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:
    [
    [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
>>> ml = flopy.modflow.Modflow()
>>> lrcsc = {0:[2, 3, 4, 10., 100.]}  #this ghb will be applied to all
>>>                                   #stress periods
>>> ghb = flopy.modflow.ModflowGhb(ml, 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.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 **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)

Returns

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: *flopy.pakbase.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 gmg 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.
- **ioutgmg** (*int*) – is a flag that controls the output of the gmg solver. The possible values of ioutgmg are.

  If ioutgmg = 0, then only the solver inputs are printed.

  If ioutgmg = 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 gmg calls, pcg iterations, and a running total of pcg iterations for all time/stress periods are printed.

  If ioutgmg = 2, then the convergence history of the pcg iteration is printed, showing the l2norm of the residual and the convergence factor for each iteration.

  ioutgmg = 3 is the same as ioutgmg = 1 except output is sent to the terminal instead of the modflow list output file.

  ioutgmg = 4 is the same as ioutgmg = 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 ector 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 gmg 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 x 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.

Returns

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.

Returns

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: *flopy.pakbase.Package*

MODFLOW HFB6 - Horizontal Flow Barrier Package

Parameters

- **model** (*model object*) – The model object (of type: class:*flopy.modflow.mf.Modflow*) 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). 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],
           ].
```

- **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.

Returns

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*, *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.
- **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_dataDefault 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*, *hobdry=0*, *tomulth=1.0*, *obs_data=None*, *hobname=None*, *extension='hob'*, *no_print=False*, *options=None*, *unitnumber=None*, *filenames=None*)

Bases: *flopy.pakbase.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).

- **hobdry** (*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, hobdry=-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

Returns

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: *flopy.pakbase.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.

Returns

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: *flopy.pakbase.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 (NSSITER, 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.

- **sscncr** (*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 (L3 /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.parsenamefile*.

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.

Returns

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: *flopy.pakbase.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.

Returns

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*)

Bases: *flopy.pakbase.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 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 STRTBOT 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.

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)

Returns

Return type None

flopy.modflow.mfmlt module

mfmlt 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: *flopy.pakbase.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()
>>> mltdict = 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.

Returns

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*, *wel1_bynode_qsum=None*, *losstype='skin'*, *stress_period_data=None*, *dtype=None*, *mnwname=None*, *extension='mnw1'*, *unitnumber=None*, *filenames=None*)
Bases: *flopy.pakbase.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
- **wel1_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
>>> ml = flopy.modflow.Modflow()
>>> mnw1 = flopy.modflow.ModflowMnw1(ml, ...)
```

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.

Returns

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 <= P <= 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 pumplay) 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

Returns

Return type None

by_node_variables = ['k', 'i', 'j', 'ztop', 'zbotm', 'rw', 'rskin', 'kskin', 'B', 'C',

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.

Returns

Return type None

static sort_node_data(*node_data*)

class ModflowMnw2(*model*, *mnwmax=0*, *nodtot=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: *flopy.pakbase.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.
- **nodtot** (*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
>>> ml = flopy.modflow.Modflow()
>>> mnw2 = flopy.modflow.ModflowMnw2(ml, ...)
```

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

Returns

Return type None

make_node_data(*mnwobjs*)

Make node_data recarray from Mnw objects

Parameters **mnwobjs** (*Mnw object*) –

Returns

Return type None

make_stress_period_data(*mnwobjs*)

Make stress_period_data recarray from Mnw objects

Parameters **mnwobjs** (*Mnw object*) –

Returns

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** –

Returns

Return type None

flopy.modflow.mfmnwi module

class ModflowMnwi(*model*, *wel1flag=None*, *qsumflag=None*, *byndflag=None*, *mnwobs=1*, *wellid_unit_qndflag_qhbflag_concflag=None*, *extension='mnwi'*, *unitnumber=None*, *filenames=None*)
Bases: *flopy.pakbase.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.
- **wel1flag** (*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
>>> ml = flopy.modflow.Modflow()
>>> ghb = flopy.modflow.ModflowMnwi(ml, ...)
```

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.

Returns

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.

Returns

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*, *hclosexmd=0.0001*, *mxiterxmd=50*, *extension='nwt'*, *unitnumber=None*, *filenames=None*)
Bases: *flopy.pakbase.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 underrelaxation) 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.

Returns

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: *flopy.pakbase.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.

Returns

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 **iubud** – 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 ot 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.
- **iddnun** (*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 of 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 nnlay 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.

Returns

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.

Returns

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_ehead=None*, *cosines=None*, *extension='pbc'*, *unitnumber=None*, *zerobase=True*)

Bases: *flopy.pakbase.Package*

Periodic boundary condition class

write_file()

Write the package file.

Returns

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*, *dampt=1.0*, *ihcofadd=0*, *extension='pcg'*, *unitnumber=None*, *filenames=None*)

Bases: *flopy.pakbase.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.)

- **dampt** (*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.

Returns

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: *flopy.pakbase.Package*

MODFLOW Pcgn 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 < damp_lb < damp. (default is 0.001)
- **rate_d** (*float*) – is a rate parameter; generally, 0 < 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 < 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 ffected; 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.

Returns

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*, *dampt=1.0*, *relax=0.97*, *ifill=0*, *droptol=0.0*, *hclose=0.001*, *rclose=0.1*, *l2norm=None*, *iprpks=0*, *mutpks=3*, *mpi=False*, *partopt=0*, *novlapimpsol=1*, *stenimpsol=2*, *verbose=0*, *partdata=None*, *extension='pks'*, *unitnumber=None*, *filenames=None*)

Bases: *flopy.pakbase.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.)
- **dampt** (*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.

Returns

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: *flopy.pakbase.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.

Returns

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: *flopy.pakbase.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 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.
- **RTmin** (*float*) – Minimum product of recharge and transmissivity. Default is 2e-8
- **RTmax** (*float*) – Maximum product of recharge and transmissivity. Default is 2e-4

Returns

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)

Returns

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: *flopy.pakbase.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]
```

(continues on next page)

(continued from previous page)

```
    ],
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.

Returns

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)

Returns

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: *flopy.pakbase.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 isfropt > 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 isfropt > 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 isfropt > 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.
- **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
>>> ml = flopy.modflow.Modflow()
>>> sfr2 = flopy.modflow.ModflowSfr2(ml, ...)
```

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 = strtop - 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.

Returns

Return type None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow.load('model.nam')
>>> m.sfr2.check()
```

const

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

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

Returns

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 strtop (streambed top) and rchlen (reach length) columns of reach_data. The slope for a reach n is computed as strtop(n+1) - strtop(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*)

graph
Dictionary of routing connections between segments.

heading = '# Streamflow-Routing (SFR2) file for MODFLOW, generated by Flopy'

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.

nper

nsfrpar = 0

nss

nstrm

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.

Returns

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: *flopy.pakbase.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
>>> ml = flopy.modflow.Modflow()
>>> sip = flopy.modflow.ModflowSip(ml, 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.

Returns

Return type None

flopy.modflow.mfsms module

mfsms module. This is the solver for MODFLOW-USG. Contains the ModflowSms class. Note that the user can access the ModflowSms class as *flopy.modflow.ModflowSms*.

class ModflowSms(*model*, *hclose=0.0001*, *hiclose=0.0001*, *mxiter=100*, *iter1=20*, *iprsms=2*, *nonlinmeth=0*, *linmeth=2*, *theta=0.7*, *akappa=0.1*, *gamma=0.2*, *amomentum=0.001*, *numtrack=20*, *btol=10000.0*, *breduc=0.2*, *reslim=100.0*, *iacl=2*, *norder=0*, *level=7*, *north=2*, *iredsys=0*, *rrctol=0.0*, *idroptol=0*, *epsrn=0.001*, *clin='bcgs'*, *ipc=3*, *iscl=0*, *iord=0*, *rclosepcgu=0.1*, *relaxpcgu=1.0*, *extension='sms'*, *options=None*, *unitnumber=None*, *filenames=None*)

Bases: *flopy.pakbase.Package*

MODFLOW Sms Package Class.

Parameters

- **model** (*model object*) – The model object (of type *flopy.modflow.mf.Modflow*) to which this package will be added.
- **hclose** (*float*) – is 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 HCLOSE, iteration stops. Commonly, HCLOSE equals 0.01.
- **hiclose** (*float*) – is 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 HICLOSE, the matrix solver assumes convergence. Commonly, HICLOSE is set an order of magnitude less than HCLOSE.
- **mxiter** (*int*) – is the maximum number of outer (nonlinear) iterations – that is, calls to the solution routine. For a linear problem MXITER should be 1.
- **iter1** (*int*) – is the maximum number of inner (linear) iterations. The number typically depends on the characteristics of the matrix solution scheme being used. For nonlinear problems, ITER1 usually ranges from 60 to 600; a value of 100 will be sufficient for most linear problems.
- **iprsms** (*int*) – is a flag that controls printing of convergence information from the solver: 0 is print nothing; 1 is print only the total number of iterations and nonlinear residual reduction summaries; 2 is print matrix solver information in addition to above.
- **nonlinmeth** (*int*) – is a flag that controls the nonlinear solution method and under-relaxation schemes. 0 is Picard iteration scheme is used without any under-relaxation schemes involved. > 0 is Newton-Raphson iteration scheme is used with under-relaxation. Note that the Newton-Raphson linearization scheme is available only for the upstream weighted solution scheme of the BCF and LPF packages. < 0 is Picard iteration scheme is used with under-relaxation. The absolute value of NONLINMETH determines the underrelaxation scheme used. 1 or -1, then Delta-Bar-Delta under-relaxation is used. 2 or -2 then Cooley under-relaxation scheme is used. Note that the under-relaxation schemes are used in conjunction with gradient based methods, however, experience has indicated that the Cooley under-relaxation and damping work well also for the Picard scheme with the wet/dry options of MODFLOW.
- **linmeth** (*int*) – is a flag that controls the matrix solution method. 1 is the XMD solver of Ibaraki (2005). 2 is the unstructured pre-conditioned conjugate gradient solver of White and Hughes (2011).
- **theta** (*float*) – is the reduction factor for the learning rate (under-relaxation term) of the delta-bar-delta algorithm. The value of THETA is between zero and one. If the change in the variable (head) is of opposite sign to that of the previous iteration, the under-relaxation term is reduced by a factor of THETA. The value usually ranges from 0.3 to 0.9; a value of 0.7 works well for most problems.

