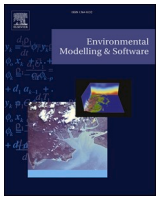
Environmental Modelling and Software 148 (2022) 105257

Contents lists available at ScienceDirect

Environmental Modelling and Software

journal homepage: www.elsevier.com/locate/envsoft



The MODFLOW Application Programming Interface for simulation control and software interoperability

Joseph D. Hughes [a,*,1], Martijn J. Russcher [b], Christian D. Langevin [c,2], Eric D. Morway [d,3], Richard R. McDonald [e,4]

[a] U.S. Geological Survey, Earth Systems Modeling Branch, 927 W Belle Plaine Ave, Chicago, IL, USA
[b] Deltares, Boussinesqweg 1, 2629, HV Delft, the Netherlands
[c] U.S. Geological Survey, Earth Systems Modeling Branch, 2280, Woodale Dr, Mounds View, MN, USA
[d] U.S. Geological Survey, Nevada Water Science Center, 2730, N. Deer Run Rd, Carson City, NV, USA
[e] U.S. Geological Survey, Earth Systems Modeling Branch, Mail Stop 412, PO Box 25046, Denver, CO, USA

ARTICLE INFO

Keywords:
MODFLOW 6
MODFLOW API
Basic model interface
MetaSWAP
PRMS
MODSIM

ABSTRACT

The MODFLOW API allows other programs to control MODFLOW and interactively change variables without having to modify the source code. The MODFLOW API is based on the Basic Model Interface (BMI), which is a set of conventions that define how to initialize a simulation, update the model state by advancing in time, and finalize the run. For many existing MODFLOW coupling applications, the information provided to MODFLOW must be updated multiple times in a time step. As this capability to modify variables within a time step is not defined by the BMI, an extension to BMI was developed. This eXtended Model Interface is part of the MODFLOW API and allows such a tight coupling to other models. Examples are included for a variety of use cases, including new flexibility for users to develop custom packages without modifying the MODFLOW source code and coupling MODFLOW with other models and optimization libraries.

1. Introduction

For over 30 years, the MODFLOW program has been widely used by academics, private consultants, and government scientists to accurately, reliably, and efficiently simulate groundwater flow and related processes. Due to its widespread popularity, modular structure, and thorough documentation, MODFLOW has been successfully coupled with many other physical process models and programs. For example, MODFLOW has been coupled with watershed models, including PRMS (Markstrom et al., 2008), SWAT (Kim et al., 2008), and HSPF (Davis, 2001). MODFLOW has been combined with solute transport models, including MOC3D (Winston et al., 2018) and MT3D (Langevin et al., 2008). Optimization routines have been coupled with MODFLOW to maximize groundwater extraction subject to various constraints (Ahlfeld et al., 2005) and to identify optimum surface water deliveries in order to meet water demands (Morway et al., 2016). MODFLOW has been extended to include pipe flow models to represent karst conditions (Shoemaker et al., 2008) and flow in the unsaturated zone (Twarakavi et al., 2008). MODFLOW has also been coupled to hydrodynamic surface-water models, including HEC-RAS (Rodriguez et al., 2008), SWIFT2D (Wang et al., 2007), and BRANCH (Swain and Wexler, 1996). The majority of previous coupling approaches used with MODFLOW

Abbreviations: API, Application Programming Interface; BMI, Basic Model Interface; CCA, Common Component Architecture; CSDMS, Community Surface Dynamics Modeling System; ESMF, Earth System Modeling Framework; EVT, Evapotranspiration; GWF, Groundwater Flow; GWT, Groundwater Transport; HRU, Hydrologic Response Unit; OpenMI, Open Modeling Interface; PRMS, Precipitation-Runoff Modeling System; SFR, Streamflow Routing; UZF, Unsaturated Zone Flow; XMI, eXtended Model Interface.

* Corresponding author.
E-mail addresses: jdhughes@usgs.gov (J.D. Hughes), Martijn.Russcher@deltares.nl (M.J. Russcher), langevin@usgs.gov (C.D. Langevin), emorway@usgs.gov (E.D. Morway), rmcd@usgs.gov (R.R. McDonald).
[1] https://www.usgs.gov/staff-profiles/joseph-hughes.
[2] https://www.usgs.gov/staff-profiles/christian-langevin.
[3] https://www.usgs.gov/staff-profiles/eric-morway.
[4] https://www.usgs.gov/staff-profiles/richard-mcdonald.

https://doi.org/10.1016/j.envsoft.2021.105257
Received 28 May 2021; Received in revised form 22 September 2021; Accepted 16 November 2021
Available online 30 November 2021

have been to combine the codes into a single program. This single-program coupling approach has often led to coupled programs that could not keep up with advances of the individual codes. As a result, many of these custom MODFLOW-variants have become stale over time and do not support recent MODFLOW advances or advances to the other process model.

Efforts to couple MODFLOW with other process models requires extensive knowledge of both models, an understanding of the types of information needed for the coupling, performance objectives, and a clear purpose for the resulting coupled MODFLOW program. These broader model integration issues are described by Belete et al. (2017) in a comprehensive overview. The overview outlines phases of the model integration effort, summarizes capabilities of existing frameworks for coupling models, and describes the challenges of designing a generalized integration framework. Overcoming these challenges can allow for new and innovative ways for coupling models, including sharing models and data over the web (Chen et al., 2020).

MODFLOW6 is the most recent version of MODFLOW and is currently the core version of MODFLOW distributed and released by the U.S. Geological Survey. MODFLOW6 is an object-oriented program and framework written in Fortran. The MODFLOW6 software provides a platform for supporting multiple models and multiple types of models within the same simulation (Langevin et al., 2017; Hughes et al., 2017). These models can be independent of one another with no interaction, they can exchange information, or they can be tightly coupled at the matrix level by adding them to the same numerical solution. Transfer of information between models is isolated to exchange objects, which contain the data and code needed to couple models. This design feature allow models to be developed, maintained, and used independently without having custom information about other models in the framework. Within this new framework, a regional-scale groundwater model may be coupled with multiple local-scale groundwater models, or a groundwater transport model may be coupled with a groundwater flow model (Langevin et al., 2020).

MODFLOW6 currently includes the Groundwater Flow (GWF) Model and the Groundwater Transport (GWT) Model each with packages to represent surface water processes, groundwater extraction, external boundaries, mass sources and sinks, and mass sorption and reactions. Morway et al. (2021) show how the advanced packages in MODFLOW 6 can be connected to represent watershed processes in managed basins. MODFLOW6 also includes advanced capabilities to simulate three-dimensional anisotropy and dispersion and correct grid errors for cell connections that violate generalized control-volume finite--difference assumptions (Panday et al., 2013; Provost et al., 2017).

To facilitate coupling with other models, including those written in another language, an Application Programming Interface (API) was developed for MODFLOW. The API was developed using the established Basic Model Interface (BMI) standard. The BMI was developed by Community Surface Dynamics Modeling System (CSDMS) to provide a standard component-based interface for Earth-science models (Peckham et al., 2013). The Common Component Architecture (CCA; Armstrong et al., 1999), the Earth System Modeling Framework (ESMF; Collins et al., 2005), and the Open Modeling Interface (OpenMI; Gregersen et al., 2007) are examples of other coupling standards. The BMI standard was selected for the MODFLOW API because it was developed specifically for Earth-science models, and it was clear how the BMI could be used to couple MODFLOW with the types of process models needed by the community. Some example applications of the BMI with Earth-science models include the coupling of 1) a river sediment transport model to a delta model that distributes the sediment (Ratliff et al., 2018) and 2) a hydrologic model to a hydrodynamic model to improve flood inundation simulations (Hoch et al., 2019). Since its initial release, development of the BMI has continued and updates have included functions for accessing variable metadata and for working with structured and unstructured grids (Hutton et al., 2020).

Although the BMI provides the foundation for the MODFLOW API, additional functionality was required to allow for a tighter coupling with other process models than is possible with the standard BMI functions. Specifically, there was a need to allow other components to be coupled with MODFLOW within the non-linear Picard iteration loop (see, for example, Ferziger and Peric (1996) for a treatment of this type of coupling and the Picard iteration). For example, GSFLOW (Markstrom et al., 2008) is a coupled version of MODFLOW and PRMS (Markstrom et al., 2015). A key feature of GSFLOW is the tight coupling of the PRMS soil zone and overland flow components with the MODFLOW groundwater flow component. It is implemented in GSFLOW by allowing the soil zone component to be calculated as part of the MODFLOW Picard iteration until convergence is achieved. In the MODFLOW API, the BMI was extended to support this same type of tight component coupling that was used in GSFLOW, but in a generic way that allows other models to be solved simultaneously with MODFLOW.

