

<sup>1</sup> FloPy Workflows for Creating and Constructing  
<sup>2</sup> Structured and Unstructured MODFLOW 6 Models

<sup>3</sup> Joseph D. Hughes<sup>1</sup>, Christian D. Langevin<sup>2</sup>, Scott R. Paulinski<sup>3</sup>, Joshua D.  
<sup>4</sup> Larsen<sup>4</sup>, and David Brakenhoff<sup>5</sup>

<sup>5</sup> <sup>1</sup>U.S. Geological Survey, Model Support and Maintenance Branch, 927 W  
<sup>6</sup> Belle Plaine Ave, Chicago, IL, USA

<sup>7</sup> <sup>2</sup>U.S. Geological Survey, Model Support and Maintenance Branch, 2280  
<sup>8</sup> Woodale Dr, Mounds View, MN, USA

<sup>9</sup> <sup>3</sup>U.S. Geological Survey, California Water Science Center, 3130 Skyway Drive,  
<sup>10</sup> Suite 602, Santa Maria, CA, USA

<sup>11</sup> <sup>4</sup>U.S. Geological Survey, California Water Science Center, 6000 J Street,  
<sup>12</sup> Placer Hall, Sacramento, CA, USA

<sup>13</sup> <sup>5</sup>Artesia Water, Korte Weistraat 12, Schoonhoven, Netherlands

<sup>14</sup> November 23, 2022

<sup>15</sup> **Abstract**

<sup>16</sup> FloPy is a popular Python package for creating, running, and post-processing MODFLOW-  
<sup>17</sup> based groundwater flow and transport models. FloPy functionality has expanded to sup-  
<sup>18</sup> port the latest version of MODFLOW (MODFLOW 6) including support for unstructured  
<sup>19</sup> grids. FloPy can be used to download MODFLOW-based and other executables for Linux,

20 MacOS, and Windows operating systems, which simplifies the process required to down-  
21 load and use these executables. Expanded FloPy capabilities include (1) full support for  
22 structured and unstructured spatial discretizations; (2) geoprocessing of spatial features  
23 and raster data to develop model input for supported discretization types; (3) the addition  
24 of functionality to provide direct access to simulated output data; (4) extension of plot-  
25 ting capabilities to unstructured MODFLOW 6 discretization types; and (5) the ability to  
26 export model data to shapefiles, NetCDF, and VTK formats for processing, analysis, and  
27 visualization by other software products. Examples of using expanded FloPy capabilities  
28 are presented for a hypothetical watershed. An unstructured groundwater flow and trans-  
29 port model, with several advanced stress packages, is presented to demonstrate how FloPy  
30 can be used to develop complicated unstructured model datasets from original source data  
31 (shapefiles and rasters), post-process model results, and plot simulated results.

## 32 Introduction

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

42 FloPy has been used to pioneer new methods and analysis tools, such as deep learning ap-  
43 proaches for improving groundwater model calibration ([Sun 2018; Zhou and Tartakovsky 2021](#)),  
44 regionalizing residence times using metamodeling ([Starn and Belitz 2018](#)), applying iterative  
45 ensemble approaches for calibration and uncertainty quantification ([White 2018](#)), and exploring  
46 alternative parameterization schemes for risk analysis ([Knowling et al. 2019](#)). There are nu-

47 merous examples of constructing MODFLOW models with FloPy to solve applied groundwater  
48 problems (Befus et al. 2017; van Engelen et al. 2018; Ebeling et al. 2019; Zipper et al. 2019;  
49 Befus et al. 2020). FloPy is used in GIS-based tools, such as FREEWAT (Rossetto et al. 2018)  
50 and other cyberinfrastructures (Essawy et al. 2018) to export models into MODFLOW datasets.  
51 FloPy can also be used as the “glue” to help couple MODFLOW to other hydrological models  
52 (Burek et al. 2020) or, for example, to agent-based models designed to quantify the effects of  
53 decision makers on environmental behavior (Jaxa-Rozen et al. 2019).

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

64 The purpose of this paper is to highlight FloPy new functionality for creating and construct-  
65 ing structured and unstructured MODFLOW models. We provide examples that demonstrate  
66 these new capabilities, and reinforce the advantages of the modern scripting workflow for devel-  
67 oping reproducible structured and unstructured MODFLOW groundwater flow and transport  
68 models that can be easily updated as new data become available. The examples also demon-  
69 strate workflows that develop different model grids for the same model domain. The important  
70 advances described here include (1) complete support for all models, packages, and options im-  
71 plemented in the core version of MODFLOW supported by the U.S. Geological Survey (Hughes  
72 et al. 2017; Langevin et al. 2017; Provost et al. 2017; Langevin et al. 2020; Morway et al. 2021;  
73 Langevin et al. 2022; Hughes et al. 2022; Mancewicz et al. 2022); (2) generalized support for  
74 models based on a regular grid consisting of layers, rows, and columns, and also for models based

75 on unstructured grids; (3) implementation of new geoprocessing capabilities to rapidly populate  
76 models with data from a variety of input sources; (4) simplified access to model results; (5)  
77 plotting capabilities for map and cross-section views of model data; and (6) export capabilities  
78 for writing model data to a variety of output formats.

## 79 FloPy Support for MODFLOW 6

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

88 MODFLOW 6 currently includes the Groundwater Flow (GWF) Model and the Groundwa-  
89 ter Transport (GWT) Model each with packages to represent surface water processes, ground-  
90 water extraction, external boundaries, mass sources and sinks, and mass sorption and reactions.  
91 GWF and GWT models can be developed using regular model grids consisting of layers, rows,  
92 and columns or they can be developed using more general unstructured grids using many of  
93 the concepts and numerical approaches available in MODFLOW-USG ([Panday et al. 2013](#)).  
94 MODFLOW 6 also includes advanced formulations to simulate three-dimensional anisotropy  
95 and dispersion ([Provost et al. 2017](#)), coupled variable-density groundwater flow and transport  
96 ([Langevin et al. 2020](#)), and a water mover package to represent natural and managed hydrologic  
97 connections ([Morway et al. 2021](#)).

98 Development and testing of the MODFLOW 6 program relies heavily on tight integration  
99 with FloPy. A key component of this tight integration is the capability to quickly support  
100 new MODFLOW 6 models and packages with FloPy. Unlike the FloPy support for previ-

ous 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 that 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 hypothetical watershed 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 executables, to generate unstructured models and calculate water budgets on user-defined zones. To facilitate getting these external executables, MODFLOW 6, and other MODFLOW-related programs (*e.g.*, MODPATH, MT3DMS, MT3D-USGS, SEAWAT, etc.) can be installed using

```
get_modflow :flopy
```

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

## 129 Managing and Creating Model Grids

130 FloPy was originally developed to support models that are based on a regular grid consisting  
131 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  
132 spatial discretization information in FloPy. Grid information is containerized into a single  
133 location and used throughout FloPy modeling tasks for geospatial processing, plotting, and  
134 exporting. Spatial discretization is now handled in FloPy through dedicated model grid classes.  
135 There is a `Grid` class, which serves as the base class for the `StructuredGrid`, `VertexGrid`, and  
136 `UnstructuredGrid` classes. Grid objects can be created by the user for preprocessing, and they  
137 are automatically generated and attached to a FloPy model object.

139 Regular MODFLOW grids can have constant row and column spacings, as shown in Figure  
140 1A, or they can have variable row and column spacings to focus resolution around an area of in-  
141 terest, as shown in Figure 1B. The following Python code shows how to create a `StructuredGrid`  
142 object in FloPy. A `StructuredGrid` object can also be created from discretization data required  
143 when instantiating a MODFLOW 6 DIS object using `flopy.mf6.ModflowGwfdis()`.

