frontiers | Frontiers in Earth Science

TYPE Technology and Code
PUBLISHED 30 September 2022
DOI 10.3389/feart.2022.903965

OPEN ACCESS

EDITED BY
Francesca Pianosi,
University of Bristol, United Kingdom

REVIEWED BY
Richard Niswonger,
United States Geological Survey (USGS), United States
Norman Jones,
Brigham Young University, United States

*CORRESPONDENCE
Andrew T. Leaf,
aleaf@usgs.gov

SPECIALTY SECTION
This article was submitted to Hydrosphere, a section of the journal Frontiers in Earth Science

RECEIVED 24 March 2022
ACCEPTED 29 July 2022
PUBLISHED 30 September 2022

CITATION
Leaf AT and Fienen MN (2022), Modflow-setup: Robust automation of groundwater model construction. *Front. Earth Sci.* 10:903965. doi: 10.3389/feart.2022.903965

Modflow-setup: Robust automation of groundwater model construction

Andrew T. Leaf* and Michael N. Fienen

U.S. Geological Survey Upper Midwest Water Science Center, Middleton, WI, United States

In an age of both big data and increasing strain on water resources, sound management decisions often rely on numerical models. Numerical models provide a physics-based framework for assimilating and making sense of information that by itself only provides a limited description of the hydrologic system. Often, numerical models are the best option for quantifying even intuitively obvious connections between human activities and water resource impacts. However, despite many recent advances in model data assimilation and uncertainty quantification, the process of constructing numerical models remains laborious, expensive, and opaque, often precluding their use in decision making. Modflow-setup aims to provide rapid and consistent construction of MODFLOW groundwater models through robust and repeatable automation. Common model construction tasks are distilled in an open-source, online code base that is tested and extensible through collaborative version control. Input to Modflow-setup consists of a single configuration file that summarizes the workflow for building a model, including source data, construction options, and output packages. Source data providing model structure and parameter information including shapefiles, rasters, NetCDF files, tables, and other (geolocated) sources to MODFLOW models are read in and mapped to the model discretization, using Flopy and other general open-source scientific Python libraries. In a few minutes, an external array-based MODFLOW model amenable to parameter estimation and uncertainty quantification is produced. This paper describes the core functionality of Modflow-setup, including a worked example of a MODFLOW 6 model for evaluating pumping impacts to a lake in central Wisconsin, United States.

KEYWORDS

groundwater modeling, Python, MODFLOW, Flopy, numerical modeling, software, automation

Introduction

Numerical groundwater models can provide water managers and other stakeholders with a powerful physics-based framework for evaluating complex hydrologic systems, which may be difficult or impossible to represent analytically (e.g., Anderson et al., 2015). In comparison to analytical methods, numerical models provide flexibility in their ability

to finely discretize natural heterogeneity and complex boundaries or structures, and in their ability to represent transient effects. In many real-world systems, these capabilities may be essential to representing and understanding questions of interest (e.g., Leaf et al., 2015; Fienen et al., 2022,2021a). Just as importantly, numerical models allow for higher dimensional parametrization that can be critical for effective data assimilation, associated model error reduction, and meaningful consideration of model uncertainty (e.g., Moore and Doherty 2005; Hunt et al., 2007; White et al., 2021, 2014). Because of these advantages, numerical groundwater models are used widely. MODFLOW (e.g., Niswonger et al., 2011; Langevin et al., 2017) and related codes are the most popular framework for numerical modeling of groundwater flow and transport worldwide.

The flexibility of numerical models, however, comes with steep costs. Disparate input data must be mapped to thousands or millions of computational cells, a process that can be cumbersome, labor-intensive, and error-prone. The number and complexity of operations presents a fundamental challenge to scientific reproducibility (e.g., Peng 2011; Fienen and Bakker, 2016), step-wise modeling (Haitjema, 1995), and the modeler's own cognitive load (e.g., Sweller 1988). The inherent difficulty of, for example, changing discretization or model structure makes it difficult to revisit these choices later in a project in response to what is learned, and carrying alternative conceptual models through a project is seldom feasible. As noted by White et al. (2021), realistic representation of model uncertainty presents an additional set of challenges that may be out of reach if the basic model inputs cannot be efficiently built. As a result of these costs, numerical groundwater models are not only expensive but can often fall short of expectations (e.g., Donoho et al., 2008; Moran 2016; Doherty and Moore, 2020). A key goal, then, is to automate repetitive, but crucial, model construction tasks such that a modeler can focus their efforts on the underlying problem and conceptualization rather than model mechanics.

Numerical groundwater models are often constructed with the help of a graphical user interface (GUI). GUIs provide an interactive environment for building and post-processing models that is especially helpful for visualization and handling of model input and output formats. Some GUIs even support grid-independent input. The reader is referred to Anderson et al. (2015) for a more thorough discussion of GUI options. Although many consider the "point and click" approach afforded by GUIs to be more intuitive than direct manipulation of model input files, most GUI workflows are not readily automatable, and therefore prone to the issues mentioned earlier. Without automation, meaningful documentation of the workflow requires additional effort on the modeler's part and may not be feasible under typical project constraints.

In recent years, open-source software tools to automate the mapping of disparate data to computational grids have become readily available and easy to use. These include Python packages for working with MODFLOW files (Bakker et al., 2016), GIS file formats and geoprocessing (Gillies 2022a,b,c,d), NetCDF data (Hoyer and Hamman, 2017), coordinate transformations (Snow et al., 2022), and general scientific algorithms (Virtanen et al., 2020); as well as software development tools that facilitate collaborative version control (e.g., Git; https://git-scm.com/and GitHub; https://github.com/), automated testing (e.g., Pytest; https://pytest.org), continuous integration, and online documentation (e.g., Sphinx; https://www.sphinx-doc.org/); and accessible tutorials that show domain scientists how to use them (e.g., https://nsls-ii.github.io/scientific-python-cookiecutter).

Script-based development of model input with a high-level language such as Python has therefore been proposed as a solution to overcome model construction challenges (e.g., Bakker et al., 2016), but in practice this is easier said than done. Ad hoc scripts must be assembled into a carefully documented workflow that can have many steps and interdependencies and is itself subject to the "ubiquity of error" (Donoho et al., 2008). Even the most well documented workflows depend on the quality of the underlying code and therefore, the fastidiousness and programming abilities of the modeler. In the end, a fully scripted workflow may be no easier to understand, repeat, or reproduce than a sequence of manual operations in a GUI or spreadsheet environment.

Fisher et al. (2016) presented what may be the best published example of a fastidious model construction workflow. In development of a groundwater model for a project in Idaho, United States, they developed code functions and assembled them into a formal R package complete with code documentation and *vignettes* (R Core Team, 2014) walking users through the workflow. Although this approach almost certainly improved reproducibility and likely carried other advantages, it was focused on a single project, and likely required considerable overhead effort that may not be readily transferable to other work.