The purpose of this paper is to describe the new MODFLOW API, which is implemented in MODFLOW6 (subsequent MODFLOW references in this paper refer specifically to the MODFLOW6 version). Although the focus of the paper is on coupling MODFLOW with other process models, the API is general and will allow MODFLOW to be called by a wide variety of software programs, such as by plotting programs or Geographical Information Systems, for example. The paper first shows how the BMI was implemented in MODFLOW, including an explanation of how a MODFLOW simulation can be controlled by an external program written in another language (*e.g.*, Python, C#, *etc.*) using model control and time functions and the way MODFLOW variables can be accessed during the simulation. The paper then describes extensions to the BMI that allow MODFLOW to be tightly coupled with other process models. Finally, five examples are presented to demonstrate a variety of uses for the MODFLOW API, including new flexibility for users to develop custom packages without modifying the MODFLOW source code and coupling MODFLOW with other process models and optimization libraries.

2. Development of the MODFLOW API

Development of the MODFLOW API required refactoring the source code into the core component. This core component can be compiled into an executable program (Fig. 1), which is provided as part of the standard distribution released by the U.S. Geological Survey. The core component can also be compiled with the API routines into a shared library that can be used for interoperable applications. The refactoring effort focused on ensuring that the routines and calls for the executable corresponded exactly to calls made to the shared library through the API and that none of the routines were duplicated. Extensive testing was performed and continues to be performed through continuous integration practices to ensure that the executable and library versions give identical numerical results and have equivalent run times with multiple compilers on the supported operating systems: Windows, Linux, and macOS.

As shown in Fig. 1 the API layer builds on the MODFLOW core. MODFLOW is considered to be either the executable version or the shared library version as the latter can be initiated and run in a manner that gives the same results as the executable. Fig. 1 shows three different implementations of an interoperable layer that sits between the MODFLOW API and applications based on it. An interoperable layer, such as the modflowapi Python package, makes it easier to control MODFLOW and transfer information. The purpose of this section is to describe the API layer shown in Fig. 1 including both the BMI and the eXtended Model Interface (XMI). The subsequent section contains examples on how to use the MODFLOW API in practice.

As shown in Fig. 1 the API layer builds on the MODFLOW core. It currently contains the eXtended Model Interface (XMI), which is the Basic Model Interface (BMI) and the extensions described below, plus supporting functionality. However, it is expected that it will continue to grow, exposing more internal MODFLOW functionality for use in other

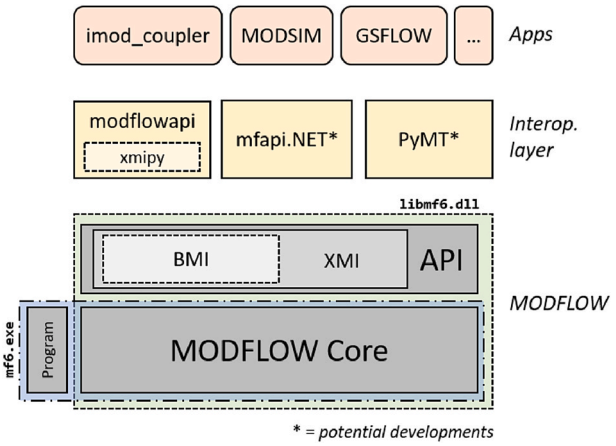


Fig. 1. Overview of software components and applications in relation to the MODFLOW API. The Core contains all simulation code and can be compiled with a driver program into the MODFLOW executable (mf6. exe) and with the API into a shared library (libmf6. dll). An interoperability layer sits between MODFLOW and possible software applications. The interaction from Python is facilitated with the modflowapi package which extends the bindings for the XMI available in xmipy. As potential developments, the API can be exposed to C#.NET (mfapi.NET) or connected to the PyMT framework (see Hutton and Piper (2020) for more information on PyMT). The Apps layer shows external simulation programs that can interact with the MODFLOW API. Developments that concern the connection of imod_coupler, MODSIM, and GSFLOW to the MODFLOW API are ongoing and presented in section 3.

applications. MODFLOW is considered to be either the executable version or the shared library version as the latter can be initiated and run in a manner that gives the same results as the executable. Fig. 1 shows three different implementations of an interoperable layer that sits between the MODFLOW API and applications based on it. An interoperable layer, such as the modflowapi Python package, makes it easier to control MODFLOW and transfer information. The purpose of this section is to describe the API layer shown in Fig. 1 including both the BMI and the eXtended Model Interface (XMI). The subsequent section contains examples on how to use the MODFLOW API in practice.

2.1. Implementation of the basic Model Interface

In order to expose the MODFLOW functionality with the main BMI control functions (initialize, update, finalize), the source code has been refactored. A high-level API is developed to aggregate the traditional MODFLOW subroutines into functional units that can be mapped directly to the BMI initialize, update, and finalize functions as shown by the diagram in Fig. 2. The main reason for the code refactoring is the requirement that the same program code is executed regardless of whether the simulation is run with the executable or with the shared library containing the newly developed BMI.

As implied by the control flow in Fig. 2, the caller of the BMI control functions is also responsible for implementing the time-step loop. The simulation time inside MODFLOW is divided into stress periods which are subdivided into time steps. Stress periods are intervals during which external stresses can be redefined and are provided as a convenience for users. The time steps constitute the potentially non-equidistant discretization of time used to solve the numerical models in the simulation. Note that as opposed to previous versions of the code, there is no explicit loop over stress periods anymore in MODFLOW which complies with the interpretation of the time loop as prescribed by the BMI. A call to the update function advances the simulation by a single time step. The start time of the simulation (get_start_time) is fixed at 0.0 for all simulations. The end time (get_end_time) is set to the total simulation time which is equal to the summed lengths of the individual time steps. The current time can be queried with a call to get_current_time but it is important to realize that its value is not updated until the internal time update (TU) is called. The current time-step length can be queried with a call to get_time_step.

The other part of the BMI implementation is its ability to access internal model component data. Nearly all array and scalar variables in MODFLOW are declared as Fortran pointers (Metcalf et al., 2018) and managed by a dedicated module called the Memory Manager, which allocates variables and tags them with a unique address string composed of the variable's name and a memory path identifying the unique name of the MODFLOW component type and for MODFLOW package variables the unique name of the package it belongs to. Examples of the address string for a number of characteristic variables is shown in Table 1. MODFLOW component types include timing, solution, model, exchange, and utility components (see Hughes et al., 2017, for more information). For user convenience and to assure future compatibility with the library, the get_var_address utility function is available to construct this string based on the syntax used internally by the Memory Manager. An inventory of all accessible variables can be retrieved with the BMI function get_input_var_names (or get_output_var_names as no distinction has been made between input or output variables in the MODFLOW API) or by setting MEMORY_PRINT_OPTION to ALL in the options block of the simulation namefile.

Although in principle it is possible to modify all variables inside the Memory Manager, this should be used with caution. For instance, the BMI could be used to change horizontal hydraulic conductivity (k11) inside the Node Property Flow (NPF) Package after the initialization phase in an attempt to simulate time-varying hydraulic properties. However, the saturated conductance (condsat) is used to calculate terms in the coefficient matrix and although it depends on k11, it is calculated only once at the beginning of the simulation (during initialize). Therefore, modifying k11 after initialization would have no effect on the simulation results. Other parameters, such as those related to the dimensionality of the coefficient matrices or to the time discretization, should probably not be modified to avoid undefined behavior and possible program failure. Section 3 contains multiple validated use cases of reading and writing data using the MODFLOW API procedures. In general, the user should consult the source code to confirm the validity of a specific application.

The implementation of the interface functions follows the CSDMS Fortran 2003 BMI specification (Piper, 2020). However, with the goal of developing a universally accessible library in mind, small modifications had to be made, mostly avoiding the use of Fortran-specific data types. The resulting API is ensured to have Standard C (Kernighan and Ritchie, 1988) interoperability.

2.2. Development of the eXtended Model Interface

BMI only allows a sequential, loose coupling of process models. To enable a tighter coupling, at the iteration level, but also to provide more fine-grained control of the simulation, we have developed the eXtended Model Interface (XMI) which consists of all BMI functionality plus the necessary extensions. A first extension is the subdivision of the update function into smaller units to allow the interactive modification of time-varying data. These data are read from file for each time step in a dedicated Read and Prepare (RP) step, as shown in Fig. 3. In the XMI function prepare_time_step the data are read and before an external call to do_time_step is made, they can still be changed using the BMI functions.

