

Research Paper /

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

⁴ Joseph D. Hughes^{1,*}, Christian D. Langevin², Scott R. Paulinski³, Joshua D.
⁵ Larsen⁴, and David Brakenhoff⁵

⁶ ¹U.S. Geological Survey, Model Support and Maintenance Branch, 927 W Belle Plaine Ave, Chicago,
⁷ IL, USA

⁸ ²U.S. Geological Survey, Model Support and Maintenance Branch, 2280 Woodale Dr, Mounds View,
⁹ MN, USA

¹⁰ U.S. Geological Survey, California Water Science Center, 3130 Skyway Drive, Suite 602, Santa
¹¹ Maria, CA, USA

¹² U.S. Geological Survey, California Water Science Center, 6000 J Street, Placer Hall, Sacramento,
¹³ CA, USA

⁵Artesia Water, Korte Weistraat 12, Schoonhoven, Netherlands

*Corresponding author jdhughes@usgs.gov

April 11, 2023

Abstract

18 FloPy is a ~~popular~~ Python package for creating, running, and post-processing
19 MODFLOW-based groundwater flow and transport models. FloPy functionality has
20 expanded to support the latest version of MODFLOW (MODFLOW 6) including
21 support for unstructured grids. FloPy can be used to download MODFLOW-based and

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

35 Introduction

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

45 FloPy has been used to pioneer new methods and analysis tools, such as deep learning
46 approaches for improving groundwater model calibration (Sun 2018; Zhou and Tartakovsky
47 2021), regionalizing residence times using metamodeling (Starn and Belitz 2018), applying
48 iterative ensemble approaches for calibration and uncertainty quantification (White 2018), and

49 exploring alternative parameterization schemes for risk analysis (Knowling et al. 2019). There
50 are numerous examples of constructing MODFLOW models with FloPy to solve applied
51 groundwater problems (Befus et al. 2017; van Engelen et al. 2018; Ebeling et al. 2019; Zipper
52 et al. 2019; Befus et al. 2020). FloPy is being used in other software and workflows to improve
53 repeatability and robustness through automated model construction (White et al. 2020;
54 Fienen et al. 2022; Larsen et al. 2022; Leaf and Fienen 2022). FloPy is also used in GIS-based
55 tools, such as FREEWAT (Rossetto et al. 2018) and other cyberinfrastructures (Essawy et al.
56 2018) to export models into MODFLOW datasets. FloPy can also be used as the “glue” to
57 help couple MODFLOW to other hydrological models (Burek et al. 2020) or, for example, to
58 agent-based models designed to quantify the effects of decision makers on environmental
59 behavior (Jaxa-Rozen et al. 2019).

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

71 The purpose of this paper is to highlight FloPy new functionality for creating and
72 constructing structured and unstructured MODFLOW models. We provide examples that
73 demonstrate these new capabilities, and reinforce the advantages of the modern scripting
74 workflow for developing reproducible structured and unstructured MODFLOW groundwater
75 flow and transport models that can be easily updated as new data become available. The
76 examples also demonstrate workflows that develop different model grids for the same model

77 domain. The important advances described here include (1) complete support for all models,
78 packages, and options implemented in the core version of MODFLOW supported by the U.S.
79 Geological Survey (Hughes et al. 2017; Langevin et al. 2017; Provost et al. 2017; Langevin
80 et al. 2020; Morway et al. 2021; Langevin et al. 2022; Hughes et al. 2022; Mancewicz et al.
81 2022); (2) generalized support for models based on a structured grid consisting of layers, rows,
82 and columns, and also for models based on unstructured grids; (3) implementation of new
83 geoprocessing capabilities to rapidly populate models with data from a variety of input
84 sources; (4) simplified access to model results; (5) plotting capabilities for map and
85 cross-section views of model data; and (6) export capabilities for writing model data to a
86 variety of output formats.

87 FloPy Support for MODFLOW 6

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

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

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

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

122 Common Modeling Tasks

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

127 Getting MODFLOW and Other Related Executables

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

```
133 get-modflow :flopy
```

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

140 Managing and Creating Model Grids

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

150 Structured MODFLOW grids can have constant row and column spacings, as shown in
151 Figure 1A, or they can have variable row and column spacings to focus resolution around an

152 area of interest, as shown in Figure 1B. The following Python code shows how to create a
153 `StructuredGrid` object in FloPy. A `StructuredGrid` object can also be created from
154 discretization data required when instantiating a MODFLOW 6 DIS object using
155 `flopy.mf6.ModflowGwfdis()`.

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

158 MODFLOW 6 was developed to ~~natively~~ support multi-model simulations (Hughes et al.
159 2017). One form of multi-model simulation is a nested grid application in which a more finely
160 discretized child model is embedded within a more coarsely discretized parent model (Mehl
161 and Hill 2006; Vilhelmsen et al. 2012; Mehl and Hill 2013; Fienen et al. 2022). The use of a
162 locally refined grid (LGR) within an encompassing parent grid offers computational benefits in
163 that the additional refinement is targeted to an area of interest. FloPy provides a `Lgr()`
164 utility class for constructing the data required to tightly couple parent and child models
165 within a single MODFLOW 6 simulation. Figure 1C shows two `StructuredGrid` objects—one
166 object represents the parent model grid and the other represents the nested child grid. The
167 `Lgr()` utility class defines the connection properties between cells in the parent model and
168 cells in the child model. Connection properties consist of distances, areas, and other geometric
169 information needed to calculate flow between cells in different models. The `Lgr()` utility class
170 is general in that the child model can have more layers than the parent model. The following
171 Python code shows the steps for creating a child `StructuredGrid` object using data for the
172 parent grid with the `Lgr()` utility class.

