Research Paper/

FloPy Workflows for Creating and Constructing Structured and Unstructured MODFLOW 6 Models

Joseph D. Hughes[1,*], Christian D. Langevin[2], Scott R. Paulinski[3], Joshua D. Larsen[4], and David Brakenhoff[5]

[1]U.S. Geological Survey, ~~Model Support and Maintenance Branch~~, 927 W Belle Plaine Ave, Chicago, IL, USA

[2]U.S. Geological Survey, ~~Model Support and Maintenance Branch~~, 2280 Woodale Dr, Mounds View, MN, USA

[3]U.S. Geological Survey, California Water Science Center, 3130 Skyway Drive, Suite 602, Santa Maria, CA, USA

[4]U.S. Geological Survey, California Water Science Center, 6000 J Street, Placer Hall, Sacramento, CA, USA

[5]Artesia Water, Korte Weistraat 12, Schoonhoven, Netherlands

[*]Corresponding author jdhughes@usgs.gov

April 11, 2023

Abstract

FloPy is a ~~popular~~ Python package for creating, running, and post-processing MODFLOW-based groundwater flow and transport models. FloPy functionality has expanded to support the latest version of MODFLOW (MODFLOW 6) including support for unstructured grids. FloPy can be used to download MODFLOW-based and other executables for Linux, MacOS, and Windows operating systems, which simplifies the process required to download and use these executables. Expanded FloPy capabilities include (1) full support for structured and unstructured spatial discretizations; (2) geoprocessing of spatial features and raster data to develop model input for supported discretization types; (3) the addition of functionality to provide direct access to simulated output data; (4) extension of plotting capabilities to unstructured MODFLOW 6 discretization types; and (5) the ability to export model data to shapefiles, NetCDF, and VTK formats for processing, analysis, and visualization by other software products. Examples of using expanded FloPy capabilities are presented for a hypothetical watershed. An unstructured groundwater flow and transport model, with several advanced stress packages, is presented to demonstrate how FloPy can be used to develop complicated unstructured model datasets from original source data (shapefiles and rasters), post-process model results, and plot simulated results.

Introduction

FloPy is a Python package for constructing, running, and post processing MODFLOW-based groundwater flow and transport models (Bakker et al. 2016). It is open-source and developed by a growing community of contributors. The combination of open-source programming languages (such as Python) with version control software (such as Git) allow the model construction process to be documented, reproducible, and easily inspected and used by others. This workflow has been recommended as one way to facilitate repeatable research and sharing of ideas (Fienen and Bakker 2016). Bakker et al. (2016) describe the general approach for working with models within the Python environment and emphasize the reproducible nature of developing models through scripting.

FloPy has been used to pioneer new methods and analysis tools, such as deep learning approaches for improving groundwater model calibration (Sun 2018; Zhou and Tartakovsky 2021), regionalizing residence times using metamodeling (Starn and Belitz 2018), applying iterative ensemble approaches for calibration and uncertainty quantification (White 2018), and exploring alternative parameterization schemes for risk analysis (Knowling et al. 2019). There are numerous examples of constructing MODFLOW models with FloPy to solve applied groundwater problems (Befus et al. 2017; van Engelen et al. 2018; Ebeling et al. 2019; Zipper et al. 2019; Befus et al. 2020). FloPy is being used in other software and workflows to improve repeatability and robustness through automated model construction (White et al. 2020; Fienen et al. 2022; Larsen et al. 2022; Leaf and Fienen 2022). FloPy is also used in GIS-based tools, such as FREEWAT (Rossetto et al. 2018) and other cyberinfrastructures (Essawy et al. 2018) to export models into MODFLOW datasets. FloPy can also be used as the "glue" to help couple MODFLOW to other hydrological models (Burek et al. 2020) or, for example, to agent-based models designed to quantify the effects of decision makers on environmental behavior (Jaxa-Rozen et al. 2019).

We use FloPy extensively to teach MODFLOW and groundwater modeling to early- and mid-career engineers and scientists. Other organizations also use FloPy to teach MODFLOW (*e.g.*, Hatari Labs and the Australian Water School). Annotated Jupyter notebooks (Kluyver et al. 2016) and example scripts demonstrate concepts and provide a resource that can be used as templates for developing real-world model applications. We routinely rely on FloPy to load and help identify problems in user model applications, and with the initial release of the MODFLOW 6 groundwater flow model (Langevin et al. 2017), we started to rely on FloPy to help with development of the MODFLOW program. We write tests that rely on FloPy to construct and run models, and then read output. We then verify that the output is as expected, by using analytical solutions, other models, or results that have been confirmed to be correct.

The purpose of this paper is to highlight FloPy new functionality for creating and constructing structured and unstructured MODFLOW models. We provide examples that demonstrate these new capabilities, and reinforce the advantages of the modern scripting workflow for developing reproducible structured and unstructured MODFLOW groundwater flow and transport models that can be easily updated as new data become available. The examples also demonstrate workflows that develop different model grids for the same model domain. The important advances described here include (1) complete support for all models, packages, and options implemented in the core version of MODFLOW supported by the U.S. Geological Survey (Hughes et al. 2017; Langevin et al. 2017; Provost et al. 2017; Langevin et al. 2020; Morway et al. 2021; Langevin et al. 2022; Hughes et al. 2022; Mancewicz et al. 2022); (2) generalized support for models based on a structured grid consisting of layers, rows, and columns, and also for models based on unstructured grids; (3) implementation of new geoprocessing capabilities to rapidly populate models with data from a variety of input sources; (4) simplified access to model results; (5) plotting capabilities for map and cross-section views of model data; and (6) export capabilities for writing model data to a variety of output formats.

FloPy Support for MODFLOW 6

The most recent version of MODFLOW (MODFLOW 6) is an object-oriented program and framework developed to provide a platform for supporting multiple models and multiple types of models within the same simulation (Hughes et al. 2017). These models can be independent of one another with no interaction, they can exchange coefficients and dependent variables (*e.g.*, head), or they can be tightly coupled at the matrix level by adding them to the same numerical solution. Transfer of information between models is isolated to exchange objects, which allow models to be developed and used independently. Within this new framework, a regional-scale groundwater model may be coupled with multiple local-scale groundwater models.

MODFLOW 6 currently includes the Groundwater Flow (GWF) Model and the Groundwater Transport (GWT) Model each with packages to represent surface water processes, groundwater extraction, external boundaries, mass sources and sinks, and mass sorption and reactions. GWF and GWT models can be developed using structured model grids consisting of layers, rows, and columns or they can be developed using more general unstructured grids using many of the concepts and numerical approaches available in

MODFLOW-USG (Panday et al. 2013). MODFLOW 6 also includes advanced formulations to simulate three-dimensional anisotropy and dispersion (Provost et al. 2017), coupled variable-density groundwater flow and transport (Langevin et al. 2020), and a water mover package to represent natural and managed hydrologic connections (Morway et al. 2021).

Development and testing of the MODFLOW 6 program relies heavily on tight integration with FloPy. A key component of this tight integration is the capability to quickly support new MODFLOW 6 models and packages with FloPy. Unlike the FloPy support for previous MODFLOW versions (*e.g.*, MODFLOW-2005, MODFLOW-NWT, MODFLOW-USG, and SEAWAT), the FloPy Python classes for MODFLOW 6 are dynamically generated from simple text files, called "definition files," that describe the input file structure. All MODFLOW 6 model input files are described using these definition files. This allows MODFLOW 6 developers to write tests for new models, packages, and functionality as they are developed. These definition files are used to programmatically generate the user input and output guide for MODFLOW 6. These same definition files are also used to generate FloPy classes, with documentation corresponding to input variable descriptions in the input and output guide. New functionality can be added by users to existing packages by modifying existing definition files using instructions provided in the MODFLOW 6 GitHub repository. The existing definition files can also be used as a template for creating classes for new MODFLOW 6 models or packages.

Common Modeling Tasks

The code snippets presented in this section demonstrate how to create model grids, geoprocess data, process output, plot model data, and export model data are available as Jupyter notebooks (Kluyver et al. 2016) at the internet address indicated in the Summary and Conclusions section.

Getting MODFLOW and Other Related Executables

FloPy for MODFLOW 6 relies on a number of helper classes, which wrap functionality available in pre-compiled external utility programs, to generate unstructured models and calculate water budgets on user-defined zones. These external utility programs (*e.g.*, GRIDGEN, Triangle, ZONEBUDGET, etc.), MODFLOW 6, and other MODFLOW-related programs (*e.g.*, MODPATH, MT3DMS, MT3D-USGS, SEAWAT, etc.) can be installed using

```
get-modflow :flopy
```

in a terminal or at the command line after installing FloPy. The `get-modflow` command detects the operating system (Linux, MacOS, or Windows) and downloads the latest operating-system-specific release of MODFLOW and related programs from an Executables GitHub repository. `get-modflow` can also download previous versions of MODFLOW 6 and the latest development version of MODFLOW 6 using instructions available on the FloPy GitHub repository.

Managing and Creating Model Grids

FloPy was originally developed to support models that are based on a structured grid consisting of layers, rows, and columns. Recent support for unstructured grids in MODFLOW (Panday et al. 2013; Langevin et al. 2017) required revisions to the underlying approach for managing spatial discretization information in FloPy. Grid information is containerized into a single location and used throughout FloPy modeling tasks for geospatial processing, plotting, and exporting. Spatial discretization is now handled in FloPy through dedicated model grid classes. There is a `Grid` class, which serves as the base class for the `StructuredGrid`, `VertexGrid`, and `UnstructuredGrid` classes. Grid objects can be created by the user for preprocessing, and they are automatically generated and attached to a FloPy model object.