- **akappa** (*float*) – is the increment for the learning rate (under-relaxation term) of the delta-bar-delta algorithm. The value of AKAPPA is between zero and one. If the change in the variable (head) is of the same sign to that of the previous iteration, the under-relaxation term is increased by an increment of AKAPPA. The value usually ranges from 0.03 to 0.3; a value of 0.1 works well for most problems.
- **gamma** (*float*) – is the history or memory term factor of the delta-bar-delta algorithm. Gamma is between zero and 1 but cannot be equal to one. When GAMMA is zero, only the most recent history (previous iteration value) is maintained. As 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.
- **amomentum** (*float*) – is the fraction of past history changes that is added as a momentum term to the step change for a nonlinear iteration. The value of AMOMENTUM 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.
- **numtrack** (*int*) – is the maximum number of backtracking iterations allowed for residual reduction computations. If NUMTRACK = 0 then the backtracking iterations are omitted. The value usually ranges from 2 to 20; a value of 10 works well for most problems.
- **numtrack** – is the maximum number of backtracking iterations allowed for residual reduction computations. If NUMTRACK = 0 then the backtracking iterations are omitted. The value usually ranges from 2 to 20; a value of 10 works well for most problems.
- **btol** (*float*) – is the tolerance for residual change that is allowed for residual reduction computations. BTOL 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 1e6 ; a value of 1e4 works well for most problems but lower values like 1.1 may be required for harder problems.
- **breduce** (*float*) – is the reduction in step size used for residual reduction computations. The value of BREDUC 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.
- **reslim** (*float*) – is the limit to which the residual is reduced with backtracking. If the residual is smaller than RESLIM, 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.
- **iacl** (*int*) – is the flag for choosing the acceleration method. 0 is Conjugate Gradient; select this option if the matrix is symmetric. 1 is ORTHOMIN. 2 is BiCGSTAB.
- **norder** (*int*) – is the flag for choosing the ordering scheme. 0 is original ordering 1 is reverse Cuthill McKee ordering 2 is Minimum degree ordering
- **level** (*int*) – is the level of fill for ILU decomposition. 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 drop tolerance.
- **north** (*int*) – is the number of orthogonalizations for the ORTHOMIN acceleration scheme. A number between 4 and 10 is appropriate. Small values require less storage

but more iteration may be required. This number should equal 2 for the other acceleration methods.

- **iredsys** (*int*) – is the index for creating a reduced system of equations using the red-black ordering scheme. 0 is do not create reduced system 1 is create reduced system using red-black ordering
- **rrctol** (*float*) – is a residual tolerance criterion for convergence. The root mean squared residual of the matrix solution is evaluated against this number to determine convergence. The solver assumes convergence if either HICLOSE (the absolute head tolerance value for the solver) or RRCTOL is achieved. Note that a value of zero ignores residual tolerance in favor of the absolute tolerance (HICLOSE) for closure of the matrix solver.
- **idroptol** (*int*) – is the flag to perform drop tolerance. 0 is do not perform drop tolerance 1 is perform drop tolerance
- **epsrn** (*float*) – is the drop tolerance value. A value of 1e-3 works well for most problems.
- **clin** (*string*) – an option keyword that defines the linear acceleration method used by the PCGU solver. CLIN is "CG", then preconditioned conjugate gradient method. CLIN is "BCGS", then preconditioned bi-conjugate gradient stabilized method.
- **ipc** (*int*) – an integer value that defines the preconditioner. IPC = 0, No preconditioning. IPC = 1, Jacobi preconditioning. IPC = 2, ILU(0) preconditioning. IPC = 3, MILU(0) preconditioning (default).
- **iscl** (*int*) – is the flag for choosing the matrix scaling approach used. 0 is no matrix scaling applied 1 is symmetric matrix scaling using the scaling method by the POLCG preconditioner in Hill (1992). 2 is symmetric matrix scaling using the l2 norm of each row of A (DR) and the l2 norm of each row of DRA.
- **iord** (*int*) – is the flag for choosing the matrix reordering approach used. 0 = original ordering 1 = reverse Cuthill McKee ordering 2 = minimum degree ordering
- **rclosepcgu** (*float*) – a real value that defines the flow residual tolerance for convergence of the PCGU linear solver. 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-USG length and time units. Usually a value of 1.0x10-1 is sufficient for the flow-residual criteria when meters and seconds are the defined MODFLOW-USG length and time.
- **relaxpcgu** (*float*) – a real value that defines the relaxation factor used by the MILU(0) preconditioner. RELAXPCGU is unitless and should be greater than or equal to 0.0 and less than or equal to 1.0. RELAXPCGU values of about 1.0 are commonly used, and experience suggests that convergence can be optimized in some cases with RELAXPCGU values of 0.97. A RELAXPCGU value of 0.0 will result in ILU(0) preconditioning. RELAXPCGU is only specified if IPC=3. If RELAXPCGU is not specified and IPC=3, then a default value of 0.97 will be assigned to RELAXPCGU.
- **extension** (*str, optional*) – File extension (default is 'sms'.
- **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 will be set to the string. Default is None.

Notes

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> sms = flopy.modflow.ModflowSms(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 **sms**

Return type ModflowSms object

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> sms = flopy.modflow.ModflowPcg.load('test.sms', m)
```

write_file()

Write the package file.

Returns

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: *flopy.pakbase.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
>>> ml = flopy.modflow.Modflow()
>>> sor = flopy.modflow.ModflowSor(ml)
```

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 **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.

Returns

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: *flopy.pakbase.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]]
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

- **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 xyx']. 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.

Returns

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: *flopy.pakbase.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 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.

Returns

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, nobs=0, iswizt=None, ipakcb=None, iswiobs=0, options=None, nsolver=1, iprsol=0, mutsol=3, solver2params={'damp': 1.0, 'dampt': 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: *flopy.pakbase.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.)

  dampt [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 nobs observations.
- **obslrc** (*list of lists*) – zero-based [layer, row, column] lists for nobs 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)

Returns

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: *flopy.pakbase.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 ModflowSwr1 object

Notes

Load method still needs to be implemented.

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> swr = flopy.modflow.ModflowSwr1.load('test.swr', m)
```

write_file()
Write the package file.

Returns

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*, *ivoid=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: *flopy.pakbase.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).
- **nystm** (*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 (nsystm)*) – 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 (nsystm, nrow, ncol)*) – thick is an array specifying the thickness of compressible sediments. (default is 1.).
- **sse** (*float or array of floats (nsystm, 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 (nsystm, 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 (nsystm, 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 (nsystm, 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 (nsystm, nrow, ncol)*) – void is an array specifying the initial void ratio of compressible beds. (default is 0.82).
- **sub** (*float or array of floats (nsystm, 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, ifl9, 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.

Returns

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: *flopy.pakbase.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 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)

Returns

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: *flopy.pakbase.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 groundwater 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., uzf1 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 uzgage)

Notes

Parameters are not supported in FloPy.

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow()
>>> uzf = flopy.modflow.ModflowUzf1(ml, ...)
```

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)
```

nuzgag

Number of uzf gages

Returns **nuzgag** – Number of uzf gages

Return type int

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.

Returns

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*)

Bases: *flopy.pakbase.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.

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)
```

phiramp_unit
Get phiramp unit

Returns **iunitramp** – unit number of phiramp file

Return type int

write_file (*f=None*)
Write the package file.

Parameters **f** – (str) optional file name

Returns

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: *flopy.pakbase.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.

Returns

Return type None

Notes

Not implemented because parameters are only supported on load

MT3DMS Packages

Contents:

flopy.mt3d.mt module

class Mt3dList(*model*, *extension='list'*, *listunit=7*)
Bases: *flopy.pakbase.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.exe'*, *structured=True*, *listunit=16*, *ftlunit=10*, *model_ws='.'*, *external_path=None*, *verbose=False*, *load=True*, *silent=0*)
Bases: *flopy.mbase.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.exe"*) – 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.exe'*, *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.exe"*) – 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.

Returns

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*)

mcomp

modelgrid

modeltime

ncol

ncomp

nlay

nper

nrow

nrow_ncol_nlay_nper

solver_tols

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*, *nph=40*, *npmin=5*, *npmax=80*, *nlsink=0*, *npsink=15*, *dchmoc=0.0001*, *extension='adv'*, *unitnumber=None*, *filenames=None*)
Bases: *flopy.pakbase.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*) – s 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 con-

trolling 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.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 **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

Returns

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: *flopy.pakbase.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.

- **prsity** (*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.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 **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

Returns

Return type None

flopy.mt3d.mtcts module

class Mt3dCts
Bases: *flopy.pakbase.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.
- **itrtinj** (*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*, *unitnumber=None*, *filenames=None*, ***kwargs*)
Bases: *flopy.pakbase.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)*) – s 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)
- **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

Returns

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: *flopy.pakbase.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.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 **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

Returns

Return type None

flopy.mt3d.mtlkt module

class Mt3dLkt(*model*, *nlkinit=0*, *mxlkbc=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: *flopy.pakbase.Package*

MT3D-USGS LaKe Transport package class

Parameters

- **model** (*model object*) – The model object (of type `flopy.mt3dms.mt.Mt3dms`) to which this package will be added.
- **nlkinit** (*int*) – is equal to the number of simulated lakes as specified in the flow simulation
- **mxlkbc** (*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 .lkcobs.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.mt.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

Returns

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: *flopy.pakbase.Package*

PHC package class for PHT3D

unitnumber = 38

write_file()
Write the package file

Returns

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, rc1=None, rc2=None, extension='rct', unitnumber=None, filenames=None, **kwargs*)
Bases: *flopy.pakbase.Package*

Chemical reaction package class.

Parameters

- **model** (*model object*) – The model object (of type `flopy.mt3dms.mt.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)
- **prsity2** (*float or array of floats (nlay, nrow, ncol)*) – prsity2 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. prsity2 is used if isothm = 5 or 6. If prsity2 is not user-specified and isothm = 5 or 6 then prsity2 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).

Other Parameters

- **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 srconc for species n is set to 0. See description of srconc 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 nlay 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.parsenamefile*.

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

Returns

Return type None

flopy.mt3d.mtsft module

class Mt3dSft(*model*, *nsfinit=0*, *mxsfbc=0*, *icbcsf=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: *flopy.pakbase.Package*

MT3D-USGS StreamFlow Transport package class

Parameters

- **model** (*model object*) – The model object (of type `flopy.mt3dms.mt.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.
- **icbcsf** (*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.mt.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
- **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 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

Returns

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: *flopy.pakbase.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

Returns

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: *flopy.pakbase.Package*

Transport Observation package class

write_file()
Write the package file

Returns

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: *flopy.pakbase.Package*

MT3D-USGS Unsaturated-Zone Transport package class

Parameters

- **model** (*model object*) – The model object (of type flopy.mt3dms.mt.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.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 **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

Returns

Return type None

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: *flopy.mbase.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*) – Location of new model workspace. If this path does not exist, it will be created. (default is None, which will be assigned to 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*, *version='seawat'*, *exe_name='swtv4'*, *verbose=False*, *model_ws='.'*, *load_only=None*)

Load an existing model.