Quality code development for robust and reproducible workflows takes time (e.g., Donoho et al., 2008; Wilson et al., 2014) that is most efficiently spent developing general code that can be reused in many different contexts. Functions and other objects that are dedicated to specific tasks and can be readily imported into a script or called repeatedly in a loop provide a local means for reusability. Functions carry the added benefit of breaking complex workflows into easily understandable pieces that can also be readily tested, thereby reducing error. At a higher level, software packages provide a well-understood framework for developing, testing, and sharing collections of functions and other objects. The Flopy project (Bakker et al., 2016) provides such a package, but at a low level that typically requires extensive ad hoc scripting and geoprocessing outside of Flopy to develop a MODFLOW model in a real-world context. MODFLOW models are not internally geolocated, so Flopy is always referenced to

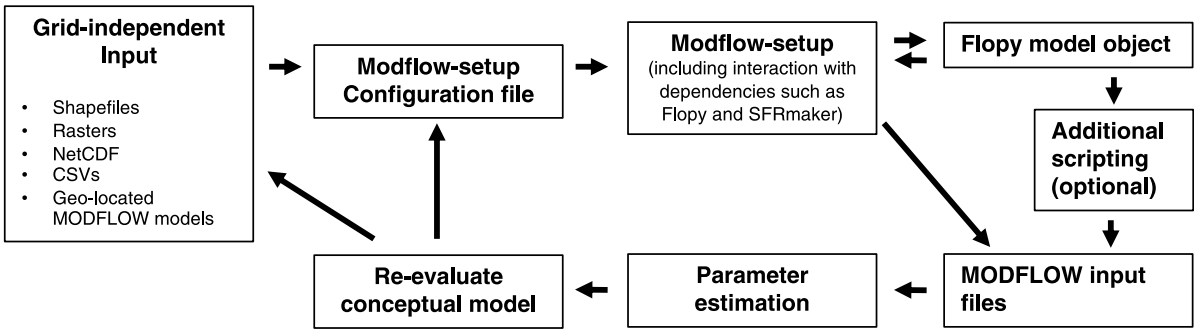

FIGURE 1
Modflow-setup in the groundwater modeling workflow. Modflow input can be written directly, or additional scripting can be performed with Flopy to customize the model. By automating the discretization of input data, Modflow-setup allows the conceptual model and model structure to be more readily revised in response to parameter estimation or new information.

MODFLOW model grids rather than geospatial coordinates. As a result, all source data must be mapped to the MODFLOW grid.

Modflow-setup provides a formal, tested, and documented Python package that builds on Flopy and the other packages referenced earlier to provide a robust, fully automated workflow for constructing MODFLOW models in a wide variety of contexts. Source data can include shapefiles, rasters, NetCDF files, and other MODFLOW models that are geolocated. We chose MODFLOW as the endpoint because it is free, open-source, easy to use (and widely used), well documented and tested, and well supported by Flopy. Modflow-setup extends the datatypes of Flopy to facilitate reading and writing MODFLOW package input and handle inter-package dependencies in memory. A key advance of Modflow-setup is the configuration file, which succinctly summarizes the data sources to a groundwater model and the methods used to process the data into model input. The information in the configuration file can be used to drive a fully automated model construction workflow, reducing the scripting needed to build a MODFLOW model to as little as a few lines of Python. This paper gives an overview of Modflow-setup, including a working example based on a published study in Wisconsin, United States (Fienen et al., 2022; 2021b).

Methods

Overview of the Modflow-setup workflow

Figure 1 illustrates the use of Modflow-setup in a groundwater modeling workflow. Grid-independent source data are preprocessed as needed and specified in a configuration file for input to Modflow-setup, along with other settings such as space and time discretization. Modflow-setup reads the configuration file, maps the input data to the model grid, and produces a modified Flopy model object. Some model inputs, such as external array text files, are written directly by Modflow-setup; other input, such as MODFLOW package input files, are written by Flopy. Prior to writing any files, additional scripting can be performed on the Flopy model object as needed, to prepare any input not supported by Modflow-setup. Parameter estimation can then be performed on the working model, which may lead to re-evaluation of the conceptual model and changes to the model structure or discretization. Modflow-setup can rapidly regenerate a new model incorporating the changes.

General paradigms

Modflow-setup supports the construction of MODFLOW 6 (Langevin et al., 2017) or MODFLOW-NWT (Niswonger et al., 2011) models from scratch (i.e., from grid-independent source data) or as an "inset" model that is coupled in one direction to a "parent" model *via* specified head or flux perimeter boundaries from the parent model solution. Parent and inset models can be mixed between MODFLOW 6 and MODFLOW-NWT. An additional "local grid refinement" (LGR) option for MODFLOW 6 models allows for specification of an inset model that is dynamically linked (in both directions) to the parent model at a finer grid resolution. Unlike previous versions of LGR (e.g., Mehl et al., 2006; Vilhelmsen et al., 2012), this inset model formulation is coupled to the parent model at the matrix level (Langevin et al., 2017), making this an efficient option for simulating both regional flow and a detailed area of interest. To facilitate array resampling and dereferencing, currently; only uniform structured grids (that may be rotated) are supported. Temporal discretization is specified in blocks that are piecewise-constant, allowing, for example, for longer spin-up periods early in a simulation, followed by a finer temporal discretization in a

period of interest. The model grid is referenced internally to a specified projected coordinate reference system (CRS; with units of feet or meters), but source data can be in any CRS; reprojection is handled automatically as needed *via* the Pyproj package (Snow et al., 2022). Similarly, length and time units can be specified for the inset and parent models, and any source data and unit conversions are handled automatically.

Time-varying specified head or specified flux boundaries can be applied to the perimeter of an inset model from a parent model solution, *via* the Constant Head and Well Packages, respectively. The parent and inset model grids need not align, but spatial alignment of the two grids can be beneficial for preserving mass when resampling recharge from one grid to another. Parent and inset model grids are located relative to one another using their respective CRS. Currently, transient inset models must have either a steady-state parent, or align temporally with a subset of the parent model stress periods.

Currently, the Streamflow Routing (SFR), Lake, and basic stress packages (Constant Head, Drain, General Head, River, and Well Packages) are fully supported for internal boundaries, with some limited support for the Multi-node Well 2 (MNW2) Package in MODFLOW-NWT. Unlike previous inset model translators such as MODTMR (Leake and Claar, 1999), internal boundary conditions are always re-discretized from their grid-independent source data (typically shapefiles), as inset models will usually carry a finer discretization than the parent. An exception is an option to translate the Well Package based on the nearest neighbor location of the model cell centers. Preparation of SFR input is handled by SFRmaker (Leaf et al., 2021). In general, the geographic extents of surface water features are specified *via* shapefiles, and any transient data such as stream inflows or pumping rates are specified *via* comma separated variable (CSV) files. Well locations can be specified with CSV or shapefiles. Transient input data are mapped to the model stress periods by computing a specified statistic (usually the mean) for values falling within each model stress period, or within a user-specified timeframe (for example, a long-term average period representing steady-state conditions).