Structured MODFLOW grids can have constant row and column spacings, as shown in Figure 1A, or they can have variable row and column spacings to focus resolution around an area of interest, as shown in Figure 1B. The following Python code shows how to create a StructuredGrid object in FloPy. A StructuredGrid object can also be created from discretization data required when instantiating a MODFLOW 6 DIS object using flopy.mf6.ModflowGwfdis().

```
>>> structured_grid = flopy.discretization.StructuredGrid(nlay=nlay,
... delr=delr, delc=delc, xoff=0.0, yoff=0.0, angrot=0.0, top=top, botm=botm)
```

MODFLOW 6 was developed to ~~natively~~ support multi-model simulations (Hughes et al. 2017). One form of multi-model simulation is a nested grid application in which a more finely discretized child model is embedded within a more coarsely discretized parent model (Mehl and Hill 2006; Vilhelmsen et al. 2012; Mehl and Hill 2013; Fienen et al. 2022). The use of a locally refined grid (LGR) within an encompassing parent grid offers computational benefits in that the additional refinement is targeted to an area of interest. FloPy provides a Lgr() utility class for constructing the data required to tightly couple parent and child models within a single MODFLOW 6 simulation. Figure 1C shows two StructuredGrid objects—one object represents the parent model grid and the other represents the nested child grid. The Lgr() utility class defines the connection properties between cells in the parent model and cells in the child model. Connection properties consist of distances, areas, and other geometric information needed to calculate flow between cells in different models. The Lgr() utility class is general in that the child model can have more layers than the parent model. The following Python code shows the steps for creating a child StructuredGrid object using data for the parent grid with the Lgr() utility class.

```
idomain_parent = np.ones((nlay_parent, nrow_parent, ncol_parent), dtype=int)
idomain_parent[0, 8:12, 13:18] = 0
ncpp, ncppl = 3, [1]
lgr = Lgr(nlay_parent, nrow_parent, ncol_parent,
```

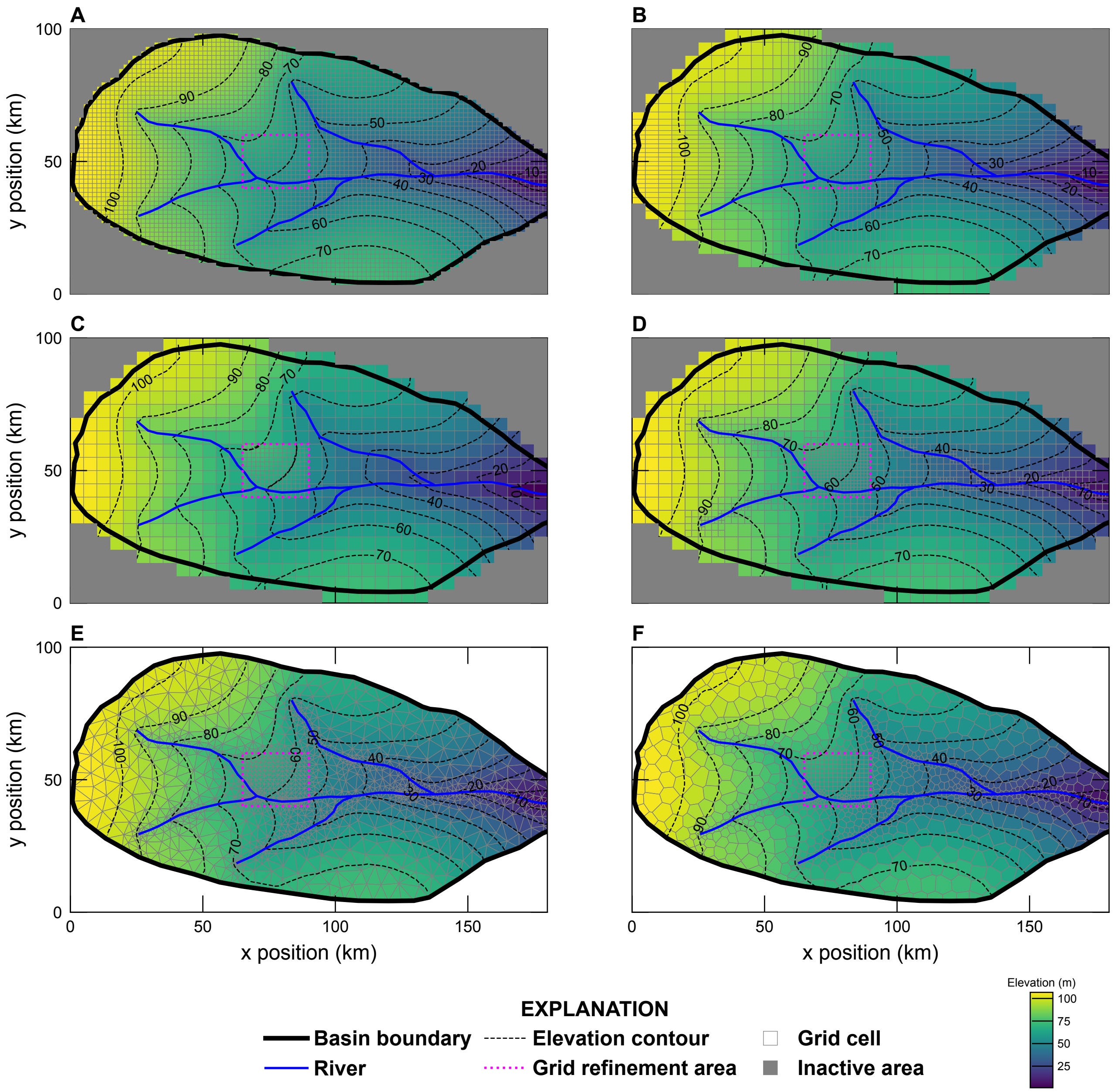

Figure 1: Examples of grids that can be generated and processed using FloPy for a hypothetical watershed, including (A) a structured MODFLOW grid with constant and equal row and column spacings, (B) a structured MODFLOW grid with variable row and column spacings, (C) a structured MODFLOW child grid nested within a structured MODFLOW parent grid, (D) a quadtree grid generated with the GRIDGEN program (Lien et al. 2014) through the FloPy wrapper, (D) a triangular grid generated with the Triangle program (Shewchuk 1996) through the FloPy wrapper, and (E) a Voronoi grid created from the triangular mesh. All of the grids have refinement in the location of the child grid in (C).

```
              delr_parent, delc_parent, topparent, botmparent,
              idomain_parent, ncpp=ncpp, ncppl=ncppl,
              xllp=0.0, yllp=0.0)
delr, delc = lgr.get_delr_delc()
xoff, yoff = lgr.get_lower_left()
structured_gridchild = StructuredGrid(delr=delr, delc=delc,
                                      xoff=xoff, yoff=yoff)
```

The child grid is created in the inactive area of the parent grid (`idomain_parent`) and the returned `lgr` object contains all of the information required to create a child `StructuredGrid` object. The connection properties needed to create the MODFLOW 6 Exchange input file for the parent and child grids can be retrieved using `lgr.get_exchange_data()`.

FloPy supports management and generation of unstructured grids. Unstructured grids are represented as layered or fully unstructured. A layered grid is one in which the same grid applies to all model layers. An unstructured grid is more general and allows the model grid to change with depth. Layered grids and unstructured grids are stored in FloPy as `VertexGrid` and `UnstructuredGrid` objects, respectively.

Layered quadtree grids can be created using the `Gridgen()` utility class, which is a wrapper around the GRIDGEN program (Lien et al. 2014). GRIDGEN starts with a structured MODFLOW grid with constant and equal row and column spacing defined by the user. The program then recursively subdivides individual cells that intersect with refinement features into quarters until a maximum level of refinement is met. Refinement features may be points, lines, or polygons. Smoothing is automatically handled so that a cell is connected to no more than two cells in any primary horizontal direction and four cells in the vertical direction. Figure 1D shows an example of a quadtree grid created with GRIDGEN in which a base grid is refined two levels along streams and in the child grid area shown in Figure 1C. The following Python code shows the steps for creating the quadtree grid with GRIDGEN.

```python
sim = flopy.mf6.MFSimulation()
gwf = flopy.mf6.ModflowGwf(sim)
dis6 = flopy.mf6.ModflowGwfdis(gwf, nrow=nrow, ncol=ncol, delr=dy, delc=dx)
g = Gridgen(dis6, model_ws=temp_path)
g.add_refinement_features([[lgr_polygon_xy]], "polygon", 2, range(1))
g.add_refinement_features(stream_points, "line", 2, range(1))
g.build(verbose=False)
gridprops_vg = g.get_gridprops_vertexgrid()
quadtree_grid = flopy.discretization.VertexGrid(**gridprops_vg)
```

FloPy also provides a wrapper utility for the Triangle mesh generation program (Shewchuk 1996). The `Triangle()` utility class writes the Triangle program input file, runs the Triangle program, and then loads the triangular mesh. Users provide the maximum area for individual triangles, angle constraints, a polygon describing the model domain, and so forth. Figure 1E shows an example of a triangular grid created with the Triangle program. The Python code for creating the triangular grid is shown below.

```python
tri = flopy.utils.triangle.Triangle(maximum_area=maximum_area,
                                    angle=30, nodes=refinement_verts,
                                    model_ws=temp_path)
tri.add_polygon(boundary_points)
tri.build(verbose=False)
cell2d = tri.get_cell2d()
vertices = tri.get_vertices()
triangular_grid = VertexGrid(vertices=vertices, cell2d=cell2d,
                             idomain=idomain, nlay=nlay, ncpl=tri.ncpl,
                             top=top, botm=botm)
```

`refinement_verts` in the triangular grid code shown above contains the user-specified stream vertices and the horizontal cell vertices for the child ~~model~~ shown in Figure 1C.

A triangular grid can be converted by FloPy into a Voronoi grid using the `VoronoiGrid()` utility class. The `VoronoiGrid()` utility class uses SciPy routines (Virtanen et al. 2020) to construct Voronoi polygons around each vertex in the triangular mesh. Figure 1F shows an example of a Voronoi grid created from the triangular mesh shown in Figure 1D. The steps for creating the Voronoi grid from the previously created `Triangle()` object (`tri`) are shown below.

```
vor = flopy.utils.voronoi.VoronoiGrid(tri)
gridprops = vor.get_gridprops_vertexgrid()
voronoi_grid = VertexGrid(**gridprops, nlay=nlay, idomain=idomain)
```

The `StructuredGrid`, `VertexGrid`, and `UnstructuredGrid` classes have useful properties and methods for accessing or mapping locations on the model grid including: (1) converting x, y pairs from local to global coordinates (`.get_coords()`) and from global to local coordinates (`.get_local_coords()`); (2) getting x, y, and z coordinates for cell centers (`.xcellcenters`, `.ycellcenters`, `.zcellcenters`, and `.xyzcellcenters`) and vertices (`.xvertices`, `.yvertices`, `.zvertices`, and `.xyzvertices`); and (3) intersecting a list of x, y pairs with the grid and returning the appropriate `cellid` (`.intersect()`). Local coordinates are model-based coordinates and global coordinates are coordinates generated after transforming local model coordinates using user-specified x-offset, y-offset, and rotation angle values; global coordinates are equal to local coordinates if the x-offset, y-offset, and rotation angle are all zero. Other useful grid class properties and methods include generating a grid object from a MODFLOW 6 binary grid file (`.from_binary_grid_file()`), retrieving cell thicknesses (`.cell_thickness`), and calculating the saturated thickness for each cell by passing a head array with dimensions consistent with the grid object (`.saturated_thickness(head)`).