Parameters

- **f** (*str*) – Path to 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, default "."*) – Model workspace. Directory name 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.

Returns

Return type *flopy.seawat.swt.Seawat*

Examples

```
>>> import flopy
>>> m = flopy.seawat.swt.Seawat.load(f)
```

mcomp

modelgrid

modeltime

ncol

ncomp

nlay

nper

nrow

nrow_ncol_nlay_nper

write_name_file()

Write the name file

Returns

Return type None

class SeawatList(*model*, *extension='list'*, *listunit=7*)

Bases: *flopy.pakbase.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*, *iwtable=1*, *densemin=0*, *densemax=0*, *dnscrit=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: *flopy.pakbase.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.
- **(or mt3drhoflg)** (*mtdnconc*) – 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.
- **mfnadvfd** (*int*) – is a flag that determines the method for calculating the internodal density values used to conserve fluid mass. If MFNADVFD = 2, internodal conductance values used to conserve fluid mass are calculated using a central-in-space algorithm. If MFNADVFD <> 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.
- **densemax** (*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 crite- rion, 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 x 10-3 kg/m4. 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 condi- tions 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.parsenamefile*.

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

Returns

Return type None

flopy.seawat.swtvsc module

class SeawatVsc(*model*, *mt3dmuflg=-1*, *viscmin=0.0*, *viscmax=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: *flopy.pakbase.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.
- **(or mt3drhoflg)** (*mt3dmuflg*) – 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.
- **viscmin** (*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.
- **viscmax** (*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.
- **viscref** (*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

Returns

Return type None

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.exe'*, *flowmodel=None*, *headfilename=None*, *budgetfilename=None*, *model_ws=None*, *verbose=False*)

Bases: *flopy.mbase.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.exe"*) – 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)
```

hdry

hnoflo

laytyp

write_name_file()
Write the name file

Returns

Return type None

class Modpath7List(*model*, *extension='list'*, *unitnumber=None*)
Bases: *flopy.pakbase.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: *flopy.pakbase.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)

Returns

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 CellDataTypes 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 subdivision data 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 CellDataTypes 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 subdivision data 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*, *localy=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 True).
- **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: flopy.modpath.mp7particlegroup._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: flopy.modpath.mp7particlegroup._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: flopy.modpath.mp7particlegroup._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: *flopy.pakbase.Package*

MODPATH Simulation File Package Class.

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 os '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 'time-series' 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)

Returns

Return type None

class budgetOpt
Bases: enum.Enum

Enumeration of different budget output options

no = 0

record_summary = 2

summary = 1

class onoffOpt
Bases: enum.Enum

Enumeration of on-off options

off = 1

on = 2

class simType
Bases: enum.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
Bases: enum.Enum

Enumeration of different stop time options

extend = 2

specified = 3

total = 1

class trackDir
Bases: enum.Enum

Enumeration of different tracking directions

backward = 2

forward = 1

class weakOpt
Bases: enum.Enum

Enumeration of different weak sink and source options

pass_through = 1

stop_at = 2

MODPATH 6 Packages

Contents:

flopy.modpath.mp6 module

class Modpath6(*modelname='modpathtest'*, *simfile_ext='mpsim'*, *namefile_ext='mpnam'*, *version='modpath'*, *exe_name='mp6.exe'*, *modflowmodel=None*, *dis_file=None*, *dis_unit=87*, *head_file=None*, *budget_file=None*, *model_ws=None*, *external_path=None*, *verbose=False*, *load=True*, *listunit=7*)
Bases: *flopy.mbase.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, default "mp6.exe"*) – 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*) – Required dis file name.
- **dis_unit** (*int, default 87*) – Optional dis file unit number.
- **head_file** (*str*) – Required filename of the MODFLOW output head file.
- **budget_file** (*str*) – Required filename of the MODFLOW output cell-by-cell budget file.
- **model_ws** (*str, optional*) – Model workspace. Directory name to create model data sets. Default is the current 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.
- **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** (*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()

mf

next_ext_unit()

Function to encapsulate next_ext_unit attribute

sim

write_name_file()

Write the name file

Returns

Return type None

class Modpath6List(*model*, *extension='list'*, *listunit=7*)

Bases: *flopy.pakbase.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=0*, *ibound=1*, *prsity=0.3*, *prsityCB=0.3*, *extension='mpbas'*, *unitnumber=86*)

Bases: *flopy.pakbase.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** (*int or list of ints*) – MODFLOW layer type (0 is convertible, 1 is confined).
- **ibound** (*array of ints, optional*) – The ibound array (the default is 1).
- **prsity** (*array of ints, optional*) – The porosity array (the default is 0.30).
- **prsityCB** (*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

Returns

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: *flopy.pakbase.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 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.

Returns

Return type None

Examples

write_file()
Write the package file

Returns

Return type None

class StartingLocationsFile(*model*, *inputstyle=1*, *extension='loc'*, *verbose=False*)
Bases: *flopy.pakbase.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')

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

7.1.3 Flopy Utilities

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: *flopy.utils.datafile.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*, *precision*, *verbose*, *kwargs*)
Bases: *flopy.utils.datafile.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 <= k < nlay; 0 <= i < nrow; 0 <= j < ncol

Examples

exception BudgetIndexError
Bases: Exception

class CellBudgetFile (*filename*, *precision='auto'*, *verbose=False*, ***kwargs*)
Bases: object

CellBudgetFile Class.

Parameters

- **filename** (*string*) – Name 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(kstpkper=(0,0), text='RIVER LEAKAGE')
```

close()
Close the file handle

create3D(*data*, *nlay*, *nrow*, *ncol*)
Convert a dictionary of {node: q, . . . } into a numpy masked array. In most cases this should not be called directly by the user unless you know what you're doing. Instead, it is used as part of the full3D keyword for get_data.

Parameters

- **data** (*dictionary*) – Dictionary with node keywords and flows (q) items.
- **nrow, ncol** (*nlay,*) – Number of layers, rows, and columns of the model grid.

Returns **out** – List contains unique simulation times (totim) in binary file.

Return type numpy masked array

get_data(*idx=None*, *kstpkper=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.
- **kstpkper** (*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 <= k < nlay; 0 <= i < nrow; 0 <= j < 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*, *text='head'*, *precision='auto'*, *verbose=False*, ***kwargs*)
Bases: *flopy.utils.binaryfile.BinaryLayerFile*

HeadFile Class.

Parameters

- **filename** (*string*) – Name 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
>>> hdobj = bf.HeadFile('model.hds', precision='single')
>>> hdobj.list_records()
>>> rec = hdobj.get_data(kstpkper=(1, 50))
```

```
>>> ddnobj = bf.HeadFile('model.ddn', text='drawdown', precision='single')
>>> ddnobj.list_records()
>>> rec = ddnobj.get_data(totim=100.)
```

class HeadUFile(*filename*, *text='headu'*, *precision='auto'*, *verbose=False*, ***kwargs*)
Bases: *flopy.utils.binaryfile.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
>>> hdobj = bf.HeadUFile('model.hds')
>>> hdobj.list_records()
>>> usgheads = hdobj.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 (not implemented).

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

Examples

class UcnFile(*filename*, *text='concentration'*, *precision='auto'*, *verbose=False*, ***kwargs*)

Bases: *flopy.utils.binaryfile.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
>>> ucnobj = bf.UcnFile('MT3D001.UCN', precision='single')
>>> ucnobj.list_records()
>>> rec = ucnobj.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*)

Determine precision of a MODFLOW head file.

Parameters

- **filename** (*str*) –
- **of binary MODFLOW file to determine precision.** (*Name*) –

Returns

- **result** (*str*)
- *Result will be unknown, single, or double*

join_struct_arrays (*arrays*)

Simple function that can join two numpy structured arrays.

flopy.utils.binarygrid_util module

Deprecated version of module to read MODFLOW 6 binary grid files (*.grb) that define the model grid binary output files. This function imports the current version of MfGrdFile from ..mf6.utils and returns the instantiated object. This version is deprecated and will be removed.

MfGrdFile(*filename*, *precision='double'*, *verbose=False*)

The deprecated 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 the screen. Default is False.

Returns **mfg** – MfGrdFile object instantiated using flopy.mf6.utils.MfGrdFile

Return type *MfGrdFile*

flopy.utils.check module

class check(*package*, *f=None*, *verbose=True*, *level=1*, *property_threshold_values={}*)

Bases: object

Check package for common errors

Parameters

- **package** (*object*) – Instance of Package 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.
- **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*) –
- **grid; for instance, a Util3d array** (*unstructured*) –
- **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, 10000

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', 'SI

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()

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. http://stackoverflow.com/questions/15182381/how-to-return-a-view-of- several-columns-in-numpy-structured-array

class mf6check(*package*, *f=None*, *verbose=True*, *level=1*, *property_threshold_values={}*)
Bases: *flopy.utils.check.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.cvfdutil module

class Point(*x*, *y*)
Bases: object

area_of_polygon(*x*, *y*)
Calculates the signed area of an arbitrary polygon given its vertices http://stackoverflow.com/a/4682656/190597 (Joe Kington) http://softsurfer.com/Archive/algorithm_0101/algorithm_0101.htm#2D%20Polygons

centroid_of_polygon(*points*)
http://stackoverflow.com/a/14115494/190597 (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 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 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*, *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*, *kstpkper=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)
- **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** (*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')

Returns

Return type None

Notes

Examples

```
>>> import flopy
>>> hdobj = flopy.utils.HeadFile('test.hds')
>>> times = hdobj.get_times()
>>> hdobj.plot(totim=times[-1])
```

to_shapefile(*filename*, *kstpkper=None*, *totim=None*, *mflay=None*, *attrib_name='lf_data'*)

Export model output data to a shapefile at a specific location in LayerFile instance.

Parameters

- **filename** (*str*) – Shapefile name to write
- **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 layer 1 will be written
- **attrib_name** (*str*) – Base name of attribute columns. (default is 'lf_data')

Returns

Return type None

Notes

Examples

```
>>> import flopy
>>> hdobj = flopy.utils.HeadFile('test.hds')
>>> times = hdobj.get_times()
>>> hdobj.to_shapefile('test_heads_sp6.shp', totim=times[-1])
```

flopy.utils.datautil module

class ArrayIndexIter(*array_shape*, *index_as_tuple=False*)
Bases: object

next()

class ConstIter(*value*)
Bases: object

next()

class DatumUtil
Bases: object

static is_basic_type(*obj*)

static is_float(*str*)

static is_int(*str*)

class FileIter(*file_path*)
Bases: object

close()

next()

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

next()

class NameIter(*name*, *first_not_numbered=True*)
Bases: object

next()

class PathIter(*path*, *first_not_numbered=True*)
Bases: object

next()

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 releated 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':

quote_list = {'"', "'"}

static reset_delimiter_used()

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_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*) –

Returns

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>"

Returns

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 http://stackoverflow.com/questions/18259393/numpy-loading-csv-too-slow-compared-to-matlab)

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

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: *flopy.utils.formattedfile.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
>>> hdobj = ff.FormattedHeadFile('model.fhd', precision='single')
>>> hdobj.list_records()
>>> rec = hdobj.get_data(kstpkper=(1, 50))
>>> rec2 = ddnobj.get_data(totim=100.)
```

class FormattedHeader(*text_ident*, *precision='single'*)

Bases: *flopy.utils.datafile.Header*

The TextHeader class is a class to read in headers from MODFLOW formatted files.