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

146 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  
147 discretized child model is embedded within a more coarsely discretized parent model ([Mehl](#)

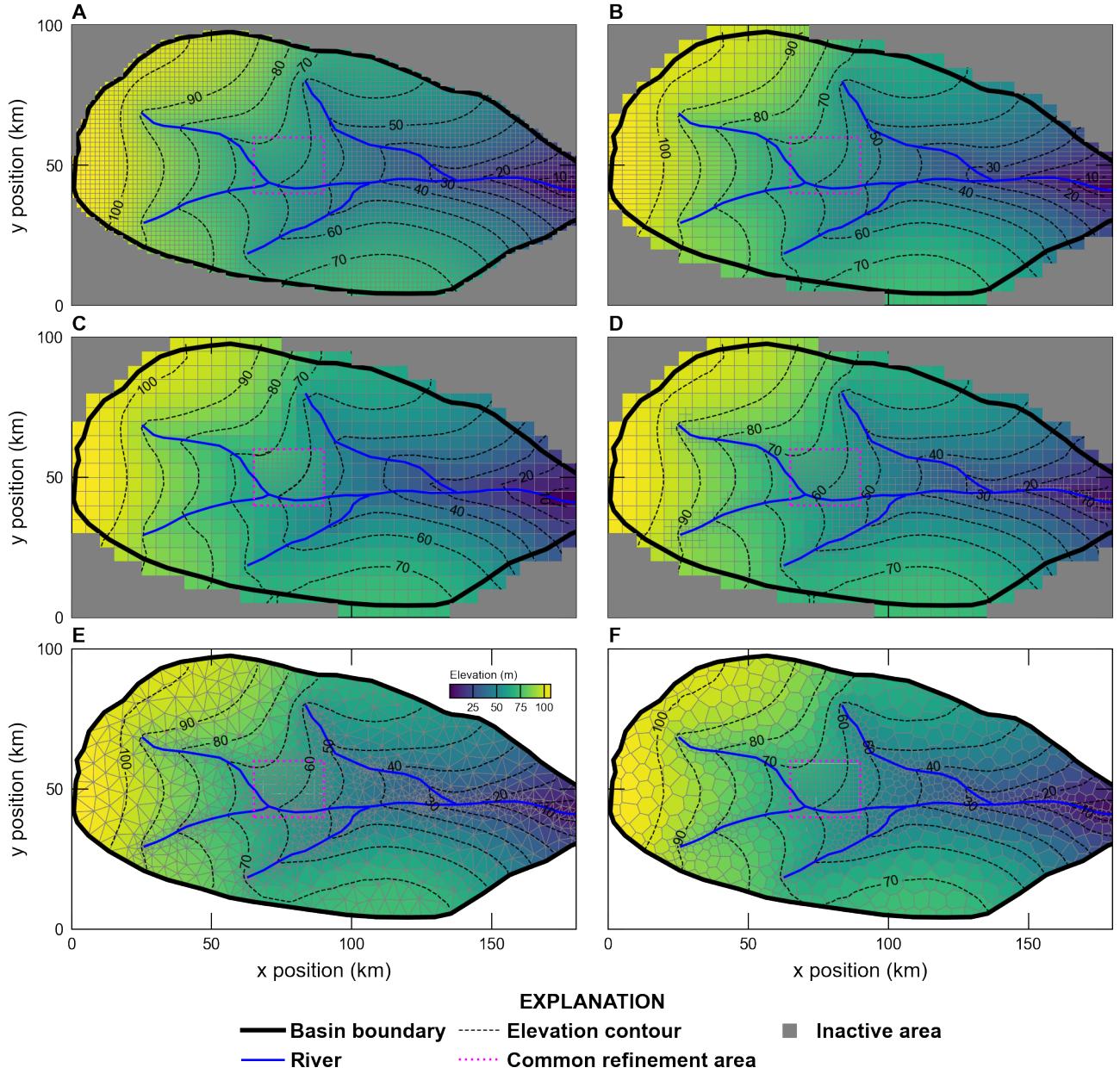


Figure 1: Examples of grids that can be generated and processed using FloPy for a hypothetical watershed, including (A) a regular MODFLOW grid with constant and equal row and column spacings, (B) a regular MODFLOW grid with variable row and column spacings, (C) a regular MODFLOW child grid nested within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

149 and Hill 2006; Vilhelmsen et al. 2012; Mehl and Hill 2013). The use of a locally refined grid  
150 (LGR) within an encompassing parent grid offers computational benefits in that the additional  
151 refinement is targeted to an area of interest. FloPy provides a `Lgr()` utility class for constructing  
152 the data required to tightly couple parent and child models within a single MODFLOW 6  
153 simulation. Figure 1C shows two `StructuredGrid` objects—one object represents the parent  
154 model grid and the other represents the nested child grid. The `Lgr()` utility class defines the  
155 connection properties between cells in the parent model and cells in the child model. Connection  
156 properties consist of distances, areas, and other geometric information needed to calculate flow  
157 between cells in different models. The `Lgr()` utility class is general in that the child model  
158 can have more layers than the parent model. The following Python code shows the steps for  
159 creating the a child `StructuredGrid` object grid using data for the parent grid with the `Lgr()`  
160 utility class.

```
161     >>> idomain_parent = np.ones((nlay_parent, nrow_parent, ncol_parent),  
162         ... dtype=int)  
163     ...  
164     >>> idomain_parent[0, 8:12, 13:18] = 0  
165     >>> ncpp, ncppl = 3, [1]  
166     >>> lgr = Lgr(nlayparent, nrow_parent, ncol_parent,  
167         ... delr_parent, delc_parent, topparent, botmparent,  
168         ... idomain_parent, ncpp=ncpp, ncppl=ncppl, xllp=0.0, yllp=0.0)  
169     ...  
170     >>> delr, delc = lgr.get_delr_delc()  
171     >>> xoff, yoff = lgr.get_lower_left()  
172     >>> regular_gridchild = StructuredGrid(  
173         ... delr=delr, delc=delc, xoff=xoff, yoff=yoff)  
174     ...
```

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

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

184 Layered quadtree grids can be created using the `Gridgen()` utility class, which is a wrapper  
185 around the GRIDGEN program (Lien et al. 2014). GRIDGEN starts with a regular MODFLOW  
186 grid defined by the user. The program then recursively subdivides individual cells that intersect  
187 with refinement features into quarters until a maximum level of refinement is met. Refinement  
188 features may be points, lines, or polygons. Smoothing is automatically handled so that a cell  
189 is connected to no more than two cells in any primary horizontal direction and four cells in the  
190 vertical direction. Figure 1D shows an example of a quadtree grid created with GRIDGEN in  
191 which a base grid is refined two levels along streams and in the child grid area shown in Figure  
192 1C. The following Python code shows the steps for creating the quadtree grid with GRIDGEN.

```
193 >>> sim = flopy.mf6.MFSimulation()  
194 >>> gwf = flopy.mf6.ModflowGwf(sim)  
195 >>> dis6 = flopy.mf6.ModflowGwfdis(gwf, nrow=nrow, ncol=ncol, delr=dy, delc=dx)  
196 >>> g = Gridgen(dis6, model_ws=temp_path)  
197 >>> g.add_refinement_features([[lgr.polygon_xy]], "polygon", 2, range(1))  
198 >>> g.add_refinement_features(stream_points, "line", 2, range(1))  
199 >>> g.build(verbose=False)  
200 >>> gridprops_vg = g.get_gridprops_vertexgrid()  
201 >>> quadtree_grid = flopy.discretization.VertexGrid(**gridprops_vg)
```

202 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  
203 program, and then loads the triangular mesh. Users provide the maximum area for individual  
204 triangles, angle constraints, a polygon describing the model domain, and so forth. Figure 1E  
205 shows an example of a triangular grid created with the Triangle program. The Python code for  
206 creating the triangular grid is shown below.  
207

```
208 >>> tri = flopy.utils.triangle.Triangle(maximum_area=maximum_area,  
209 ... angle=30, nodes=nodes, model_ws=temp_path)  
210 ...  
211 >>> tri.add_polygon(boundary_points)  
212 >>> tri.build(verbose=False)  
213 >>> cell2d = tri.get_cell2d()  
214 >>> vertices = tri.get_vertices()  
215 >>> triangular_grid = VertexGrid(vertices=vertices, cell2d=cell2d,  
216 ... idomain=idomain, nlay=nlay, ncpl=tri.ncpl, top=top, botm=botm)  
217 ...
```

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

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

```
226 >>> vor = flopy.utils.voronoi.VoronoiGrid(tri)
```

```
227     >>> gridprops = vor.get_gridprops_vertexgrid()  
228     >>> voronoi_grid = VertexGrid(**gridprops, nlay=nlay, idomain=idomain)
```

229       The `StructuredGrid`, `VertexGrid`, and `UnstructuredGrid` classes have useful properties  
230      and methods for accessing or mapping locations on the model grid including: (1) converting x,  
231      y pairs from local to global coordinates (`.get_coords()`) and from global to local coordinates  
232      (`.get_local_coords()`); (2) getting x, y, and z coordinates for cell centers (`.xcellcenters`,  
233      `.ycellcenters`, `.zcellcenters`, and `.xyzcellcenters`) and vertices (`.xvertices`, `.yvertices`,  
234      `.zvertices`, and `.xyzvertices`); and (3) intersecting a list of x, y pairs with the grid and re-  
235      turning the appropriate `cellid` (`.intersect()`). Other useful grid class properties and methods  
236      include generating a grid object from a MODFLOW 6 binary grid file (`.from_binary_grid_file()`),  
237      retrieving cell thicknesses (`.thick`), and calculating the saturated thickness for each cell by pass-  
238      ing a head array with dimensions consistent with the grid object (`.saturated_thick(head)`).