The new FloPy capabilities for generating and testing different types of model grids allows for innovation in the way a study area is discretized. For example, Guira (2018) used a Voronoi grid to add additional resolution in the vicinity of irrigation wells in the Frenchman Creek Basin in Nebraska, USA to quantify the effects of land-use change and irrigation on streamflow depletion. Furthermore, the ability to develop multi-model simulations using FloPy allows higher-resolution inset models to be added in focused areas. Fienen et al. (2022) used local grid refinement models tightly coupled to inset models that were in turn loosely coupled to a coarse regional model, to better represent lakes and quantify the effects of distant pumping on lake/groundwater interactions in the Central Sands region in Wisconsin, USA. The inset groundwater flow models with lakes (Fienen et al. 2022) were developed using `modflow-setup` (Leaf and Fienen 2022), which relies on FloPy to generate MODFLOW 6 datasets.

Geospatial Processing

Geospatial processing is often a fundamental part of creating a groundwater model. New geospatial processing functionality has been added to FloPy to help users construct models using data from common input sources. The geospatial processing functionality has been implemented to work with the different types of model grids so that it is straightforward to build and construct models with different grid resolutions or grid types. The geospatial processing routines work with all three of the model grid types (`StructuredGrid`, `VertexGrid`, and `UnstructuredGrid`).

A common geospatial processing task is resampling of raster data onto a model grid. For example, it is often necessary as part of model construction to resample a raster data set of land surface elevation onto a model grid. FloPy includes a new raster sampling utility based on the Rasterio Python package (Gillies et al. 2013). The following Python code demonstrates the steps for resampling an Esri ASCII raster format grid onto a Voronoi grid.

```
fine_topo = flopy.utils.Raster.load("./grid_data/fine_topo.asc")
top_vg = fine_topo.resample_to_grid(voronoi_grid, band=fine_topo.bands[0],
```

```
                                              method="linear", extrapolate_edges=True)
```

The result of raster resampling is a numpy array, equal in size to the number of cells in one layer of the Voronoi grid. The numpy array contains an interpolated land surface elevation for each model cell. In this Python code example, the land surface grid was interpolated to the Voronoi grid using a “linear” method, however, the method also supports “nearest”, “cubic” and other options (“mean”, “median”, “mode”, “min”, and “max”) available in the rasterstats Python package (Perry 2013) for geostatistical resampling. ~~The color floods of e~~levation in Figure 1 show the results of linear raster resampling for land surface onto a variety of structured and unstructured model grids.

Performing intersections of hydrologic features with the model grid is another common modeling task. FloPy is now equipped with robust and efficient capabilities for intersecting a model grid with points, lines, and polygons. The underlying intersection routines rely on the Shapely Python package (Gillies 2022) to determine intersection properties. When a point or collection of points is intersected with a model grid, the grid intersection routine returns the cells that intersect with the points. When a line or collection of lines is intersected with a model grid, the grid intersection routine returns the cells that intersect with the lines and the lengths of lines within each intersected cell. The line and grid intersection routine also creates and returns individual line segments of the line features within each intersected cell. When a polygon or collection of polygons is intersected with a model grid, the grid intersection routine returns the cells that intersect with the polygons and the polygon area within the cell. The polygon and grid intersection routine also creates and returns individual polygons of the original polygon features within each intersected intersected cell.

The following Python code demonstrates the steps for identifying the grid cells that intersect with a collection of line segments.

```
ixs = flopy.utils.GridIntersect(voronoi_grid)
results = []
```

```
    for points in segments:
        segment = ixs.intersect(LineString(points))
        results.extend(segment["cellids"].tolist())
```

The result of this code snippet (`results`) is a list of Voronoi grid cell numbers that intersect with the line segments. The `ixs.intersect()` method also returns the `"lengths"` of the shapelike collection intersecting each cell, the `"vertices"` corresponding to each cell that intersects a collection of shapelike objects, and a shapely object (`"ixshape"`) for each portion of the original shape (`LineString(points)`) that intersects a cell. Results of the grid intersection for a linear stream network and the six different model grids is shown in Figure 2.

Processing MODFLOW 6 output

MODFLOW 6 has many different types of output that can be created during a simulation. A GWF ~~M~~odel, for example, can write simulated heads and detailed budget information to binary files. Global model budgets are written to standard MODFLOW listing (`*.lst`) files and can be written to comma-separated value text files. Some individual GWF and GWT ~~M~~odel advanced stress packages can also write simulated output during a simulation. Advanced packages ~~are packages that~~ solve their own continuity equation and include the Lake (LAK), Stream~~F~~low Routing (SFR), Multi-Aquifer Well (MAW), Unsaturated Zone Flow (UZF), and Mover (MVR) packages. For example, the LAK Package can write simulated lake stages and detailed lake budget information to binary files. Likewise, the MAW Package can write simulated well head and well budgets to binary files. Recent improvements have been made to FloPy to allow users easier access to simulation results using `.output` routines available for MODFLOW 6 models and advanced stress packages. Prior to these improvements, users were required to instantiate head, concentration, and budget file readers using file paths and names in order to access ~~this information.~~ With the `.output` routines, the file readers are automatically generated when called by the user.

The following `.methods()` syntax shows how a user can discover the type of output

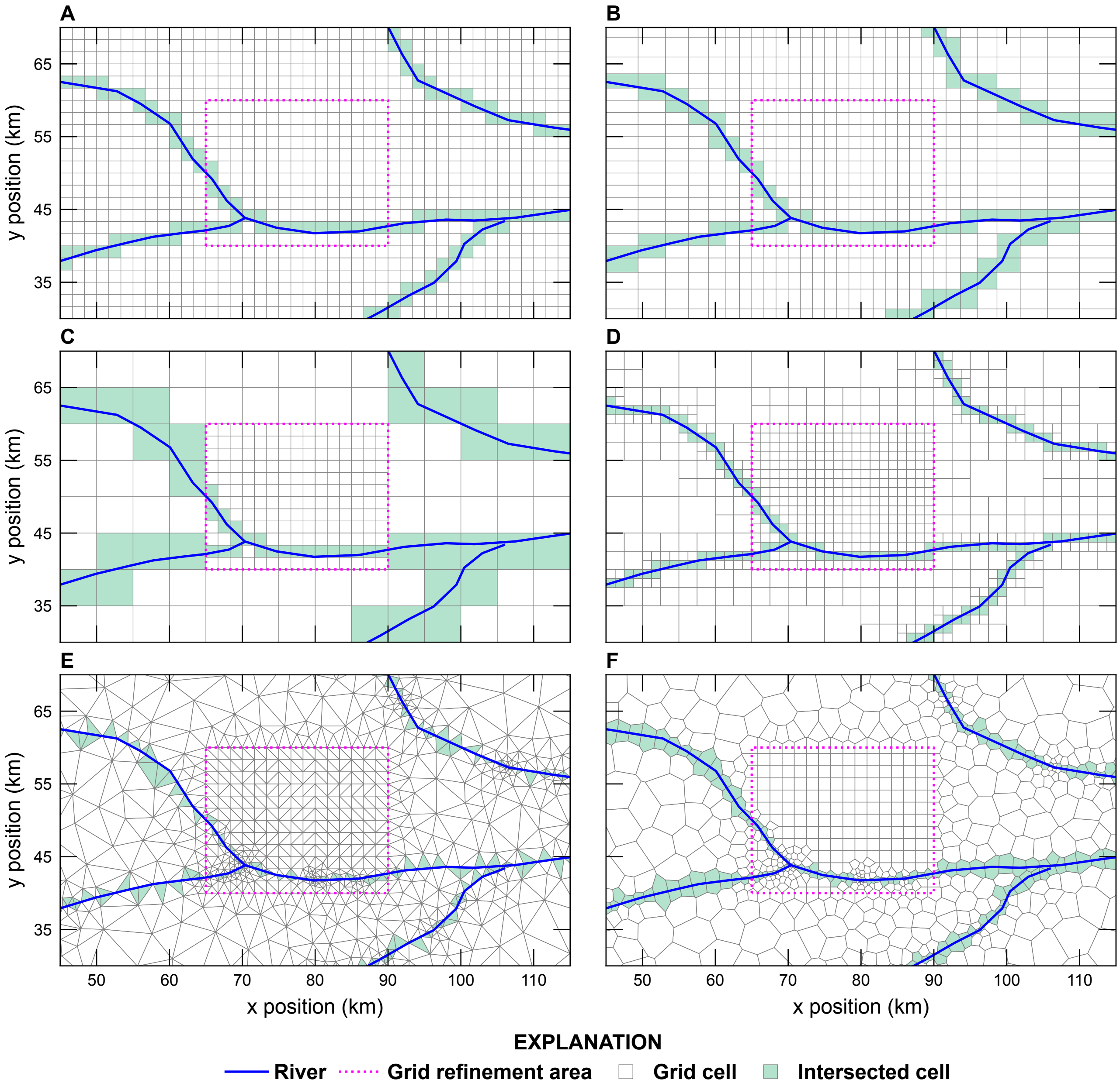

Figure 2: Examples of the intersection of a linear stream network with the model grids shown in Figure 1. Intersections were performed using FloPy for (A) a structured MODFLOW grid, (B) a structured MODFLOW grid with variable row and column spacing, (C) a structured MODFLOW child grid nested within a structured MODFLOW parent grid, (D) a quadtree grid, (D) a triangular grid, and (E) a Voronoi grid. Shaded cells represent those cells that intersect with the linear stream network. Individual plots in this figure are centered on the location of the child grid shown in Figure 1C.

332 information that is available for the specified `gwf` model.

```
>>> gwf.output.methods()
['list()', 'zonebudget()', 'budget()', 'budgetcsv()', 'head()']
```

The `.list()` method can be used to get the incremental (`incremental=True`) or cumulative budget information from the MODFLOW listing file for the `gwf` model for a user-specified simulation time, zero-based time step and stress period tuple, or zero-based index. The `.zonebudget()` method allows the user to easily build water and mass budgets for individual zones for MODFLOW 6 models, run the ZONEBUDGET program, and access ZONEBUDGET output. The `.budget()` method provides easy access to data in binary MODFLOW 6 cell-by-cell budget files. The `.budgetcsv()` method provides easy access to cumulative and incremental global budgets written by MODFLOW 6 to comma separated value files. The `.head()` method gives user access to data in the binary MODFLOW 6 head file.