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

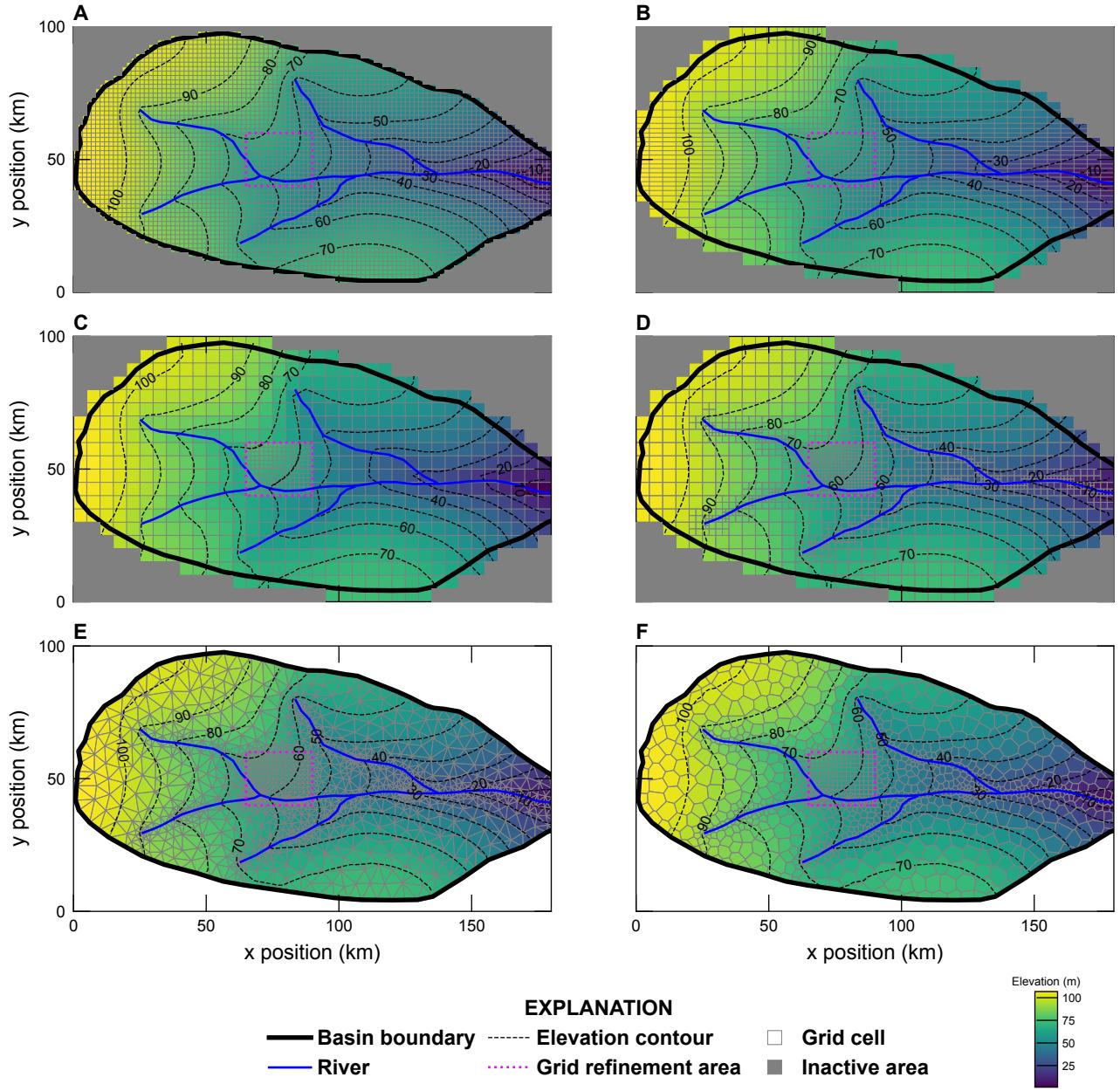


Figure 1: Examples of grids that can be generated and processed using FloPy for a hypothetical watershed, including (A) a structured MODFLOW grid with constant and equal row and column spacings, (B) a structured MODFLOW grid with variable row and column spacings, (C) a structured MODFLOW child grid nested within a structured MODFLOW parent grid, (D) a quadtree grid generated with the GRIDGEN program (Lien et al. 2014) through the FloPy wrapper, (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).

```

177     delr.parent, delc.parent, topparent, botparent,
178     idomain.parent, ncpp=ncpp, ncpl=ncpl,
179     xlfp=0.0, ylfp=0.0)
180
180     delr, delc = lgr.get_delr_delc()
181
181     xoff, yoff = lgr.get_lower_left()
182
182     structured_gridchild = StructuredGrid(delr=delr, delc=delc,
183
183             xoff=xoff, yoff=yoff)

```

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

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

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

```

203 sim = flopy.mf6.MFSimulation()
204 gwf = flopy.mf6.ModflowGwf(sim)
205 dis6 = flopy.mf6.ModflowGwfdis(gwf, nrow=nrow, ncol=ncol, delr=dy, delc=dx)
206 g = Gridgen(dis6, model_ws=temp_path)
207 g.add_refinement_features([[lgr_polygon_xy]], "polygon", 2, range(1))
208 g.add_refinement_features(stream_points, "line", 2, range(1))
209 g.build(verbose=False)
210 gridprops_vg = g.get_gridprops_vertexgrid()
211 quadtree_grid = flopy.discretization.VertexGrid(**gridprops_vg)

```

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

```

218 tri = flopy.utils.triangle.Triangle(maximum_area=maximum_area,
219                                         angle=30, nodes=refinement_verts,
220                                         model_ws=temp_path)
221 tri.add_polygon(boundary_points)
222 tri.build(verbose=False)
223 cell2d = tri.get_cell2d()
224 vertices = tri.get_vertices()
225 triangular_grid = VertexGrid(vertices=vertices, cell2d=cell2d,
226                               idomain=idomain, nlay=nlay, ncpl=tri.ncpl,
227                               top=top, botm=botm)

```

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

230 A triangular grid can be converted by FloPy into a Voronoi grid using the
231 `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
233 1F shows an example of a Voronoi grid created from the triangular mesh shown in Figure 1D.
234 The steps for creating the Voronoi grid from the previously created `Triangle()` object (`tri`)
235 are shown below.

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

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

253 The new FloPy capabilities for generating and testing different types of model grids

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

265 Geospatial Processing

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

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

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

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

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

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

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

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

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

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

315 Processing MODFLOW 6 output

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

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

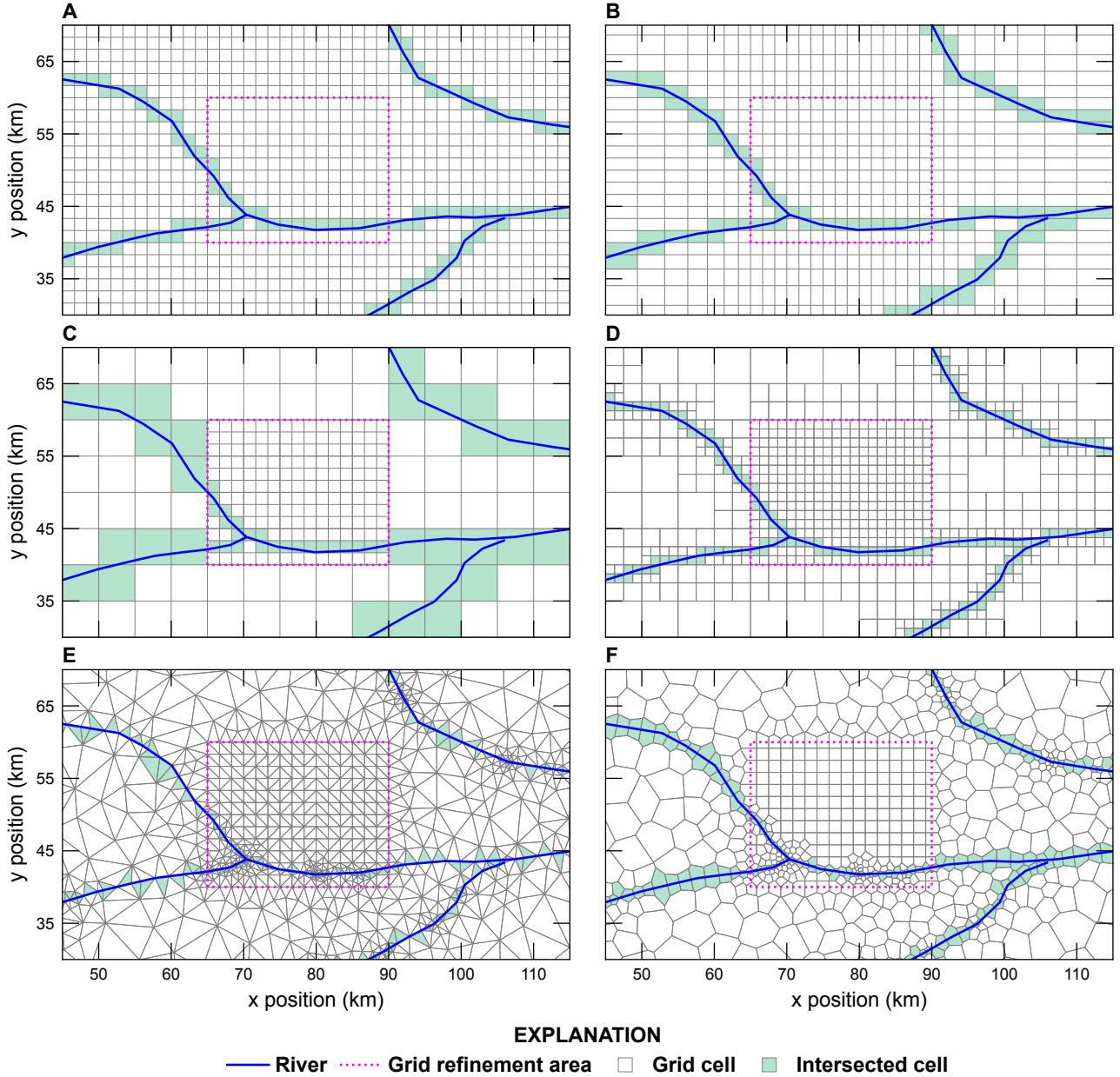


Figure 2: Examples of the intersection of a linear stream network with the model grids shown in Figure 1. Intersections were performed using FloPy for (A) a structured MODFLOW grid, (B) a structured MODFLOW grid with variable row and column spacing, (C) a structured MODFLOW child grid nested within a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