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

## 249     **Geospatial Processing**

250       Geospatial processing is often a fundamental part of creating a groundwater model. New geospa-  
251      tial processing functionality has been added to FloPy to help users construct models using data  
252      from common input sources. The geospatial processing functionality has been implemented to

253 work with the different types of model grids so that it is straightforward to build and construct  
254 models with different grid resolutions or grid types. The geospatial processing routines work  
255 with all three of the model grid types (`StructuredGrid`, `VertexGrid`, and `UnstructuredGrid`).

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

```
261     >>> fine_topo = flopy.utils.Raster.load("./grid_data/fine_topo.asc")
262
263     >>> top_vg = fine_topo.resample_to_grid(voronoi_grid, band=fine_topo.bands[0],
264 ...     method="linear", extrapolate_edges=True)
265
266     ...
267
268
269
270
271
272
273
274
275
276
277
278
```

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

272 Performing intersections of hydrologic features with the model grid is another common mod-  
273 eling task. FloPy is now equipped with robust and efficient capabilities for intersecting a model  
274 grid with points, lines, and polygons. The underlying intersection routines rely on the Shapely  
275 Python package ([Gillies 2022](#)) to determine intersection properties. When a point or collection  
276 of points is intersected with a model grid, the grid intersection routine returns the cells that  
277 intersect with the points. When a line or collection of lines is intersected with a model grid,  
278 the grid intersection routine returns the cells that intersect with the lines and the lengths of

279 lines within each intersected cell. The line and grid intersection routine also creates and returns  
280 individual line segments of the line features within each intersected cell. When a polygon or  
281 collection of polygons is intersected with a model grid, the grid intersection routine returns the  
282 cells that intersect with the polygons and the polygon area within the cell. The polygon and  
283 grid intersection routine also creates and returns individual polygons of the original polygon  
284 features within each intersected intersected cell.

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

```
287 >>> ixs = flopy.utils.GridIntersect(voronoi_grid)
288
289 >>> results = []
290
291 >>> for points in segments:
292     ...     segment = ixs.intersect(LineString(points))
293     ...     results.extend(segment["cellids"].tolist())
294
295     ...
296
297
298
```

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

## 299 Processing MODFLOW 6 output

300 MODFLOW 6 has many different types of output that can be created during a simulation. A  
301 GWF Model, for example, can write simulated heads and detailed budget information to binary  
302 files. Global model budgets are written to standard MODFLOW listing (\*.lst) files and can be  
303 written to comma-separated value text files. Some individual GWF and GWT Model advanced

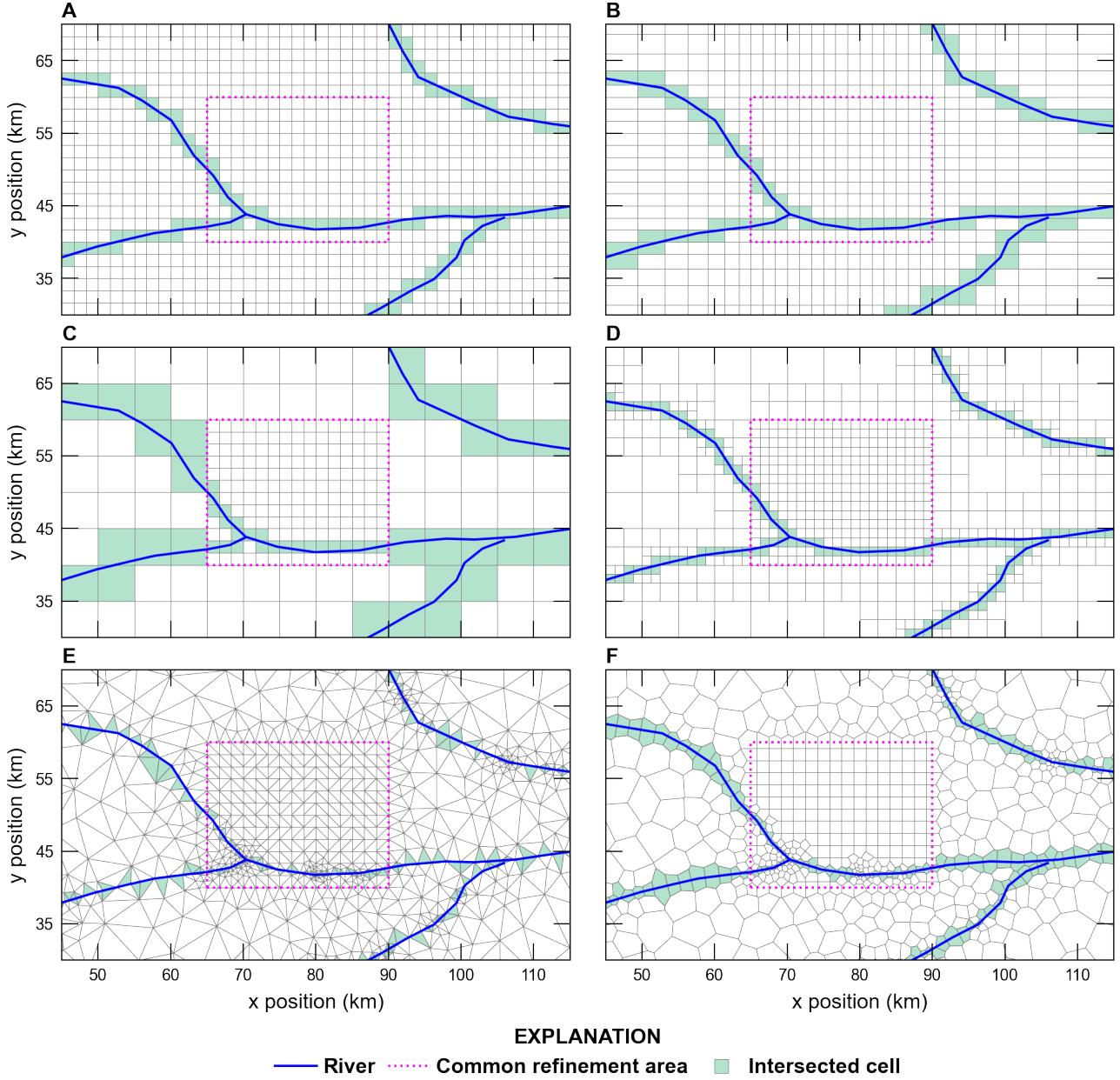


Figure 2: Examples of the intersection of a linear stream network with the model grids shown in Figure 1. Intersections were performed using FloPy for (A) a regular MODFLOW grid, (B) a regular MODFLOW grid with variable row and column spacing, (C) a regular MODFLOW child grid nested within a regular 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.

stress packages can also write simulated output during a simulation. For example, the Lake (LAK) Package can write simulated lake stages and detailed lake budget information to binary

306 files. Likewise, the Multi-Aquifer Well (MAW) Package can write simulated well head and  
307 well budgets to binary files. Recent improvements have been made to FloPy to allow users  
308 easier access to simulation results using `.output` routines available for MODFLOW 6 models  
309 and advanced stress packages. Prior to these improvements, users were required to instantiate  
310 head, concentration, and budget file readers using file paths and names in order to access this  
311 information. With the `.output` routines, the file readers are automatically generated when  
312 called by the user.

313 The following `.methods()` syntax shows how a user can discover the type of output infor-  
314 mation that is available for the specified `gwf` model.

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

317 The `.list()` method can be used to get the incremental (`incremental=True`) or cumulative  
318 budget information from the MODFLOW listing file for the `gwf` model for a user-specified simu-  
319 lation time, zero-based time step and stress period tuple, or zero-based index. The `.zonebudget()`  
320 method allows the user to easily build a Zone Budget for MODFLOW 6 instance, run the Zone  
321 Budget program, and access Zone Budget output. The `.budget()` method gives user access to  
322 data in binary MODFLOW 6 cell-by-cell budget files. The `.budgetcsv()` method gives user  
323 access to cumulative and incremental global budget data saved to a comma separated values  
324 file. The `.head()` method gives user access to data in the binary MODFLOW 6 head file.