Parameters

- **is the text string in the header that identifies the type** (*text_ident*) –
- **data (eg. 'head') precision is the precision of the floating point** (*of*) –
- **in the file** (*data*) –

read_header(*text_file*)

Read header information from a formatted file

Parameters

- **is an open file object currently at the beginning of** (*text_file*) –
- **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: *flopy.utils.datafile.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 <= k < nlay; 0 <= i < nrow; 0 <= 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

bounds
Method to calculate the bounding box of the collection

Returns

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

Returns

Return type matplotlib.axes object

class LineString(*coordinates*)
Bases: *flopy.utils.geometry.Shape*

bounds

has_z = False

plot(*ax=None*, ***kwargs*)

pyshp_parts

shapeType = 3

type = 'LineString'

x

y

z

class MultiLineString(*linestrings=()*)
Bases: *flopy.utils.geometry.Collection*

Container for housing and describing multilinestring geometries (e.g. to be read or written to shapefiles or other geographic data formats)

polygons [list] list of flopy.utils.geometry.LineString objects

class MultiPoint(*points=()*)
Bases: *flopy.utils.geometry.Collection*

Container for housing and describing multipoint geometries (e.g. to be read or written to shapefiles or other geographic data formats)

polygons [list] list of flopy.utils.geometry.Point objects

class MultiPolygon(*polygons=()*)
Bases: *flopy.utils.geometry.Collection*
Container for housing and describing multipolygon geometries (e.g. to be read or written to shapefiles or other geographic data formats)

polygons [list] list of flopy.utils.geometry.Polygon objects

class Point(**coordinates*)
Bases: *flopy.utils.geometry.Shape*

bounds

has_z = False

plot(*ax=None*, ***kwargs*)

pyshp_parts

shapeType = 1

type = 'Point'

x

y

z

class Polygon(*exterior*, *interiors=None*)
Bases: *flopy.utils.geometry.Shape*

bounds

get_patch(***kwargs*)

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).:

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
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

Returns

Return type *Polygon*, *LineString*, or *Point*

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(xl) 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.

shape(*pyshp_shpobj*)

Convert a pyshp geometry object to a flopy geometry object.

Parameters **pyshp_shpobj** (*shapefile._Shape instance*) –

Returns shape

Return type flopy.utils.geometry Polygon, Linestring, or *Point*

Notes

Currently only regular Polygons, LineStrings and Points (pyshp types 5, 3, 1) supported.

Examples

```
>>> import shapefile as sf
>>> from flopy.utils.geometry import shape
>>> sfobj = sf.Reader('shapefile.shp')
>>> flopy_geom = shape(list(sfobj.iterShapes())[0])
```

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 : shapefile name 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], ...]

- **shapetype** (*list*) – optional list of shapetypes that is required when vertices are supplied to the class as the obj parameter

flopy_geometry

Property that returns a flopy.util.geometry.Collection object

Returns

Return type flopy.util.geometry.Collectionnos object

geojson

Property that returns a geojson.GeometryCollection object to the user

Returns

Return type geojson.GeometryCollection

points

Property returns a multidimensional list of vertices

Returns

Return type list of vertices

shape
Property that returns a shapefile.Shapes object

Returns

Return type shapefile.Shapes object

shapely
Property that returns a shapely.geometry.collection.GeometryCollection object to the user

Returns

Return type shapely.geometry.collection.GeometryCollection object

shapetype
Returns a list of shapetypes to the user

Returns

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

flopy_geometry
Returns a flopy geometry object to the user

Returns

Return type flopy.utils.geometry.<Shape>

geojson
Returns a geojson object to the user

Returns

Return type geojson.<shape>

points
Returns a list of vertices to the user

Returns

Return type list

shape
Returns a shapefile.Shape object to the user

Returns

Return type shapefile.shape

shapely
Returns a shapely.geometry object to the user

Returns

Return type shapely.geometry.<shape>

shapetype
Shapetype string for a geometry

Returns

Return type str

flopy.utils.gridgen module

class Gridgen(*dis*, *model_ws='.'*, *exe_name='gridgen'*, *surface_interpolation='replicate'*, *vertical_pass_through=False*)
Bases: object

Class to work with the gridgen program to create layered quadtree grids.

Parameters

- **dis** (*flopy.modflow.ModflowDis*) – Flopy discretization object
- **model_ws** (*str*) – 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)

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.

Returns

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.

Returns

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)

Returns

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)

Returns

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(*nodenumber*)
Return the cell center x and y coordinates

Parameters **nodenumber** –

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 will 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()

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_recarray()
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(*nodenumber*)
Return a list of 5 vertices for the cell. The first vertex should be the same as the last vertex.

Parameters **nodenumber** –

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.

Returns

Return type None

to_disu6(*fname*, *writevertices=True*)

to_disv6(*fname*, *verbose=False*)

features_to_shapefile(*features*, *featuretype*, *filename*)

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** (*string*) – name of the shapefile to write

Returns

Return type None

get_ia_from_iac(*iac*)

get_isym(*ia*, *ja*)

is_symmetrical(*isym*, *a*, *atol=0*)

ndarray_to_asciigrid(*fname*, *a*, *extent*, *nodata=1e+30*)

read1d(*f*, *a*)

Quick file to array reader for reading gridgen output. Much faster than the read1d 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 shapes (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. It is possible to perform intersects without building the STR-tree by setting *rtree=False*.
- The optimized routines for structured grids will often outperform the shapely routines 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.

static filter_query_result(*qresult*, *shp*)

Filter query result to obtain grid cells that intersect with shape. Used to (further) reduce query result to cells that definitely intersect with shape.

Parameters

- **qresult** (*iterable*) – query result, iterable of polygons
- **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 qfiltered

intersect(*shp*, ***kwargs*)

Method to intersect a shape with a model grid

Parameters

- **shp** (*shapely.geometry, geojson object, shapefile.Shape,*) – or flopy geomerty object
- **sort_by_cellid** (*bool*) – Sort results by cellid
- **keepzerolengths** (*bool*) – boolean method to keep zero length intersections for linestring intersection

Returns a record array containing information about the intersection

Return type numpy.recarray

intersects(*shp*)

Return cellIDs for shapes that intersect with shape.

Parameters **shp** (*shapely.geometry, geojson geometry, shapefile.shape,*) – or flopy geometry object shape to intersect with the grid

Returns **rec** – a record array containing cell IDs of the gridcells the shape intersects with

Return type numpy.recarray

static plot_linestring(*rec*, *ax=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
- ****kwargs** – passed to the plot function

Returns **ax** – 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 **ax** – 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 **ax** – 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 list or generator containing grid cells in query result

Return type list or generator expression

static sort_gridshapes(*shape_iter*)
Sort query result by node id.

Parameters **shape_iter** (*iterable*) – list or iterable of gridcells

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_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 shptyp.

Parameters

- **collection** (*list*) – state variable for storing result, generally an empty list
- **ix_result** (*shapely.geometry type*) – any shapely intersection result
- **shptyp** (*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 **collection** – list containing shapely geometries of type shptyp

Return type list

flopy.utils.lgrutil module

class Lgr(*nlayp*, *nrowp*, *ncolp*, *delrp*, *delcp*, *topp*, *botmp*, *idomainp*, *ncpp=3*, *ncppl=1*, *xllp=0.0*, *yllp=0.0*)

Bases: object

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()