Array-based input can be specified from rasters, shapefiles, NetCDF files, or the parent MODFLOW model. Rasters can be used to assign values to specific layers or stress periods; input is resampled to the model grid at the cell center locations using either a nearest neighbor or linear interpolation approach. Shapefiles are generally only used for delineating discrete features such as the active model area and are mapped using the `rasterize` method in the Rasterio package (Gillies, 2022b). NetCDF files provide a convenient mechanism for array-based input with many two-dimensional time slices, for example, daily estimates of net infiltration from Soil-Water-Balance code (SWB; Westenbroek et al., 2018). Similar to other transient inputs, NetCDF time slices are mapped to the model time discretization by computing period statistics. As with other data, unit conversions are performed automatically if the units are specified. Finally, arrays from a parent MODFLOW model can be resampled to the inset grid in time or space. Inset-parent layer mapping is typically specified for static inputs such as aquifer properties or cell top and bottom elevations. The coarser parent model values are then upsampled by layer to the inset model resolution, using a barycentric scheme similar to the `griddata` method in Scipy (Virtanen et al., 2020). In the case of perimeter boundary conditions, fields of head or flux components from the parent model are upsampled by stress period to the inset model grid, using the same barycentric interpolation scheme but in three dimensions, which simplifies specification of the system state along the inset model perimeter when the inset and parent grids do not align exactly (e.g., Leake and Claar, 1999).

Another key feature of Modflow-setup is the creation of MODFLOW observation input. Head observation locations can be supplied *via* a CSV file and are then mapped to the closest model cell center. Observations are set up in each model layer at the mapped locations, to allow for subsequent post-processing of model output to derive simulated head equivalents for the well open intervals (for example, using the transmissivity-based weighting functionality in Modflow-obs; https://github.com/aleaf/modflow-obs). Head observation input is created for the observation utility in MODFLOW 6 (Langevin et al., 2017) or the HYDMOD Package in MODFLOW-NWT (Hanson et al., 1999). Streamflow observations can also be supplied, with either coordinate locations or unique identifiers referencing them to a specific flowline within the input hydrography. SFRmaker will then locate the observations within the SFR package and create the relevant SFR package observation input (Leaf et al., 2021). Finally, other types of observations can be set up automatically based on lake numbers (for the Lake Package) or boundnames (for the basic stress packages in MODFLOW 6; Langevin et al., 2017).

Version control presents a fundamental challenge to reproducibility and robustness in numerical models. Even if a version control system such as Git is used to track model construction, the model files may inevitably get copied or modified outside of the Git framework, leading to confusion about their provenance. Modflow-setup records up to three levels of version information in the comment headers of the produced MODFLOW input files: 1) the Flopy version, 2) the Modflow-setup version (including the commit hash), and 3) the model version, if the model is being tracked by Git, or if a version is specified in the configuration file. In the former case, Git versioning information is read by Modflow-setup using an approach similar to the Versioneer package (https://github.com/python-versioneer/python-versioneer). This way, the methods used to generate a particular model input file can be understood and reproduced, even if the code base and model have changed.

Software implementation

Modflow-setup is implemented as a Python package that works on Linux, OSX, or Windows. The version of the code documented in

this study is available as a USGS software release (Leaf et al., 2022); the current development version that incorporates bug fixes and other improvements is available through GitHub (https://github.com/doi-usgs/modflow-setup) or the Python Package Index (PyPI). It should be noted that Modflow-setup has software dependencies that must be installed prior to its use. Detailed instructions on how to install the dependencies and Modflow-setup are available in the online documentation (https://doi-usgs.github.io/modflow-setup). Similar to any Python package, Modflow-setup consists of objects that can be imported into a Python session and therefore used within scripts or other Python code. The use of Modflow-setup does not require extensive knowledge of Python, however. In the simplest use case, input can be specified in a configuration file, and a MODFLOW model can be built from the configuration file using only a few lines of Python, as illustrated in the example.

Code organization

The core function of Modflow-setup is to automate mapping of disparate, grid-independent data to a finite difference grid. At a basic level, the Modflow-setup package houses general objects (functions, classes, and methods) to do this. The objects are organized into modules that loosely correspond to the components of a MODFLOW model (e.g., "oc.py" for output control) or specific functionality (e.g., "interpolate.py" for interpolation). Ideally, each module has a corresponding test module in the "tests" folder, and each object a corresponding test within that test module. In practice, much of the testing follows an integration approach where entire packages or models are built within a single test, effectively testing the interactions of multiple objects at once.

While Modflow-setup may evolve to include more functionality as a library of stand-alone components, the current development focus is on an integrated workflow that builds Flopy model objects from information provided in a configuration file. Three model classes, each contained in their own module, are central to this focus. The `MF6model` and `MFnwtModel` classes subclass the Flopy `ModflowGwf` and `Modflow` classes, respectively, to add additional model construction functionality for MODFLOW 6 and MODFLOW-NWT models. Both `MF6model` and `MFnwtModel` also subclass a shared `MFsetupMixin` class that contains core functionality common to any MODFLOW version. The model classes themselves contain a number of methods centered around various arrays and packages, which, in turn, interact with functions and other objects in the remaining Modflow-setup modules.

The configuration file

Most user interaction with Modflow-setup is through the configuration file, which is specified in the YAML format (yaml.org). YAML maps key:value pairs similar to a Python dictionary (https://docs.python.org/3/tutorial/datastructures.html), except that whitespace and newlines can often be used in the place of commas and brackets to delimit structures. YAML input in the configuration file is organized into blocks that generally follow the MODFLOW input structure, with primary blocks representing specific MODFLOW packages or model components, and sub-blocks representing MODFLOW 6 input blocks or features in Modflow-setup. The naming of blocks and variables is intended to follow MODFLOW and Flopy conventions as closely as possible, with MODFLOW given preference where these conflict. For example, this block (from the example problem discussed below) describes the MODFLOW 6 simulation:

```yaml
simulation:
  sim_name: 'pleasant_lgr'
  version: 'mf6'
  sim_ws: 'pleasant_lgr/'
```

In the model setup workflow, input from the configuration file is loaded by Modflow-setup into a configuration dictionary attached to the model object. For example, the `simulation:` block shown above would be loaded as:

```
cfg['simulation'] = {'sim_name: 'mfsim',
                     'version': 'mf6',
                     'sim_ws': 'pleasant_lgr/'}
```

The above dictionary would then be fed to the Flopy `MFSimulation` class constructor to create a simulation instance. Within package blocks, input to MODFLOW can be specified directly using the appropriate variables and structures described in the MODFLOW input instructions (Niswonger et al., 2011; Langevin et al., 2017). For example, in the block below, the dimensions: and griddata: sub-blocks would be fed directly to the MODFLOW 6 Discretization Package constructor in Flopy:

```yaml
dis:
  options:
    length_units: 'meters'
  dimensions:
    nlay: 2
    nrow: 30
    ncol: 35
  griddata:
    delr: 1000.
    delc: 1000.
    top: 2.
    botm: [1, 0]
```

Such direct input might also contain paths to external text file arrays that are consistent with the model grid. Alternatively, *source_data*: sub-blocks can be used to reference grid-independent data (shapefiles, rasters, or comma separated variable files, etc.) that need to be mapped to the model grid. The Pleasant Lake example described below includes a DIS package block that references GeoTIFF rasters as input for layer tops and bottoms. More details on configuration file input options are available in the online documentation (https://doi-usgs.