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

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

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

335 **Processing simulated dependent-variable data**

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

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

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

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

346 or a zero-based index

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

348 **Processing simulated cell-by-cell budget results**

349 Similar to head output, cell-by-cell budget information can be accessed using FloPy. Unlike the  
350 simulated head file, the cell-by-cell budget file can have data for more than one item and these

351 items may be stored in the file as arrays or lists of data. The data in the cell-by-cell budget file  
352 can be determined using

```
353     >>> gwf.output.budget().list_unique_records()  
354     RECORD           IMETH  
355     -----  
356     FLOW-JA-FACE      1  
357     DATA-SPDIS        6  
358     DATA-SAT          6  
359     WEL                6  
360     DRN                6  
361     RCHA               6  
362     EVTA               6  
363     SFR                6  
364     LAK                6
```

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

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

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

```
374     >>> qx, qy, qz = flopy.utils.postprocessing.get_specific_discharge(spdis, gwf,
```

```
375     ... head=head)
```

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

### 377 Performing zone budgets analyses

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

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

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

## 393 Plotting

394 FloPy plotting capabilities have been refined and updated to support plotting both struc-  
395 tured and unstructured models in map and cross-section view using the `.PlotMapView()` and  
396 `.PlotCrossSection()` classes, respectively. The plotting methods are wrappers around the  
397 matplotlib plotting methods ([Hunter 2007](#)) and allow fine-grained control using matplotlib key-

398 word arguments (`kwargs`). The following Python code demonstrates the steps for plotting a map  
399 of simulated heads, the model grid, the location of drain (DRN) package cells, specific-discharge  
400 vectors, and head contours for the `gwf` model.

```
401 >>> mm = flopy.plot.PlotMapView(model=gwf)
402 >>> mm.plot_array(head, edgecolor="0.5")
403 >>> mm.plot_bc("DRN")
404 >>> mm.plot_grid()
405 >>> cs = mm.contour_array(head)
406 >>> mm.ax.clabel(cs)
407 >>> mm.plot_vector(qx, qy, normalize=True)
408 >>> plt.show()
```

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

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

```
421 >>> fx = flopy.plot.PlotCrossSection(model=gwf,
422 ... line={"line": [(0, 42500), (186801, 42500)]})
```

```
423     ...
424     >>> fx.plot_array(head, head=head)
425     >>> fx.plot_grid()
426     >>> plt.show()
```

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

## 435 Exporting Grid Data to Other Formats

436 Model input and output can be exported in a variety of standard formats using the `.export()`  
437 method, which is available for FloPy model objects, package objects, binary dependent-variable  
438 (`head`, concentration, *etc.*), and cell-by-cell output files. Standard output formats that are cur-  
439 rently supported include shapefiles ([ESRI 1998](#)), NetCDF files ([Rew et al. 2006](#); [Rew and Davis](#)  
440 [1990](#)), and Visualization Tool Kit (VTK) files ([Schroeder et al. 2006](#)). Entire models, pack-  
441 ages, individual package arrays, binary dependent-variables (*e.g.*, heads), or three-dimensional  
442 representations of binary cell-by-cell data can be exported. Shapefile and VTK output can be  
443 exported for all grid types but NetCDF files can currently only be exported for structured grids.  
444 The NetCDF output capability has been used to convert entire models and associated output  
445 so that it can be rendered in the GWWebFlow viewer ([U.S. Geological Survey 2018](#)).

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

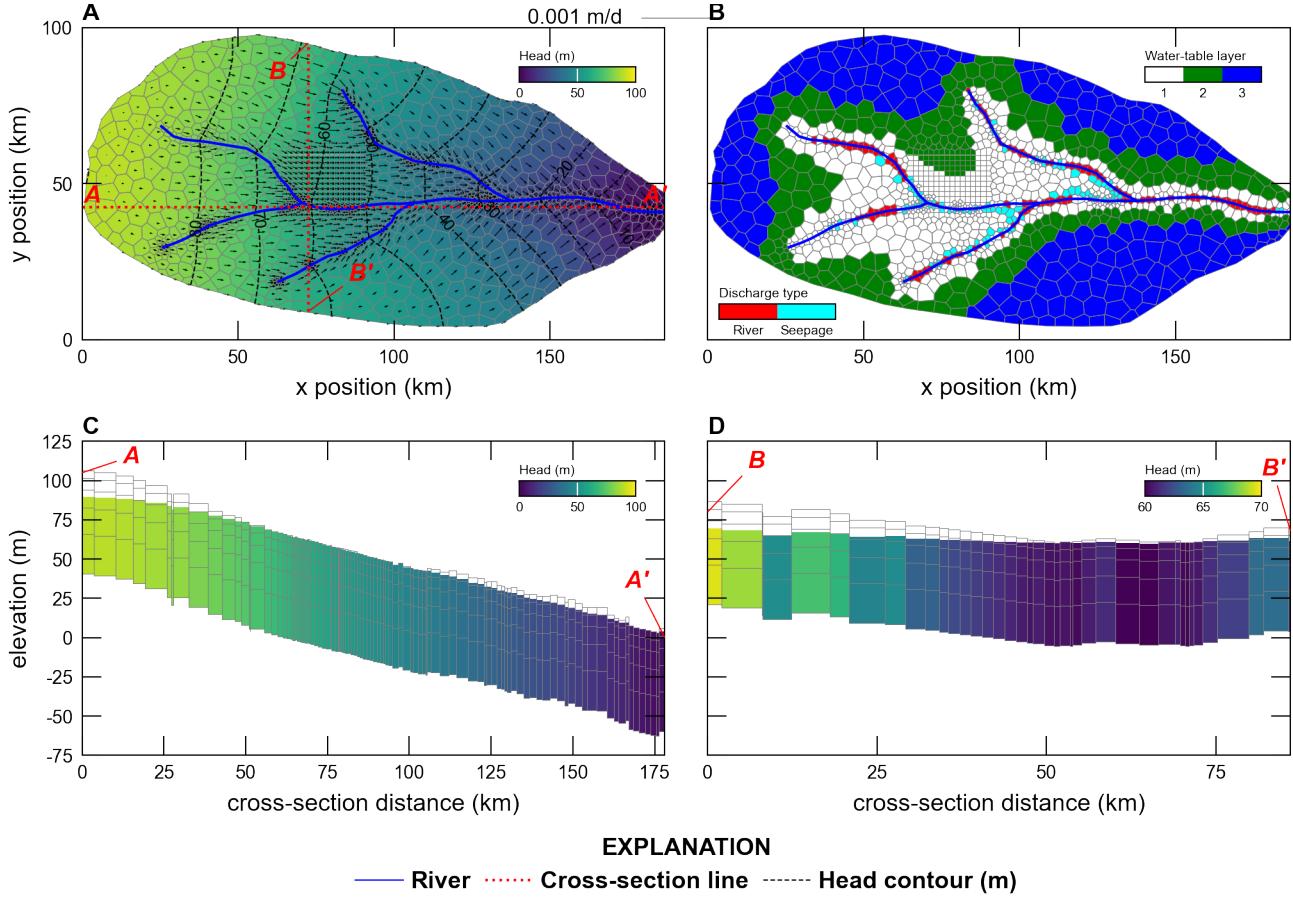


Figure 3: Examples of FloPy map and cross-section plotting capabilities for a model discretized using a Voronoi grid (Figure 1F). (A) Map showing simulated heads and specific-discharge vectors in the upper-most saturated cells. (B) Map showing the layer containing the water table, the location of cells 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.

```

448     >>> gwf.export("temp_vtk/vtk_staircase", fmt='vtk', smooth=False,
449             ... vertical_exaggeration=500.0, pvd=True)
450

```

VTK models can also be exported with smooth cell tops and bottoms using elevations interpolated to the cell vertices (`smooth=True`). Other supported export formats can be created

453 by specifying the file extension to be `.shp` for shapefiles, `.nc` for NetCDF files, or if the `fmt`  
454 keyword is `vtk` (as shown above) for VTK files. Figure 4 shows stair-case and smooth VTK  
455 exports of the model described in the [Plotting](#) section and rendered with Paraview ([Ahrens](#)  
456 [et al. 2005](#)).

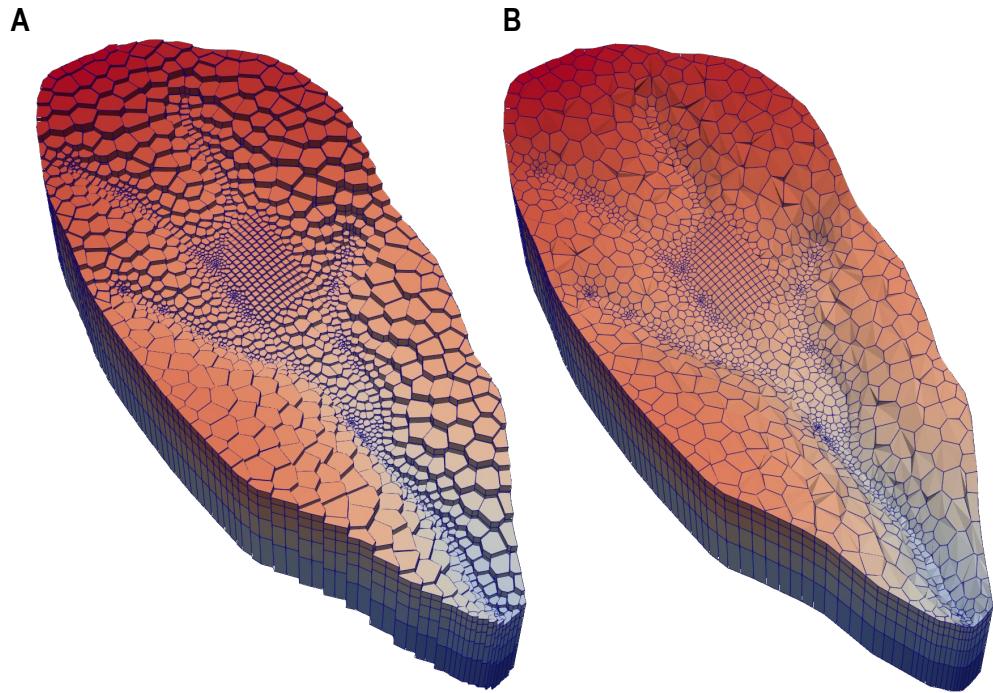


Figure 4: Two different graphical renderings of the Voronoi model grid: (A) stair-cased representation in which cell have flat tops and bottoms and (B) smooth representation in which elevations for cell vertices are interpolated using cell top and bottom elevations. Renderings were created using Paraview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