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_in, df_out = 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(*kstpkper=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).
- **kstpkper** (*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, cumulativedf = 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 most 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()
```

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: *flopy.utils.mflistfile.ListBudget*

set_budget_key()

class MfListBudget(*file_name*, *budgetkey=None*, *timeunit='days'*)
Bases: *flopy.utils.mflistfile.ListBudget*

set_budget_key()

class MfusgListBudget(*file_name*, *budgetkey=None*, *timeunit='days'*)
Bases: *flopy.utils.mflistfile.ListBudget*

set_budget_key()

class SwrListBudget(*file_name*, *budgetkey=None*, *timeunit='days'*)
Bases: *flopy.utils.mflistfile.ListBudget*

set_budget_key()

class SwtListBudget(*file_name*, *budgetkey=None*, *timeunit='days'*)
Bases: *flopy.utils.mflistfile.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

attribs_from_namfile_header(*namefile*)

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

- **namefilename** (*str*) – 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. Prior to Python version 3.6 the return object is an OrderedDict to retain the order of items in the nam file.

Return type dict or OrderedDict

Raises

- IOError: – 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 is 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', 'particleg

write_shapefile(*endpoint_data=None*, *shpname='endpoints.shp'*, *direction='ending'*, *mg=None*, *epsg=None*, *sr=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 epgs code associated with mg will be used.

kwargs : keyword arguments to flopy.export.shapefile_utils.recarray2shp

class PathlineFile(*filename*, *verbose=False*)

Bases: `flopy.utils.modpathfile._ModpathSeries`

PathlineFile Class.

Parameters

- **filename** (*string*) – Name 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_recarray=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_recarray** (*bool*) – Boolean that controls returned pthldest. If to_recarray is True, a single recarray with all of the pathlines that intersect dest_cells are returned. If to_recarray 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',

write_shapefile(*pathline_data=None*, *one_per_particle=True*, *direction='ending'*, *shpname='pathlines.shp'*, *mg=None*, *epsg=None*, *sr=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 epgs code associated with mg will be used.

kwargs : keyword arguments to flopy.export.shapefile_utils.recarray2shp

class TimeseriesFile(*filename*, *verbose=False*)

Bases: `flopy.utils.modpathfile._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', 'ti

write_shapefile(*timeseries_data=None*, *one_per_particle=True*, *direction='ending'*, *shpname='pathlines.shp'*, *mg=None*, *epsg=None*, *sr=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 epgs 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")
>>> incremental, cumulative = mt_list.get_budget()
>>> 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=', '*)

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 ","

nobs

Method to get the number of observations

Returns

Return type int

obsnames

Method to get the observation names

Returns

Return type list

static read_csv(*fobj*, *dtype*, *delimiter=', '*)

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 ","

Returns

Return type np.recarray

class HydmodObs(*filename*, *verbose=False*, *hydlbl_len=20*)

Bases: *flopy.utils.observationfile.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)

Returns

Return type None

class Mf6Obs(*filename*, *verbose=False*, *isBinary='auto'*)

Bases: *flopy.utils.observationfile.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

Returns

Return type None

class ObsFiles

Bases: *flopy.utils.utils_def.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 nobs+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 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_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: *flopy.utils.observationfile.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

Returns

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':

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 OptionUtil

Bases: object

static isfloat(*s*)

Simple method to check that a string is a valid floating point variable

Parameters s (*str*) –

Returns

Return type bool

static isint(*s*)

Simple data check method to check that a string is a valid integer

Parameters s (*str*) –

Returns

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*) –

Returns

Return type bool

flopy.utils.postprocessing module

get_extended_budget(*cbcfile*, *precision='single'*, *idx=None*, *kstpkper=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.
- **kstpkper** (*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_saturated_thickness(*heads*, *m*, *nodata*, *per_idx=None*)

Calculates the saturated thickness for each cell 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** (*float, list*) – HDRY value indicating dry cells and/or HNOFLO values.
- **per_idx** (*int or sequence of ints*) – stress periods to return. If None, returns all stress periods (default).

Returns **sat_thickness** – Array of saturated thickness

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*, *nodata*, *per_idx=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.
- **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 is None).

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.u

INT64 = (<class 'numpy.int64'>, <class 'numpy.uint64'>)

INT8 = (<class 'numpy.int8'>, <class 'numpy.uint8'>)

bands
Returns a tuple of raster bands

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

- **polygon** (*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

Returns

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*)
Static method to load a raster file into the raster object

Parameters **raster** (*str*) –

Returns

Return type Raster object

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*) – scipy interpolation methods

  `linear` for bi-linear interpolation

  `nearest` for nearest neighbor

  `cubic` for bi-cubic interpolation

  `mean` for mean sampling

  `median` for median sampling
- **multithread** (*bool*) – boolean flag indicating if multithreading should be used with the `mean` and `median` sampling methods

- **thread_pool** (*int*) – 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.

Returns

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*)
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

Returns

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

xcenters
Returns a np.ndarray of raster x cell centers

ycenters
Returns a np.ndarray of raster y cell centers

flopy.utils.recarray_utils module

create_empty_recarray(*length*, *dtype*, *default_value=0*)
Create a empty recarray with a defined default value for floats.

Parameters

- **length** (*int*) – Shape of the empty recarray.

- **dtype** (*np.dtype*) – dtype of the empty recarray.
- **default_value** (*float*) – default value to use for floats in recarray.

Returns **r** – Recarray of type dtype with shape length.

Return type np.recarray

Examples

```
>>> import numpy as np
>>> import flopy
>>> dtype = np.dtype([('x', np.float32), ('y', np.float32)])
>>> ra = flopy.utils.create_empty_recarray(10, dtype)
```

ra_slice (*ra*, *cols*)

Create a slice of a recarray

Parameters

- **ra** (*np.recarray*) – recarray 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.recarray

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: '

itmuni_values = {'days': 4, 'hours': 3, 'minutes': 2, 'seconds': 1, 'undefined': 0, 'y

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')
```

df

dtypes = {'column': <class 'int'>, 'layer': <class 'int'>, 'reach': <class 'int'>, 'ro

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 **kstpkper** – list of kstp, kper tuples

Return type tuple

flopy.utils.swroutputfile module

class SwrBudget(*filename*, *precision='double'*, *verbose=False*)

Bases: *flopy.utils.swroutputfile.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: *flopy.utils.swroutputfile.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: *flopy.utils.utils_def.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)
- **kswrkstpkper** (*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 kswrkstpkper and totim are None, will return the last entry

Examples

get_kswrkstpkper()
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, nitems). The first column in the data array will contain time (totim). nitems 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: *flopy.utils.swroutputfile.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: *flopy.utils.swroutputfile.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: *flopy.utils.swroutputfile.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

Returns

Return type None

add_hole(*hole*)

Add a point that will turn enclosing polygon into a hole

Parameters **hole** (*tuple*) – (x, y)

Returns

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

Returns

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

Returns

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)

Returns

Return type None

clean()
Remove the input and output files created by this class and by the Triangle program

Returns

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 <= n < 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()

Returns

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()

Returns

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.

Returns

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()

Returns

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()

Returns

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()

Returns

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)

fortran
fortran 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 if 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

array_free_format

binary

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()

format

fortran

free

static get_default_numpy_fmt(*dtype*)

classmethod integer()

npl

numpy

py

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: flopy.datbase.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

array

build_transient_sequence()
parse self.__value into a dict{kper:Util2d}

data_type

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.

Returns

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*)

model

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()
```

plottable

to_shapefile (*filename*)

Export transient 2D data to a shapefile (as polygons). Adds an attribute for each unique Util2d instance in self.data

Parameters **filename** (*str*) – Shapefile name to write

Returns

Return type None

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.rch.rech.as_shapefile('test_rech.shp')
```

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: flopy.datbase.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

array

build_transient_sequence()
parse self.__value into a dict{kper:Util3d}

data_type

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*)

model

name

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: `flopy.datbase.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","openclose"

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()

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.

cnstnt_str

data_type

dtype

export(*f*, ***kwargs*)

filename

format

get_constant_cr(*value*)

get_external_cr()

get_file_entry(*how=None*)

get_internal_cr()

get_openclose_cr()

get_value()

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.

Returns

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*) –

Returns

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.

Returns

Return type 1-D or 2-D array

model

model_file_path
where the model expects the file to be

Returns file_path (str)

Return type path relative to the name file

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()
```

plottable

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*)

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()

to_shapefile(*filename*)
Export 2-D model data to a shapefile (as polygons) of self.array

Parameters **filename** (*str*) – Shapefile name to write

Returns

Return type None

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.dis.top.as_shapefile('test_top.shp')
```

unique()

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: flopy.datbase.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 defaut 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

array
Return a numpy array of the 3D shape. If an unstructured model, then return an array of size nodes.

build_2d_instances()

data_type

dtype

export(*f*, ***kwargs*)

get_file_entry()

get_value()

classmethod load(*f_handle*, *model*, *shape*, *dtype*, *name*, *ext_unit_dict=None*, *array_format=None*)

model

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()
```

plottable

to_shapefile(*filename*)

Export 3-D model data to shapefile (polygons). Adds an attribute for each Util2d in self.u2ds

Parameters **filename** (*str*) – Shapefile name to write

Returns

Return type None

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.lpf.hk.to_shapefile('test_hk.shp')
```

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: flopy.datbase.DataInterface, flopy.datbase.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}

array

attribute_by_kper(*attr*, *function=<function mean>*, *idx_val=None*)

binary

check_kij()

data

data_type

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

dtype

export(*f*, ***kwargs*)

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}

Returns

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}

masked_4D_arrays

masked_4D_arrays_itr()

mg

model

mxact

name

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 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)
```

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)

Returns

Return type None

Notes

Examples

```
>>> import flopy
>>> ml = flopy.modflow.Modflow.load('test.nam')
>>> ml.wel.to_shapefile('test_hk.shp', kper=1)
```

vtype

write_transient (*f*, *single_per=None*, *forceInternal=False*)

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_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

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

flopy.utils.voronoi module

class VoronoiGrid(*points*, ***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

- **points** (*ndarray*) – Two dimensional array of points with x in column 0 and y in column 1. These points will become cell centers in the voronoi grid.
- **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

The points passed into this class are marked as the cell center locations. Along the edges, these cell centers do not correspond to the centroid location of the cell polygon. Instead, the cell centers are along the edge.

This class does not yet support holes, which are supported by the Triangle class. This is a feature that could be added in the future.

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 keyward 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*)

get_voronoi_grid(*points*, ***kwargs*)

point_in_cell(*point*, *vertices*)

sort_vertices(*vlist*)

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 "L^3/T"(True, default) or if the data have been processed to volumetric values "L^3" (False)

class ZoneBudget(*cbc_file*, *z*, *kstpkper=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.
- **kstpkper** (*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.

Returns

Return type None

Examples