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

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

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

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

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

355 Processing simulated dependent variables

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

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

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

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

366 or a zero-based index

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

368 Processing simulated cell-by-cell budgets

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

```
373     >>> gwf.output.budget().list_unique_records()
```

```
374     RECORD           IMETH
```

375

```
376 FLOW-JA-FACE      1  
377 DATA-SPDIS        6  
378 DATA-SAT          6  
379 WEL               6  
380 DRN               6  
381 RCHA              6  
382 EVTA              6  
383 SFR               6  
384 LAK               6
```

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

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

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

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

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

397 **Performing zone budget analyses**

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

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

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

413 **Plotting**

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

```

421 mm = flopy.plot.PlotMapView(model=gwf)
422 mm.plot_array(head, edgecolor="0.5")
423 mm.plot_bc("DRN")
424 mm.plot_grid()
425 cs = mm.contour_array(head)
426 mm.ax.clabel(cs)
427 mm.plot_vector(qx, qy, normalize=True)
428 plt.show()

```

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

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

```

442 fx = flopy.plot.PlotCrossSection(model=gwf,
443                                     line={"line": [(0, 42500), (186801, 42500)]})
444 fx.plot_array(head, head=head)
445 fx.plot_grid()
446 plt.show()

```

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

455 Exporting Grid Data to Other Formats

456 Model input and output can be exported in a variety of standard formats using the `.export()`
457 method, which is available for FloPy model objects, package objects, binary
458 dependent-variable files (head, concentration, *etc.*), and cell-by-cell output files. Standard
459 output formats that are currently supported include shapefiles ([ESRI 1998](#)), NetCDF files
460 ([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
462 (*e.g.*, heads), or three-dimensional representations of binary cell-by-cell data can be exported.
463 Shapefile and VTK output can be exported for all grid types, but currently, NetCDF output
464 can only be exported for structured grids. The NetCDF output capability has been used to
465 convert entire models and associated output so that it can be rendered in the GWWebFlow
466 viewer ([U.S. Geological Survey 2018](#)).

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

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

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

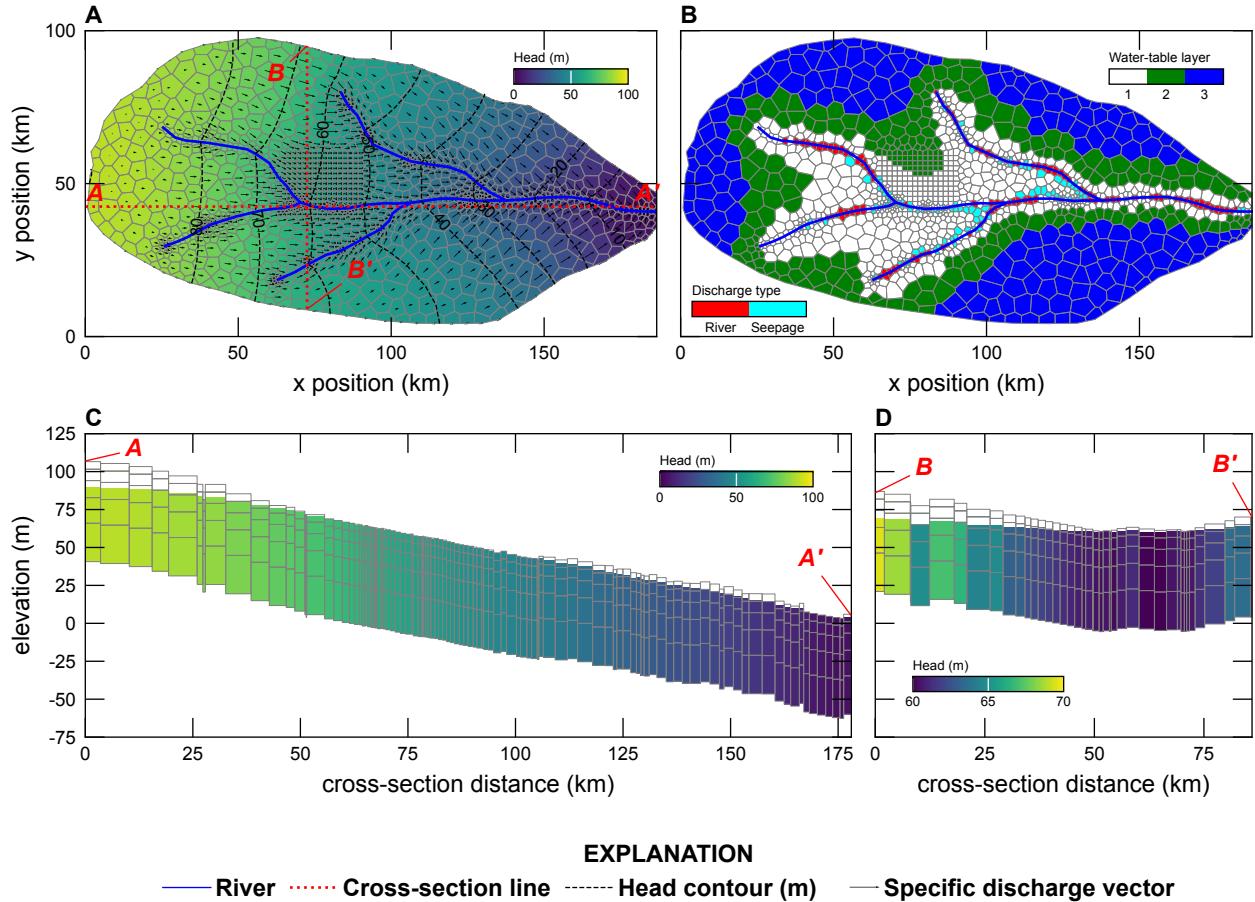


Figure 3: Examples of FloPy map and cross-section plotting capabilities for a model discretized using a Voronoi grid (Figure 1F). (A) Map showing simulated heads and specific-discharge vectors in the upper-most saturated cells. (B) Map showing the layer containing the water table, the location of cells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

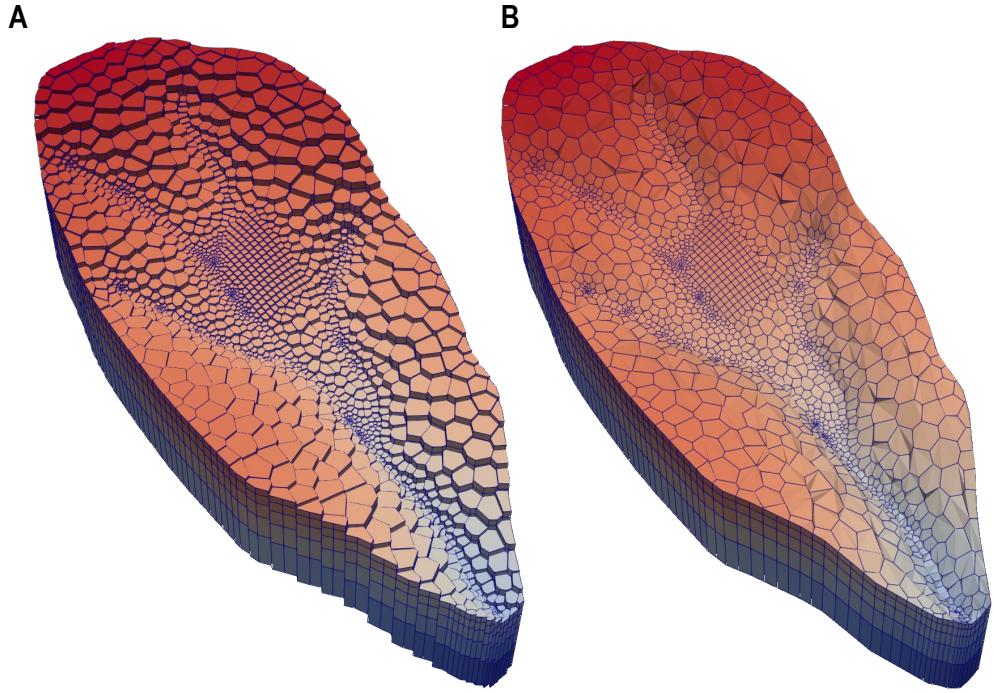


Figure 4: Two different graphical renderings of the Voronoi model grid: (A) stair-cased representation in which cell have flat tops and bottoms and (B) smooth representation in which elevations for cell vertices are interpolated using cell top and bottom elevations. Renderings were created using Paraview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