## 457 **Scripting MODFLOW 6 Model Development Using Python** 458 **and FloPy**

459 In this section, FloPy is used to construct, run, and post process a MODFLOW 6 model.  
460 All pre- and post-processing was done using FloPy grid, geospatial processing, MODFLOW 6  
461 processing, and plotting functionality discussed previously. Figures 5, 6, 7, and 8 were created

462 using a combination of FloPy plotting functionality and matplotlib plotting methods (Hunter  
463 2007). Jupyter notebooks (Kluyver et al. 2016) showing the commands for creating the model  
464 data sets, processing model results, and plotting these figures are available at the Synthetic  
465 Valley internet address indicated in the **Summary and Conclusions** section.

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

## 475 MODFLOW 6 Model Setup

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

482 In this example both groundwater flow and solute transport are simulated. To better rep-  
483 resent solute transport, the lower aquifer has been discretized into 3 layers (instead of one).  
484 Confining units have to be explicitly simulated in MODFLOW 6, therefore, a total of six layers  
485 are simulated. The bottom of layers 1, 2, 3, and 4 were set to constant values of -1.53, -15.24,  
486 -15.55 and -30.48 m, respectively. The `IDOMAIN` concept (Langevin et al. 2017) was used to  
487 eliminate cells in model layer 3 (by setting `IDOMAIN=-1`) where the confining unit does not  
488 exist. In these areas, the thickness of layer 3 was set to zero and `IDOMAIN` was set to -1, which

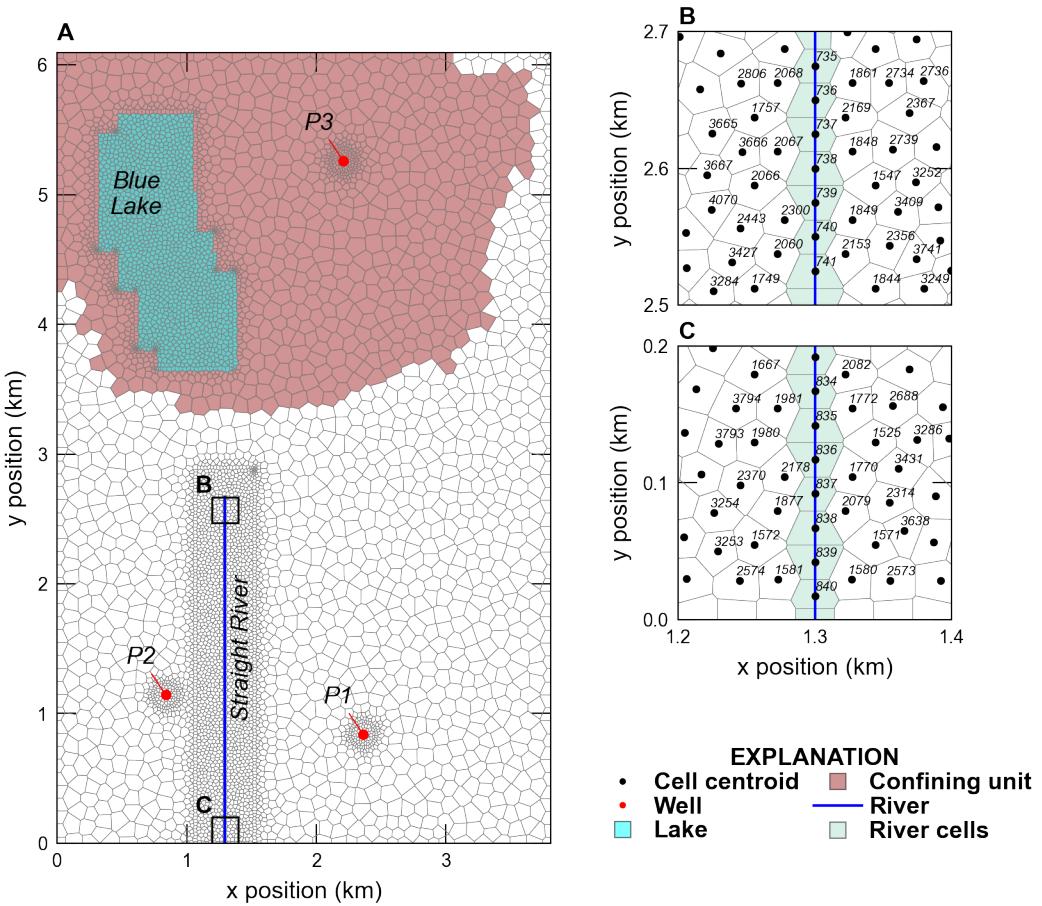


Figure 5: Synthetic Valley model used to demonstrate the MODFLOW 6 capabilities of FloPy. (A) Map showing the Voronoi grid used to discretized the model domain and the location of Blue Lake, Straight River, and the areal extent of the confining unit separating the upper and lower aquifer units. (B) Map 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).

489 marks these cells in layer 3 as “vertical pass through cells” and results in cells in layer 2 being  
 490 directly connected to cells in layer 4.

491 The bottom of the model (layer 6) is based on Hill et al. (1998) and the bottom of layer 5  
 492 was specified to be half the distance between the bottom of layers 4 and 6. The top of the model  
 493 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.