Similarly, the following `.methods()` syntax shows how a user can discover the type of output information that is available for an advanced stress package, such as the LAK package.

```
>>> gwf.lak.output.methods()
['zonebudget()', 'budget()', 'budgetcsv()', 'package_convergence()', 'obs()',
 'stage()']
```

The `.package_convergence()` method can be used to get the convergence information for an advanced stress package. The `.obs()` method can be used to get observation data saved for a model or stress package as a numpy record array or pandas data frame. The `.stage()` method provides access to the dependent variable calculated by the LAK package and behaves similarly to the `.head()` method for the `gwf` model.

Processing simulated dependent variables

Simulated output for dependent variables are written by MODFLOW 6 to binary files. Simulated heads and concentrations written by the GWF and GWT models, respectively, can be accessed using the `.output` method on the FloPy `gwf` or `gwt` objects. To access the simulated head output, for example, a call can be made to the head file reader to retrieve data for a specified simulation time using the `.get_data()` method as follows.

```
head = gwf.output.head().get_data(totim=1.0)
```

In this case, the `head` variable is retrieved for a user-specified simulation time (`totim=` and is a numpy array equal in size to the size of the model grid. Head data can also be accessed for a zero-based time step–stress period tuple

```
head = gwf.output.head().get_data(kstpkper=(0,0))
```

or a zero-based index

```
head = gwf.output.head().get_data(idx=0)
```

Processing simulated cell-by-cell budgets

Similar to head output, cell-by-cell budget information can be accessed using FloPy. Unlike the simulated head file, the cell-by-cell budget file can have data for more than one item and these items may be stored in the file as arrays or lists of data. The data in the cell-by-cell budget file can be determined using

```
>>> gwf.output.budget().list_unique_records()
RECORD           IMETH
----------------------
```

```
FLOW-JA-FACE            1
DATA-SPDIS              6
DATA-SAT                6
WEL                     6
DRN                     6
RCHA                    6
EVTA                    6
SFR                     6
LAK                     6
```

The IMETH code indicates if the data is stored in the file as an array (`IMETH=1`) or if it is list based (`IMETH=6`). Cell-by-cell specific-discharge data can be extracted using

```
spdis = gwf.output.budget().get_data(totim=1.0, text="DATA-SPDIS")[0]
```

Simulated values for specific discharge for each cell are returned as a list containing a numpy record array for the user-specified simulation time (`totim=`). Like MODFLOW head data, all of the data in the cell-by-cell data file for a user-specified simulation time (`totim=`), zero-based time step and stress period tuple (`kstpkper=`), or zero-based index (`idx=`) can also be extracted. Simulated specific discharge information can be processed into a form that can be plotted with FloPy using

```
qx, qy, qz = flopy.utils.postprocessing.get_specific_discharge(spdis, gwf,
                                                                head=head)
```

The optional argument `head=` above sets the specific discharge in inactive or dry cells to NaN.

Performing zone budget analyses

`zonebudget()` output methods are available for both the `gwf` model and the `gwf.lak` advanced stress package examples shown above ~~since~~ they both solve a continuity equation. Other flow and transport advanced stress packages (*e.g.*, SFR, Streamflow Transport (SFT), Unsaturated Zone Flow (UZF), Unsaturated Zone Transport (UZT), MAW, and Multi-Aquifer Well Transport (MWT)) also solve continuity equations and can be used with this zone budget functionality. The `zonebudget()` output method can be used to perform a zone budget analysis on the LAK advanced stress package using

```
>>> zonbud = gwf.lak.output.zonebudget(zarr)
>>> zonbud.write_input()
>>> zonbud.run_model(silent=True)
(True, [])
```

`zarr` in the `gwf.lak.output.zonebudget()` is a numpy array that defines an integer zone for each lake or group of lakes in the LAK advanced stress package. Zone budget output can be returned as a numpy record array (`.get_budget()` or `.get_volumetric_budget()`) or a Pandas dataframe (`.get_dataframes()`).

Plotting

FloPy plotting capabilities have been refined and updated to support plotting both structured and unstructured models in map and cross-section view using the `.PlotMapView()` and `.PlotCrossSection()` classes, respectively. The plotting methods are wrappers around the ~~m~~atplotlib plotting methods (Hunter 2007) and allow fine-grained control using ~~m~~atplotlib keyword arguments (`kwargs`). The following Python code demonstrates the steps for plotting a map of simulated heads, the model grid, the location of drain (DRN) package cells, specific-discharge vectors, and head contours for the `gwf` model.

```python
mm = flopy.plot.PlotMapView(model=gwf)
mm.plot_array(head, edgecolor="0.5")
mm.plot_bc("DRN")
mm.plot_grid()
cs = mm.contour_array(head)
mm.ax.clabel(cs)
mm.plot_vector(qx, qy, normalize=True)
plt.show()
```

Figure 3A shows the outcome of the Python code demonstrated above with additional geographic features and fine-grained control of grid lines, text, annotations, tick locations, and axis labels. Results shown in Figure 3 are for a steady-state model discretized into three convertible layers, with isotropic hydraulic properties, a hydraulic conductivity of 1 m/d, with rivers represented as drain cells, ~~with~~ drains located on the top of the model in layer 1, and ~~with~~ an areal recharge rate of 0.000001 m/d. Figure 3B shows use of the `.plot_array()` method to create a map of the layer containing the water table, drain cells where the groundwater is discharging to a river, and cells where groundwater is discharging to the surface.

The following Python code demonstrates the steps for plotting a cross section of simulated heads and the model grid for the `gwf` model along an arbitrary line defined using a list of x, y coordinate pairs (tuples) defining the vertices of the line. For structured grids, cross sections can also be specified along a row or column.

```python
fx = flopy.plot.PlotCrossSection(model=gwf,
                                 line={"line": [(0, 42500), (186801, 42500)]})
fx.plot_array(head, head=head)
fx.plot_grid()
plt.show()
```

The `head=` keyword option for the `plot_array()` method above causes the plotting routine to draw and fill only the the saturated part of the model cell (determined using the simulated head and cell information). Without the `head=` keyword option, the entire cell from top to bottom would be color filled based on the head value. Figure 3C and D show the outcome of the Python code demonstrated along cross-section lines A–A' and B–B' (shown in Figure 3A) ~~above~~ with fine-grained control of grid lines, text, annotations, tick locations, and axis labels. Note that the color flood of head in Figure 3C and D shows that unconfined conditions occur in higher elevation cells or cells adjacent to river cells.

Exporting Grid Data to Other Formats

Model input and output can be exported in a variety of standard formats using the `.export()` method, which is available for FloPy model objects, package objects, binary dependent-variable files (head, concentration, *etc.*), and cell-by-cell output files. Standard output formats that are currently supported include shapefiles (~~ESRI~~ 1998), NetCDF files (Rew et al. 2006; Rew and Davis 1990), and Visualization Tool Kit (VTK) files (Schroeder et al. 2006). Entire models, packages, individual package arrays, binary dependent-variables (*e.g.*, heads), or three-dimensional representations of binary cell-by-cell data can be exported. Shapefile and VTK output can be exported for all grid types, but currently, NetCDF output can only be exported for structured grids. The NetCDF output capability has been used to convert entire models and associated output so that it can be rendered in the GWWebFlow viewer (U.S. Geological Survey 2018).

The following Python code demonstrates the steps for exporting the `gwf` model as a VTK dataset with flat cell tops and bottoms (stair-case representation).

```
gwf.export("temp_vtk/vtk_staircase", fmt='vtk', smooth=False,
                vertical_exageration=500.0, pvd=True)
```

VTK models can also be exported with smooth cell tops and bottoms using elevations

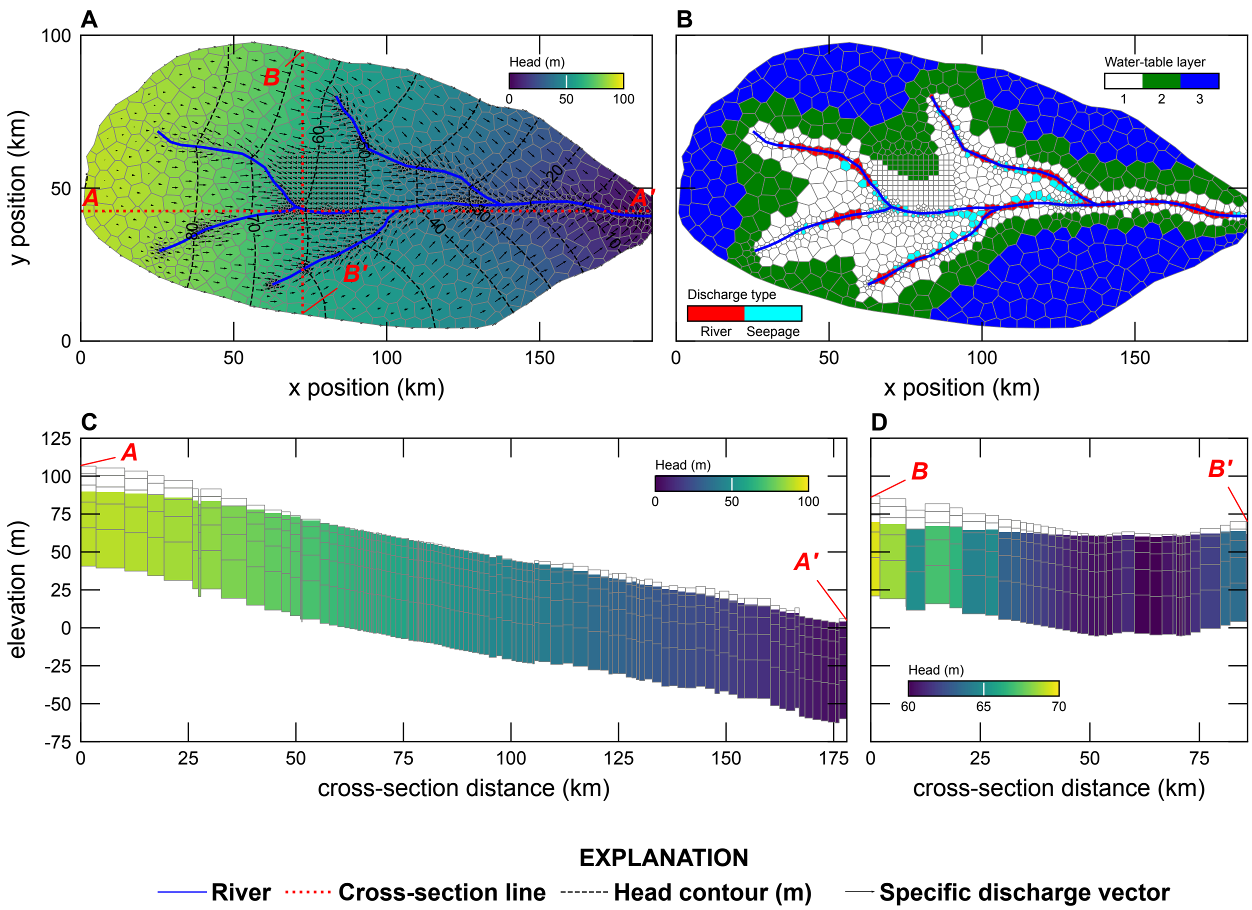

Figure 3: Examples of FloPy map and cross-section plotting capabilities for a model discretized using a Voronoi grid (Figure 1F). (A) Map showing simulated heads and specific-discharge vectors in the upper-most saturated cells. (B) Map showing the layer containing the water table, the location of cells where the aquifer is discharged to rivers represented as drain cells, and the location of cells where groundwater is discharging to the land surface. (C) East-West cross-section along line A–A', shown on Figure 3A, showing the model grid, simulated heads, and cells where water-table conditions exist. (D) North-South cross-section along line B–B', shown on Figure 3A, showing the model grid, simulated heads, and cells where water-table conditions exist.

interpolated to the cell vertices (`smooth=True`). Other supported export formats can be created by specifying the file extension to be `.shp` for shapefiles, `.nc` for NetCDF files, or if the `fmt` keyword is `vtk` (as shown above) for VTK files. Figure 4 shows stair-case and smooth VTK exports of the model described in the Plotting section and rendered with Paraview (Ahrens et al. 2005).