A second extension is the explicit exposure of the non-linear convergence loop of a numerical solution. This is shown schematically in Fig. 4. With such fine grained control, it is now possible to tightly-couple MODFLOW to other model components and have the solution converge at the outer iteration level, i.e., within a single time step. This is often preferable to a loosely-coupled sequential alternative, where one model finishes a time step and only then provides data that is used as boundary conditions for the other model. The latter can cause the

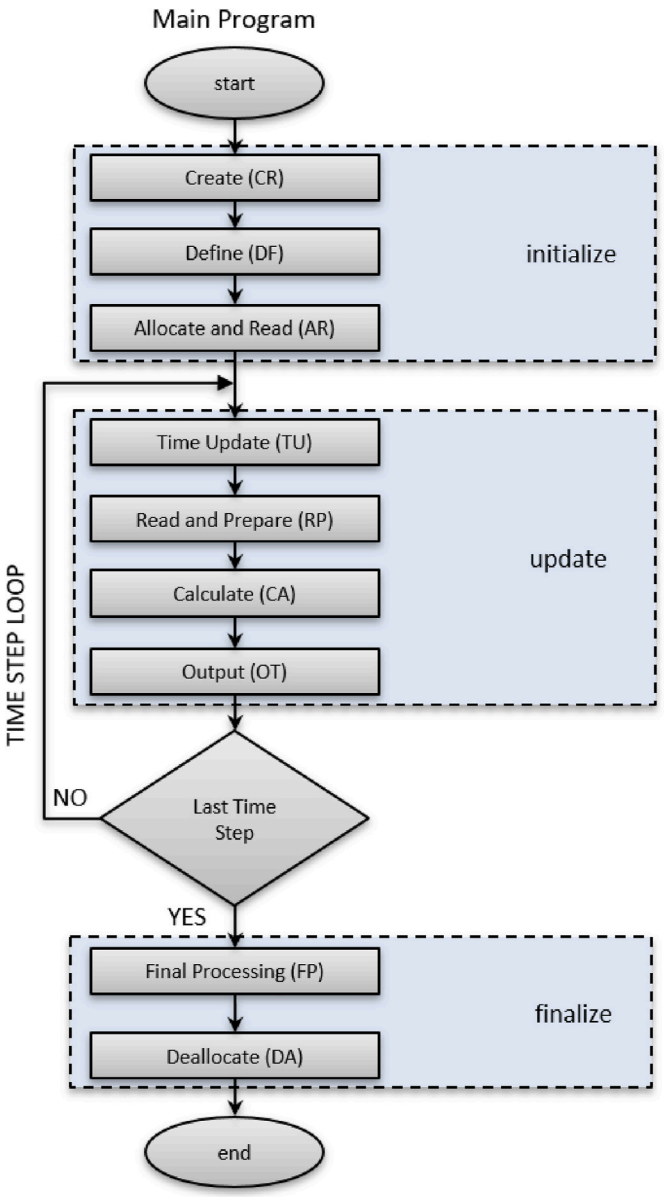


Fig. 2. Grouping of MODFLOW routines into the standard BMI initialize, update, and finalize functions. To replicate the behavior of the main program using these BMI functions, the calling program should implement a loop to execute the time steps, analogous to the TIME STEP LOOP shown in figure. A description of the subroutines mentioned in the diagram (Create (CR), Define (DF), etc.) can be found in Hughes et al. (2017).

solution to oscillate in time and result in a simulation that is sensitive to the order in which the components are executed. The value of having this extra functionality in the MODFLOW API is clearly demonstrated by the fact that most of the example applications presented in Section 3 require this tight-coupling scheme.

It is important to realize that the flow chart shown in Fig. 3 corresponds to a simulation with a single numerical solution. Although this might be the most frequent use case, MODFLOW accommodates a simulation with multiple solutions, for example, when running groundwater flow and transport simultaneously. To replicate the behavior of do_time_step in this case, an enveloping loop running over the total number of solutions should be implemented in the calling program or script.

One of the key benefits of having the XMI available as an Standard C compliant library is the capability to control the simulation from

Table 1
The composition of MODFLOW API memory addresses for variables inside a solution (SLN), the dependent variable for the groundwater flow and transport models, three distinct MODFLOW model packages (GWF/NPF, GWF/RIV, and GWT/MST), and a groundwater flow exchange (GWF-GWF). Note that the MODFLOW component and package types have unique names, which are used as part of the address string. This is not an exhaustive list of MODFLOW variables.

Variable	Description	MODFLOW Component Type	MODFLOW Package Type	Memory Address
mxiter	Maximum outer iterations	SLN	- -	*"SLN$_1$/MXITER"*
x[a]	Simulated heads	GWF	- -	*"MYGWF/X"*
k22	K_H for the second ellipsoid axis	GWF	NPF	*"MYGWF/NPF/K22"*
x[a]	Simulated concentrations	GWT	- -	*"GWT$_1$/X"*
Porosity	Porosity	GWT	MST	*"GWT$_1$/MST/POROSITY"*
Condsat	Saturated connection conductance	GWF-GWF	- -	*"EXG$_M$1M2/CONDSAT"*
Nodelist	1-D node numbers for *bound*	GWF	RIV	*"MYGWF/RIVA/NODELIST"*
bound[b]	2-D boundary data array	GWF	RIV	*"MYGWF/RIVA/BOUND"*

[a] The dependent variable name for the GWF (head) and GWT (concentration) models is x for both models.
[b] The bound variable contains the stress package data for the standard stress packages. For example, columns in bound would contain the volumetric well rate for the WEL package and the boundary head and conductance for the GHB package.

programming environments other than Fortran. This design makes the MODFLOW API straightforward to integrate into programs written in, for example, C++, Python, Java, and C#.NET, or to run simulations interactively from a Jupyter notebook (Pérez and Granger, 2007; Kluyver et al., 2016). Because Python is such a widespread and highly valued scripting language, the xmipy (Russcher et al., 2020) and modflowapi (Hughes et al., 2021) Packages have been developed to facilitate use of the MODFLOW API. The xmipy Package is a key part of the interoperable layer shown in Fig. 1 as it contains the complete set of Python bindings for the native BMI and XMI functions in MODFLOW. It fully encapsulates the complexity of type marshaling between Python and Fortran and the memory management of the data, which allows users to work with comprehensible functions and standard Python data types. The modflowapi Package extends the xmipy Package and includes the Python binding for the get_var_address Memory Manager convenience function. In the future, it will be extended with additional functionality that is made available through the MODFLOW API.

3. Example applications

We present a few examples below that demonstrate use of the MODFLOW API. The first example simulates evapotranspiration using different approaches and represents an example of how the API can be used to rapidly prototype new MODFLOW functionality. The second example shows how the MODFLOW API can be used to optimize groundwater pumpage with standard optimization methods available in the Python SciPy Package (Virtanen et al., 2020). The third example is a tight coupling with MetaSWAP (van Walsum and Groenendijk, 2008; van Walsum and Veldhuizen, 2011), to simulate unsaturated zone flow processes and groundwater recharge. The fourth example demonstrates the sequential loose coupling of PRMS using only BMI functionality in the MODFLOW API and includes mapping data from non-rectangular PRMS control volumes to grid-based MODFLOW control volumes. The fifth example demonstrates use of the MODFLOW API in the river

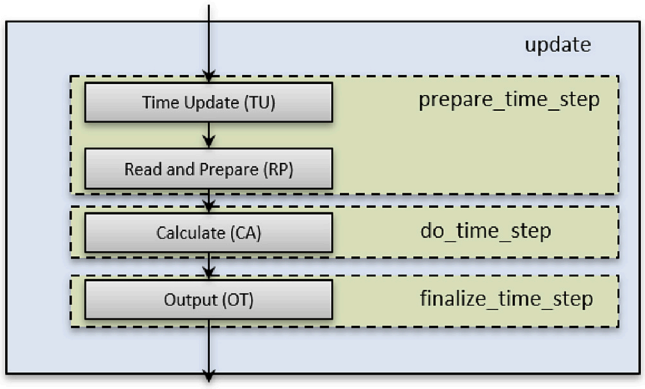


Fig. 3. Subdivision of the BMI update function into prepare_time_step, do_time_step, and finalize_time_step functions as part of the XMI. These functions allow data read from files in the Read and Prepare (RP) step to be replaced using the BMI data access routines.

operations model MODSIM (Labadie et al., 2000), which is coded in C#, to optimize irrigation diversions, represented with the MODFLOW Streamflow Routing (SFR) Package, to meet prior appropriation constraints. Jupyter notebooks are available for all of the examples and include further details on the model setup, the coupling, and the software required to reproduce the examples (see Section 4).