477 Scripting MODFLOW 6 Model Development Using 478 Python and FloPy

479 In this section, FloPy is used to construct, run, and post process a MODFLOW 6 model. All
480 pre- and post-processing was done using FloPy grid, geospatial processing, MODFLOW 6
481 processing, and plotting functionality discussed previously. Figures 5, 6, 7, and 8 were created
482 using a combination of FloPy plotting functionality and `matplotlib` plotting methods (Hunter
483 2007). Jupyter notebooks (Kluyver et al. 2016) showing the commands for creating the model
484 data sets, processing model results, and plotting these figures are available at the Synthetic
485 Valley internet address indicated in the [Summary and Conclusions](#) section.

486 Hill et al. (1998) present a synthetic test case (Synthetic Valley) of an undeveloped
487 alluvial valley surrounded by low permeability bedrock. The model includes the Blue Lake
488 and Straight River surface water features (Figure 5A). The model in Hill et al. (1998) was
489 calibrated and simulated using MODFLOWP (Hill 1992) using a structured grid with a
490 constant 152.4 m grid spacing, three model layers, and 1,000 active cells per layer. The upper
491 two layers represent an unconfined aquifer, and the third layer represents a lower aquifer unit
492 that is separated from the overlying aquifer by a confining unit in the northern part of the
493 model domain (Figure 5A). The confining unit was not explicitly represented by Hill et al.
494 (1998); instead a quasi-3D approach (low vertical conductance) between layers 2 and 3 was
495 used to represent the confining unit.

496 MODFLOW 6 Model Setup

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

503 In this example both groundwater flow (Langevin et al. 2017) and solute transport
504 (Langevin et al. 2022) are simulated. To better represent solute transport, the lower aquifer
505 has been discretized into 3 layers (instead of one). Confining units have to be explicitly
506 simulated in MODFLOW 6, therefore, a total of six layers are simulated. The bottom of layers
507 1, 2, 3, and 4 were set to constant values of -1.53, -15.24, -15.55 and -30.48 m, respectively.
508 Model layer 3 represents the confining unit and is relatively thin (0.3 m). The `IDOMAIN`
509 concept (Langevin et al. 2017) was used to eliminate cells in model layer 3 (by setting
510 `IDOMAIN=-1`) where the confining unit does not exist. In these areas, the thickness of layer 3
511 was set to zero and `IDOMAIN` was set to -1, which marks these cells in layer 3 as “vertical pass
512 through cells” and results in cells in layer 2 being directly connected to cells in layer 4.

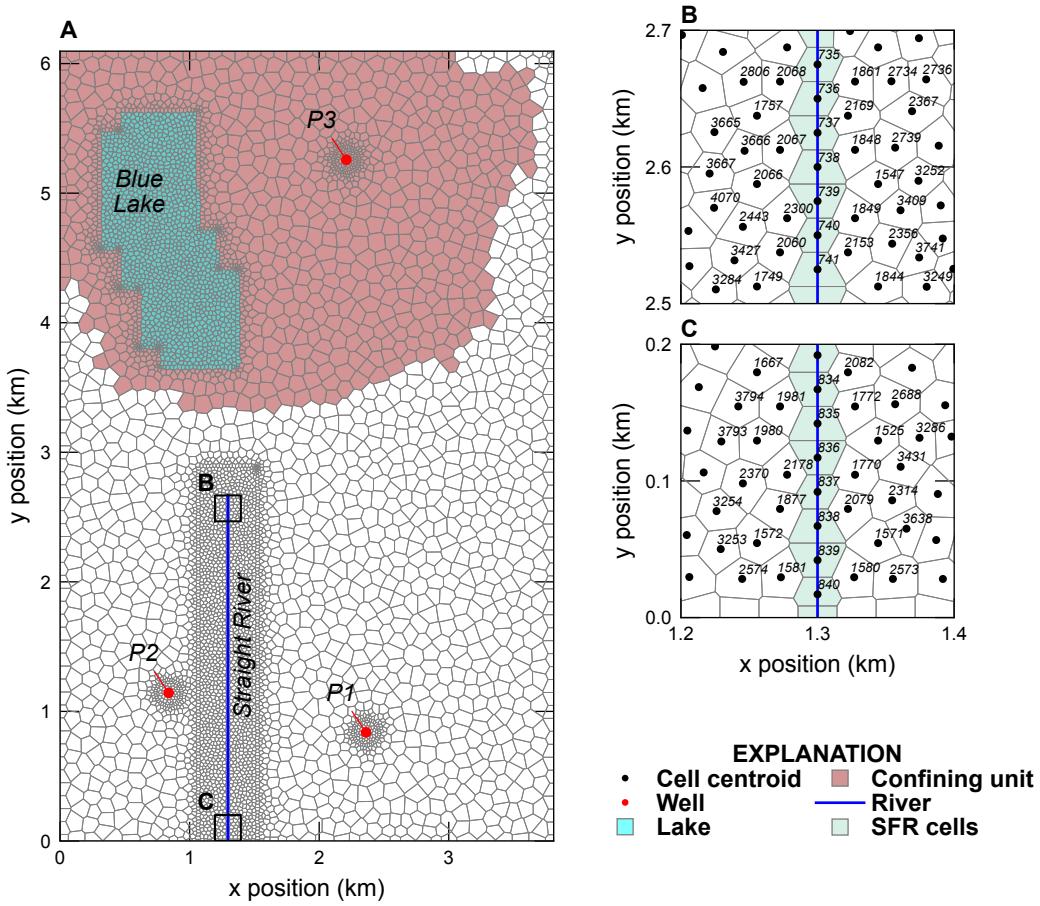


Figure 5: Synthetic Valley model used to demonstrate the MODFLOW 6 capabilities of FloPy. (A) Map showing the Voronoi grid used to discretized the model domain and the location of Blue Lake, Straight River, and the areal extent of the confining unit separating the upper and lower aquifer units. (B) Map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

513 The bottom of the model (layer 6) is based on Hill et al. (1998) and the bottom of layer 5
 514 was specified to be half the distance between the bottom of layers 4 and 6. The top of the
 515 model was developed from topographic contours developed for the model that was used as the
 516 starting point for Hill et al. (1998) (Pollock 2014); the top of the model is shown in Figure 6A.
 517 The top of the model and the bottom of layer 6 were resampled from the data used in the

518 structured grid model using the `.resample_to_grid()` method available on the `VertexGrid`
 519 ~~for the model~~ and linear interpolation. Figure 7 shows the vertical discretization along
 520 cross-section lines A–A' and B–B', which are shown in Figure 6A.