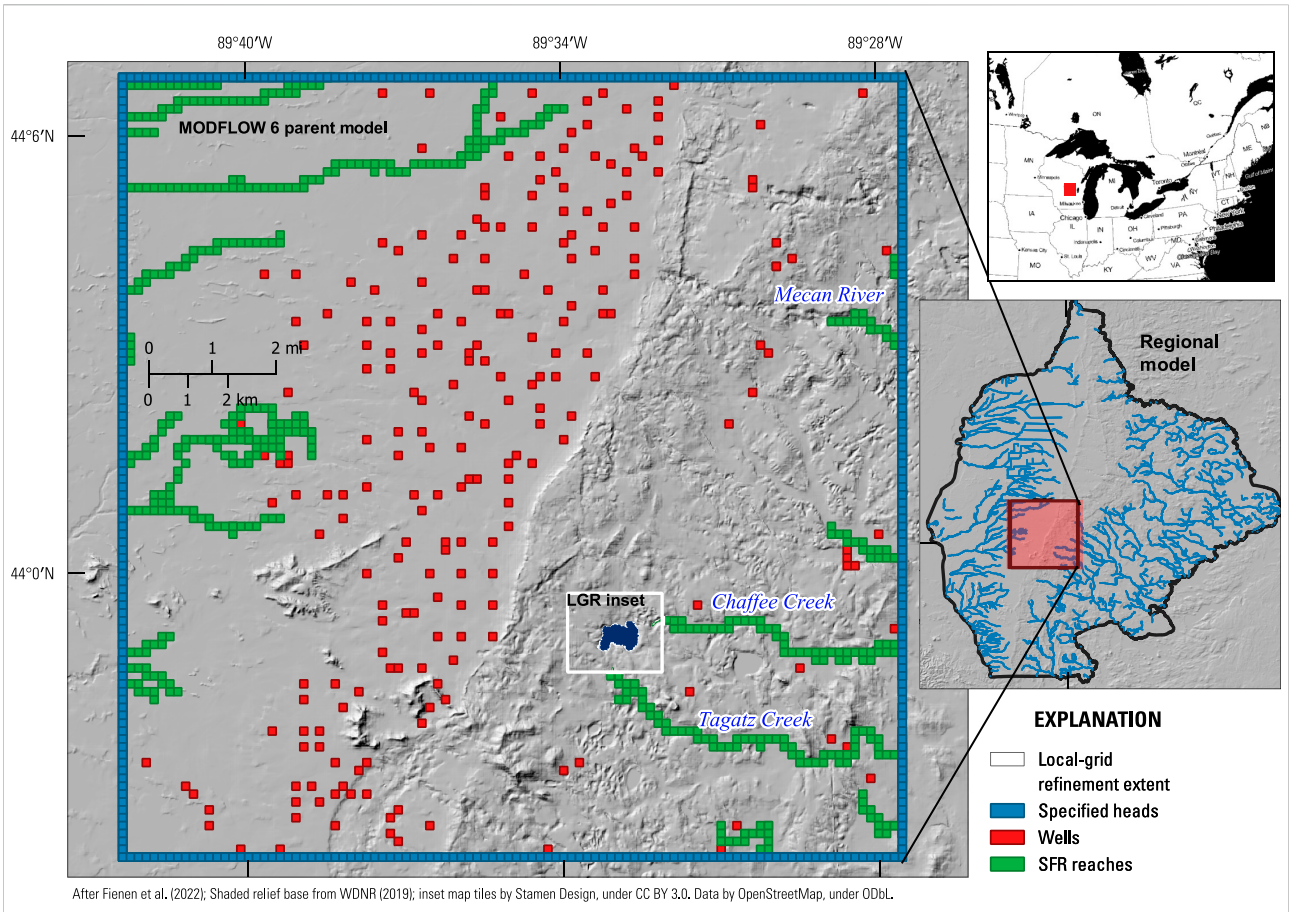

FIGURE 2
The full Pleasant Lake model domain with location map, showing the relationship between the regional, intermediate and LGR inset models, as well as the irrigation wells considered.

github.io/modflow-setup), which includes a gallery of working configuration files for various models in the Modflow-setup test suite.

An example model build script

With an appropriate configuration file, a Python script to build a MODFLOW model can be as simple as the following three lines of Python:

```
from mfsetup import MF6model
model = MF6model.setup_from_yaml('config_file.yml')
model.write_input()
```

In this example, the model object class is imported, similar to Flopy, and the `setup_from_yaml` constructor method is called with the configuration file. An `MF6model` instance, which is essentially a Flopy model object with additional functionality, is returned. The `MF6model` instance can be used to write the model input files or as the basis for additional custom scripting.

Example: Setup of the Pleasant Lake model

The Pleasant Lake model (Fienen et al., 2022) is a MODFLOW 6 simulation that was constructed using Modflow-setup. We show this example both because this project motivated the development of the Modflow-setup code and because it highlights a complex workflow that benefits greatly from the scripting approach. A simplified version of this workflow with a smaller model domain is available on the Modflow-setup GitHub site (https://github.com/doi-usgs/modflow-setup); the published, fully detailed models for Pleasant Lake are available from Fienen et al. (2021b). Another worked example including uncertainty analysis and a decision support outcome is available from Fienen and Corson-Dosch (2021; https://github.com/usgs/neversink_workflow).

The goal of the Pleasant Lake model, part of the Central Sands Lake Study (Fienen et al., 2022), was to address connections between groundwater abstraction and the ecological function of a lake in central Wisconsin, United States (WDNR 2021; Figure 2). This required modeling at multiple scales. Fine discretization was needed near the

lake for accurate simulation of water levels and groundwater–lake flux. A large model domain was also needed to simulate farfield water-use activity (chiefly irrigated agriculture), in order to delineate a limit of connection, as well as to incorporate distant hydrologic boundaries. Adopting a fine enough discretization for the lake detail throughout the farfield would have resulted in a model with more cells than could be practically managed. To mitigate this, three models were combined: a large regional model built with MODFLOW-NWT, an intermediate MODFLOW 6 model inset within the regional model to simulate the irrigated agriculture area, and a refined MODFLOW 6 inset model (nested within the intermediate model) to simulate the lake (Figure 2; Fienen et al., 2022). Regional groundwater flow and the effects of distant boundaries were simulated with the MODFLOW-NWT model, which was coupled sequentially (one-way) to the MODFLOW 6 models through time-varying specified head boundaries along the intermediate MODFLOW 6 model perimeter. The two MODFLOW 6 models were coupled dynamically (both ways) within the groundwater flow solution, allowing for feedback between the models. Estimates of groundwater recharge for the MODFLOW models were provided by a SWB simulation that could represent alternative assumptions of climate and land use. Net infiltration estimates from the SWB model in the NetCDF format were read directly by Modflow-setup to produce Recharge Packages for the MODFLOW models. Climate-based estimates of irrigation demand from SWB were also passed to the Well Package for simulations that considered future scenarios.