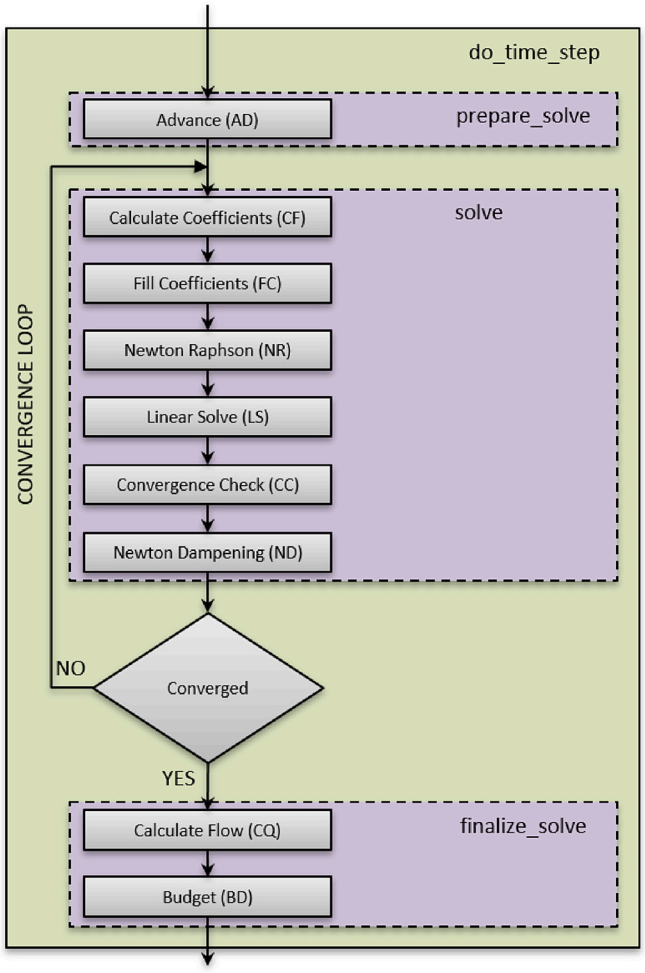


Fig. 4. Subdivision of the XMI do_time_step into prepare_solve, solve, and finalize_solve functions. These XMI functions allow the modification of data at each outer iteration using standard BMI data access functions and permits a tight coupling of MODFLOW with other codes.

3.1. Development of a custom MODFLOW package using python

MODFLOW has simplified equations for dynamically calculating evapotranspiration as a function of the simulated water table depth. This example shows how the API can be used to implement alternative evapotranspiration functions without modifying the MODFLOW source code. A Python routine was written to simulate evapotranspiration that varies from a maximum value at or above defined elevation and decays to zero at a defined extinction depth using constant, linear, and exponential functions (Fig. 5). For this simple example, the MODFLOW model has 10 layers, 1 row, and 1 column. A constant grid spacing of 10 m was used in the row and column directions. The top and bottom were set to 0 and −5 m, respectively, and the layers were discretized using a constant 0.5 m thickness. A constant horizontal conductivity of 1 m d^{-1}, specific storage of 1.5×10^{-5} m^{-1}, specific yield of 0.2, and initial head of 1 m above land surface were specified in each cell. The model was transient and ran for 1000 days using 1000 variable length time steps and a time step multiplier of 1.05; time step lengths ranged from 3.234×10^{-20} d at the beginning of the simulation and increased to 47.619 d at the end of the simulation. To improve model convergence the Newton-Raphson formulation was used.

A maximum evapotranspiration rate (q_{max}) of 6×10^{-4} m d^{-1}, an evapotranspiration surface elevation of −0.25 m (depth of 0.25 m below land surface), and a evapotranspiration extinction elevation of 3 m (depth of 3 m below land surface) were specified to simulate evapotranspiration in the model (Fig. 5). The linear evapotranspiration function is identical to the MODFLOW Evapotranspiration (EVT) Package (Langevin et al., 2017). The constant and exponential evapotranspiration functions are not directly available in the EVT Package. The constant evapotranspiration function was implemented in Python using

$$Q_{ET} = \begin{cases} Q_{max} & h \geq z_e \\ 0 & h < z_e \end{cases}, \quad (1)$$

where Q_{ET} is the volumetric evapotranspiration rate (m^3 d^{-1}), Q_{max} is the maximum volumetric evapotranspiration rate (m^3 d^{-1}), h is the simulated groundwater head (m), and z_e is the evapotranspiration extinction elevation (m). The volumetric evapotranspiration rate (Q_{max}) is the product of the horizontal cell area (A) and q_{max}. The linear evapotranspiration function was implemented in Python using

$$Q_{ET} = \begin{cases} Q_{max} & h \geq z_s \\ Q_{max}(1 - \widehat{d}) & z_s > h \geq z_e \\ 0 & h < z_e \end{cases}, \quad (2)$$

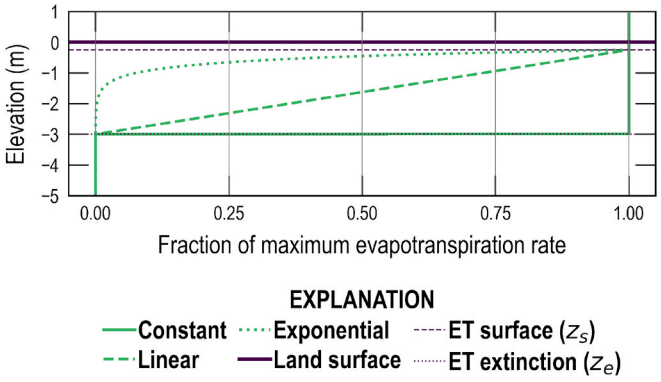


Fig. 5. Functions used to demonstrate use of the Basic Model Interface to simulate different head-dependent evapotranspiration models in MODFLOW. The land surface elevation, the elevation above which maximum evapotranspiration rates occur (z_s), and the elevation below which evapotranspiration rates are zero (z_e) used in the model are also shown.

where z_s is the evapotranspiration surface elevation (m), $\widehat{d}$ is the depth of the simulated head below z_s normalized by the thickness of the interval over which Q_{ET} varies from $Q_{\max}$ to 0 (unitless). $\widehat{d}$ is calculated as

$$\widehat{d} = \frac{z_s - h}{z_s - z_e}. \tag{3}$$

The normalized depth varies from 0 at z_s to 1 at z_e. The exponential evapotranspiration function was implemented in Python using

$$Q_{ET} = \begin{cases} Q_{max} & h \geq z_s \\ Q_{max} e^{-3\pi\widehat{d}} & z_s > h \geq z_e \\ 0 & h < z_e \end{cases}. \tag{4}$$

The Newton-Raphson method was used to linearize equations (1), (2) and (4) and calculate the hcof and rhs variables for the API Package, which was used to represent evapotranspiration in the model. The API Package was developed to provide a generic way to add terms to the MODFLOW system of equations when using the MODFLOW API.

The MODFLOW API for the evapotranspiration example is accessed in Python by first instantiating a ModflowApi object (mf6), which is imported from modflowapi, using

```
mf6 = ModflowApi("libmf6.dll")
```

The mf6 object has access to all of the methods exposed in the MODFLOW API. Next, the base MODFLOW model is initialized from the mfsim. nam file in the working directory using

```
mf6.initialize()
```

Program memory is allocated and static data are read when initialize is called. The current time (0 d when called right after initialize) and simulation time at the end of the simulation are used to control the MODFLOW time step loop and are determined using

```
current_time = mf6.get_current_time()
end_time = mf6.get_end_time()
```

The maximum number of non-linear iterations specified in the Iterative Model Solution (IMS) are determined using

```
max_iter = mf6.get_value(
    mf6.get_var_address("MXITER", "SLN_1")
)
```

An array pointing to simulated heads for the evapotranspiration model is set using

```
head = mf6.get_value_ptr(
    mf6.get_var_address("X", gwfname)
)
```

In this statement, head is the name of the Python variable of type numpy. ndarray, a data type in the Python Numpy Package (Harris et al., 2020), that points to the head array in MODFLOW, and gwfname is the user-specified name of the groundwater flow model, which is "etmodel" in this example. The evapotranspiration variables hcof and rhs and the model layer containing the water table are both calculated using the values in head. The variable addresses to access the data in the API Package are

```
node_addr = mf6.get_var_address(
    "NODELIST", gwfname, "BMI-ET"
)
hcof_addr = mf6.get_var_address(
    "HCOF", gwfname, "BMI-ET"
)
rhs_addr = mf6.get_var_address(
    "RHS", gwfname, "BMI-ET"
)
```

node_addr, hcof_addr, and rhs_addr are used in the set_value method to modify the cell evapotranspiration is extracted from and the coefficient matrix and right-hand side terms used to represent evapotranspiration in the system of equations, respectively. Node numbers are calculated from user-specified cell IDs, which are (layer, row, column) in this example, and range from 1 to the number of cells in the groundwater flow model.