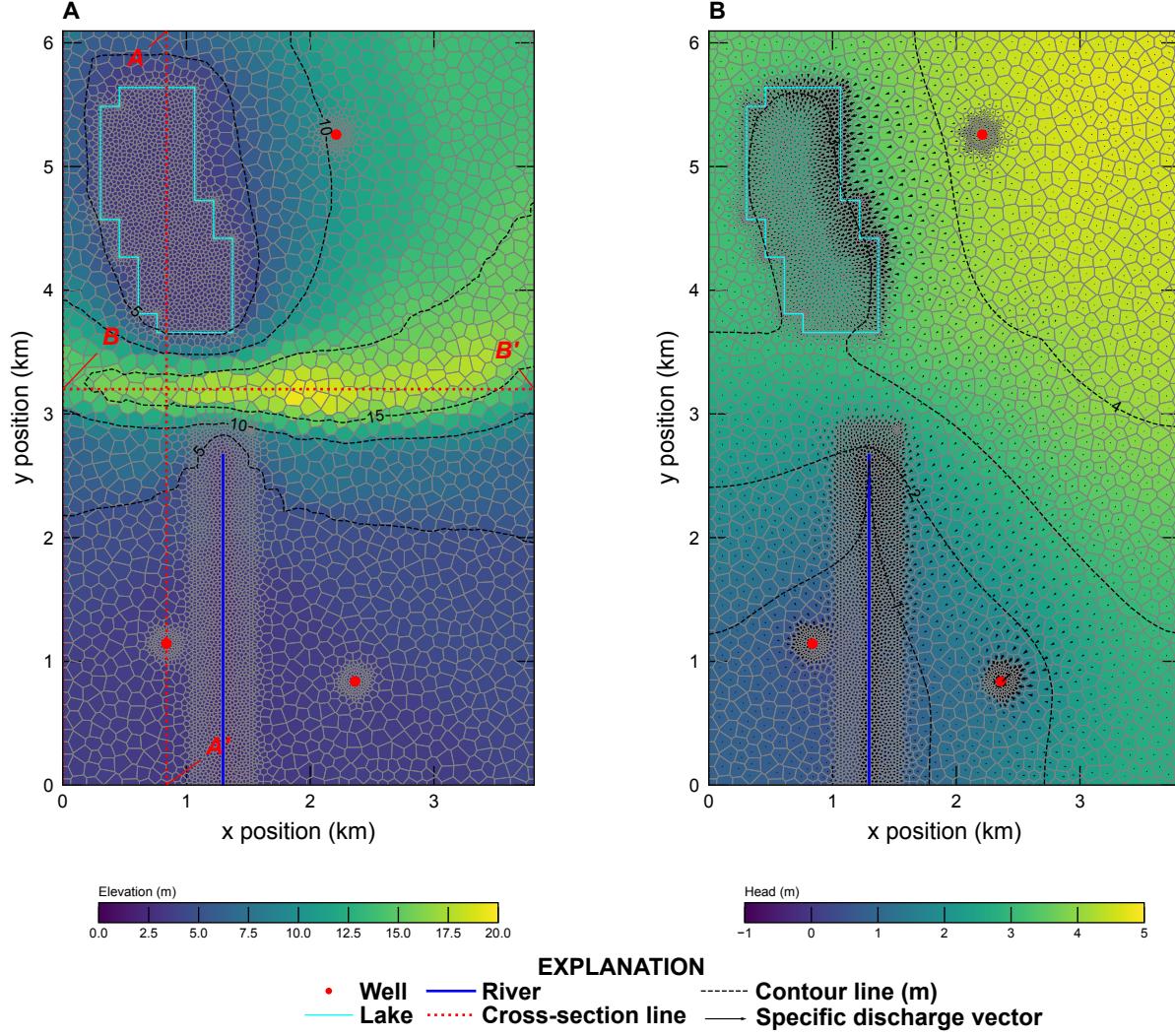


Figure 6: Map showing Synthetic Valley model (A) topography and (B) simulated steady-state heads and specific discharge rates in model layer 1. Cross-section lines A–A' and B–B' shown in Figure 7 are also shown on (A).

521 Hydraulic properties for the model were resampled from the data used in the structured
 522 grid model that was used as the starting point for Hill et al. (1998) (Pollock 2014). The
 523 horizontal hydraulic conductivity was discretized into five zones with values of 45.72, 50.29,
 524 60.96, 83.82, and 121.92 m/d; the lowest hydraulic conductivity zone was located south of

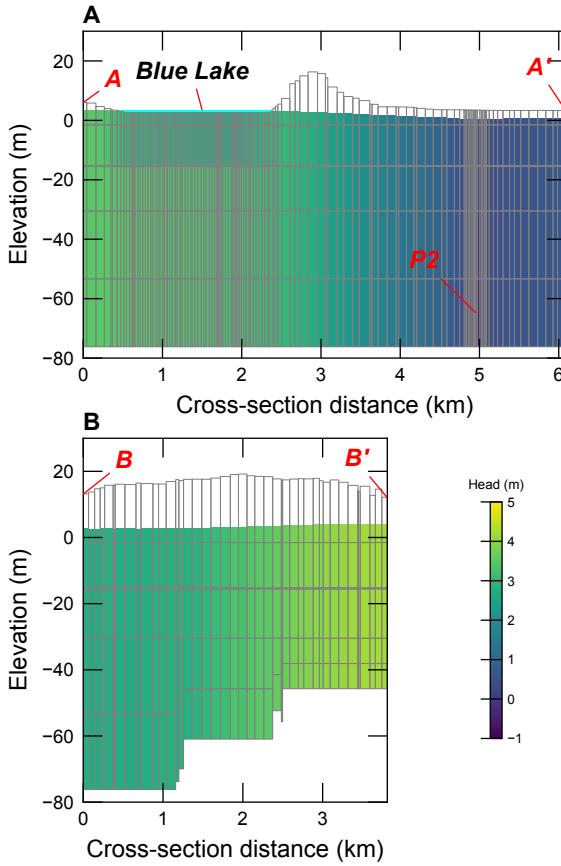


Figure 7: Cross-section of Synthetic Valley model grid and simulated steady-state heads along cross-section line (A) A–A' and (B) B–B'. The simulated Blue Lake steady-state stage (3.46 m) and pumping well P-2 are also shown on (A).

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

532 For the groundwater transport model, the porosity was set to 0.2 in the upper and lower

533 aquifer and 0.4 for cells in the confining unit. For the transport model, the Total Variation
534 Diminishing scheme was used to simulate advection. Dispersion was simulated using a
535 longitudinal dispersivity of 75 m and a transverse dispersivity of 7.5 m. Molecular diffusion
536 was not represented.

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

543 Straight River was discretized into 108 SFR reaches. Cells that intersect the northern and
544 southern end of Straight River are shown in Figures 5B and C. The bed thickness and width
545 of each SFR reach was specified to be 0.3048 and 3.048 m, respectively. The leakance for each
546 SFR reach was calculated using the bed thickness, reach width, and reach length in each cell
547 and based on a total Straight River conductance of 50,971.72 m²/d. A specified rainfall rate of
548 0.0025 m/d and a potential evaporation rate of 0.0019 m/d was defined for each Straight River
549 reach.

550 Blue Lake was simulated as a lake on top of the model grid and only had vertical
551 connections to 1,406 cells in the underlying upper aquifer (model layer 1). A bed leakance of
552 0.0013 1/d was specified for each cell connected to Blue Lake. A specified rainfall rate of
553 0.0025 m/d and a potential evaporation rate of 0.0019 m/d was defined for Blue Lake.

554 Drain (DRN) cells were specified in each cell in model layer 1 that was not connected to
555 Blue Lake to prevent water levels from exceeding the top of the model. The conductance of
556 each DRN cell was based on the horizontal cell area, a thickness of 0.3048 m, and a vertical
557 hydraulic conductivity of 0.03048 m/d. Linear scaling of the drainage conductance was
558 applied to improve model convergence and ranged from 0 m²/d when groundwater levels were
559 greater than or equal to 1 m below the top of the model to the specified conductance when
560 groundwater water levels were greater than or equal to the top of the model.