The MODFLOW 6 models were set up using the Modflow-setup LGR feature, which uses the LGR utility in Flopy (Bakker et al., 2016) to create input for the Groundwater Flow Exchange Package, which dynamically links MODFLOW 6 models within the same matrix solution *via* fluxes across their shared boundaries (Langevin et al., 2017). The Modflow-setup LGR feature also sets up the Water Mover Package to maintain continuity in the SFR Package streamflow across the linked model boundaries. Construction of the LGR inset model is activated by an `lgr:` sublock within the parent model configuration file, which points to a second configuration file for the LGR inset. Full versions of the parent and inset model configuration files for the example are available in the online documentation (https://doi-usgs.github.io/modflow-setup). An abbreviated version of the example LGR inset model configuration file is reproduced in snippets here for illustration.

As noted earlier, the `simulation:` block provides input to the Flopy MFSimulation constructor, and, critically, the `version:` argument that also tells Modflow-setup which version of MODFLOW to use. Similarly, the `model:` block contains input to the Flopy `ModflowGwf` constructor and ultimately, the MODFLOW 6 Name file (Langevin et al., 2017). The `packages:` argument tells Modflow-setup which packages to build. Since this model is an LGR inset, the parent model is already known to Modflow-setup and does not need to be re-specified here. Similarly, any packages included in the package list, but not specified in the inset model configuration file, are simply built (on the inset model grid) from the input in the parent model configuration file.

```yaml
simulation:
  sim_name: 'pleasant_lgr'
  version: 'mf6'
  sim_ws: 'pleasant_lgr/'

model:
  simulation: 'pleasant_lgr'
  modelname: 'plsnt_lgr_inset'
  options:
    print_input: True
    save_flows: True
    newton: True
    newton_under_relaxation: True
  packages: ['dis', 'ic', 'npf', 'oc', 'sto', 'rch', 'sfr',
             'lak', 'obs', 'wel', 'ims']
```

The `setup_grid:` block specifies the orientation and discretization of the LGR inset grid. Model grids in Modflow-setup can be defined explicitly or using a buffer around a feature of interest. If the model is associated with a parent model, the model discretization is aligned with the parent model grid by default (this is required for LGR models). A `snap_to_parent:` option allows for unaligned grids. Unrotated models with a grid spacing that is a factor of 1,000 m can also be aligned with the National Hydrogeologic Grid, a framework intended to facilitate the development and use of national-scale hydrogeologic datasets in the United States (Clark et al., 2018).

In this case, a polygon for Pleasant Lake is provided *via* a shapefile, and Modflow-setup is instructed to create a regular 40-m mesh within a 1000-m buffer of the lake. The projected CRS for the model grid is Wisconsin Transverse Mercator (indicated by EPSG code 3070). The vertical discretization is specified in a `dis:` (Discretization Package) block. A digital elevation model (DEM) in units of meters is specified for the model top. As in Python, numbering for layers or stress periods is zero-based. Since no bottom elevation grid is supplied for layer 0, the bottom of that layer will be set halfway between the model top and the specified bottom of layer 1. Additional layers could be similarly subdivided by specifying the desired layer number for the next bottom surface elevation.

```yaml
setup_grid:
  source_data:
    features_shapefile:
      filename: 'data/pleasant/source_data/shps/all_lakes.shp'
      id_column: 'HYDROID'
      include_ids: [600059060]
  dxy: 40
  buffer: 1000
  epsg: 3070

dis:
  options:
    length_units: 'meters'
  dimensions:
    nlay: 5
  source_data:

top:
  filename: 'data/pleasant/source_data/rasters/dem40m.tif'
  elevation_units: 'meters'
botm:
  filenames:
    1: 'data/pleasant/source_data/rasters/botm0.tif'
    2: 'data/pleasant/source_data/rasters/botm1.tif'
    3: 'data/pleasant/source_data/rasters/botm2.tif'
    4: 'data/pleasant/source_data/rasters/botm3.tif'
```

The Lake Package (`lak:`) block includes shapefile input to delineate the horizontal extent of the lake, and optionally, a

bathymetry_raster: input to delineate bottom depths that are subtracted off the initial model top (which is assumed to represent the water surface, typically the case for DEMs). Alternatively, a *stage_area_volume_file:* can be specified to allow for more accurate representation of lake volume and surface area as lake levels change. Initial values for lakebed leakance can be input for both a littoral zone extending a specified distance from shore around the perimeter of the lake, and a lower permeability profundal zone in the lake interior (e.g., after Hunt et al., 2013; Leaf and Haserodt, 2020). Finally, climate input, including daily precipitation and mean air temperatures, are supplied in a text file downloaded from the PRISM Climate Group (2019); PRISM provides modeled climate time series at any point location within the United States. Precipitation is used directly by the Lake Package to compute the lake water balance; Modflow-setup uses the Hamon (1961) method to convert daily mean air temperatures to estimates of lake surface evaporation (Harwell, 2012). A *period_stats:* sub-block specifies how the climate input should be aggregated to the model stress periods. For the initial steady-state period, 2012–2018 averages of the daily precipitation and lake surface evaporation are used; subsequently, the average values within each monthly stress period are used. Alternatively, lake climate information can be input directly or supplied in a general CSV format.

```
lak:
  options:
    boundnames: True
    save_flows: True
    surfdep: 0.1
  source_data:
    littoral_leakance: 0.045 # 1/d
    profundal_leakance: 0.025 # 1/d
    littoral_buffer_zone_width: 40
    lakes_shapefile:
      filename: 'data/pleasant/source_data/shps/all_lakes.shp'
      id_column: 'HYDROID'
      include_ids: [600059060] # pleasant lake
    climate:
      filenames:
        600059060: 'data/pleasant/source_data/PRISM_ppt_tmean_stable_4km.csv'
      format: 'prism'
      period_stats:
        0: ['mean', '2012-01-01', '2018-12-31']
        1: 'mean'
    bathymetry_raster:
      filename: 'data/pleasant/source_data/rasters/pleasant_bathymetry.tif'
      length_units: 'meters'
    stage_area_volume_file:
      filename: 'data/pleasant/source_data/tables/area_stage_vol_Pleasant.csv'
      length_units: 'meters'
        id_column: 'hydroid'
        column_mappings:
          volume_m3: 'volume'
    external_files: False  # option to write connectiondata table to external file
```

The *sfr:* block instructs Modflow-setup to generate an SFR Package for the LGR inset model area, using SFRmaker (Leaf et al., 2021). Since this is an LGR inset model, Modflow-setup will automatically set up the Water Mover Package as needed to connect the SFR network across the boundary with the enclosing parent model.

```
sfr:
  options:
    save_flows: True
  source_data:
    flowlines:
      nhdplus_paths: ['data/pleasant/source_data/shps']
    dem:
      filename: 'data/pleasant/source_data/rasters/dem40m.tif'
      elevation_units: 'meters'
  sfrmaker_options:
    set_streambed_top_elevations_from_dem: True
```

To simplify input as much as possible, Modflow-setup includes configuration files of default settings for MODFLOW 6 and MODFLOW-NWT models. In constructing a model, the default configuration files are read first, and the settings within them are recursively updated with user-specified input. Therefore, many settings are optional. For example, `save_flows: True` in the *sfr:* block earlier is also specified in the default configuration, making it technically redundant, although perhaps useful as a placeholder to turn the setting on or off. Other examples of default configurations include the Output Control Package, which is generated by default to save output on the last timestep of each stress period, and initial heads, which are set to the model top by default if no configuration is specified. The default configuration files can be viewed in the online documentation.