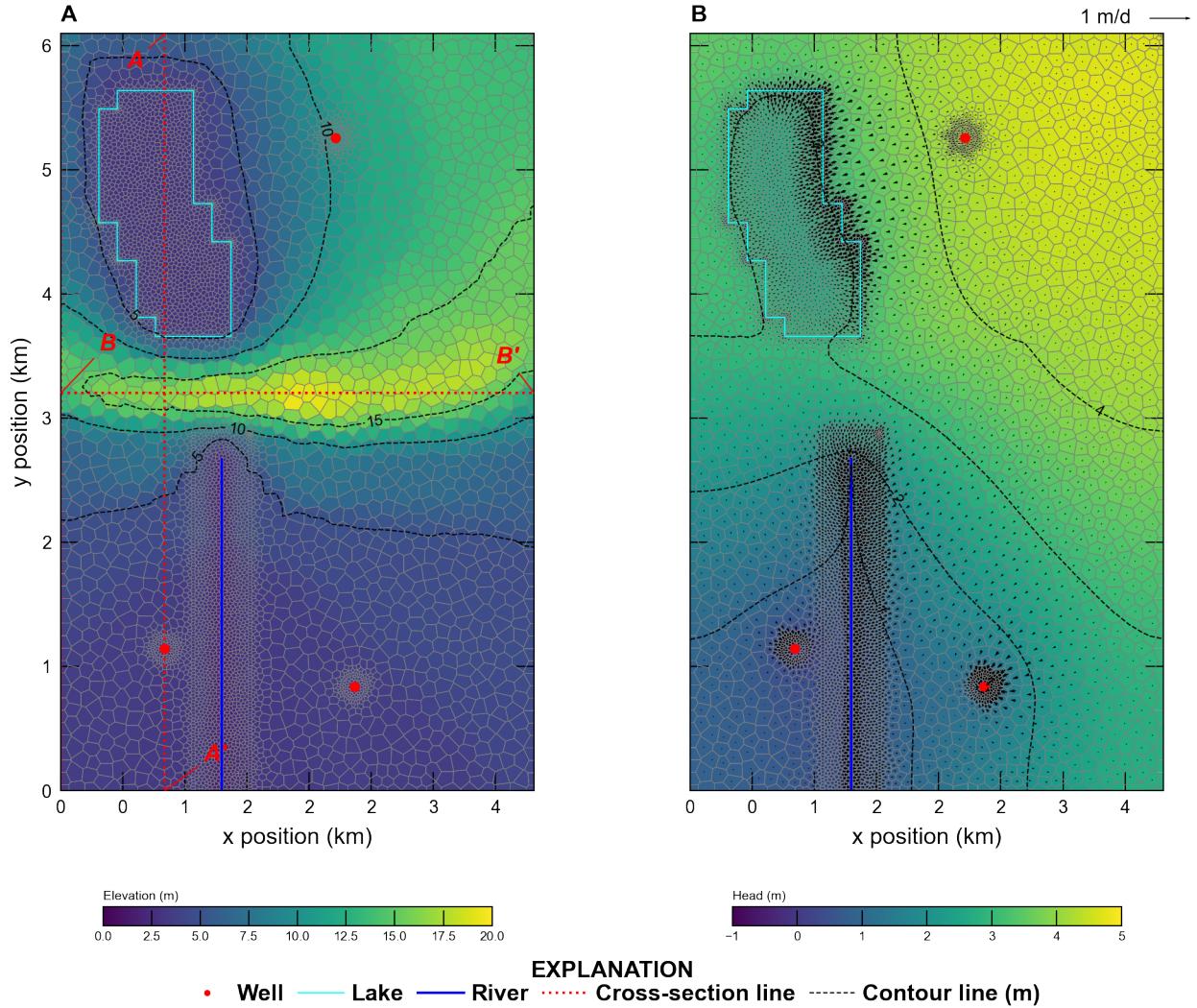


Figure 6: Map showing Synthetic Valley model (A) topography and (B) simulated steady-state heads and specific discharge rates in model layer 1. Cross-section lines A–A' and B–B' shown in Figure 7 are also shown on (A).

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

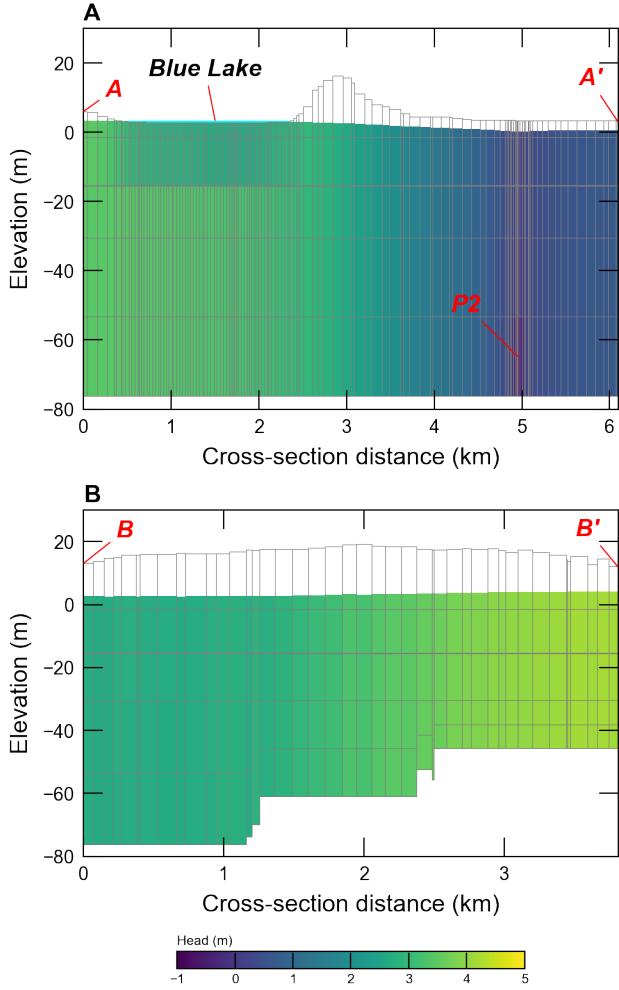


Figure 7: Cross-section of Synthetic Valley model grid and simulated steady-state heads along cross-section line (A) A–A' and (B) B–B'. The simulated Blue Lake steady-state stage (3.46 m) and pumping well P-2 are also shown on (A).

501 hydraulic conductivity was discretized into five zones with values of 45.72, 50.29, 60.96, 83.82,  
 502 and 121.92 m/d; the lowest hydraulic conductivity zone was located south of Blue Lake and  
 503 the highest hydraulic conductivity zone was located beneath Blue Lake. The vertical hydraulic  
 504 conductivity in the upper and lower aquifer was specified to be one quarter of the horizontal  
 505 hydraulic conductivity. The horizontal and vertical hydraulic conductivity in the confining  
 506 unit was set equal to  $9.14 \times 10^{-4}$  m/d. The horizontal and vertical hydraulic conductivity were  
 507 resampled from the data used in the structured grid model using the `.resample_to_grid()`  
 508 method on the `VertexGrid modelgrid` for the model and the nearest neighbor algorithm.

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

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

520 Straight River was discretized into 108 SFR reaches. Cells that intersect the northern and  
521 southern end of Straight River are shown in Figures 5B and C. The bed thickness and width  
522 of each SFR reach was specified to be 0.3048 and 3.048 m, respectively. The leakance for each  
523 SFR reach was calculated using the bed thickness, reach width, and reach length in each cell  
524 and based on a total Straight River conductance of 50,971.72 m<sup>2</sup>/d. A specified rainfall rate of  
525 0.0025 m/d and a potential evaporation rate of 0.0019 m/d was defined for each Straight River  
526 reach.

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

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

537 water levels were greater than or equal to the top of the model.

538 Areal specified recharge and potential evapotranspiration rates of 0.0025 and 0.0019 m/d  
539 were specified using the recharge (RCH) and evapotranspiration (EVT) packages, respectively.  
540 The EVT surface was specified to be the top of the model and the EVT extinction depth was  
541 specified to be 1 m.

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

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

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

## 550 Simulated Results

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

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

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

562 Simulated concentrations at the end of 30-years in all six model layers are shown in Figure 8.  
563 Simulated concentrations are highest beneath Blue Lake in model layer 1 and do not vary much

564 in model layers 1 and 2. Simulated concentrations in model layer 3 are limited to the extent  
565 of the confining unit because the remaining cells in the layer are defined to be vertical pass  
566 through cells (`IDOMAIN=-1`). The lateral extent of the solute plume does not vary much south  
567 of Blue Lake because of the lack of confinement in these areas.

## 568 Summary and Conclusions

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

- 574 • FloPy supports the creation of many different types of groundwater models, including  
575 models that use MODFLOW 6, MODFLOW-2005, MODFLOW-NWT, MODFLOW-  
576 USG, MT3D, MT3D-USGS, and SEAWAT. FloPy support for MODFLOW 6 is based  
577 on an entirely new approach designed to automatically support all MODFLOW 6 models,  
578 packages, and options. The underlying FloPy classes for MODFLOW 6 are program-  
579 matically generated from the same input definition files that are used to construct the  
580 MODFLOW 6 user guide. This correspondence ensures that the FloPy classes are consis-  
581 tent and in-sync with MODFLOW 6 input.
- 582 • FloPy has been extended to support unstructured model grids in addition to regular grids  
583 defined by layers, rows, and columns. FloPy has several different routines for creating  
584 unstructured grids. FloPy includes a wrapper for the GRIDGEN program (Lien et al.  
585 2014), which can be used to create layered quadtree grids. FloPy also includes a wrapper  
586 for the Triangle program (Shewchuk 1996), which can be used to create triangular meshes.  
587 A triangular mesh can be converted by FloPy into a Voronoi grid. Grid information  
588 is stored for each FloPy model created by the user. This model grid object is used