The Python code shown in Fig. 6 determines the node evapotranspiration is extracted from, calculates the evapotranspiration terms, and solves the GWF Model for a time step until convergence is achieved and then proceeds to the next time step until the entire simulation is complete. The get_node function returns the one-based model node number for the cell containing the water-table. The et_terms function returns the hcof and rhs variables that are used to calculate the current head-dependent volumetric evapotranspiration rate (Q_{ET}), based on equation (1), 2, or 4. The hcof and rhs variables are added by MODFLOW to the diagonal of the coefficient matrix and the right-hand side, respectively, when the system of equations are formulated for the current outer iteration of a time step (for more information see Hughes et al., 2017; Langevin et al., 2017).

Simulated results for the three evapotranspiration functions are shown in Fig. 7. Simulated water levels decline from the initial value of 1 m for all three evapotranspiration functions (Fig. 7*A*). The cumulative evapotranspiration for all three functions is shown in Fig. 7*B*. As expected, the cumulative evapotranspiration is highest for the constant evapotranspiration function (60.0 m^3) and results in the lowest water level (−3.0 m) at the end of the simulation. The cumulative evapotranspiration is smallest for the exponential evapotranspiration function (18.6 m^3) and results in the highest water level (−0.9 m) at the end of the simulation. The cumulative evapotranspiration and water level for

```
while current_time < end_time:
    dt = mf6.get_time_step()
    mf6.prepare_time_step(dt)
    kiter = 0
    mf6.prepare_solve()
    while kiter < max_iter:
        tnode[:] = get_node(head)
        hcof[:], rhs[:] = et_terms(
            head,
            function="exponential")
        mf6.set_value(node_addr, tnode)
        mf6.set_value(hcof_addr, hcof)
        mf6.set_value(rhs_addr, rhs)
        has_converged = mf6.solve()
        kiter += 1
        if has_converged:
            break
    mf6.finalize_solve()
    mf6.finalize_time_step()
    current_time = mf6.get_current_time()
    if not has_converged:
        print("model did not converge")
        break
try:
    mf6.finalize()
    success = True
except:
    raise RuntimeError
```

Fig. 6. Python code used to run the MODFLOW model with the selected head-dependent evapotranspiration function as part of the non-linear outer iteration loop. Although this code snippet does not run on its own, it demonstrates the concept of getting and setting MODFLOW variables, such as the node number (nodelist array of the API Package) and the evapotranspiration terms (hcof and rhs arrays of the API Package). The Python code has been modified slightly from the working example for illustrative purposes.

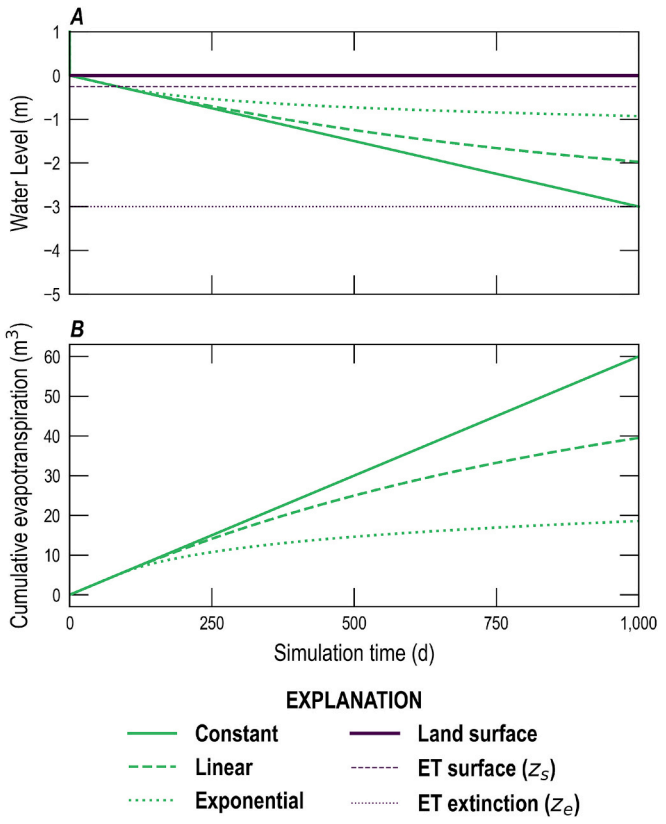


Fig. 7. Simulated water level and cumulative evapotranspiration rates calculated using the constant, linear, and exponential functions. *A*. Simulated water levels. The land surface elevation, evapotranspiration surface, and evapotranspiration extinction elevation are also shown. *B*. Simulated cumulative evapotranspiration rates.

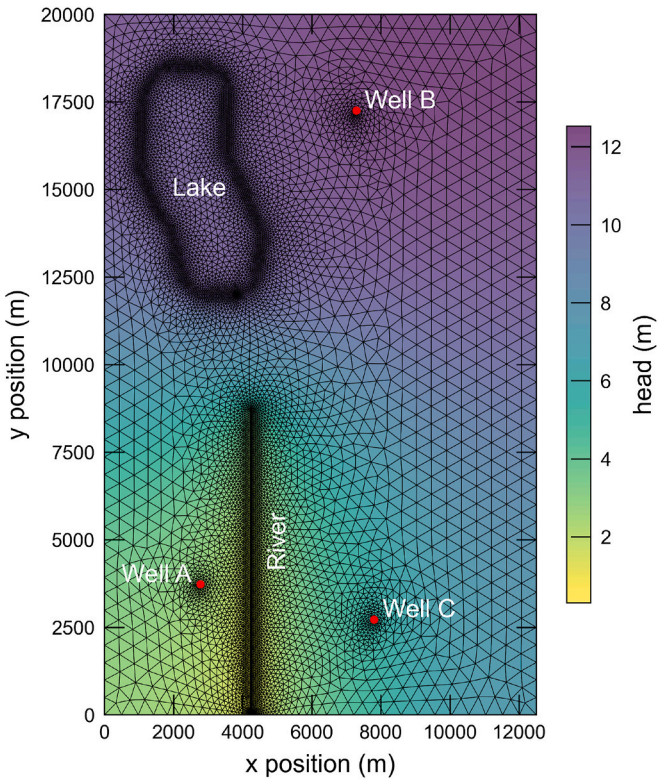


Fig. 8. Model grid used to demonstrate optimization of groundwater withdrawals subject to drawdown constraints. Cells are shaded using simulated head in layer 1 for a simulation without groundwater withdrawals. The five-layer model is characterized by a lake, a straight river, and three optimization wells, labeled A, B, and C. Resolution of the triangular mesh increases along the lake shoreline, along the river, and around each well.

the linear evapotranspiration function were 39.5 m^3 and −2.0 m, respectively, and are identical to model results using the EVT Package.

Neither the constant nor the exponential evapotranspiration functions could be exactly represented using the standard EVT Package. The MODFLOW API allows a user to implement these functions without having to modify the MODFLOW program.

3.2. Optimization of groundwater withdrawals

Groundwater models are often used within an optimization context to maximize groundwater withdrawals subject to drawdown constraints, or to optimize mitigation strategies for contaminant plume containment, for example. Optimization strategies require many forward runs with the groundwater model to calculate response coefficients and recalculate them as necessary for non-linear problems. The MODFLOW API allows a single model instance to be solved as many times as necessary without reloading the model from files. Each model solution can correspond to altered model variables, such as pumping rates. This approach is efficient for optimization because the overhead associated with loading a model can be restricted to just the very first model solution.