The *obs:* block here illustrates how head observation locations can be supplied from multiple CSV files. In this case, no x and y column arguments are needed, because both files have the default column names of "x" and "y." Non-default column names can be specified with the *column_mappings:* argument. In this example, the column names "obsprefix" and "common_name" are mapped to the default "obsname," column for observation names.

```
obs:
  source_data:
    filenames: ['data/pleasant/source_data/tables/lake_sites.csv'
                'data/pleasant/source_data/tables/wdnr_gw_sites.csv']
    column_mappings:
      obsname: ['obsprefix', 'common_name']
  drop_observations: ['10019209_lk']
```

Since this is an LGR inset model that shares a MODFLOW 6 simulation with the enclosing parent model, the simulation-level Temporal Discretization and Iterative Model Solution Packages are specified in the parent model configuration file. The remaining unspecified packages (Initial Conditions, Output Control, Node Property Flow, Storage, Recharge, and Well packages) are generated for the LGR inset model using the input blocks specified for the parent (MODFLOW 6) model, as described previously. The simplified example version of the Pleasant Lake model from the online documentation is shown in Figure 3.

Discussion

After setting up this framework for model construction and linkage, it is straightforward to evaluate some of the many decisions that are often made once in a modeling workflow and not revisited again, such as spatial discretization, time discretization, changing data sources, or hypothesis testing. In the Pleasant Lake example, the key goal of establishing a causal connection between human water use (chiefly irrigation abstraction) and lake levels required evaluation of multiple conditions. Under a unified representative climate, we evaluated recharge and irrigation-required water abstraction for three land use scenarios: 1) no irrigated agriculture, 2) irrigated agriculture in the footprint of current conditions, and 3) potential maximum irrigated agriculture.

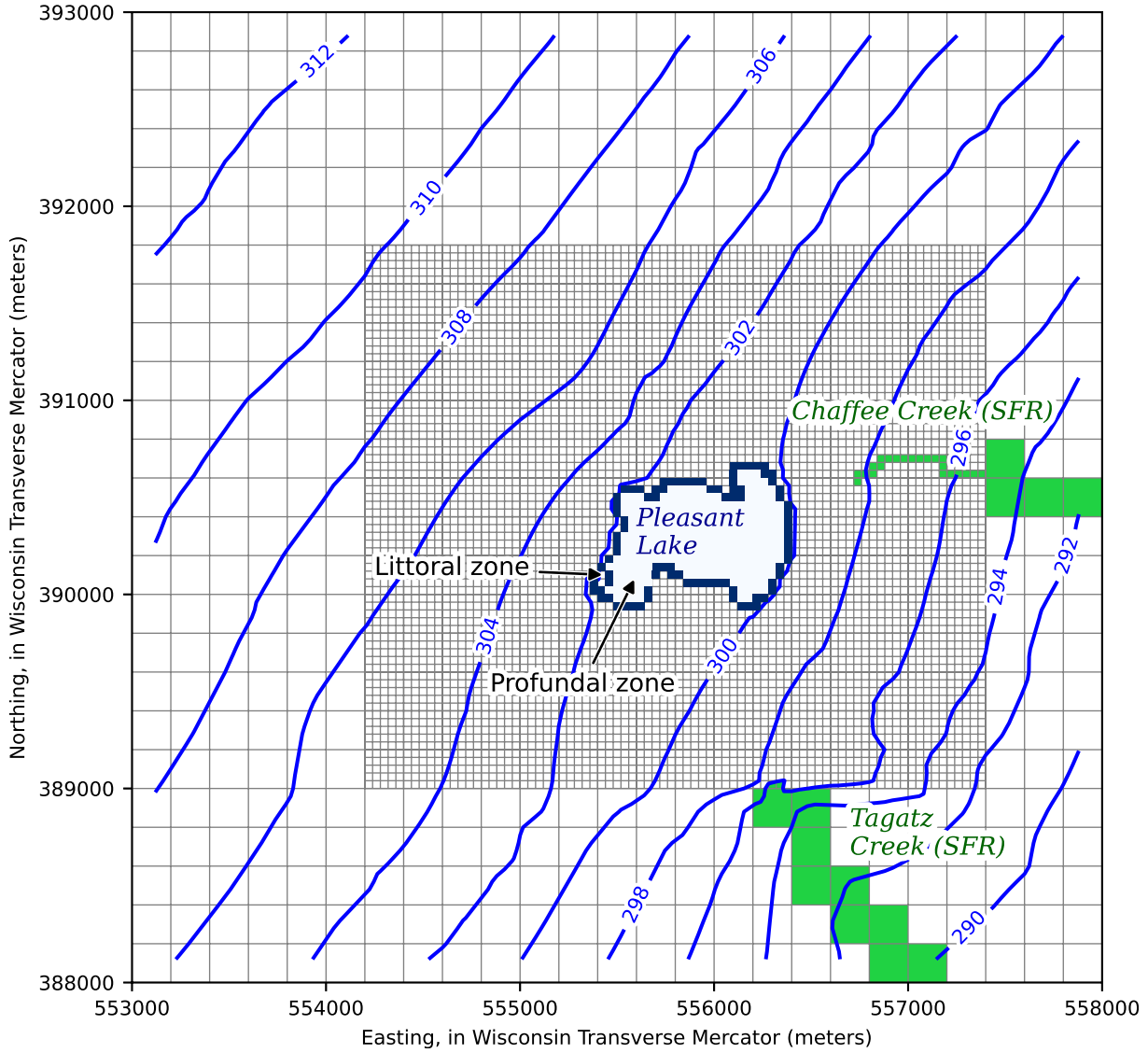

FIGURE 3
Close-up of the example local grid refinement inset model configuration, showing the discretization and combined water table solution, as well as Streamflow Routing (SFR) Package cells and the littoral and profundal lakebed leakance zones.

Consideration of these multiple hypotheses required multiple instances of the SWB model, with the outputs from each instance ingested as recharge and water-use inputs to multiple MODFLOW 6 models. The robust and repeatable nature of the Modflow-setup framework enabled efficient evaluation of the scenarios and has yielded similar benefits to other projects involving multiple numerical models or advanced analyses (e.g., Fienen et al., 2022; 2021a).