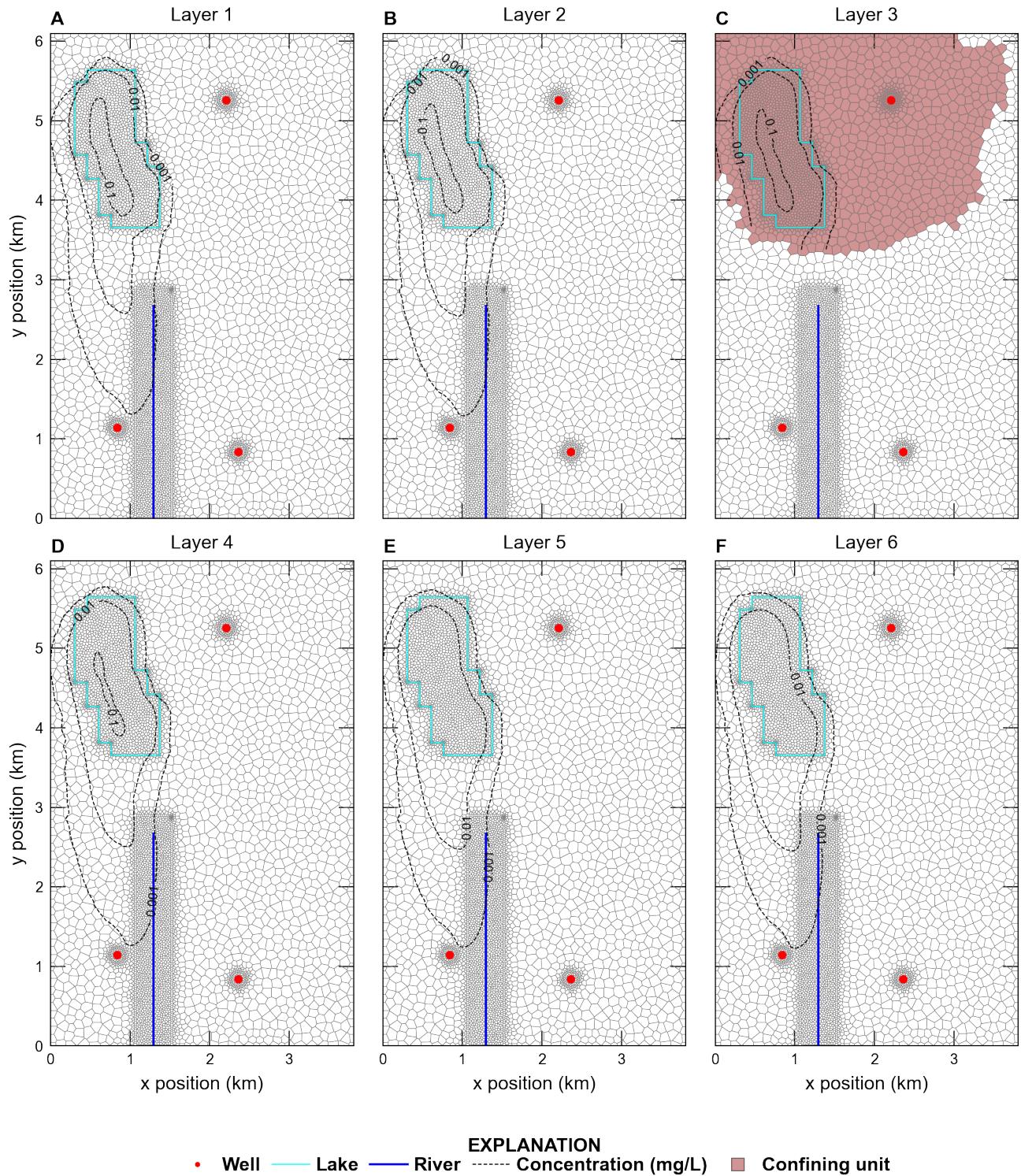


Figure 8: Maps showing Synthetic Valley simulated concentrations at the end of 30 years in model layer (A) 1, (B) 2, (C) 3, (D) 4, (E) 5, and (F) 6. The extent of the confining unit in model layer 3 is also shown on (C).

589 systemically throughout FloPy for geospatial operations, plotting, and exporting model  
590 information to supported formats.

- 591 • Geospatial intersections of points, lines, and polygons with model grids and raster resam-  
592 pling onto model grids are common steps in model construction. FloPy fully supports  
593 these geospatial operations through its grid intersection and raster resampling routines.
- 594 • Access to model output using FloPy has been simplified for MODFLOW 6 models. The  
595 new output access routines makes it possible to quickly extract simulated results from  
596 binary and text model output files.
- 597 • FloPy supports plan-view map and cross-section plotting of model grids, boundary con-  
598 ditions, and simulated results. These plotting routines work with structured and unstruc-  
599 tured models and can be customized to produce high quality figures.
- 600 • FloPy supports the export of model information to shapefiles, VTK files, and NetCDF files.  
601 These exported files can then be loaded into other software programs, such as geographic  
602 information systems or advanced visualization programs for additional processing.

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

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

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

## 621 Acknowledgments

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

## 626 Authors' Note

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

## 628 Disclaimer

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

## 631 References

- 632 Ahrens, J., B. Geveci, and C. Law. 2005. ParaView: An End-User Tool for Large Data Visual-  
633 ization. *Visualization Handbook*. Elsevier.
- 634 Bakker, M., V. Post, C.D. Langevin, J.D. Hughes, J. White, J. Starn, and M.N. Fienen. 2016.

- 635 Scripting MODFLOW model development using Python and FloPy. *Groundwater* 54, no. 5:  
636 733–739, <https://doi.org/10.1111/gwat.12413>.
- 637 Befus, K.M., P.L. Barnard, D.J. Hoover, J.A. Finzi Hart, and C.I. Voss. 2020. Increasing threat  
638 of coastal groundwater hazards from sea-level rise in California. *Nature Climate Change* 10,  
639 no. 10: 946–952, <https://doi.org/10.1038/s41558-020-0874-1>.
- 640 Befus, K.M., K.D. Kroeger, C.G. Smith, and P.W. Swarzenski. 2017. The Magnitude and Origin  
641 of Groundwater Discharge to Eastern U.S. and Gulf of Mexico Coastal Waters. *Geophysical*  
642 *Research Letters* 44, no. 20: 10,396–10,406, <https://doi.org/10.1002/2017GL075238>.
- 643 Burek, P., Y. Satoh, T. Kahil, T. Tang, P. Greve, M. Smilovic, L. Guillaumot, F. Zhao, and  
644 Y. Wada. 2020. Development of the Community Water Model (CWatM v1.04) – a high-  
645 resolution hydrological model for global and regional assessment of integrated water resources  
646 management. *Geoscientific Model Development* 13, no. 7: 3267–3298, <https://doi.org/10.5194/gmd-13-3267-2020>.
- 648 Ebeling, P., F. Handel, and M. Walther. 2019. Potential of mixed hydraulic barriers to remediate  
649 seawater intrusion. *Science of The Total Environment* 693: 133478, <https://doi.org/10.1016/j.scitotenv.2019.07.284>.
- 651 ESRI. 1998. ESRI Shapefile Technical Description, an ESRI white paper. <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf> (accessed August 29, 2022).
- 653 Essawy, B.T., J.L. Goodall, W. Zell, D. Voce, M.M. Morsy, J. Sadler, Z. Yuan, and T. Malik.  
654 2018. Integrating scientific cyberinfrastructures to improve reproducibility in computational  
655 hydrology: Example for HydroShare and GeoTrust. *Environmental Modelling & Software* 105:  
656 217–229, <https://doi.org/10.1016/j.envsoft.2018.03.025>.
- 657 Fienen, M.N. and M. Bakker. 2016. HESS Opinions: Repeatable research: what hydrologists  
658 can learn from the Duke cancer research scandal. *Hydrology and Earth System Sciences* 20,  
659 no. 9: 3739–3743, <https://doi.org/10.5194/hess-20-3739-2016>.

- 660 Fienen, M.N., M.J. Haserodt, A.T. Leaf, and S.M. Westenbroek. 2022. Simulation of regional  
661 groundwater flow and groundwater/lake interactions in the Central Sands, Wisconsin. Scien-  
662 tific Investigations Report 2022-5046, 111 p. <https://doi.org/10.3133/sir20225046>.
- 663 Gillies, S.. 2022. The shapely user manual. [https://shapely.readthedocs.io/en/stable/  
664 manual.html](https://shapely.readthedocs.io/en/stable/manual.html) (accessed August 28, 2022).
- 665 Gillies, S. et al.. 2013. Rasterio: geospatial raster I/O for Python programmers. <https://github.com/rasterio/rasterio> (accessed October 6, 2022).
- 666 Guira, M.. 2018. Numerical Modeling Of The Effects Of Land Use Change And Irrigation  
667 On Streamflow Depletion Of Frenchman Creek, Nebraska. Master's thesis, University of  
668 Nebraska, Lincoln, NE.
- 669 Hill, M.C.. 1992. A computer program (MODFLOWP) for estimating parameters of a tran-  
670 sient, three-dimensional, ground-water flow model using nonlinear regression. U.S. Geological  
671 Survey Open-File Report 91-484, 358 p.
- 672 Hill, M.C., R.L. Cooley, and D.W. Pollock. 1998. A Controlled Experiment in Ground Water  
673 Flow Model Calibration. *Groundwater* 36, no. 3: 520–535, <https://doi.org/10.1111/j.1745-6584.1998.tb02824.x>.
- 674 Hughes, J.D., C.D. Langevin, and E.R. Banta. 2017. Documentation for the MODFLOW  
675 framework. U.S. Geological Survey Techniques and Methods, book 6, chap. A57, 36 p.  
<https://doi.org/10.3133/tm6A57>.
- 676 Hughes, J.D., S.A. Leake, D.L. Galloway, and J.W. White. 2022. Documentation for the Skeletal  
677 Storage, Compaction, and Subsidence (CSUB) Package of MODFLOW 6. U.S. Geological Sur-  
678 vey Techniques and Methods, book 6, chap. A62, 57 p. <https://doi.org/10.3133/tm6A62>.
- 682 Hunter, J.D.. 2007. Matplotlib: A 2D graphics environment. *Computing in science & engineer-  
683 ing* 9, no. 03: 90–95.