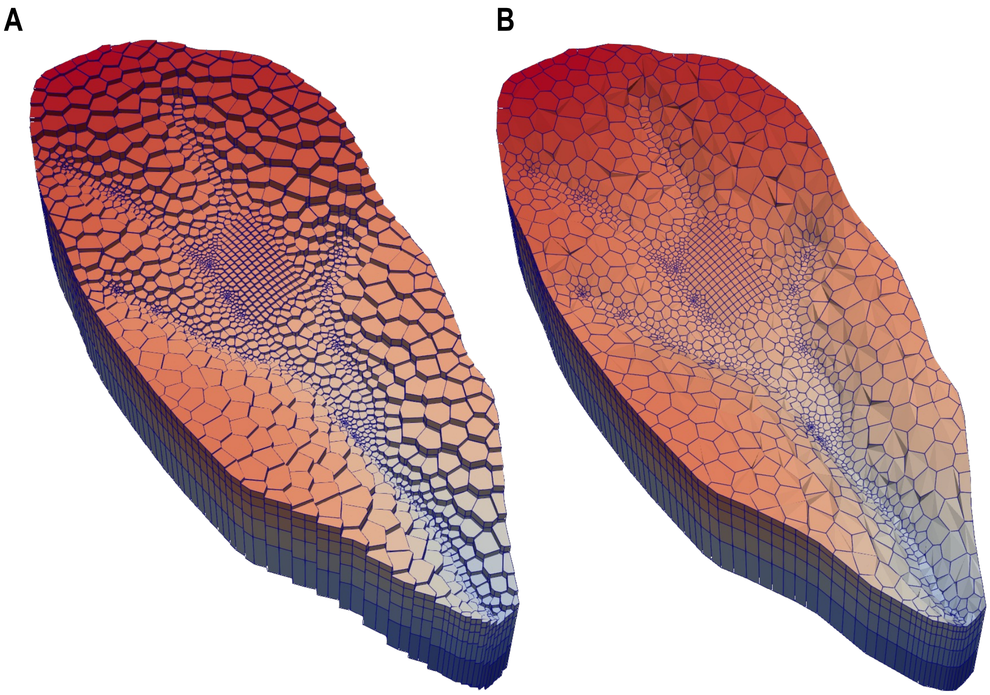

Figure 4: Two different graphical renderings of the Voronoi model grid: (A) stair-cased representation in which cell have flat tops and bottoms and (B) smooth representation in which elevations for cell vertices are interpolated using cell top and bottom elevations. Renderings were created using Paraview (Ahrens et al. 2005) and Visualization Tool Kit (Schroeder et al. 2006) files exported from FloPy.

Scripting MODFLOW 6 Model Development Using Python and FloPy

In this section, FloPy is used to construct, run, and post process a MODFLOW 6 model. All pre- and post-processing was done using FloPy grid, geospatial processing, MODFLOW 6 processing, and plotting functionality discussed previously. Figures 5, 6, 7, and 8 were created using a combination of FloPy plotting functionality and matplotlib plotting methods (Hunter 2007). Jupyter notebooks (Kluyver et al. 2016) showing the commands for creating the model data sets, processing model results, and plotting these figures are available at the Synthetic Valley internet address indicated in the Summary and Conclusions section.

Hill et al. (1998) present a synthetic test case (Synthetic Valley) of an undeveloped alluvial valley surrounded by low permeability bedrock. The model includes the Blue Lake and Straight River surface water features (Figure 5A). The model in Hill et al. (1998) was calibrated and simulated ~~using~~ MODFLOWP (Hill 1992) using a structured grid with a constant 152.4 m grid spacing, three model layers, and 1,000 active cells per layer. The upper two layers represent an unconfined aquifer, and the third layer represents a lower aquifer unit that is separated from the overlying aquifer by a confining unit in the northern part of the model domain (Figure 5A). The confining unit was not explicitly represented by Hill et al. (1998); instead a quasi-3D approach (low vertical conductance) between layers 2 and 3 was used to represent the confining unit.

MODFLOW 6 Model Setup

To demonstrate the capabilities of FloPy and MODFLOW 6 the 6,096 m x 3,810 m model domain is discretized using a Voronoi grid, with 6,343 active cells per layer, and the discretization by vertices (DISV) package (Figure 5A). The model grid was developed using the `Triangle()` and `VoronoiGrid()` utility classes. The model grid was refined within Blue Lake, around Straight River using a 750 m buffer, and around pumping wells P1, P2, and P3 using a 100 m buffer.

In this example both groundwater flow (Langevin et al. 2017) and solute transport (Langevin et al. 2022) are simulated. To better represent solute transport, the lower aquifer has been discretized into 3 layers (instead of one). Confining units have to be explicitly simulated in MODFLOW 6, therefore, a total of six layers are simulated. The bottom of layers 1, 2, 3, and 4 were set to constant values of -1.53, -15.24, -15.55 and -30.48 m, respectively. Model layer 3 represents the confining unit and is relatively thin (0.3 m). The `IDOMAIN` concept (Langevin et al. 2017) was used to eliminate cells in model layer 3 (by setting `IDOMAIN=-1`) where the confining unit does not exist. In these areas, the thickness of layer 3 was set to zero and `IDOMAIN` was set to -1, which marks these cells in layer 3 as “vertical pass through cells” and results in cells in layer 2 being directly connected to cells in layer 4.

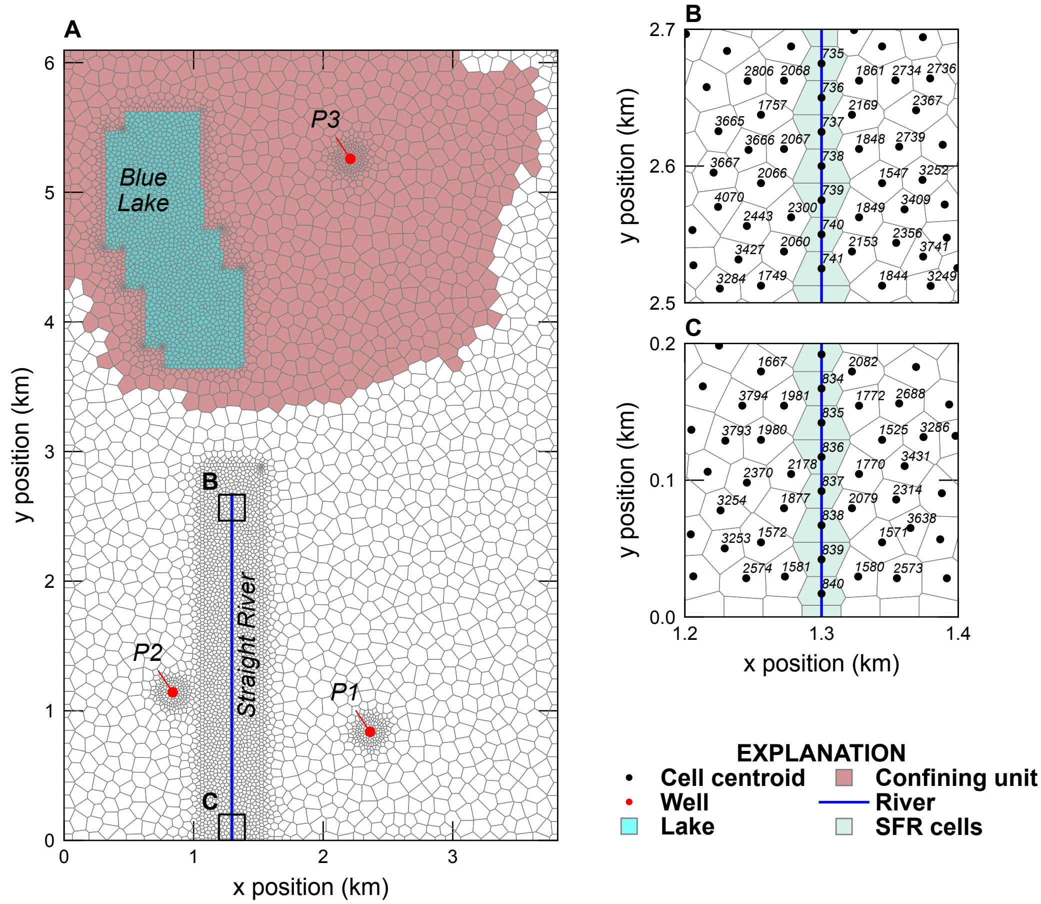

Figure 5: Synthetic Valley model used to demonstrate the MODFLOW 6 capabilities of FloPy. (A) Map showing the Voronoi grid used to discretize~~d~~ the model domain and the location of Blue Lake, Straight River, and the areal extent of the confining unit separating the upper and lower aquifer units. (B) Map showing model cells intersecting the northern end of Straight River. (C) Map showing model cells intersecting the southern end of Straight River. The cell centroid and cell numbers in the inset areas at the northern and southern end of Straight River are also shown on (B) and (C).