```
>>> from flopy.utils.zonbud import ZoneBudget, read_zbarray
>>> zon = read_zbarray('zone_input_file')
>>> zb = ZoneBudget('zonebudtest.cbc', zon, kstpkper=(0, 0))
>>> zb.to_csv('zonebudtest.csv')
>>> zb_mgd = zb * 7.48052 / 1000000
```

copy()
Return a deepcopy of the 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, read_zbarray
>>> zon = read_zbarray('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

Returns

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.

Returns

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.

Returns

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) 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 ... INTERNAL (10I7) 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 ...

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 zonebuget

Parameters

- **name** (*str*) – model name for zonebudget
- **model_ws** (*str*) – path to model
- **exe_name** (*str*) – excutable 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

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

Returns

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 ZoneBudget, read_zbarray
>>> zon = read_zbarray('zone_input_file')
>>> zb = ZoneBudget('zonebudtest.cbc', zon, kstpkper=(0, 0))
>>> df = zb.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

Returns

Return type pd.DataFrame

static load (*nam_file*, *model_ws='.'*)
Method to load a zonebudget model from namefile

Parameters

- **nam_file** (*str*) – zonebudget name file
- **model_ws** (*str*) – model workspace path

Returns

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

Returns

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 izone array

class ZoneBudgetOutput (*f*, *dis*, *zones=None*)
Bases: object

DEPRECATED: Class method to process zonebudget output into volumetric budgets

Parameters

- **f** (*str*) – zonebudget output file path
- **dis** (*flopy.modflow.ModflowDis object*) –
- **zones** (*np.ndarray*) – numpy array of zones

dataframe
Returns a net flux dataframe of the zonebudget output

dataframe_to_netcdf_fmt(*df*, *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*) –
- **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)
- **zone_array** (*np.ndarray*) – zonebudget zones array

Returns

Return type ZBNetOutput 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

Returns

Return type flopy.export.netcdf.NetCdf object

volumetric_flux(*extrapolate_kper=False*)
Method to generate a volumetric budget table based on flux information

Parameters **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

Returns

Return type pd.DataFrame

zone_array
Property method to get the zone array

zones
Get a unique list of zones

class ZoneFile6(*model*, *izone*, *extension='.zon'*, *aliases=None*)
Bases: object

Class to build, read, write and edit MODFLOW 6 zonebudget zone files

Parameters

- **model** (*ZoneBudget6 object*) – model object
- **izone** (*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*, *model*)

Method to load a Zone file for zonebudget 6.

f [str] zone file name

model [ZoneBudget6 object] zonebudget 6 model object

Returns

Return type ZoneFile6 object

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 izone array

read_zbarray (*fname*)

Reads an ascii array in a format readable by the zonebudget program executable.

Parameters **fname** (*str*) – The path and name of the file to be written.

Returns **zones** – An integer array of the zones.

Return type numpy ndarray

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*)

write_zbarray (*fname*, *X*, *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 (10I8) 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 ... INTERNAL (10I8) 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 199 ...

Parameters

- **X** (*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.

Plotting Utilities

Contents:

flopy.plot.crosssection module

class DeprecatedCrossSection(*ax=None*, *model=None*, *modelgrid=None*, *line=None*, *extent=None*)

Bases: *flopy.plot.crosssection.PlotCrossSection*

Deprecation handler for the PlotCrossSection class

Parameters

- **ax** (*matplotlib.pyplot.axes object*) –
- **model** (*flopy.modflow.Modflow object*) –
- **modelgrid** (*flopy.discretization.Grid 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.

class ModelCrossSection

Bases: object

DEPRECATED. Class to create a cross section of the 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)
- **dis** (*flopy.modflow.ModflowDis object*) – flopy discretization object. (Default is None)
- **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.
- **xul** (*float*) – x coordinate for upper left corner
- **yul** (*float*) – y coordinate for upper left corner. The default is the sum of the delc array.
- **rotation** (*float*) – Angle of grid rotation around the upper left corner. A positive value indicates clockwise rotation. Angles are in degrees. Default is None

- **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.

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 mimick 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_discharge(*frf*, *fff*, *flf=None*, *head=None*, *kstep=1*, *hstep=1*, *normalize=False*, ***kwargs*)
DEPRECATED. Use plot_vector() instead, which should follow after postprocessing.get_specific_discharge().

Use quiver to plot vectors.

Parameters

- **frf** (*numpy.ndarray*) – MODFLOW's 'flow right face'
- **fff** (*numpy.ndarray*) – MODFLOW's 'flow front face'
- **flf** (*numpy.ndarray*) – MODFLOW's 'flow lower face' (Default is None.)
- **head** (*numpy.ndarray*) – MODFLOW's head array. If not provided, then will assume confined conditions in order to calculated saturated thickness.
- **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 discharge vectors should be normalized using the magnitude of the specific discharge in each cell. (default is False)
- **kwargs** (*dictionary*) – Keyword arguments passed to plt.quiver()

Returns **quiver** – Vectors

Return type matplotlib.pyplot.quiver

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_specific_discharge(*spdis*, *head=None*, *kstep=1*, *hstep=1*, *normalize=False*, ***kwargs*)
DEPRECATED. Use plot_vector() instead, which should follow after postprocessing.get_specific_discharge().

Use quiver to plot vectors.

Parameters

- **spdis** (*np.recarray*) – numpy recarray of specific discharge information. This can be grabbed directly from the CBC file if SAVE_SPECIFIC_DISCHARGE is used in the MF6 NPF file.
- **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 discharge vectors should be normalized using the magnitude of the specific discharge in each cell. (default is False)
- **kwargs** (*dictionary*) – Keyword arguments passed to plt.quiver()

Returns **quiver** – Vectors

Return type matplotlib.pyplot.quiver

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

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 DeprecatedMapView(*model=None*, *modelgrid=None*, *ax=None*, *layer=0*, *extent=None*)

Bases: *flopy.plot.map.PlotMapView*

Deprecation handler for the PlotMapView class

Parameters

- **model** (*flopy.modflow.Modflow object*) –
- **modelgrid** (*flopy.discretization.Grid object*) –
- **ax** (*matplotlib.pyplot.axes object*) –
- **layer** (*int*) – model layer to plot, default is layer 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.

plot_discharge(*frf*, *fff*, *dis=None*, *flf=None*, *head=None*, *istep=1*, *jstep=1*, *normalize=False*, ***kwargs*)

Use quiver to plot vectors. Deprecated method that uses the old function call to pass the method to PlotMapView

Parameters

- **frf** (*numpy.ndarray*) – MODFLOW's 'flow right face'
- **fff** (*numpy.ndarray*) – MODFLOW's 'flow front face'
- **dis** (*flopy.modflow.ModflowDis package*) – Depricated parameter
- **flf** (*numpy.ndarray*) – MODFLOW's 'flow lower face' (Default is None.)
- **head** (*numpy.ndarray*) – MODFLOW's head array. If not provided, then will assume confined conditions in order to calculated saturated thickness.
- **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 discharge vectors should be normalized using the magnitude of the specific discharge in each cell. (default is False)
- **kwargs** (*dictionary*) – Keyword arguments passed to plt.quiver()

Returns **quiver** – Vectors of specific discharge.

Return type matplotlib.pyplot.quiver

class ModelMap
Bases: object

DEPRECATED. ModelMap acts as a PlotMapView factory object. Please migrate to PlotMapView for plotting functionality and future code compatibility

Parameters

- **sr** (*flopy.utils.reference.SpatialReference*) – The spatial reference class (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)
- **dis** (*flopy.modflow.ModflowDis object*) – flopy discretization object. (Default is None)
- **layer** (*int*) – Layer to plot. Default is 0. Must be between 0 and nlay - 1.
- **xul** (*float*) – x coordinate for upper left corner
- **yul** (*float*) – y coordinate for upper left corner. The default is the sum of the delc array.
- **rotation** (*float*) – Angle of grid rotation around the upper left corner. A positive value indicates clockwise rotation. Angles are in degrees.
- **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.
- **length_multiplier** (*float*) – scaling factor for conversion from model units to another unit length base ex. ft to m.

Notes

ModelMap must know the position and rotation of the grid in order to make the plot. This information is contained in the SpatialReference class (sr), which can be passed. If sr is None, then it looks for sr in dis. If dis is None, then it looks for sr in model.dis. If all of these arguments are none, then it uses xul, yul, and rotation. If none of these arguments are provided, then it puts the lower-left-hand corner of the grid at (0, 0).

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

contour_array_cvfd(*vertc*, *a*, *masked_values=None*, ***kwargs*)
Contour a cvfd array. If the array is three-dimensional, then the method will contour the layer tied to this class (self.layer). The vertices must be in the same coordinates as the rotated and offset grid.

Parameters

- **vertc** (*np.ndarray*) – Array with of size (nc, 2) with centroid location of cvfd
- **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

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 **quadmesh** – 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 **quadmesh**

Return type matplotlib.collections.QuadMesh

plot_cvfd(*verts*, *iverts*, ***kwargs*)

Plot a cvfd grid. The vertices must be in the same coordinates as the rotated and offset grid.

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
- **kwargs** (*dictionary*) – Keyword arguments passed to plotutil.plot_cvfd()

plot_discharge(*frf=None*, *fff=None*, *flf=None*, *head=None*, *istep=1*, *jstep=1*, *normalize=False*, ***kwargs*)

DEPRECATED. Use plot_vector() instead, which should follow after postprocessing.get_specific_discharge().

Use quiver to plot vectors.

Parameters

- **frf** (*numpy.ndarray*) – MODFLOW's 'flow right face'
- **fff** (*numpy.ndarray*) – MODFLOW's 'flow front face'
- **flf** (*numpy.ndarray*) – MODFLOW's 'flow lower face' (Default is None.)
- **head** (*numpy.ndarray*) – MODFLOW's head array. If not provided, then will assume confined conditions in order to calculated saturated thickness.
- **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 discharge vectors should be normalized using the magnitude of the specific discharge in each cell. (default is False)
- **kwargs** (*dictionary*) – Keyword arguments passed to plt.quiver()

Returns **quiver** – Vectors of specific discharge.

Return type matplotlib.pyplot.quiver

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** (*string or pyshp shapefile object*) – Name 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 name 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()

Returns

Return type matplotlib.Collection object

plot_specific_discharge(*spdis*, *istep=1*, *jstep=1*, *normalize=False*, ***kwargs*)

DEPRECATED. Use plot_vector() instead, which should follow after postprocessing.get_specific_discharge().

Method to plot specific discharge from discharge vectors provided by the cell by cell flow output file. In MODFLOW-6 this option is controled in the NPF options block. This method uses matplotlib quiver to create a matplotlib plot of the output.

Parameters

- **spdis** (*np.recarray*) – specific discharge recarray from cbc file
- **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 discharge vectors should be normalized using the magnitude of the specific discharge in each cell. (default is False)
- **kwargs** (*matplotlib.pyplot keyword arguments for the*) – plt.quiver method.

Returns **quiver** – quiver plot of discharge vectors

Return type matplotlib.pyplot.quiver

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.

exception PlotException(*message*)
Bases: Exception

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*)

Patch for vertex cross section plotting when we have an irregular shape type throughout the model grid or multiple shape types.

Parameters

- **xverts** (*list*) – xvertices
- **yverts** (*list*) – yvertices

Returns

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 caluculate 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*)

Parameters

- **recarray** –
- **travel_time** –

Returns

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 crossectional 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.pylot 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*) – Name of the shapefile to plot.
- **ax** (*matplolib.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

Returns

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*, *radius=500.0*, *idx=None*)

Create a patch collection from the shapes in a shapefile

Parameters

- **shp** (*string*) – 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*)

Set the legend title for a matplotlib legend object

Parameters

- **leg** (*legend object*) – matplotlib legend object
- **title** (*str*) – title 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

Returns

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 test
- **italic** (*bool*) – flag to use italic text
- **kwargs** (*dict*) – keyword arguments for the matplotlib set_xlabel method

Returns

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 test
- **italic** (*bool*) – flag to use italic text
- **kwargs** (*dict*) – keyword arguments for the matplotlib set_xlabel method

Returns

Return type None

Export Utilities

Contents:

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/ http://cfconventions.org/cf-conventions/v1.6.0/cf-conventions.html#description-of-file-contents http://wiki.esipfed.org/index.php/Attribute_Convention_for_Data_Discovery

bounds

creator

creator_url

geospatial_bounds

Describes the data's 2D or 3D geospatial extent in OGC's Well-Known Text (WKT) Geometry format

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

references

time_coverage

vertical_datum

Try to parse the vertical datum from the xml info

xmlfile

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*, *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*) – Name 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
- **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)*) –

Returns

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

Returns

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

Returns

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*) –

Returns

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.json'*)
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

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

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

class CRS(*prj=None*, *esri_wkt=None*, *epsg=None*)
Bases: object

Container to parse and store coordinate reference system parameters, and translate between different formats.

crs
Dict mapping crs attributes to proj4 parameters

static get_spatialreference(*epsg*, *text='esriwkt'*)
Gets text for given epsg code and text format from spatialreference.org Fetches the reference text using the url:

https://spatialreference.org/ref/epsg/<epsg code>/<text>/

See: https://www.epsg-registry.org/

Parameters

- **epsg** (*int*) – epsg code for coordinate system
- **text** (*str*) – string added to url

Returns **url**

Return type str

static getprj(*epsg*, *addlocalreference=True*, *text='esriwkt'*)
Gets projection file (.prj) text for given epsg code from spatialreference.org See: https://www.epsg-registry.org/

Parameters

- **epsg** (*int*) – epsg code for coordinate system
- **addlocalreference** (*boolean*) – adds the projection file text associated with epsg to a local database, epsgref.json, located in the user's data directory.

Returns **prj** – text for a projection (*.prj) file.

Return type str

static getproj4(*epsg*)
Gets projection file (.prj) text for given epsg code from spatialreference.org. See: https://www.epsg-registry.org/

Parameters **epsg** (*int*) – epsg code for coordinate system

Returns **prj** – text for a projection (*.prj) file.

Return type str

grid_mapping_attribs
Map parameters for CF Grid Mappings http://http://cfconventions.org/cf-conventions/cf-conventions.html, Appendix F: Grid Mappings

parse_wkt()

proj4
Not implemented yet

class EpsgReference
Bases: object

Sets up a local database of text representations of coordinate reference systems, keyed by EPSG code.

The database is epsgref.json, located in the user's data directory. If optional 'appdirs' package is available, this is in the platform-dependent user directory, otherwise in the user's 'HOME/.flopy' directory.

add(*epsg*, *prj*)
add an epsg code to epsgref.json

get(*epsg*)
returns prj from a epsg code, otherwise None if not found

remove(*epsg*)
removes an epsg entry from epsgref.json

reset(*verbose=True*)

static show()

to_dict()
returns dict with EPSG code integer key, and WKT CRS text

enforce_10ch_limit(*names*)
Enforce 10 character limit for fieldnames. Add suffix for duplicate names starting at 0.

Parameters **names** (*list of strings*) –

Returns **names**

Return type list of unique strings of len <= 10.

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.

import_shapefile(*check_version=True*)
Import shapefile module from pyshp.

Parameters **check_version** (*bool*) – Checks to ensure that pyshp is at least version 2. Default True, which is usually required for Writer (which has a different API), but can be False if only using Reader.

Returns

Return type module

Raises `ImportError` – If shapefile module is not found, or major version is less than 2.

model_attributes_to_shapefile(*filename*, *ml*, *package_names=None*, *array_dict=None*, ***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

- **filename** (*string*) – name of the shapefile to write
- **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)
- ****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

  epsg [int] epsg projection information

  prj [str] user supplied prj file

Returns

Return type None

Examples