As is typical in modeling projects, a single iteration of this workflow was insufficient, as all modeling requires refinement of datasets, testing of hypotheses, and incorporation of lessons learned (e.g., Anderson et al., 2015). For example, examination of model history matching results pointed to the need to better represent headwater springs near the lake. This required rebuilding the SFR package, a task that would be prohibitively time-consuming in a traditional modeling workflow but that was easily done with Modflow-setup. In addition, Modflow-setup allowed for multiple updates to the layering and geological structures represented in the model as new data became available during the course of the project. By opening the numerical model structure to testing and improvement, the

automated workflow enabled by the Modflow-setup can maximize the assimilation of data and ultimately provide models that are better suited for decision support.

It is important to note that while Modflow-setup aims to be general, development is ongoing on the project GitHub site, and to date has focused primarily on meeting project needs through iterative improvement, instead of building a comprehensive tool from the ground up. Some features are incomplete, and others haven't been developed yet. While the configuration file interface is mostly established, it may change somewhat going forward to accommodate new features or improve the user experience. The internal code structure is almost certain to change. While the online documentation is also a work in progress, it aims to accurately describe the current state of the project and how to use it. Contributions and ideas at all levels are encouraged and can be submitted through issues and pull requests on the project GitHub page, or *via* email. In any case, the integration of Modflow-setup with the general Python interface provided by Flopy allows for custom code to be added to a model construction workflow as needed.

Finally, like many open-source software projects, Modflow-setup depends on a large "stack" of other software that is constantly changing. Regular continuous integration testing helps ensure functionality by executing the test suite in freshly built Python environments encompassing the last two minor versions of Python (e.g., 3.10 and 3.9), across the supported platforms. For reproducibility, a project-specific Python environment built from a configuration file works well (for example, a Conda environment file; https://docs.conda.io/). Long-term archives that are meant to persist over years may consider packaging this environment into a stand-alone Python distribution, for example using Conda-pack (https://github.com/conda/conda-pack).

Conclusions

Modflow-setup provides a rapid, reproducible, and robust framework for building MODFLOW models from grid-independent source data. Common model construction tasks are distilled in an open-source, online code base that is tested and extensible through collaborative version control. The workflow for building the model—including input data, construction options, and output packages—is summarized in a single configuration file in the human-readable YAML format. Integration with Flopy allows for additional customization of the model construction workflow as needed. The benefits of Modflow-setup include reduced time and labor required to build a groundwater model, reduced potential for error, improved reproducibility, expanded ability to explore alternative conceptual models or hypotheses, and a reduction in cognitive load that allows the modeler to focus on the most important aspects of the analysis. In the case of the Pleasant Lake model, the robust automation enabled by Modflow-setup allowed for efficient exploration of cumulative pumping impacts to lake levels from hundreds of wells, across multiple scenarios.

Data availability statement

The original contributions presented in the study are included in the article/Supplementary Material; further inquiries can be directed to the corresponding author.

Author contributions

AL has led the code development of Modflow-setup, and contributed approximately 2/3 of the writing to the manuscript. MF has contributed code and ideas to the Modflow-setup project and has tested/applied it extensively in the project work that is cited in the manuscript. MF contributed approximately 1/3 of the writing in the manuscript.

Funding

This project was funded by the U.S. Geological Survey Mississippi Alluvial Plain Project, the Wisconsin Department of Natural Resources, and the Aarhus University Department of Geoscience.

Acknowledgments

The authors would like to thank Aaron Pruitt, Jonathan Traylor, Leslie Duncan, Meg Haserodt, Moussa Guira, Nick Corson-Dosch, Rasmus Frederiksen, Troels Vilhelmsen, and J.R. Rigby for their enthusiastic support and beta testing of Modflow-setup. Any use of trade, firm, or product names is for descriptive purposes only and does not imply endorsement by the U.S. Government.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Anderson, M. P., Woessner, W. W., and Hunt, R. J. (2015). *Applied groundwater modeling*. Second Edition. San Diego: Academic Press.

Bakker, M., Post, V., Langevin, C. D., Hughes, J. D., White, J. T., Starn, J. J., et al. (2016). Scripting modflow model development using python and flopy. *Groundwater* 54 (5), 733–739. doi:10.1111/gwat.12413

Clark, B. R., Barlow, P. M., Peterson, S. M., Hughes, J. D., Reeves, H. W., and Viger, R. J. (2018). *National-scale grid to support regional groundwater availability studies and a national hydrogeologic database*. New York: U.S. Geological Survey data release. doi:10.5066/F7P84B24

Doherty, J., and Moore, C. (2020). Decision support modeling: Data assimilation, uncertainty quantification, and strategic abstraction. *Groundwater* 58, 327–337. doi:10.1111/gwat.12969

Donoho, D. L., Maleki, A., Rahman, I. U., Shahram, M., and Stodden, V. (2008). Reproducible research in computational harmonic analysis. *Comput. Sci. Eng.* 11 (1), 8–18. doi:10.1109/MCSE.2009.15

Fienen, M., and Corson-Dosch, N. (2021). *Groundwater model archive and workflow for neversink/rondout basin*. New York: Source Water Delineation: U.S. Geological Survey Data Release. doi:10.5066/P9HWSOHP

Fienen, M. N., and Bakker, M. (2016). Hess opinions: Repeatable research: What hydrologists can learn from the duke cancer research scandal. *Hydrol. Earth Syst. Sci.* 20 (9), 3739–3743. doi:10.5194/hess-20-3739-2016

Fienen, M. N., Corson-Dosch, N. T., White, J. T., Leaf, A. T., and Hunt, R. J. (2021a). Risk-based wellhead protection decision support: A repeatable workflow approach. *Groundwater* 60, 71–86. doi:10.1111/gwat.13129

Fienen, M. N., Haserodt, M. J., and Leaf, A. T. (2021b). *MODFLOW models used to simulate groundwater flow in the Wisconsin Central Sands Study Area, 2012-2018*. New York: U.S. Geological Survey Data Release. doi:10.5066/P9BVFSGJ

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

Fisher, J. C., Bartolino, J. R., Wylie, A. H., Sukow, J., and McVay, M. (2016). *Groundwater-flow model of the Wood River Valley aquifer system, south-central Idaho*. U.S. Geological Survey Scientific Investigations Report 2016– 5080, 71. doi:10.3133/sir20165080

Gillies, S. (2022b). Rasterio: Access to geospatial raster data. Available at: https://rasterio.readthedocs.io/en/latest/(Accessed January 28, 2022).

Gillies, S. (2022c). Rtree: Spatial indexing for python. Available at: https://toblerity.org/rtree/(Accessed January 28, 2022).

Gillies, S. (2022a). The fiona user manual. Available at: https://fiona.readthedocs.io/en/latest/manual.html (Accessed January 28, 2022).