The bottom of the model (layer 6) is based on Hill et al. (1998) and the bottom of layer 5 was specified to be half the distance between the bottom of layers 4 and 6. The top of the model was ~~developed~~ from topographic contours developed for the model that was used as the starting point for Hill et al. (1998) (Pollock 2014); the top of the model is shown in Figure 6A. The top of the model and the bottom of layer 6 were resampled from the data used in the structured grid model using the `.resample_to_grid()` method available on the `VertexGrid` ~~for the model~~ and linear interpolation. Figure 7 shows the vertical discretization along cross-section lines A–A' and B–B', which are shown in Figure 6A.

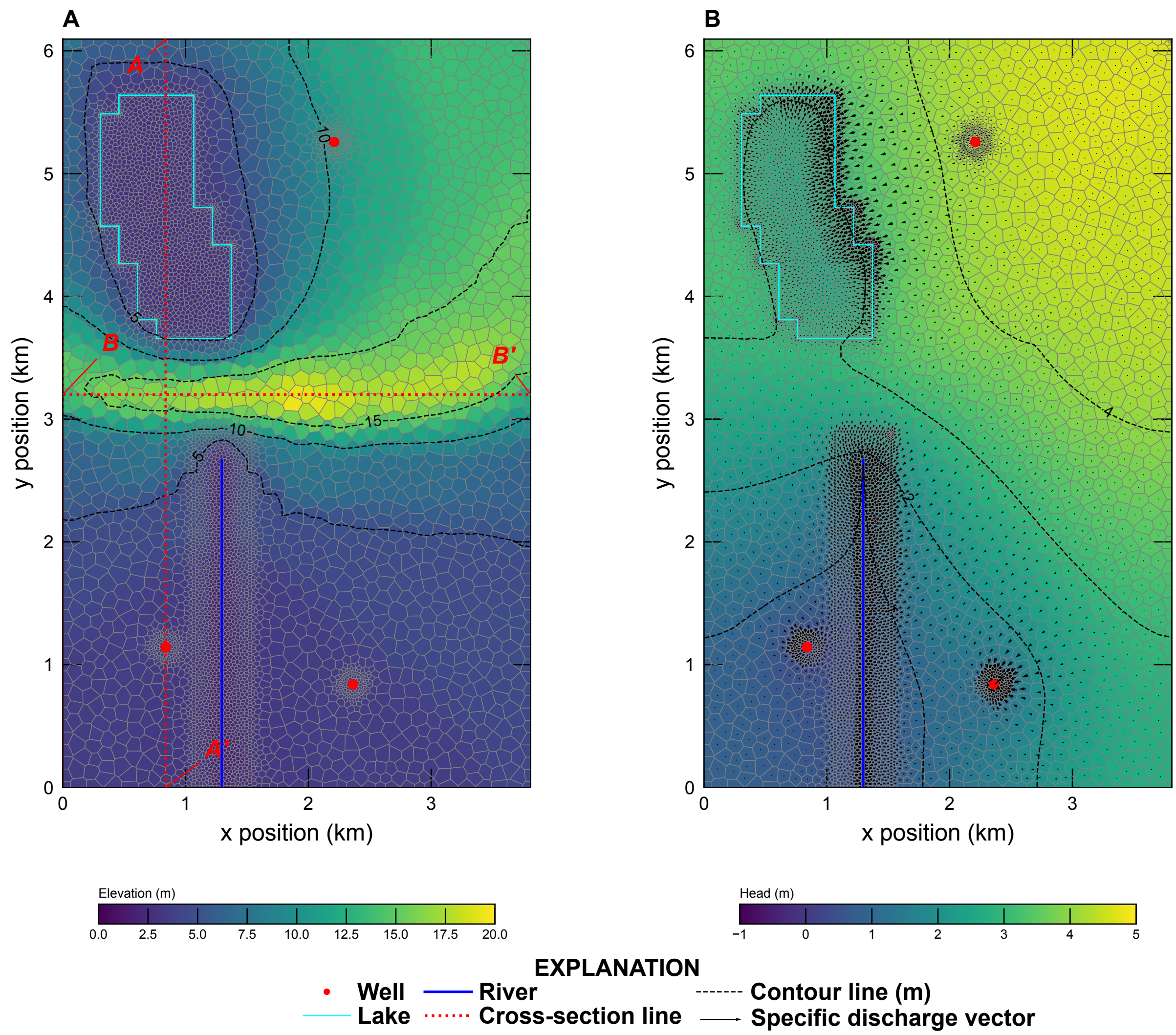

Figure 6: Map showing Synthetic Valley model (A) topography and (B) simulated steady-state heads and specific discharge rates in model layer 1. Cross-section lines A–A' and B–B' shown in Figure 7 are also shown on (A).

Hydraulic properties for the model were resampled from the data used in the structured grid model that was used as the starting point for Hill et al. (1998) (Pollock 2014). The horizontal hydraulic conductivity was discretized into five zones with values of 45.72, 50.29, 60.96, 83.82, and 121.92 m/d; the lowest hydraulic conductivity zone was located south of

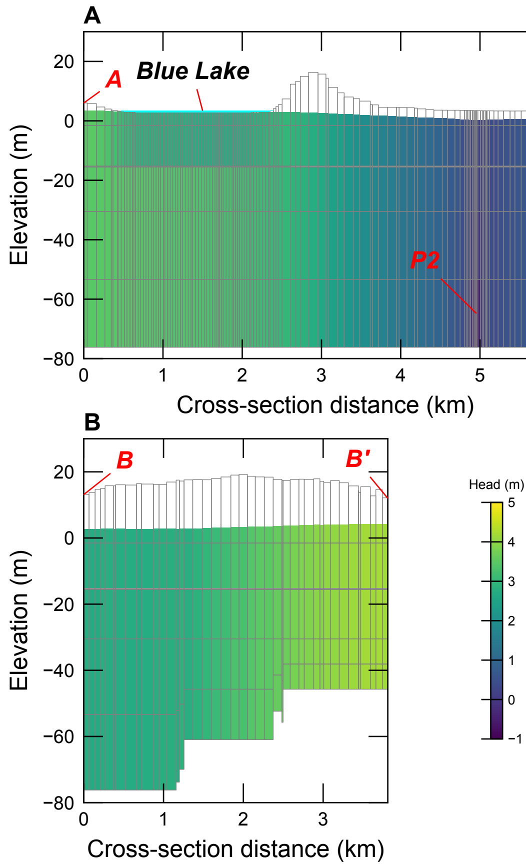

Figure 7: Cross-section of Synthetic Valley model grid and simulated steady-state heads along cross-section line (A) A–A' and (B) B–B'. The simulated Blue Lake steady-state stage (3.46 m) and pumping well P-2 are also shown on (A).

Blue Lake and the highest hydraulic conductivity zone was located beneath Blue Lake. The vertical hydraulic conductivity in the upper and lower aquifer was specified to be one quarter of the horizontal hydraulic conductivity. The horizontal and vertical hydraulic conductivity in the confining unit was set equal to 9.14×10^{-4} m/d. The horizontal and vertical hydraulic conductivity were resampled from the data used in the structured grid model using the `.resample_to_grid()` method on the `VertexGrid modelgrid` ~~for the model~~ and the nearest neighbor algorithm.

For the groundwater transport model the porosity was set to 0.2 in the upper and lower aquifer and 0.4 for cells in the confining unit. For the transport model, the Total Variation Diminishing scheme was used to simulate advection. Dispersion was simulated using a longitudinal dispersivity of 75 m and a transverse dispersivity of 7.5 m. Molecular diffusion was not represented.

In the Hill et al. (1998) representation of Synthetic Valley, the Straight River was simulated as head-dependent river (RIV) package cells, and Blue Lake was simulated as a high-hydraulic conductivity feature in model layer 1. In this recreation, Straight River is simulated using the streamflow routing (SFR) package, and Blue Lake is simulated using the LAK package. The SFR and LAK package cells were determined using `GridIntersect().intersect()` FloPy functionality (Figure 5A).

Straight River was discretized into 108 SFR reaches. Cells that intersect the northern and southern end of Straight River are shown in Figures 5B and C. The bed thickness and width of each SFR reach was specified to be 0.3048 and 3.048 m, respectively. The leakance for each SFR reach was calculated using the bed thickness, reach width, and reach length in each cell and based on a total Straight River conductance of 50,971.72 m^2/d. A specified rainfall rate of 0.0025 m/d and a potential evaporation rate of 0.0019 m/d was defined for each Straight River reach.

Blue Lake was simulated as a lake on top of the model grid and only had vertical connections to 1,406 cells in the underlying upper aquifer (model layer 1). A bed leakance of 0.0013 1/d was specified for each cell connected to Blue Lake. A specified rainfall rate of 0.0025 m/d and a potential evaporation rate of 0.0019 m/d was defined for Blue Lake.

Drain (DRN) cells were specified in each cell in model layer 1 that was not connected to Blue Lake to prevent water levels from exceeding the top of the model. The conductance of each DRN cell was based on the horizontal cell area, a thickness of 0.3048 m, and a vertical hydraulic conductivity of 0.03048 m/d. Linear scaling of the drainage conductance was applied to improve model convergence and ranged from 0 m^2/d when groundwater levels were greater than or equal to 1 m below the top of the model to the specified conductance when groundwater water levels were greater than or equal to the top of the model.

Uniform recharge and potential evapotranspiration rates were specified using the recharge (RCH) and evapotranspiration (EVT) packages, respectively, and are equal to the rates specified in the SFR and LAK packages (0.0025 and 0.0019 m/d). The EVT surface was specified to be the top of the model and the EVT extinction depth was specified to be 1 m.

The location of pumping wells P1, P2, and P3 were determined using `GridIntersect().intersect()` FloPy functionality (Figure 5A). Pumping rates of -7,600, -7,600, and -1,900 m^3/d were specified for pumping wells P1, P2, and P3, respectively.

Transport was not simulated in the LAK and SFR packages. Instead, a specified concentration condition with a concentration of 1.0 mg/L was specified for Blue Lake. All other stress packages were assumed to have a concentration of 0 mg/L.

An initial head of 11 m was specified for every cell. An initial stage of 3.44 m was specified for Blue Lake. ~~An initial concentration of 0 mg/L was specified for the transport model.~~

Simulated Results

The groundwater flow model used the Newton-Raphson Formulation with Newton under-relaxation to improve convergence. The groundwater flow and transport models used the Bi-conjugate Stabilized (`bicgstab`) linear accelerator and `complexity="simple"` settings.

The groundwater flow and transport models were run for a total of 30 years. The groundwater flow model used a single steady-state time step and groundwater flow results were used to run the transport model with a total of 360 time steps with a constant length of 30.4375 days.