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

565 The location of pumping wells P1, P2, and P3 were determined using

566 `GridIntersect().intersect()` FloPy functionality (Figure 5A). Pumping rates of -7,600,
567 -7,600, and -1,900 m³/d were specified for pumping wells P1, P2, and P3, respectively.

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

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

573 Simulated Results

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

577 The groundwater flow and transport models were run for a total of 30 years. The
578 groundwater flow model used a single steady-state time step and groundwater flow results
579 were used to run the transport model with a total of 360 time steps with a constant length of
580 30.4375 days.

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

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

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

592 Summary and Conclusions

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

- 598 • FloPy supports the creation of many different types of groundwater models, including
599 models that use MODFLOW 6, MODFLOW-2005, MODFLOW-NWT,
600 MODFLOW-USG, MT3D, MT3D-USGS, and SEAWAT. FloPy support for MODFLOW
601 6 is based on an entirely new approach designed to automatically support all
602 MODFLOW 6 models, packages, and options. The underlying FloPy classes for
603 MODFLOW 6 are programmatically generated from the same input definition files that
604 are used to construct the MODFLOW 6 user guide. This correspondence ensures that
605 the FloPy classes are consistent and in-sync with MODFLOW 6 input.
- 606 • FloPy has been extended to support unstructured model grids in addition to structured
607 grids defined by layers, rows, and columns. FloPy has several different routines for
608 creating unstructured grids. FloPy includes a wrapper for the GRIDGEN program (Lien
609 et al. 2014), which can be used to create layered quadtree grids. FloPy also includes a
610 wrapper for the Triangle program (Shewchuk 1996), which can be used to create
611 triangular meshes. A triangular mesh can be converted by FloPy into a Voronoi grid.
612 Grid information is stored for each FloPy model created by the user. This model grid

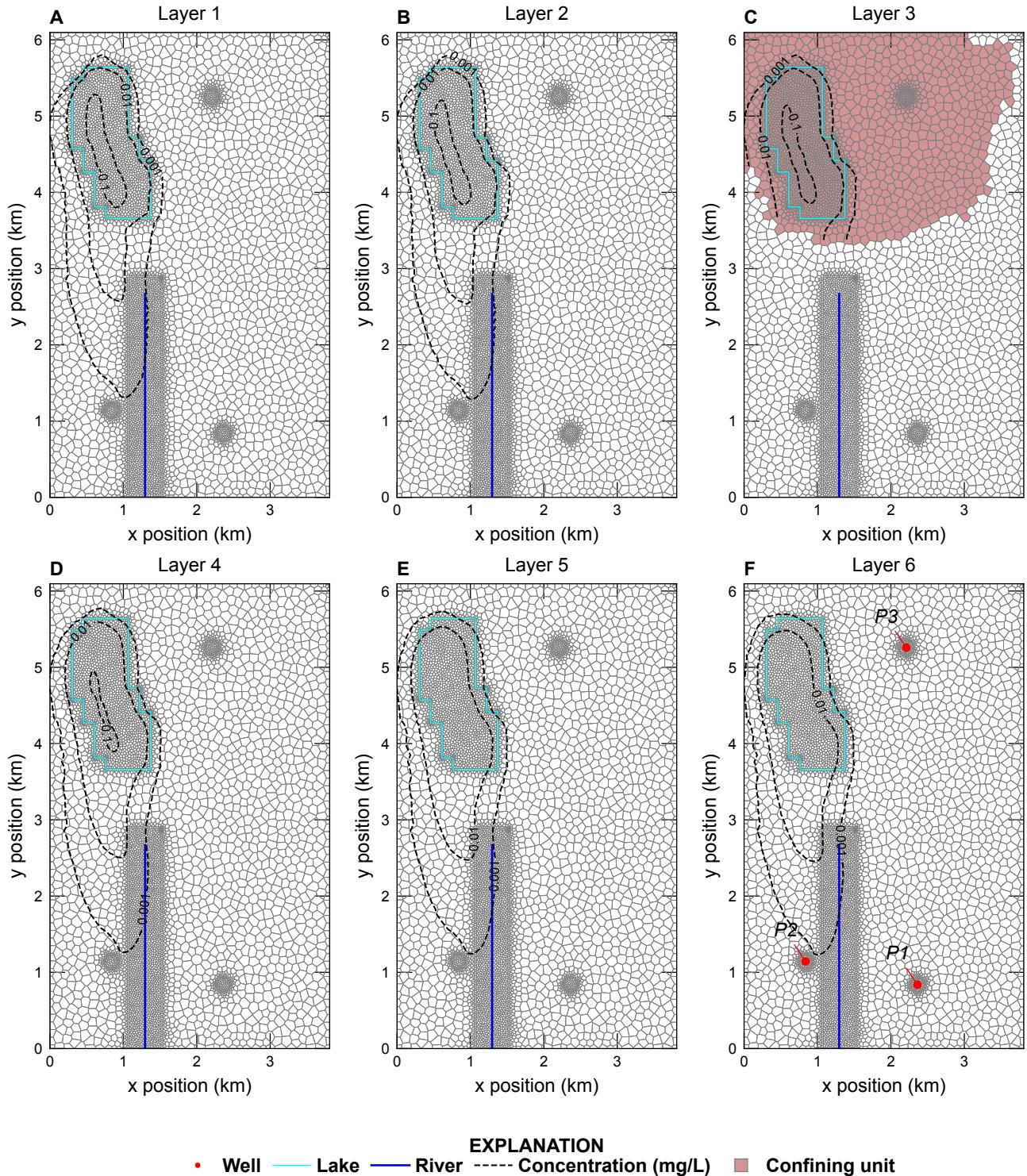


Figure 8: Maps showing Synthetic Valley simulated concentrations at the end of 30 years in model layer (A) 1, (B) 2, (C) 3, (D) 4, (E) 5, and (F) 6. The extent of the confining unit in model layer 3 is also shown on (C).

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

- 615 • Geospatial intersections of points, lines, and polygons with model grids and raster
616 resampling onto model grids are common steps in model construction. FloPy fully
617 supports these geospatial operations through its grid intersection and raster resampling
618 routines.
- 619 • Access to model output using FloPy has been simplified for MODFLOW 6 models. The
620 new output access routines makes it possible to quickly extract simulated results from
621 binary and text model output files.
- 622 • FloPy supports plan-view map and cross-section plotting of model grids, boundary
623 conditions, and simulated results. These plotting routines work with structured and
624 unstructured models and can be customized to produce high quality figures.
- 625 • FloPy supports the export of model information to shapefiles, VTK files, and NetCDF
626 files. These exported files can then be loaded into other software programs, such as
627 geographic information systems or advanced visualization programs for additional
628 processing.

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

633 The ability to script groundwater model construction and post-processing increases
634 robustness, ensures reproducibility, provides a record of the data processing and model
635 construction steps, and provides a means to improve the model and extend the simulation
636 period as new data becomes~~s~~ available. The new geospatial processing routines make it
637 possible to change model resolution as part of the model construction script. This allows one
638 to prototype fast running models with coarse resolution and use finer resolution as the model

639 starts to behave as intended. This workflow also allows one to conduct grid convergence
640 studies to ensure that the grid is not the cause of unintended model behavior.