The MODFLOW API was used to optimize groundwater pumping rates for a hypothetical steady-state groundwater flow model patterned after the example described by Hill et al. (1998). The original model was based on a regular grid. For the application here, however, a 5-layer triangular mesh with 20,400 cells per layer was created to increase spatial resolution around the shoreline of the lake, along the river, and around three groundwater pumping wells (Fig. 8). For this simple example, the lake is represented as a specified head boundary in layer 1, the river is specified as a head-dependent flow boundary in layer 1, and the wells are represented as specified flows in layer 5. Net recharge is uniformly applied to the top of the model.

The goal of the optimization example is to determine the maximum pumping rate for each well subject to a minimum allowable head constraint for each well, as shown in Table 2. Pumping rates cannot be optimized individually for each well, because the effect of pumping at one well has an effect on the simulated head at the other wells. Specialized versions of MODFLOW have been written to solve these types of optimization problems. For example MODFLOW–GWM couples an earlier version of MODFLOW (Harbaugh, 2005) with optimization routines into a single program (Ahlfeld et al., 2005). MODFLOW–GWM has not been kept up to date with new MODFLOW versions, and therefore it cannot be used with new features in MODFLOW, such as support for unstructured grids, for example. Alternatively, the model-agnostic PESTPP-OPT software (White et al., 2018) could be used for this problem, but without the computational advantage of repeatedly solving the flow model without having to initialize the model from input files. The MODFLOW API allows any general optimization routine, such as the simplex method available in the Python SciPy Package, to be used with a groundwater model. As described by Ahlfeld et al. (2005) the simplex method can be used to solve the following linearized form of the optimization equations,

Table 2
Maximum groundwater withdrawal rates and head constraint information for the optimization problem. Well rates are in $m^3\ d^{-1}$; head values are in m.

Well	Max. Q	Head (Q = 0)	Min. Head
A	268,000	2.66	1.00
B	67,000	11.46	10.00
C	268,000	5.34	1.00

$$\begin{aligned} &\text{Minimize } Z = \mathbf{c}^t\mathbf{x} \\ &\text{subject to } \mathbf{A}\mathbf{x} = \mathbf{b} \\ &\text{where } \mathbf{0} \le \mathbf{x} \le \mathbf{u}, \end{aligned} \tag{5}$$

where Z is the value of the objective function, $\mathbf{c}^t$ is a transposed column vector of coefficients associated with the decision variables; $\mathbf{x}$ is a column vector of pumping rates; $\mathbf{A}$ is the response coefficients, calculated here through perturbation of groundwater withdrawal rates and forward runs with the MODFLOW model through the run_model function (Fig. 9); $\mathbf{b}$ is the column vector of constraints equal to the head at each well cell without pumping minus the head at the well cell with pumping; and $\mathbf{u}$ is a vector containing the maximum pumping rate for each well. For non-linear problems in which the response coefficient matrix depends on the rates of groundwater withdrawals, the optimization problem can be solved repeatedly until some acceptable solution tolerance is achieved.

The run_model function shown in Fig. 9 is an important part of the optimization routine implemented to demonstrate the MODFLOW API. In this function the mf6 object is of type ModflowApi and interfaces between Python and the initialized and running MODFLOW model. Well pumping rates stored in the WEL Package bound array are accessed through a pointer (wrapped inside a numpy. ndarray), which is used by run_model to change the pumping rates. The maximum number of MODFLOW iterations is also accessed and used as part of a loop to make repeated calls to mf6. solve() until convergence is achieved. Finally, the solution vector containing simulated groundwater head for the provided well pumping rates (wellq) are accessed as a pointer and returned to the calling program. For those applications tasked with running a model repeatedly, the run_model function is intuitively appealing because it shows how a Python script (or any other program) can send a running MODFLOW instance one or more new parameter values and receive back the resulting heads, and this can all be done through memory without any file access.

```
def run_model(wellq):
    address = ["BOUND", "GWF", "WEL-1"]
    wbaddr = mf6.get_var_address(*address)
    bound = mf6.get_value_ptr(wbaddr)
    bound[:, 0] = -wellq
    address = ["MXITER", "SLN_1"]
    mxitaddr = mf6.get_var_address(*address)
    mxit = mf6.get_value_ptr(mxitaddr)
    kiter = 0
    mf6.prepare_solve()
    while kiter < mxit:
        has_converged = mf6.solve()
        kiter += 1
        if has_converged:
            break
    if not has_converged:
        return None
    mf6.finalize_solve()
    address = ["X", "GWF"]
    haddr = mf6.get_var_address(*address)
    head = mf6.get_value_ptr(haddr)
    head = head.reshape((5, 20400))
    return head
```

Fig. 9. Python function used to run the MODFLOW model one time using the provided well pumping rates. Although this code snippet does not run on its own, it demonstrates the concept of getting and setting pointers to MODFLOW variables, such as the "bound" array of the WEL Package, the "MXITER" variable of the solver, and the "X" variable (head) of the Groundwater Flow (GWF) Model. This function solves the GWF Model for the specified well withdrawal rates, and returns the head array for the full model grid. Function has been modified slightly from the working example for illustrative purposes.

The Python code used to solve this optimization problem formulates the terms shown in equation (5) and then uses the revised simplex method in scipy. optimize to solve for the well withdrawal rates using

```
scipy.optimize.linprog(c, A_ub=A, b_ub=b,
    bounds=bounds, method='revised simplex')
```

For this problem, the maximum pumping rates were determined to be 44,975, 33,656, and 90,191 m^3d^{-1} for wells A, B, and C, respectively. Although this example problem used simple head constraints, the scripting approach presented here is highly customizable and could be used with many other types of constraints, such as baseflow to the river, minimum head within a model subregion, and so forth. Likewise, pumping rates are used as the decision variable here; however, there are many other model decision variables that could be optimized in order to meet problem constraints.

3.3. Coupling MODFLOW to MetaSWAP

Accurate simulation of the flow of water in the unsaturated zone is important in many water quality and quantity studies. In most cases, the preferred approach involves numerical solution of the Richards' equation, which is computationally intensive and challenging in terms of robustness (see Farthing and Ogden, 2017, for a recent overview on application of the Richards' equation to unsaturated flow problems). The Unsaturated Zone Flow (UZF) Package, which solves a simplified form of Richards' equation based on kinematic waves, is available in MODFLOW. However, this approximation affects the ability to simulate capillary effects and the accuracy of simulated soil moisture dynamics. These dynamics are particularly important in lowland regions, such as the Netherlands, where the groundwater levels are within 2 m of the soil surface in most of the country. As a result, the current national-scale hydrological model of the Netherlands (De Lange et al., 2014) relies on a dedicated coupling between a previous, customized version of MODFLOW (Vermeulen et al., 2020) and MetaSWAP, a quasi steady-state simulation of the unsaturated zone based on Richards' equation (van Walsum and Groenendijk, 2008). A new implementation of this model is being developed and will now build on the MODFLOW API, avoiding the need to develop a proprietary version of MODFLOW as was done in the past.

This example presents a hypothetical model with characteristics common to hydrologic conditions in a large part of the Netherlands. The software driving this example is the Python package imod_coupler, which uses xmipy to control both MODFLOW and MetaSWAP components using their XMI-enabled shared libraries. In short after every solution of the groundwater heads within the non-linear convergence loop (see section 2.2), MetaSWAP determines the unsaturated zone flux and the associated primary storage coefficients while, at the same time, ensuring mass balance for the shared control volume. Both variables (groundwater recharge and storage coefficients) are then communicated to MODFLOW and this sequence is repeated until the MODFLOW convergence criteria are met. See van Walsum and Veldhuizen (2011) for more detail on the shared control volume approach used to couple MODFLOW and MetaSWAP.

The groundwater model consists of a rectangular grid consisting of 3 layers and 81 (9×9) cells per layer, covering an area of 8100 m^2 with the soil surface located at an elevation of 0.0 m. A general head boundary condition (GHB) is assigned to the outer columns, and the active cells in the top layer are all coupled to a corresponding cell in MetaSWAP. The simulation is run for an exceptionally dry year (2018) using daily stress periods and 1 time step per stress period. Precipitation and reference evapotranspiration (Makkink, 1957), both forcing data for MetaSWAP, are from the Dutch national weather service 'De Bilt' station. The land use type in the model domain is agricultural and the crop type is potatoes, which have a relatively large seasonal transpiration rate and require groundwater irrigation. The MetaSWAP irrigation process is

enabled and extracts water from model layer 3 of the MODFLOW model component using the WEL Package. The soil type for the MetaSWAP model component is specified as peat, which is type number 1 in the MetaSWAP soil database.