Simulated heads and vectors of specific discharge in model layer 1 are shown in Figure 6B. Specific discharge is greatest on the east side of Blue Lake and in the vicinity of the three pumping wells and Straight River. ~~Cross sections showing s~~imulated heads along cross-sections A–A' and B–B' are shown in Figure 7. The cross sections show that water table conditions occur in most of the model domain except in the vicinity of Blue Lake.

Simulated concentrations at the end of 30-years in all six model layers are shown in Figure 8. Simulated concentrations are highest beneath Blue Lake in model layer 1 and do not vary much in model layers 1 and 2. Simulated concentrations in model layer 3 are limited to the extent of the confining unit because the remaining cells in the layer are defined to be vertical pass through cells (`IDOMAIN=-1`). The lateral extent of the solute plume does not vary much south of Blue Lake because of the lack of confinement in these areas.

Summary and Conclusions

FloPy is a ~~popular~~ Python package for building, running, and post processing groundwater models. It is open source and developed with input from a growing community of contributors. This paper summarizes ~~important~~ new FloPy capabilities that have been added since the package was first described by Bakker et al. (2016). The new and updated capabilities can be summarized as follows.

- FloPy supports the creation of many different types of groundwater models, including models that use MODFLOW 6, MODFLOW-2005, MODFLOW-NWT, MODFLOW-USG, MT3D, MT3D-USGS, and SEAWAT. FloPy support for MODFLOW 6 is based on an entirely new approach designed to automatically support all MODFLOW 6 models, packages, and options. The underlying FloPy classes for MODFLOW 6 are programmatically generated from the same input definition files that are used to construct the MODFLOW 6 user guide. This correspondence ensures that the FloPy classes are consistent and in-sync with MODFLOW 6 input.

- FloPy has been extended to support unstructured model grids in addition to structured grids defined by layers, rows, and columns. FloPy has several different routines for creating unstructured grids. FloPy includes a wrapper for the GRIDGEN program (Lien et al. 2014), which can be used to create layered quadtree grids. FloPy also includes a wrapper for the Triangle program (Shewchuk 1996), which can be used to create triangular meshes. A triangular mesh can be converted by FloPy into a Voronoi grid. Grid information is stored for each FloPy model created by the user. This model grid

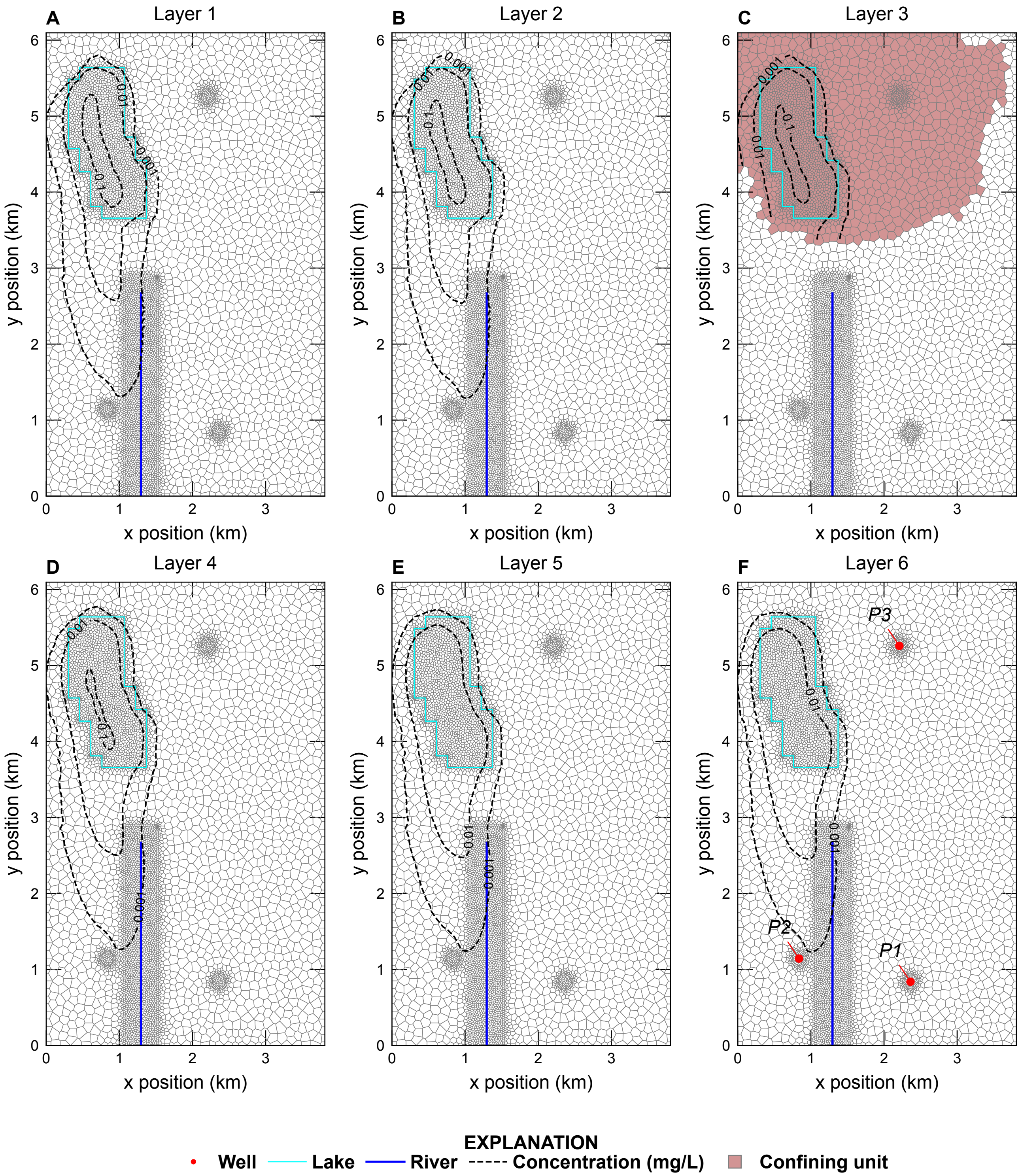

Figure 8: Maps showing Synthetic Valley simulated concentrations at the end of 30 years in model layer (A) 1, (B) 2, (C) 3, (D) 4, (E) 5, and (F) 6. The extent of the confining unit in model layer 3 is also shown on (C).

object is used systemically throughout FloPy for geospatial operations, plotting, and exporting model information to supported formats.

- Geospatial intersections of points, lines, and polygons with model grids and raster resampling onto model grids are common steps in model construction. FloPy fully supports these geospatial operations through its grid intersection and raster resampling routines.

- Access to model output using FloPy has been simplified for MODFLOW 6 models. The new output access routines makes it possible to quickly extract simulated results from binary and text model output files.

- FloPy supports plan-view map and cross-section plotting of model grids, boundary conditions, and simulated results. These plotting routines work with structured and unstructured models and can be customized to produce high quality figures.

- FloPy supports the export of model information to shapefiles, VTK files, and NetCDF files. These exported files can then be loaded into other software programs, such as geographic information systems or advanced visualization programs for additional processing.

FloPy makes it possible to construct, and reproduce the construction, of a groundwater model from ~~native~~ data in any format that can be accessed using Python. The robust new features in FloPy allow users to quickly try different model grids, different model spatial and temporal resolution, and different model configurations.

The ability to script groundwater model construction and post-processing increases robustness, ensures reproducibility, provides a record of the data processing and model construction steps, and provides a means to improve the model and extend the simulation period as new data become~~s~~ available. The new geospatial processing routines make it possible to change model resolution as part of the model construction script. This allows one to prototype fast running models with coarse resolution and use finer resolution as the model starts to behave as intended. This workflow also allows one to conduct grid convergence studies to ensure that the grid is not the cause of unintended model behavior.

FloPy is open source and we welcome bug reports, code contributions, or improvements to the documentation from the community. The FloPy Python package can be installed using the `conda` or `pip` package managers. The source code, code documentation, tutorials, and examples can be found in the FloPy GitHub repository. The Synthetic Valley example is available as a MODFLOW 6 example and the hypothetical watershed grid examples are available on the FloPy GitHub repository.

Acknowledgments

The authors gratefully acknowledge the efforts of Mark Bakker and Vincent E.A. Post for initially developing FloPy and their continued efforts improving FloPy. Funding for this research was provided by the Enterprise Capacity ~~(EC)~~ project of the U.S. Geological Survey Integrated Water Prediction program.

Authors' Note

The authors do not have any conflicts of interest to report.

Disclaimer

Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

References

Ahrens, J., B. Geveci, and C. Law. 2005. ParaView: An End-User Tool for Large Data Visualization. Visualization Handbook. Elsevier.

Bakker, M., V. Post, C.D. Langevin, J.D. Hughes, J. White, J. Starn, and M.N. Fienen. 2016. Scripting MODFLOW model development using Python and FloPy. *Groundwater* 54, no. 5: 733–739, https://doi.org/10.1111/gwat.12413.

Befus, K.M., P.L. Barnard, D.J. Hoover, J.A. Finzi Hart, and C.I. Voss. 2020. Increasing threat of coastal groundwater hazards from sea-level rise in California. *Nature Climate Change* 10, no. 10: 946–952, https://doi.org/10.1038/s41558-020-0874-1.

Befus, K.M., K.D. Kroeger, C.G. Smith, and P.W. Swarzenski. 2017. The Magnitude and Origin of Groundwater Discharge to Eastern U.S. and Gulf of Mexico Coastal Waters. *Geophysical Research Letters* 44, no. 20: 10,396–10,406, https://doi.org/10.1002/2017GL075238.

Burek, P., Y. Satoh, T. Kahil, T. Tang, P. Greve, M. Smilovic, L. Guillaumot, F. Zhao, and Y. Wada. 2020. Development of the Community Water Model (CWatM v1.04) – a high-resolution hydrological model for global and regional assessment of integrated water resources management. *Geoscientific Model Development* 13, no. 7: 3267–3298, https://doi.org/10.5194/gmd-13-3267-2020.

Ebeling, P., F. Handel, and M. Walther. 2019. Potential of mixed hydraulic barriers to remediate seawater intrusion. *Science of The Total Environment 693*: 133478, https://doi.org/10.1016/j.scitotenv.2019.07.284.

ESRI. 1998. ESRI Shapefile Technical Description, an ESRI white paper. http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf (accessed August 29, 2022).

Essawy, B.T., J.L. Goodall, W. Zell, D. Voce, M.M. Morsy, J. Sadler, Z. Yuan, and T. Malik. 2018. Integrating scientific cyberinfrastructures to improve reproducibility in computational hydrology: Example for HydroShare and GeoTrust. *Environmental Modelling & Software 105*: 217–229, https://doi.org/10.1016/j.envsoft.2018.03.025.