- 684 Jaxa-Rozen, M., J.H. Kwakkel, and M. Bloemendal. 2019. A coupled simulation architecture  
685 for agent-based/geohydrological modelling with NetLogo and MODFLOW. *Environmental*  
686 *Modelling & Software* 115: 19–37, <https://doi.org/10.1016/j.envsoft.2019.01.020>.
- 687 Kluyver, T., B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley,  
688 J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. 2016. Jupyter  
689 Notebooks – a publishing format for reproducible computational workflows. In F. Loizides  
690 and B. Schmidt (Eds.), *Positioning and Power in Academic Publishing: Players, Agents and*  
691 *Agendas*, pp. 87 – 90. IOS Press, <https://doi.org/10.3233/978-1-61499-649-1-87>.
- 692 Knowling, M.J., J.T. White, and C.R. Moore. 2019. Role of model parameterization in risk-  
693 based decision support: An empirical exploration. *Advances in Water Resources* 128: 59–73,  
694 <https://doi.org/10.1016/j.advwatres.2019.04.010>.
- 695 Langevin, C.D., J.D. Hughes, A.M. Provost, E.R. Banta, R.G. Niswonger, and S. Panday. 2017.  
696 Documentation for the MODFLOW 6 Groundwater Flow (GWF) Model. U.S. Geological  
697 Survey Techniques and Methods, book 6, chap. A55, 197 p. <https://doi.org/10.3133/tm6A55>.
- 698
- 699 Langevin, C.D., S. Panday, and A.M. Provost. 2020. Hydraulic-Head Formulation for Density-  
700 Dependent Flow and Transport. *Groundwater* 58, no. 3: 349–362.
- 701 Langevin, C.D., A.M. Provost, S. Panday, and J.D. Hughes. 2022. Documentation for the  
702 MODFLOW 6 Groundwater Transport (GWT) Model. U.S. Geological Survey Techniques  
703 and Methods, book 6, chap. A61, 56 p. <https://doi.org/10.3133/tm6A55>.
- 704 Leaf, A.T. and M.N. Fienen. 2022. Modflow-setup: Robust automation of groundwater model  
705 construction. *Frontiers in Earth Science* 10: 903965, <https://doi.org/10.3389/feart.2022.903965>.
- 706
- 707 Lien, J.M., G. Liu, and C.D. Langevin. 2014. GRIDGEN Version 1.0: A computer program

- 708 for generating unstructured finite-volume grids. U.S. Geological Survey Open-File Report  
709 2014–1109, 26 p. <https://doi.org/10.3133/ofr20141109>.
- 710 Mancewicz, L.K., A. Mayer, C.D. Langevin, and J. Gulley. 2022. Improved method for simulating  
711 groundwater inundation using the MODFLOW 6 Lake Transport Package. *Groundwater*,  
712 <https://doi.org/10.1111/gwat.13254>.
- 713 Mehl, S.W. and M.C. Hill. 2006. MODFLOW-2005, the US Geological Survey modular ground-  
714 water model-documentation of shared node local grid refinement (LGR) and the boundary  
715 flow and head (BFH) package. U.S. Geological Survey Techniques and Methods, book 6,  
716 chap. A12, 78 p. <https://doi.org/10.3133/tm6A12>.
- 717 Mehl, S.W. and M.C. Hill. 2013. MODFLOW-LGR—Documentation of ghost node local grid  
718 refinement (LGR2) for multiple areas and the boundary flow and head (BFH2) package. U.S.  
719 Geological Survey Techniques and Methods, book 6, chap. A44, 43 p. [https://doi.org/10.](https://doi.org/10.3133/tm6A44)  
720 [3133/tm6A44](https://doi.org/10.3133/tm6A44).
- 721 Morway, E.D., C.D. Langevin, and J.D. Hughes. 2021. Use of the MODFLOW 6 water mover  
722 package to represent natural and managed hydrologic connections. *Groundwater* 59, no. 6:  
723 913–924, <https://doi.org/10.1111/gwat.13117>.
- 724 Panday, S., C.D. Langevin, R.G. Niswonger, M. Ibaraki, and J.D. Hughes. 2013. MODFLOW-  
725 USG version 1—An unstructured grid version of MODFLOW for simulating groundwater  
726 flow and tightly coupled processes using a control volume finite-difference formulation. U.S.  
727 Geological Survey Techniques and Methods, book 6, chap. A45, 66 p.
- 728 Pollock, D.W.. 2014. Personal communication. Reston, VA.
- 729 Provost, A.M., C.D. Langevin, and J.D. Hughes. 2017. Documentation for the “XT3D” Op-  
730 tion in the Node Property Flow (NPF) Package of MODFLOW 6. U.S. Geological Survey  
731 Techniques and Methods, book 6, chap. A56, 46 p. <https://doi.org/10.3133/tm6A56>.

- 732 Rew, R. and G. Davis. 1990. NetCDF: an interface for scientific data access. *IEEE computer*  
733       *graphics and applications* 10, no. 4: 76–82.
- 734 Rew, R., E. Hartnett, J. Caron, et al.. 2006. NetCDF-4: Software implementing an enhanced  
735       data model for the geosciences. In *22nd International Conference on Interactive Information*  
736       *Processing Systems for Meteorology, Oceanograph, and Hydrology*, Volume 6.
- 737 Rossetto, R., G. De Filippis, I. Borsi, L. Foglia, M. Cannata, R. Criollo, and E. Vázquez-Suñé.  
738       2018. Integrating free and open source tools and distributed modelling codes in GIS environ-  
739       ment for data-based groundwater management. *Environmental Modelling & Software* 107:  
740       210–230, <https://doi.org/10.1016/j.envsoft.2018.06.007>.
- 741 Schroeder, W., K. Martin, and B. Lorensen. 2006. *The Visualization Toolkit* (4th ed.). Kitware.
- 742 Shewchuk, J.R.. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Tri-  
743       angulator. In *Applied Computational Geometry, Towards Geometric Engineering, FCRC'96*  
744       *Workshop, WACG'96, Philadelphia, PA, USA, May 27-28, 1996, Selected Papers*, pp. 203–  
745       222. <https://doi.org/10.1007/BFb0014497>.
- 746 Starn, J.J. and K. Belitz. 2018. Regionalization of Groundwater Residence Time Using Meta-  
747       modeling. *Water Resources Research* 54, no. 9: 6357–6373, <https://doi.org/10.1029/2017WR021531>.
- 749 Sun, A.Y.. 2018. Discovering State-Parameter Mappings in Subsurface Models Using Generative  
750       Adversarial Networks. *Geophysical Research Letters* 45, no. 20: 11,137–11,146, <https://doi.org/10.1029/2018GL080404>.
- 752 U.S. Geological Survey. 2018. GWWebFlow—a browser-based groundwater model viewer. <https://webapps.usgs.gov/gwwebflow/>.
- 754 van Engelen, J., G.H. Oude Essink, H. Kooi, and M.F. Bierkens. 2018. On the origins of  
755       hypersaline groundwater in the Nile Delta aquifer. *Journal of Hydrology* 560: 301–317, <https://doi.org/10.1016/j.jhydrol.2018.03.029>.

- 757 Vilhelmsen, T.N., S. Christensen, and S.W. Mehl. 2012. Evaluation of MODFLOW-LGR in  
758 connection with a synthetic regional-scale model. *Groundwater* 50, no. 1: 118–132.
- 759 Virtanen, P., R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau,  
760 E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson,  
761 K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat,  
762 Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A.  
763 Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, and  
764 SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing  
765 in Python. *Nature Methods* 17: 261–272, <https://doi.org/10.1038/s41592-019-0686-2>.
- 766 White, J.T.. 2018. A model-independent iterative ensemble smoother for efficient history-  
767 matching and uncertainty quantification in very high dimensions. *Environmental Modelling  
768 & Software* 109: 191–201, <https://doi.org/10.1016/j.envsoft.2018.06.009>.
- 769 Zhou, Z. and D.M. Tartakovsky. 2021. Markov chain Monte Carlo with neural net-  
770 work surrogates: application to contaminant source identification. *Stochastic Environ-  
771 mental Research and Risk Assessment* 35, no. 10: 639–651, <https://doi.org/10.1007/s00477-020-01888-9>.
- 773 Zipper, S.C., T. Gleeson, B. Kerr, J.K. Howard, M.M. Rohde, J. Carah, and J. Zimmerman.  
774 2019. Rapid and Accurate Estimates of Streamflow Depletion Caused by Groundwater Pump-  
775 ing Using Analytical Depletion Functions. *Water Resources Research* 55, no. 7: 5807–5829,  
776 <https://doi.org/10.1029/2018WR024403>.