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

647 Acknowledgments

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

652 Authors' Note

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

654 Disclaimer

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

657 References

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

- 660 Bakker, M., V. Post, C.D. Langevin, J.D. Hughes, J. White, J. Starn, and M.N. Fienen. 2016.
661 Scripting MODFLOW model development using Python and FloPy. *Groundwater* 54, no. 5:
662 733–739, <https://doi.org/10.1111/gwat.12413>.
- 663 Befus, K.M., P.L. Barnard, D.J. Hoover, J.A. Finzi Hart, and C.I. Voss. 2020. Increasing
664 threat of coastal groundwater hazards from sea-level rise in California. *Nature Climate
665 Change* 10, no. 10: 946–952, <https://doi.org/10.1038/s41558-020-0874-1>.
- 666 Befus, K.M., K.D. Kroeger, C.G. Smith, and P.W. Swarzenski. 2017. The Magnitude and
667 Origin of Groundwater Discharge to Eastern U.S. and Gulf of Mexico Coastal Waters.
668 *Geophysical Research Letters* 44, no. 20: 10,396–10,406,
669 <https://doi.org/10.1002/2017GL075238>.
- 670 Burek, P., Y. Satoh, T. Kahil, T. Tang, P. Greve, M. Smilovic, L. Guillaumot, F. Zhao, and
671 Y. Wada. 2020. Development of the Community Water Model (CWatM v1.04) – a
672 high-resolution hydrological model for global and regional assessment of integrated water
673 resources management. *Geoscientific Model Development* 13, no. 7: 3267–3298,
674 <https://doi.org/10.5194/gmd-13-3267-2020>.
- 675 Ebeling, P., F. Handel, and M. Walther. 2019. Potential of mixed hydraulic barriers to
676 remediate seawater intrusion. *Science of The Total Environment* 693: 133478,
677 <https://doi.org/10.1016/j.scitotenv.2019.07.284>.
- 678 ESRI. 1998. ESRI Shapefile Technical Description, an ESRI white paper.
679 <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf> (accessed August 29,
680 2022).
- 681 Essawy, B.T., J.L. Goodall, W. Zell, D. Voce, M.M. Morsy, J. Sadler, Z. Yuan, and T. Malik.
682 2018. Integrating scientific cyberinfrastructures to improve reproducibility in computational
683 hydrology: Example for HydroShare and GeoTrust. *Environmental Modelling &
684 Software* 105: 217–229, <https://doi.org/10.1016/j.envsoft.2018.03.025>.

- 685 Fienen, M.N. and M. Bakker. 2016. HESS Opinions: Repeatable research: what hydrologists
686 can learn from the Duke cancer research scandal. *Hydrology and Earth System Sciences* 20,
687 no. 9: 3739–3743, <https://doi.org/10.5194/hess-20-3739-2016>.
- 688 Fienen, M.N., N.T. Corson-Dosch, J.T. White, A.T. Leaf, and R.J. Hunt. 2022. Risk-Based
689 Wellhead Protection Decision Support: A Repeatable Workflow Approach. *Groundwater* 60,
690 no. 1: 71–86, <https://doi.org/10.3389/feart.2022.903965>.
- 691 Fienen, M.N., M.J. Haserodt, A.T. Leaf, and S.M. Westenbroek. 2022. Simulation of regional
692 groundwater flow and groundwater/lake interactions in the Central Sands, Wisconsin. U.S.
693 Geological Survey Scientific Investigations Report 2022-5046, 111 p.
694 <https://doi.org/10.3133/sir20225046>.
- 695 Gillies, S. 2022. The shapely user manual.
696 <https://shapely.readthedocs.io/en/stable/manual.html> (accessed August 28, 2022).
- 697 Gillies, S. et al. 2013. Rasterio: geospatial raster I/O for Python programmers.
698 <https://github.com/rasterio/rasterio> (accessed October 6, 2022).
- 699 Guira, M. 2018. Numerical Modeling Of The Effects Of Land Use Change And Irrigation On
700 Streamflow Depletion Of Frenchman Creek, Nebraska. Master's thesis, University of
701 Nebraska, Lincoln, NE.
- 702 Hill, M.C. 1992. A computer program (MODFLOWP) for estimating parameters of a
703 transient, three-dimensional, ground-water flow model using nonlinear regression. U.S.
704 Geological Survey Open-File Report 91-484, 358 p.
- 705 Hill, M.C., R.L. Cooley, and D.W. Pollock. 1998. A Controlled Experiment in Ground Water
706 Flow Model Calibration. *Groundwater* 36, no. 3: 520–535,
707 <https://doi.org/10.1111/j.1745-6584.1998.tb02824.x>.
- 708 Hughes, J.D., C.D. Langevin, and E.R. Banta. 2017. Documentation for the MODFLOW 6

- 709 framework. U.S. Geological Survey Techniques and Methods, book 6, chap. A57, 36 p.
- 710 <https://doi.org/10.3133/tm6A57>.
- 711 Hughes, J.D., S.A. Leake, D.L. Galloway, and J.W. White. 2022. Documentation for the
712 Skeletal Storage, Compaction, and Subsidence (CSUB) Package of MODFLOW 6. U.S.
713 Geological Survey Techniques and Methods, book 6, chap. A62, 57 p.
- 714 <https://doi.org/10.3133/tm6A62>.
- 715 Hunter, J.D. 2007. Matplotlib: A 2D graphics environment. *Computing in science &*
716 *engineering* 9, no. 03: 90–95.
- 717 Jaxa-Rozen, M., J.H. Kwakkel, and M. Bloemendal. 2019. A coupled simulation architecture
718 for agent-based/geohydrological modelling with NetLogo and MODFLOW. *Environmental*
719 *Modelling & Software* 115: 19–37, <https://doi.org/10.1016/j.envsoft.2019.01.020>.
- 720 Kluyver, T., B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley,
721 J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, and C. Willing. 2016.
722 Jupyter Notebooks – a publishing format for reproducible computational workflows. In
723 F. Loizides and B. Schmidt (Eds.), *Positioning and Power in Academic Publishing: Players,*
724 *Agents and Agendas*, pp. 87 – 90. IOS Press,
725 <https://doi.org/10.3233/978-1-61499-649-1-87>.
- 726 Knowling, M.J., J.T. White, and C.R. Moore. 2019. Role of model parameterization in
727 risk-based decision support: An empirical exploration. *Advances in Water Resources* 128:
728 59–73, <https://doi.org/10.1016/j.advwatres.2019.04.010>.
- 729 Langevin, C.D., J.D. Hughes, A.M. Provost, E.R. Banta, R.G. Niswonger, and S. Panday.
730 2017. Documentation for the MODFLOW 6 Groundwater Flow (GWF) Model. U.S.
731 Geological Survey Techniques and Methods, book 6, chap. A55, 197 p.
- 732 <https://doi.org/10.3133/tm6A55>.

- 733 Langevin, C.D., S. Panday, and A.M. Provost. 2020. Hydraulic-Head Formulation for
734 Density-Dependent Flow and Transport. *Groundwater* 58, no. 3: 349–362.
- 735 Langevin, C.D., A.M. Provost, S. Panday, and J.D. Hughes. 2022. Documentation for the
736 MODFLOW 6 Groundwater Transport (GWT) Model. U.S. Geological Survey Techniques
737 and Methods, book 6, chap. A61, 56 p. <https://doi.org/10.3133/tm6A55>.
- 738 Larsen, J.D., A.H. Alzraiee, D. Martin, and R.G. Niswonger. 2022. Rapid Model Development
739 for GSFLOW With Python and pyGSFLOW. *Frontiers in Earth Science* 10,
740 <https://doi.org/10.3389/feart.2022.907533>.
- 741 Leaf, A.T. and M.N. Fienen. 2022. Modflow-setup: Robust automation of groundwater model
742 construction. *Frontiers in Earth Science* 10: 903965,
743 <https://doi.org/10.3389/feart.2022.903965>.
- 744 Lien, J.M., G. Liu, and C.D. Langevin. 2014. GRIDGEN Version 1.0: A computer program
745 for generating unstructured finite-volume grids. U.S. Geological Survey Open-File Report
746 2014-1109, 26 p. <https://doi.org/10.3133/ofr20141109>.
- 747 Mancewicz, L.K., A. Mayer, C.D. Langevin, and J. Gulley. 2022. Improved method for
748 simulating groundwater inundation using the MODFLOW 6 Lake Transport Package.
749 *Groundwater*, <https://doi.org/10.1111/gwat.13254>.
- 750 Mehl, S.W. and M.C. Hill. 2006. MODFLOW-2005, the U.S. Geological Survey modular
751 ground-water model—documentation of shared node local grid refinement (LGR) and the
752 boundary flow and head (BFH) package. U.S. Geological Survey Techniques and Methods,
753 book 6, chap. A12, 78 p. <https://doi.org/10.3133/tm6A12>.
- 754 Mehl, S.W. and M.C. Hill. 2013. MODFLOW-LGR—Documentation of ghost node local grid
755 refinement (LGR2) for multiple areas and the boundary flow and head (BFH2) package.
756 U.S. Geological Survey Techniques and Methods, book 6, chap. A44, 43 p.
757 <https://doi.org/10.3133/tm6A44>.