Precipitation and actual evapotranspiration (ET_{act}) are shown in Fig. 10A. Actual evapotranspiration is greatest during the growing season with peaks appearing at weekly intervals, resulting from increased soil water evaporation during the sprinkler irrigation, which is modeled to occur on weekly basis. The simulated recharge from the unsaturated zone to the groundwater model (q_{rch}) for the centermost cell in the top layer is shown in Fig. 10B. The inset in Fig. 10B shows how the system transitions from the unsaturated zone being a source of water for the groundwater system to a sink for the groundwater system, as a result of capillary rise, during the dry summer period. Simulated groundwater head in the centermost cell in the top layer is shown in Fig. 10C and shows the response of the water table to groundwater recharge (q_{rch}). Irrigation effects are not noticeable in Fig. 10C because the well extracts water from model layer 3.

Although this example is hypothetical and oversimplified, it does illustrate how the MODFLOW API enables seamless coupling of two existing component models without needing to modify the source code of either component model. In this example, MetaSWAP makes it possible to simulate the unsaturated zone in more detail than possible with the MODFLOW UZF Package and to simulate groundwater irrigation that is a function of soil moisture. The explicit control of the outer iteration loop, a specific feature of XMI, is required for this example and can serve as a blueprint for other applications that require tight coupling with MODFLOW.

3.4. Coupling MODFLOW to PRMS

The MODFLOW API and BMI for PRMS were used to simulate integrated surface-water and groundwater processes in the Sagehen Creek watershed located on the east slope of the northern Sierra Nevada mountains in California. The Sagehen Creek application described here is based on the GSFLOW model of the same area (Markstrom et al., 2008). Some key differences between this Sagehen Creek application and the GSFLOW Sagehen Creek model application include: 1) 128 watershed-based hydrologic response units (HRUs) were used instead of cell based HRUs, 2) the PRMS cascade module was not used to route surface runoff and interflow from upslope HRUs to downslope HRUs, and 3) the PRMS soil component is only solved once per time step instead of being recalculated each MODFLOW outer non-linear iteration as is done in GSFLOW.

As part of a separate effort, the BMI was recently implemented in PRMS with the program being split into separate surface, soil, groundwater, and streamflow domain components. The surface and soil domain components were the only PRMS domain components used in this example.

The PRMS surface domain component was used to simulate processes above the soil surface including: 1) rain and snow; 2) potential evapotranspiration; 3) snow sublimation; 4) canopy interception, storage, evaporation, and throughfall; and 5) surface runoff. The PRMS soil domain component was used to simulate storage, inflow, and outflow within the soil zone reservoir. Soil zone inflows include infiltration and outflows include: 1) evapotranspiration; 2) interflow and preferential flow; and 3) groundwater recharge. The MODFLOW UZF Package was used to simulate vertical unsaturated groundwater flow below the PRMS soil domain component soil-zone and subsurface reservoirs. The MODFLOW SFR Package was used to simulate streamflow in a total of 201 connected reaches. The MODFLOW drain (DRN) Package was used to simulate groundwater seepage to land surface in areas where the groundwater levels exceed land surface.

PRMS HRUs and parameters are from Markstrom et al. (2006); the HRUs simulated in this model application are shown in Fig. 11. MODFLOW parameters are from GSFLOW Sagehen Creek model application. The groundwater domain component has a total of 2 layers, 73 rows, and 81 columns and was discretized using a constant grid cell size of 90 m in the row and column directions. The active model domain is restricted to the lateral extent of the Sagehen Creek watershed, with a total of 3392 active cells per layer and covering a 27,475,200 m^2 area. UZF Package cells were included in active cells in both model layers and in cases where the water level in cells in model layer 1 is below the bottom of the cell, unsaturated zone flow at the bottom of the cell is routed to the underlying UZF cell in layer 2. Lateral groundwater discharge out of the watershed was simulated using the Time-Varying Constant Head (CHD) Package and constant head cells at the downstream end of the watershed; a constant head of 1915 m and 1900 m was specified in 3 cells in model layer 1 and 3 cells in model layer 2, respectively. See Markstrom et al. (2006) and Markstrom et al. (2008) for additional information on PRMS and MODFLOW parameters.

MODFLOW and PRMS were sequentially coupled during a time step by first running the update() functions for the PRMS surface and soil domain components, extracting results from PRMS and mapping PRMS

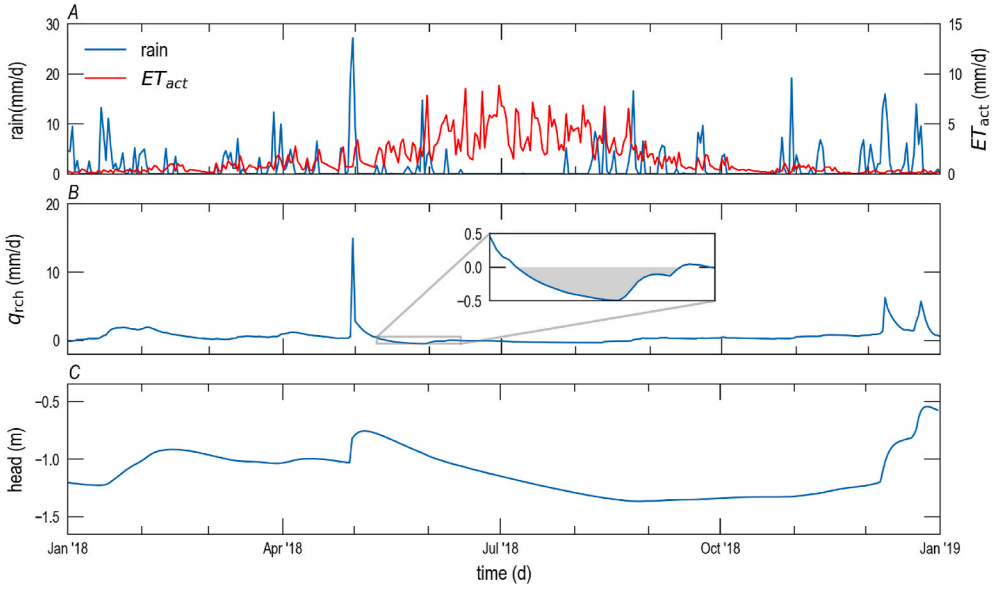


Fig. 10. Time series data from the hypothetical example in which a saturated groundwater system is coupled with an unsaturated zone. The simulation is implemented in the Python package imod_coupler and integrates MODFLOW and MetaSWAP relying on xmipy for control of the individual components and synchronization of coupled variables. ***A***. Input precipitation and the magnitude of the realized evapotranspiration ET_{act} in MetaSWAP, as daily sums. ***B***. The groundwater recharge from the unsaturated zone q_{rch}, converted to a flux rate for an easier comparison to the precipitation data. The data shown is for the centermost cell in the top layer of the grid. The inset shows the effect of capillary rise on the direction of the flux during the summer period. ***C***. The hydraulic head, also a coupled variable, as determined by the groundwater simulation for the centermost cell in the top layer of the model grid.

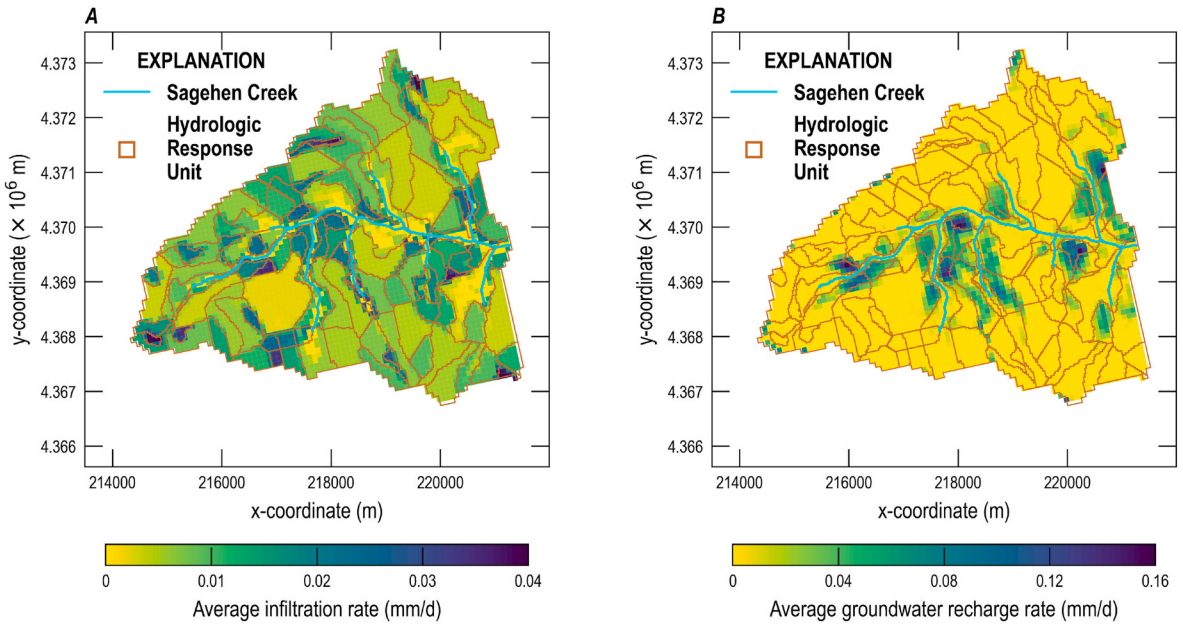


Fig. 11. Simulated average infiltration and groundwater recharge for October 1992 through September 1996. ***A***. Average infiltration rate in mm d^{-1}. ***B***. Average groundwater recharge rate in mm d^{-1}. The location of the 128 HRUs and the SFR stream network are also shown.