Fienen, M.N. and M. Bakker. 2016. HESS Opinions: Repeatable research: what hydrologists can learn from the Duke cancer research scandal. *Hydrology and Earth System Sciences* 20, no. 9: 3739–3743, https://doi.org/10.5194/hess-20-3739-2016.

Fienen, M.N., N.T. Corson-Dosch, J.T. White, A.T. Leaf, and R.J. Hunt. 2022. Risk-Based Wellhead Protection Decision Support: A Repeatable Workflow Approach. *Groundwater* 60, no. 1: 71–86, https://doi.org/10.3389/feart.2022.903965.

Fienen, M.N., M.J. Haserodt, A.T. Leaf, and S.M. Westenbroek. 2022. Simulation of regional groundwater flow and groundwater/lake interactions in the Central Sands, Wisconsin. U.S. Geological Survey Scientific Investigations Report 2022-5046, 111 p. https://doi.org/10.3133/sir20225046.

Gillies, S. 2022. The shapely user manual. https://shapely.readthedocs.io/en/stable/manual.html (accessed August 28, 2022).

Gillies, S. et al. 2013. Rasterio: geospatial raster I/O for Python programmers. https://github.com/rasterio/rasterio (accessed October 6, 2022).

Guira, M. 2018. Numerical Modeling Of The Effects Of Land Use Change And Irrigation On Streamflow Depletion Of Frenchman Creek, Nebraska. Master's thesis, University of Nebraska, Lincoln, NE.

Hill, M.C. 1992. A computer program (MODFLOWP) for estimating parameters of a transient, three-dimensional, ground-water flow model using nonlinear regression. U.S. Geological Survey Open-File Report 91-484, 358 p.

Hill, M.C., R.L. Cooley, and D.W. Pollock. 1998. A Controlled Experiment in Ground Water Flow Model Calibration. *Groundwater* 36, no. 3: 520–535, https://doi.org/10.1111/j.1745-6584.1998.tb02824.x.

Hughes, J.D., C.D. Langevin, and E.R. Banta. 2017. Documentation for the MODFLOW 6 framework. U.S. Geological Survey Techniques and Methods, book 6, chap. A57, 36 p. https://doi.org/10.3133/tm6A57.

Hughes, J.D., S.A. Leake, D.L. Galloway, and J.W. White. 2022. Documentation for the Skeletal Storage, Compaction, and Subsidence (CSUB) Package of MODFLOW 6. U.S. Geological Survey Techniques and Methods, book 6, chap. A62, 57 p. https://doi.org/10.3133/tm6A62.

Hunter, J.D. 2007. Matplotlib: A 2D graphics environment. *Computing in science & engineering* 9, no. 03: 90–95.

Jaxa-Rozen, M., J.H. Kwakkel, and M. Bloemendal. 2019. A coupled simulation architecture for agent-based/geohydrological modelling with NetLogo and MODFLOW. *Environmental Modelling & Software 115*: 19–37, https://doi.org/10.1016/j.envsoft.2019.01.020.

Kluyver, T., B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt (Eds.), *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pp. 87 – 90. IOS Press, https://doi.org/10.3233/978-1-61499-649-1-87.

Knowling, M.J., J.T. White, and C.R. Moore. 2019. Role of model parameterization in risk-based decision support: An empirical exploration. *Advances in Water Resources 128*: 59–73, https://doi.org/10.1016/j.advwatres.2019.04.010.

Langevin, C.D., J.D. Hughes, A.M. Provost, E.R. Banta, R.G. Niswonger, and S. Panday. 2017. Documentation for the MODFLOW 6 Groundwater Flow (GWF) Model. U.S. Geological Survey Techniques and Methods, book 6, chap. A55, 197 p. https://doi.org/10.3133/tm6A55.

Langevin, C.D., S. Panday, and A.M. Provost. 2020. Hydraulic-Head Formulation for Density-Dependent Flow and Transport. *Groundwater* 58, no. 3: 349–362.

Langevin, C.D., A.M. Provost, S. Panday, and J.D. Hughes. 2022. Documentation for the MODFLOW 6 Groundwater Transport (GWT) Model. U.S. Geological Survey Techniques and Methods, book 6, chap. A61, 56 p. https://doi.org/10.3133/tm6A55.

Larsen, J.D., A.H. Alzraiee, D. Martin, and R.G. Niswonger. 2022. Rapid Model Development for GSFLOW With Python and pyGSFLOW. *Frontiers in Earth Science 10*, https://doi.org/10.3389/feart.2022.907533.

Leaf, A.T. and M.N. Fienen. 2022. Modflow-setup: Robust automation of groundwater model construction. *Frontiers in Earth Science 10*: 903965, https://doi.org/10.3389/feart.2022.903965.

Lien, J.M., G. Liu, and C.D. Langevin. 2014. GRIDGEN Version 1.0: A computer program for generating unstructured finite-volume grids. U.S. Geological Survey Open-File Report 2014–1109, 26 p. https://doi.org/10.3133/ofr20141109.

Mancewicz, L.K., A. Mayer, C.D. Langevin, and J. Gulley. 2022. Improved method for simulating groundwater inundation using the MODFLOW 6 Lake Transport Package. *Groundwater*, https://doi.org/10.1111/gwat.13254.

Mehl, S.W. and M.C. Hill. 2006. MODFLOW-2005, the U.S. Geological Survey modular ground-water model-documentation of shared node local grid refinement (LGR) and the boundary flow and head (BFH) package. U.S. Geological Survey Techniques and Methods, book 6, chap. A12, 78 p. https://doi.org/10.3133/tm6A12.

Mehl, S.W. and M.C. Hill. 2013. MODFLOW-LGR—Documentation of ghost node local grid refinement (LGR2) for multiple areas and the boundary flow and head (BFH2) package. U.S. Geological Survey Techniques and Methods, book 6, chap. A44, 43 p. https://doi.org/10.3133/tm6A44.

Morway, E.D., C.D. Langevin, and J.D. Hughes. 2021. Use of the MODFLOW 6 water mover package to represent natural and managed hydrologic connections. *Groundwater* 59, no. 6: 913–924, https://doi.org/10.1111/gwat.13117.

Panday, S., C.D. Langevin, R.G. Niswonger, M. Ibaraki, and J.D. Hughes. 2013. MODFLOW-USG version 1—An unstructured grid version of MODFLOW for simulating groundwater flow and tightly coupled processes using a control volume finite-difference formulation. U.S. Geological Survey Techniques and Methods, book 6, chap. A45, 66 p.

Perry, M. 2013. Rasterstats python library. https://github.com/perrygeo/python-rasterstats.

Pollock, D.W. 2014. Personal communication. Reston, VA.

Provost, A.M., C.D. Langevin, and J.D. Hughes. 2017. Documentation for the "XT3D" Option in the Node Property Flow (NPF) Package of MODFLOW 6. U.S. Geological Survey Techniques and Methods, book 6, chap. A56, 46 p. https://doi.org/10.3133/tm6A56.

Rew, R. and G. Davis. 1990. NetCDF: an interface for scientific data access. *IEEE computer graphics and applications* 10, no. 4: 76–82.

Rew, R., E. Hartnett, J. Caron, et al. 2006. NetCDF-4: Software implementing an enhanced data model for the geosciences. In *22nd International Conference on Interactive Information Processing Systems for Meteorology, Oceanograph, and Hydrology*, Volume 6.

Rossetto, R., G. De Filippis, I. Borsi, L. Foglia, M. Cannata, R. Criollo, and E. Vázquez-Suñé. 2018. Integrating free and open source tools and distributed modelling codes in GIS environment for data-based groundwater management. *Environmental Modelling & Software 107*: 210–230, https://doi.org/10.1016/j.envsoft.2018.06.007.

Schroeder, W., K. Martin, and B. Lorensen. 2006. *The Visualization Toolkit* (4th ed.). Kitware.

Shewchuk, J.R. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry, Towards Geometric Engineering, FCRC'96 Workshop, WACG'96, Philadelphia, PA, USA, May 27-28, 1996, Selected Papers*, pp. 203–222. https://doi.org/10.1007/BFb0014497.

Starn, J.J. and K. Belitz. 2018. Regionalization of Groundwater Residence Time Using Metamodeling. *Water Resources Research* 54, no. 9: 6357–6373, https://doi.org/10.1029/2017WR021531.

Sun, A.Y. 2018. Discovering State-Parameter Mappings in Subsurface Models Using Generative Adversarial Networks. *Geophysical Research Letters* 45, no. 20: 11,137–11,146, https://doi.org/10.1029/2018GL080404.

U.S. Geological Survey. 2018. GWWebFlow–a browser-based groundwater model viewer. https://webapps.usgs.gov/gwwebflow/.

van Engelen, J., G.H. Oude Essink, H. Kooi, and M.F. Bierkens. 2018. On the origins of hypersaline groundwater in the Nile Delta aquifer. *Journal of Hydrology 560*: 301–317, https://doi.org/10.1016/j.jhydrol.2018.03.029.

Vilhelmsen, T.N., S. Christensen, and S.W. Mehl. 2012. Evaluation of MODFLOW-LGR in connection with a synthetic regional-scale model. *Groundwater* 50, no. 1: 118–132, https://doi.org/10.1111/j.1745-6584.2011.00826.x.

Virtanen, P., R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods 17*: 261–272, https://doi.org/10.1038/s41592-019-0686-2.

White, J.T. 2018. A model-independent iterative ensemble smoother for efficient history-matching and uncertainty quantification in very high dimensions. *Environmental Modelling & Software 109*: 191–201, https://doi.org/10.1016/j.envsoft.2018.06.009.

White, J.T., L.K. Foster, M.N. Fienen, M.J. Knowling, B. Hemmings, and J.R. Winterle. 2020. Toward reproducible environmental modeling for decision support: A worked example. *Frontiers in Earth Science 8*: 50, https://doi.org/10.3389/feart.2020.00050.

Zhou, Z. and D.M. Tartakovsky. 2021. Markov chain Monte Carlo with neural network surrogates: application to contaminant source identification. *Stochastic Environmental Research and Risk Assessment* 35, no. 10: 639–651, https://doi.org/10.1007/s00477-020-01888-9.

Zipper, S.C., T. Gleeson, B. Kerr, J.K. Howard, M.M. Rohde, J. Carah, and J. Zimmerman. 2019. Rapid and Accurate Estimates of Streamflow Depletion Caused by Groundwater Pumping Using Analytical Depletion Functions. *Water Resources Research* 55, no. 7: 5807–5829, https://doi.org/10.1029/2018WR024403.

Figure captions