- 758 Morway, E.D., C.D. Langevin, and J.D. Hughes. 2021. Use of the MODFLOW 6 water mover
759 package to represent natural and managed hydrologic connections. *Groundwater* 59, no. 6:
760 913–924, <https://doi.org/10.1111/gwat.13117>.
- 761 Panday, S., C.D. Langevin, R.G. Niswonger, M. Ibaraki, and J.D. Hughes. 2013.
762 MODFLOW-USG version 1—An unstructured grid version of MODFLOW for simulating
763 groundwater flow and tightly coupled processes using a control volume finite-difference
764 formulation. U.S. Geological Survey Techniques and Methods, book 6, chap. A45, 66 p.
- 765 Perry, M. 2013. Rasterstats python library.
766 <https://github.com/perrygeo/python-rasterstats>.
- 767 Pollock, D.W. 2014. Personal communication. Reston, VA.
- 768 Provost, A.M., C.D. Langevin, and J.D. Hughes. 2017. Documentation for the “XT3D”
769 Option in the Node Property Flow (NPF) Package of MODFLOW 6. U.S. Geological Survey
770 Techniques and Methods, book 6, chap. A56, 46 p. <https://doi.org/10.3133/tm6A56>.
- 771 Rew, R. and G. Davis. 1990. NetCDF: an interface for scientific data access. *IEEE computer
772 graphics and applications* 10, no. 4: 76–82.
- 773 Rew, R., E. Hartnett, J. Caron, et al. 2006. NetCDF-4: Software implementing an enhanced
774 data model for the geosciences. In *22nd International Conference on Interactive
775 Information Processing Systems for Meteorology, Oceanograph, and Hydrology*, Volume 6.
- 776 Rossetto, R., G. De Filippis, I. Borsi, L. Foglia, M. Cannata, R. Criollo, and E. Vázquez-Suñé.
777 2018. Integrating free and open source tools and distributed modelling codes in GIS
778 environment for data-based groundwater management. *Environmental Modelling &
779 Software* 107: 210–230, <https://doi.org/10.1016/j.envsoft.2018.06.007>.
- 780 Schroeder, W., K. Martin, and B. Lorensen. 2006. *The Visualization Toolkit* (4th ed.).
781 Kitware.

- 782 Shewchuk, J.R. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay
783 Triangulator. In *Applied Computational Geometry, Towards Geometric Engineering,*
784 *FCRC'96 Workshop, WACG'96, Philadelphia, PA, USA, May 27-28, 1996, Selected Papers*,
785 pp. 203–222. <https://doi.org/10.1007/BFb0014497>.
- 786 Starn, J.J. and K. Belitz. 2018. Regionalization of Groundwater Residence Time Using
787 Metamodelling. *Water Resources Research* 54, no. 9: 6357–6373,
788 <https://doi.org/10.1029/2017WR021531>.
- 789 Sun, A.Y. 2018. Discovering State-Parameter Mappings in Subsurface Models Using
790 Generative Adversarial Networks. *Geophysical Research Letters* 45, no. 20: 11,137–11,146,
791 <https://doi.org/10.1029/2018GL080404>.
- 792 U.S. Geological Survey. 2018. GWWebFlow—a browser-based groundwater model viewer.
793 <https://webapps.usgs.gov/gwwebflow/>.
- 794 van Engelen, J., G.H. Oude Essink, H. Kooi, and M.F. Bierkens. 2018. On the origins of
795 hypersaline groundwater in the Nile Delta aquifer. *Journal of Hydrology* 560: 301–317,
796 <https://doi.org/10.1016/j.jhydrol.2018.03.029>.
- 797 Vilhelmsen, T.N., S. Christensen, and S.W. Mehl. 2012. Evaluation of MODFLOW-LGR in
798 connection with a synthetic regional-scale model. *Groundwater* 50, no. 1: 118–132,
799 <https://doi.org/10.1111/j.1745-6584.2011.00826.x>.
- 800 Virtanen, P., R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau,
801 E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson,
802 K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey,
803 İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman,
804 I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa,
805 P. van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for
806 Scientific Computing in Python. *Nature Methods* 17: 261–272,
807 <https://doi.org/10.1038/s41592-019-0686-2>.

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

811 White, J.T., L.K. Foster, M.N. Fienan, M.J. Knowling, B. Hemmings, and J.R. Winterle.
812 2020. Toward reproducible environmental modeling for decision support: A worked
813 example. *Frontiers in Earth Science* 8: 50, <https://doi.org/10.3389/feart.2020.00050>.

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

818 Zipper, S.C., T. Gleeson, B. Kerr, J.K. Howard, M.M. Rohde, J. Carah, and J. Zimmerman.
819 2019. Rapid and Accurate Estimates of Streamflow Depletion Caused by Groundwater
820 Pumping Using Analytical Depletion Functions. *Water Resources Research* 55, no. 7:
821 5807–5829, <https://doi.org/10.1029/2018WR024403>.

822 Figure captions

823 1 Examples of grids that can be generated and processed using FloPy for a hypothetical
824 watershed, including (A) a structured MODFLOW grid with constant and equal
825 row and column spacings, (B) a structured MODFLOW grid with variable row
826 and column spacings, (C) a structured MODFLOW child grid nested within
827 a structured MODFLOW parent grid, (D) a quadtree grid generated with the
828 GRIDGEN program (Lien et al. 2014) through the FloPy wrapper, (D) a triangular
829 grid generated with the Triangle program (Shewchuk 1996) through the FloPy
830 wrapper, and (E) a Voronoi grid created from the triangular mesh. All of the
831 grids have refinement in the location of the child grid in (C).

832	2 Examples of the intersection of a linear stream network with the model grids 833 shown in Figure 1. Intersections were performed using FloPy for (A) a structured 834 MODFLOW grid, (B) a structured MODFLOW grid with variable row and 835 column spacing, (C) a structured MODFLOW child grid nested within a structured 836 MODFLOW parent grid, (D) a quadtree grid, (D) a triangular grid, and (E) a 837 Voronoi grid. Shaded cells represent those cells that intersect with the linear 838 stream network. Individual plots in this figure are centered on the location of the 839 child grid shown in Figure 1C.	15
840	3 Examples of FloPy map and cross-section plotting capabilities for a model discretized 841 using a Voronoi grid (Figure 1F). (A) Map showing simulated heads and specific- 842 discharge vectors in the upper-most saturated cells. (B) Map showing the layer 843 containing the water table, the location of cells where the aquifer is discharged 844 to rivers represented as drain cells, and the location of cells where groundwater 845 is discharging to the land surface. (C) East-West cross-section along line A– 846 A', shown on Figure 3A, showing the model grid, simulated heads, and cells 847 where water-table conditions exist. (D) North-South cross-section along line B– 848 B', shown on Figure 3A, showing the model grid, simulated heads, and cells where 849 water-table conditions exist.	22
850	4 Two different graphical renderings of the Voronoi model grid: (A) stair-cased 851 representation in which cell have flat tops and bottoms and (B) smooth representation 852 in which elevations for cell vertices are interpolated using cell top and bottom 853 elevations. Renderings were created using Paraview (Ahrens et al. 2005) and 854 Visualization Tool Kit (Schroeder et al. 2006) files exported from FloPy.	23

855	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).	25
862	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).	26
864	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).	27
869	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).	31
871			