```
>>> import flopy
>>> m = flopy.modflow.Modflow()
>>> flopy.utils.model_attributes_to_shapefile('model.shp', m)
```

recarray2shp(*recarray*, *geoms*, *shpname='recarray.shp'*, *mg=None*, *epsg=None*, *prj=None*, ***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*) – Path for the output shapefile
- **epsg** (*int*) – EPSG code. See https://www.epsg-registry.org/ or spatialreference.org
- **prj** (*str*) – Existing projection file to be used with new shapefile.

Notes

Uses pyshp. epsg code requires an internet connection the first time to get the projection file text from spatialreference.org, but then stashes the text in the file epsgref.json (located in the user's data directory) for subsequent use. See flopy.reference for more details.

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** (*string*) – 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)

Returns

Return type String

Examples

```
>>> import flopy
>>> name = flopy.utils.shape_attr_name('averylongstring')
>>> name
>>> 'averyl'
```

shp2recarray (*shpname*)

Read a shapefile into a numpy recarray.

Parameters **shpname** (*str*) – ESRI Shapefile.

Returns **recarray**

Return type np.recarray

write_grid_shapefile (*filename*, *mg*, *array_dict*, *nan_val=nan*, *epsg=None*, *prj=None*)

Method to write a shapefile of gridded input data

Parameters

- **filename** (*str*) – shapefile file name path
- **mg** (*flopy.discretization.Grid object*) – flopy model grid
- **array_dict** (*dict*) – dictionary of model input arrays
- **nan_val** (*float*) – value to fill nans
- **epsg** (*str, int*) – epsg code
- **prj** (*str*) – projection file name path

Returns

Return type None

write_gridlines_shapefile (*filename*, *mg*)

Write a polyline shapefile of the grid lines - a lightweight alternative to polygons.

Parameters

- **filename** (*string*) – name of the shapefile to write

- **mg** (*model grid*) –

Returns

Return type None

write_prj (*shpname*, *mg=None*, *epsg=None*, *prj=None*, *wkt_string=None*)

flopy.export.utils module

array2d_export (*f*, *u2d*, *fmt=None*, ***kwargs*)
export helper for Util2d instances

Parameters

- **f** (*str*) – filename or existing export instance type (NetCdf only for now)
- **u2d** (*Util2d 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

array3d_export (*f*, *u3d*, *fmt=None*, ***kwargs*)
export helper for Transient2d instances

Parameters

- **f** (*str*) – 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*, *outputs_filename*, *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*, *a*, *nodata=-9999*, *fieldname='value'*, ***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*) – Path of output file. Export format is determined by file extention. '.asc' Arc Ascii grid '.tif' GeoTIFF (requries 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')
- **kwargs** – keyword arguments to np.savetxt (ascii) rasterio.open (GeoTIFF) or flopy.export.shapefile_utils.write_grid_shapefile2

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*, *a*, *fieldname='level'*, *interval=None*, *levels=None*, *maxlevels=1000*, *epsg=None*, *prj=None*, ***kwargs*)

Contour an array using matplotlib; write shapefile of contours.

Parameters

- **modelgrid** (*flopy.discretization.Grid object*) – model grid object
- **filename** (*str*) – 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
- **epsg** (*int*) – EPSG code. See https://www.epsg-registry.org/ or spatialreference.org
- **prj** (*str*) – Existing projection file to be used with new shapefile.
- ****kwargs** (*keyword arguments to flopy.export.shapefile_utils.recarray2shp*) –

export_contourf(*filename*, *contours*, *fieldname='level'*, *epsg=None*, *prj=None*, ***kwargs*)

Write matplotlib filled contours to shapefile. This utility requires that shapely is installed.

Parameters

- **filename** (*str*) – 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'.
- **epsg** (*int*) – EPSG code. See https://www.epsg-registry.org/ or spatialreference.org
- **prj** (*str*) – Existing projection file to be used with new shapefile.
- ****kwargs** (*keyword arguments to flopy.export.shapefile_utils.recarray2shp*) –

Returns

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(*modelgrid*, *filename*, *contours*, *fieldname='level'*, *epsg=None*, *prj=None*, ***kwargs*)

Convert matplotlib contour plot object to shapefile.

Parameters

- **modelgrid** (*flopy.discretization.Grid*) – flopy modelgrid instance
- **filename** (*str*) – 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
- **epsg** (*int*) – EPSG code. See https://www.epsg-registry.org/ or spatialreference.org
- **prj** (*str*) – Existing projection file to be used with new shapefile.
- ****kwargs** (*key-word arguments to flopy.export.shapefile_utils.recarray2shp*) –

Returns **df**

Return type dataframe of shapefile contents

generic_array_export(*f*, *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*) – 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*, *mfl*, ***kwargs*)
export helper for MfList instances

Parameters

- **f** (*str*) – filename 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

model_export (*f*, *ml*, *fmt=None*, ***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 ....
- **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

  epsg [int] epsg projection code

  prj [str] prj file name

  if fmt is set to 'vtk', parameters of vtk.export_model

output_helper (*f*, *ml*, *oudic*, ***kwargs*)
Export model outputs using the model spatial reference info.

Parameters

- **f** (*str*) – filename for output - must have .shp or .nc extension
- **ml** (*flopy.mbase.ModelInterface derived type*) –
- **oudic** (*dict*) – output_filename,flopy datafile/cellbudgetfile instance
- ****kwargs** (*keyword arguments*) –

  modelgrid [flopy.discretizaiton.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

Returns

Return type None

casts down double precision to single precision for netCDF files

package_export (*f*, *pak*, *fmt=None*, ***kwargs*)
Method to export a package to shapefile or netcdf

Parameters

- **f** (*str*) – output file name (ends in .shp for shapefile or .nc for netcdf)
- **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

  epsg [int] epsg projection code

  prj [str] prj file name

  if fmt is set to 'vtk', parameters of vtk.export_package

Returns f

Return type NetCdf object or None

transient2d_export(*f*, *t2d*, *fmt=None*, ***kwargs*)

export helper for Transient2d instances

Parameters

- **f** (*str*) – 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.discretization.Grid

  model grid instance which will supercede the flopy.model.modelgrid

  if fmt is set to 'vtk', parameters of vtk.export_transient

flopy.export.vtk module

class Vtk(*model*, *verbose=None*, *nanval=-1e+20*, *smooth=False*, *point_scalars=False*, *vtk_grid_type='auto'*, *true2d=False*, *binary=False*)

Bases: object

Class to build VTK object for exporting flopy vtk

Parameters

- **model** (MFModel) – flopy model instance
- **verbose** (*bool*) – If True, stdout is verbose
- **nanval** (*float*) – no data value, default 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

arrays
Stores data arrays added to VTK object

Type dict

add_array (*name*, *a*, *array2d=False*)
Adds an array to the vtk object.

Parameters

- **name** (*str*) – Name of the array.
- **a** (*flopy array*) – The array to be added to the vtk object. The shape should match either grid cells or grid vertices.
- **array2d** (*bool*) – true if the array is 2d and represents the first layer, default is False

add_vector (*name*, *v*, *array2d=False*)
Adds a vector (i.e., a tuple of arrays) to the vtk object.

Parameters

- **name** (*str*) – Name of the vector.
- **v** (*tuple of arrays*) – The vector to be added to the vtk object. The shape of each component should match either grid cells or grid vertices.
- **array2d** (*bool*) – true if the vector components are 2d arrays and represent the first layer, default is False

Notes

If the grid is rotated, the vector will be rotated too, assuming that the first and second components are along x and y directions, respectively.

write (*output_file*, *timeval=None*)
Writes the stored arrays to vtk file in XML format.

Parameters

- **output_file** (*str*) – output file name without extension (extension is determined automatically)
- **timeval** (*scalar*) – model time value to be stored in the time section of the vtk file, default is None

class XmlWriterAscii (*file_path*)
Bases: *flopy.export.vtk.XmlWriterInterface*

Helps writing ascii vtk files.
Parameters **file_path** (*str*) – output file path
write_array (*array*, *actwcells=None*, ***kwargs*)
Write an array to the file.

Parameters

- **array** (*ndarray*) – the data array being output

- **actwcells** (*array*) – array of the active cells
- **kwargs** (*dictionary*) – Attributes to be added to the DataArray element

write_string(*string*)
Write a string to the file.

class XmlWriterBinary(*file_path*)
Bases: *flopy.export.vtk.XmlWriterInterface*

Helps writing binary vtk files.
Parameters **file_path** (*str*) – output file path

final()
Finalize the file. Must be called.

write_array(*array*, *actwcells=None*, ***kwargs*)
Write an array to file.

Parameters

- **array** (*ndarray*) – the data array being output
- **actwcells** (*array*) – array of the active cells
- **kwargs** (*dictionary*) – Attributes to be added to the DataArray element

write_string(*string*)
Write a string to the file.

class XmlWriterInterface(*file_path*)
Bases: object

Helps writing vtk files.
Parameters **file_path** (*str*) – output file path

add_attributes(***kwargs*)

close_element(*tag=None*)

final()
Finalize the file. Must be called.

open_element(*tag*)

write_array(*array*, *actwcells=None*, ***kwargs*)
Write an array to the file.

Parameters

- **array** (*ndarray*) – the data array being output
- **actwcells** (*array*) – array of the active cells
- **kwargs** (*dictionary*) – Attributes to be added to the DataArray element

write_line(*text*)

write_string(*string*)
Write a string to the file.

export_array(*model*, *array*, *output_folder*, *name*, *nanval=-1e+20*, *array2d=False*, *smooth=False*, *point_scalars=False*, *vtk_grid_type='auto'*, *true2d=False*, *binary=False*)
Export array to vtk
Parameters

- **model** (*flopy model instance*) – the flopy model instance
- **array** (*flopy array*) – flopy 2d or 3d array

- **output_folder** (*str*) – 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*, *otfolder*, *precision='single'*, *verbose=False*, *nanval=-1e+20*, *kstpkper=None*, *text=None*, *smooth=False*, *point_scalars=False*, *vtk_grid_type='auto'*, *true2d=False*, *binary=False*)