Gillies, S. (2022d). The shapley user manual. Available at: https://shapely.readthedocs.io/en/latest/manual.html (Accessed January 28, 2022).

Haitjema, H. M. (1995). *Analytic element modeling of groundwater flow*. San Diego, California: Academic Press.

Hamon, W. R. (1961). Estimating potential evapotranspiration: Journal of hydraulics division. *J. Hydr. Div.* 87, 107–120. doi:10.1061/JYCEAJ.0000599

Hanson, R. T., and Leake, S. A. (1999). *Documentation for HYDMOD, a program for extracting and processing time-series data from the U.S. Geological Survey's modular three-dimensional finite-difference ground-water flow model*. U.S. Geological Survey Open-File Report 98-564, 57. doi:10.3133/ofr98564

Harwell, G. R. (2012). *Estimation of evaporation from open water—a review of selected studies, summary of U.S. Army Corps of Engineers data collection and methods, and evaluation of two methods for estimation of evaporation from five reservoirs in Texas*. U.S. Geological Survey Scientific Investigations Report 2012– 5202, 96. doi:10.3133/sir20125202

Hoyer, S., and Hamman, J. (2017). xarray: N-D labeled Arrays and Datasets in Python. *J. Open Res. Softw.* 5 (1), 10. doi:10.5334/jors.148

Hunt, R. J., Doherty, J., and Tonkin, M. J. (2007). Are models too simple? *Ground Water* 45 (3), 254–262. doi:10.1111/j.1745-6584.2007.00316.x

Hunt, R. J., Walker, J. F., Selbig, W. R., Westenbroek, S. M., and Regan, R. S. (2013). *Simulation of climate-change effects on streamflow, lake water budgets, and stream temperature using GSFLOW and SNTEMP, Trout Lake Watershed, Wisconsin*. U.S. Geological Survey Scientific Investigations Report 2013– 5159, 118. doi:10.3133/sir20135159

Langevin, C. D., Hughes, J. D., Banta, E. R., Niswonger, R. G., Panday, S., and Provost, A. M. (2017). *Documentation for the MODFLOW 6 groundwater flow model*. U.S. Geological Survey Techniques and Methods, book 6, 197. chap. A55. doi:10.3133/tm6A55

Leaf, A. T., Fienen, M. N., Hunt, R. J., and Buchwald, C. A. (2015). *Groundwater/surface-water interactions in the bad river watershed, Wisconsin*. U.S. Geological Survey Scientific Investigations Report 2015– 5162, 110. doi:10.3133/sir20155162

Leaf, A. T., and Fienen, M. N. (2022). *Modflow-setup version 0.1*. U.S. Geological Survey Software Release, 1 Aug. 2022. doi:10.5066/P9O3QWQ1

Leaf, A. T., Fienen, M. N., and Reeves, H. W. (2021). SFRmaker and linesink-maker: Rapid construction of streamflow routing networks from hydrography data. *Groundwater* 59, 761–771. doi:10.1111/gwat.13095

Leaf, A. T., and Haserodt, M. J., 2020, Hydrology of haskell lake and investigation of a groundwater contamination plume, lac du Flambeau reservation, Wisconsin: U.S. Geological Survey Scientific Investigations Report 2020– 5024, 79. doi:10.3133/sir20205024

Leake, S. A., and Claar, D. V. (1999). *Procedures and computer programs for telescopic mesh refinement using MODFLOW*. U.S. Geological Survey Open-File Report 99-238, 53. doi:10.3133/ofr99238

Mehl, S., Hill, M. C., and Leake, S. A. (2006). Comparison of local grid refinement methods for MODFLOW. *Ground Water* 44, 792–796. doi:10.1111/j.1745-6584.2006.00192.x

Moore, C., and Doherty, J. (2005). Role of the calibration process in reducing model predictive error. *Water Resour. Res.* 41, W05020. doi:10.1029/2004WR003501

Moran, T. (2016). *Projecting forward: A framework for groundwater model development under the sustainable groundwater management act*. Stanford, CA, USA: Stanford Woods Institute for the Environment, 56. Available at: https://waterinthewest.stanford.edu/sites/default/files/Groundwater-Model-Report.pdf (Accessed July 25, 2022).

Niswonger, R. G., Panday, S., and Ibaraki, M., 2011, *MODFLOW–NWT—a Newton formulation for MODFLOW–2005*. U.S. Geological Survey Techniques and Methods, book 6, 44. chap. A37. doi:10.3133/tm6A45

Peng, R. D. (2011). Reproducible research in computational science. *Science* 334 (6060), 1226–1227. doi:10.1126/science.1213847

PRISM Climate GroupOregon State University (2019). Time series values for individual locations. Available at: https://prism.oregonstate.edu/explorer/(Accessed December 10, 2019).

R Core Team (2014). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Available at:http://www.R-project.org/(Accessed March 9, 2016).

Snow, A. D., Whitaker, J., et al. (2020). *Pyproj documentation*. Available at:https://pyproj4.github.io/pyproj/stable/ (Accessed August 5, 2020).

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cogn. Sci.* 12 (2), 257–285. doi:10.1207/s15516709cog1202_4

Vilhelmsen, T. N., Christensen, S., and Mehl, S. W. (2012). Evaluation of MODFLOW-LGR in connection with a synthetic regional-scale model. *Ground Water* 50, 118–132. doi:10.1111/j.1745-6584.2011.00826.x

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nat. Methods* 17, 261–272. doi:10.1038/s41592-019-0686-2

Westenbroek, S. M., Engott, J. A., Kelson, V. A., and Hunt, R. J. (2018). *SWB Version 2.0—a soil-water-balance code for estimating net infiltration and other water-budget components*. U.S. Geological Survey Techniques and Methods, book 6, 118. chap. A59. doi:10.3133/tm6A59

White, J. T., Doherty, J. E., and Hughes, J. D. (2014). Quantifying the predictive consequences of model error with linear subspace analysis. *Water Resour. Res.* 50, 1152–1173. doi:10.1002/2013WR014767

White, J. T., Hemmings, B., Fienen, M. N., and Knowling, M. J. (2021). Towards improved environmental modeling outcomes: Enabling low-cost access to high-dimensional, geostatistical-based decision-support analyses. *Environ. Model. Softw.* 139, 105022. doi:10.1016/j.envsoft.2021.105022

Wilson, G., Aruliah, D. A., Brown, C. T., Chue Hong, N. P., Davis, M., Guy, R. T., et al. (2014). Best practices for scientific computing. *PLoS Biol.* 12 (1), e1001745. doi:10.1371/journal.pbio.1001745

Wisconsin Department of Natural Resources (WDNR) (2021). Central Sands Lake study report: Findings and recommendations. *Rep. Wis. State Legislature*. doi:10.5281/zenodo.5708791

Wisconsin Department of Natural Resources [WDNR] (2019). Digital elevation model (DEM) – 10 meter. Available at: https://data-wi-dnr.opendata.arcgis.com/search?q=DEM (Accessed July 25, 2022).