Exports cell by cell file to vtk

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
- **kstpkper** (*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*, *otfolder*, *text='head'*, *precision='auto'*, *verbose=False*, *nanval=-1e+20*, *kstpkper=None*, *smooth=False*, *point_scalars=False*, *vtk_grid_type='auto'*, *true2d=False*, *binary=False*)

Exports binary head file to vtk

Parameters

- **model** (MFModel) – the flopy model instance
- **hdsfile** (*str*) – binary heads file
- **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
- **kstpkper** (*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*, *otfolder*, *package_names=None*, *nanval=-1e+20*, *smooth=False*, *point_scalars=False*, *vtk_grid_type='auto'*, *true2d=False*, *binary=False*, *kpers=None*)

Exports model to vtk

Parameters

- **model** (*flopy model instance*) – flopy model
- **ot_folder** (*str*) – 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*, *otfolder*, *vtkobj=None*, *nanval=-1e+20*, *smooth=False*, *point_scalars=False*, *vtk_grid_type='auto'*, *true2d=False*, *binary=False*, *kpers=None*)

Exports package to vtk

Parameters

- **pak_model** (*flopy model instance*) – the model of the package
- **pak_name** (*str*) – the name of the package
- **otfolder** (*str*) – 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*, *name*, *nanval=-1e+20*, *array2d=False*, *smooth=False*, *point_scalars=False*, *vtk_grid_type='auto'*, *true2d=False*, *binary=False*, *kpers=None*)

Export transient 2d array to vtk

Parameters

- **model** (MFModel) – the flopy model instance
- **array** (*Transient instance*) – flopy transient array
- **output_folder** (*str*) – 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.

export_vector(*model*, *vector*, *output_folder*, *name*, *nanval=-1e+20*, *array2d=False*, *smooth=False*, *point_scalars=False*, *vtk_grid_type='auto'*, *true2d=False*, *binary=False*)

Export vector (i.e., a tuple of arrays) to vtk

Parameters

- **model** (*flopy model instance*) – the flopy model instance
- **vector** (*tuple of arrays*) – vector to be exported
- **output_folder** (*str*) – output folder to write the data
- **name** (*str*) – name of vector
- **nanval** (*scalar*) – no data value, default value is -1e20
- **array2d** (*bool*) – true if the vector components are 2d arrays and represent 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

trans_dict (*in_dict*, *name*, *trans_array*, *array2d=False*)
Builds or adds to dictionary trans_array

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

Discretization Utilities

Contents:

flopy.discretization.grid module

class CachedData(*data*)
Bases: object

data

data_nocopy

update_data(*data*)

class Grid(*grid_type=None*, *top=None*, *botm=None*, *idomain=None*, *lenuni=None*, *epsg=None*, *proj4=None*, *prj=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** (*ndarray (float)*) – top elevations of cells in topmost layer
- **botm** (*ndarray (float)*) – bottom elevations of all cells
- **idomain** (*ndarray (int)*) – ibound/idomain value for each cell
- **lenuni** (*ndarray (int)*) – model length units
- **espg** (*str, int*) – optional espg projection code
- **proj4** (*str*) – optional proj4 projection string code
- **prj** (*str*) – optional projection file name path
- **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 ndarray(float)

botm
bottom elevations of all cells

Type ndarray(float)

idomain
ibound/idomain value for each cell

Type ndarray(int)

proj4
spatial reference locates the grid in a coordinate system

Type proj4 SpatialReference

epsg
spatial reference locates the grid in a coordinate system

Type epsg SpatialReference

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

ycenters
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

angrot

angrot_radians

attribs_from_namfile_header (*namefile*)

botm

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_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 prefered 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*

cross_section_vertices

epsg

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_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*)

grid_lines

grid_type

idomain

intersect(*x*, *y*, *local=False*, *forgive=False*)

is_complete

is_valid

iverts

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

map_polygons

ncpl

nnodes

nvert

prj

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

saturated_thick(*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 **thick**

Return type calculated saturated thickness

set_coord_info(*xoff=None*, *yoff=None*, *angrot=None*, *epsg=None*, *proj4=None*, *merge_coord_info=True*)

shape

thick
Get the cell thickness for a structured, vertex, or unstructured grid.

Returns **thick**

Return type calculated thickness

top

top_botm

units

verts

write_shapefile(*filename='grid.shp'*, *epsg=None*, *prj=None*)
Write a shapefile of the grid with just the row and column attributes.

xcellcenters

xoffset

xvertices

xyzcellcenters

xyzextent

xyzvertices

ycellcenters

yoffset

yvertices

zcellcenters

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

nper

nstp

perlen

start_datetime

steady_state

time_units

totim

tslen

tsmult

flopy.discretization.structuredgrid module

class StructuredGrid(*delc=None*, *delr=None*, *top=None*, *botm=None*, *idomain=None*, *lenuni=None*, *epsg=None*, *proj4=None*, *prj=None*, *xoff=0.0*, *yoff=0.0*, *angrot=0.0*, *nlay=None*, *nrow=None*, *ncol=None*, *laycbd=None*)

Bases: *flopy.discretization.grid.Grid*

class for a structured model grid

Parameters

- **delc** – delc array
- **delr** – delr array
- **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

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)

delc

delr

delz

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(*gridspec_file*, *lenuni=0*)

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.

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 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_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*)

grid_lines
Get the grid lines as a list

intersect(*x*, *y*, *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
- **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

is_complete

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).

is_regular
Test if the grid spacing is regular and equal in x, y and z directions.

is_regular_x
Test whether the grid spacing is regular in the x direction.

is_regular_xy
Test if the grid spacing is regular and equal in x and y directions.

is_regular_xz
Test if the grid spacing is regular and equal in x and z directions.

is_regular_y
Test whether the grid spacing is regular in the y direction.

is_regular_yz
Test if the grid spacing is regular and equal in y and z directions.

is_regular_z
Test if the grid spacing is regular in z direction.

is_valid

iverts

map_polygons
Get a list of matplotlib Polygon patches for plotting

Returns

Return type list of Polygon objects

ncol

ncpl

nlay

nnodes

nrow

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

shape

top_botm

top_botm_withnan
Same as top_botm array but with NaN where idomain==0 both above and below a cell.

verts

xycenters
Return a list of two numpy one-dimensional float arrays for center x and y coordinates in model space - not offset or rotated.

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

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

xyzvertices
Method to get all grid vertices in a layer

Returns [] 2D array

zedges
Return zedges for (column, row)==(0, 0).

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*, *epsg=None*, *proj4=None*, *prj=None*, *xoff=0.0*, *yoff=0.0*, *angrot=0.0*)
Bases: *flopy.discretization.grid.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

- **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 beneath the top layer.
- **top** (*list or ndarray*) – top elevations for all cells in the grid.
- **botm** (*list or ndarray*) – bottom elevations for all cells in the grid.
- **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 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 prefered 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*

cross_section_vertices
Method to get vertices for cross-sectional plotting

Returns

Return type xvertices, yvertices

extent

classmethod from_argus_export(*fname*, *nlay=1*)
Create a new UnstructuredGrid from an Argus One Trimesh file

Parameters

- **fname** (*string*) – File name
- **nlay** (*int*) – Number of layers to create

Returns

Return type *flopy.discretization.unstructuredgrid.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*

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 **nplottable** – 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*)

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

grid_varies_by_layer

ia

intersect(*x*, *y*, *local=False*, *forgive=False*)

is_complete

is_valid

iverts

ja

map_polygons

Property to get Matplotlib polygon objects for the modelgrid

Returns

Return type list or dict of matplotlib.collections.Polygon

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

nlay

nnodes

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*)

shape

top_botm

verts

xyzcellcenters
Method to get cell centers and set to grid

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*, *epsg=None*, *proj4=None*, *prj=None*, *xoff=0.0*, *yoff=0.0*, *angrot=0.0*, *nlay=None*, *ncpl=None*, *cell1d=None*)
Bases: *flopy.discretization.grid.Grid*

class for a vertex model grid

Parameters

- **vertices** – list of vertices that make up the grid
- **cell2d** – list of cells and their vertices
- **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.

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*)

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*, *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

- **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

is_complete

is_valid

iverts

map_polygons
Get a list of matplotlib Polygon patches for plotting

Returns

Return type list of Polygon objects

ncpl

nlay

nnodes

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

shape

top_botm

verts

xyzcellcenters
Method to get cell centers and set to grid

xyzvertices
Method to get all grid vertices in a layer, arranged per cell

Returns list of size sum(nvertices per cell)

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

f

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z