HRU results to MODFLOW grid cells, and finally running the MODFLOW update() function. Groundwater recharge calculated by the PRMS soil domain component for the soil zone and subsurface reservoirs for each of the 128 HRUs was applied as infiltration to 3386 underlying UZF Package cells in model layer 1. Unsatisfied potential evapotranspiration, calculated as the difference between the potential evapotranspiration calculated in the PRMS surface domain component and the actual evapotranspiration calculated by the PRMS surface and soil domain components, was applied as potential evapotranspiration to underlying the UZF cells in model layer 1. The HRUs were intersected with the MODFLOW grid to calculate the area-based HRU-UZF weights used to map PRMS results to UZF cells. The HRUs were also intersected with the stream network to map PRMS runoff and interflow from 128 HRUs as runoff to 201 SFR reaches; equal weighting rather than reach-length based weighting was applied to calculate HRU-SFR weights. In addition to mapping PRMS HRU data to MODFLOW UZF cells and SFR reaches, unit conversions were made to convert PRMS inch and acre units to MODFLOW m and m^2 units.

The MODFLOW Mover (MVR) Package was used to route rejected infiltration, calculated by the UZF Package, and groundwater seepage to the surface, calculated by the DRN Package, to the reaches in the SFR Package. The same approach used to develop the HRU-SFR weights was used to connect UZF cells to SFR reaches using the MVR Package. UZF cells in model layer 2 were also routed to the same SFR reach as UZF cells in model layer 1 using the MVR Package to route excess unsaturated zone flow from the overlying UZF cell (rejected infiltration) to the surface-water network.

The model application simulation period extends from October 1, 1980 through September 30, 1996 and included 5844 daily time steps. The Sagehen Creek watershed model application used daily precipitation and air temperature data from the GSFLOW Sagehen Creek model application. Precipitation in the Sagehen Creek watershed is highly variable in form and intensity and generally increases with altitude. Initial groundwater levels for the simulation were based on steady-state heads calculated by a stand-alone MODFLOW model with specified UZF Package infiltration rates from the GSFLOW Sagehen Creek model application and an initial moisture content of 0.08 in each UZF cell.

Although model application parameters are based on previous Sagehen Creek model applications, this model application is considered uncalibrated because of the previously stated differences from the GSFLOW Sagehen Creek model application. Significant storage changes were observed in the unsaturated and saturated zones in MODFLOW results during the first two years of the simulation. As a result, the period from October 1980 to October 1982 is considered a warm-up period and has been excluded from model result analyses.

The average infiltration, which is calculated as the difference between the groundwater recharge calculated by the PRMS soil domain component and rejected infiltration calculated by the UZF Package, is shown in Fig. 11A. Rejected infiltration occurs near the surface-water network and is shown in Fig. 11A as areas where average infiltration rates are different from the groundwater recharge rate calculated by PRMS for a HRU. Average groundwater recharge rates exceed average infiltration rates near topographic divides for several HRUs and in cells with elevations much higher than surrounding cells as a result of drainage of initial soil moisture in the unsaturated zone.

Simulated and observed streamflow is shown in Fig. 12A. The model application over-simulates streamflow, especially during high flow events. The contribution of runoff, interflow, groundwater seepage, and baseflow to streamflow is shown in 12B. Interflow was the largest contributor to streamflow at the end of the evaluation period followed by baseflow, groundwater seepage, and runoff. The contribution of PRMS, unsaturated zone, and saturated zone evapotranspiration to the total evapotranspiration is shown in 12C. Evapotranspiration from the surface and soil PRMS component accounted for more the 99% of the total evapotranspiration.

3.5. *Coupling MODFLOW to MODSIM*

Three recent publications showcased the integration of a river operation model with a physically-based distributed parameter hydrologic model (Brookfield et al., 2017; Dogrul et al., 2016; Morway et al., 2016). In all three integrations, reservoir release and ditch diversion(s) as calculated by the river/reservoir operation model override values within the hydrologic model and specified by a user prior to running the model. The groundwater surface-water exchange rates resulting from those operational decisions (i.e., releases and diversions) are then re-calculated by the hydrologic model for updating the appropriate values within the operations model. In summary, the respective specialty of each model complements a known weakness of the other, resulting in a modeling platform that better equips water resource

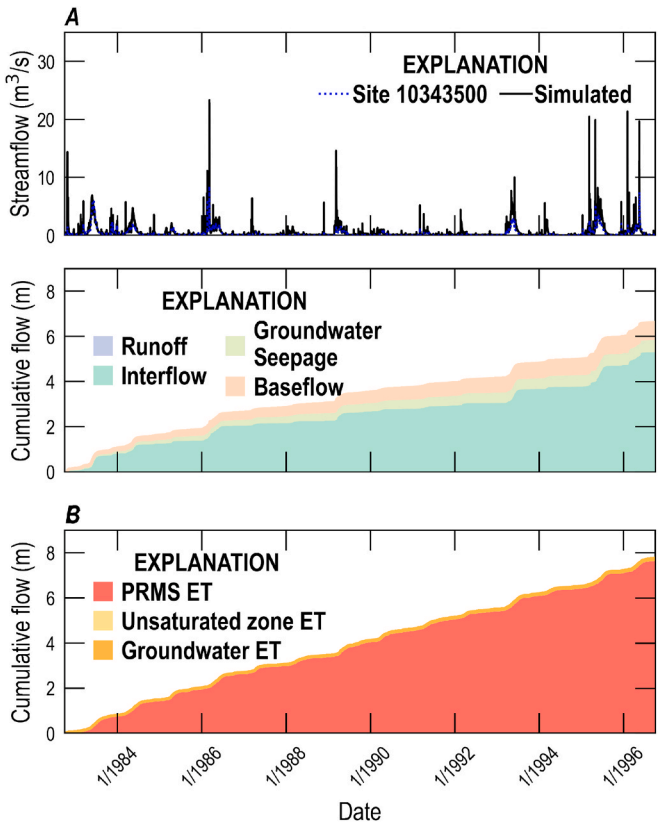


Fig. 12. Simulated and observed Sagehen Creek streamflow, simulated cumulative streamflow components, and simulated cumulative evapotranspiration components. ***A***. Simulated and observed stream flow at Sagehen Creek near Truckee, California (site 10343500). ***B***. Simulated cumulative runoff, interflow, groundwater seepage to the surface, and baseflow from the watershed. ***C***. Simulated cumulative evapotranspiration calculated by PRMS and the UZF Package from the unsaturated and saturated zones. Cumulative flows were calculated by summing volumetric flow terms for each time step and dividing by the discretized watershed area (27,475,200 m^2).

managers to conjunctively manage groundwater and surface-water as a single resource (Winter et al., 1998).

Unlike the previous river operation and hydrologic model integration efforts, no customization of the hydrologic model source code was necessary for this integration. Instead, the selected river operation model MODSIM (Labadie et al., 2000) repeatedly calls MODFLOW as needed. Unlike the other included examples, MODSIM uses the Microsoft. NET platform to run the initialize and finalize functions shown in Fig. 2; the prepare_time_step and finalize_time_step functions shown in Fig. 3; and the prepare_solve, solve, and finalize_solve functions shown in Fig. 4 as necessary. The only new code that was written to establish communication between MODSIM and MODFLOW was through the custom code interface provided with the standard distribution of MODSIM.

Use of the MODFLOW API with a river operations model is demonstrated with a hypothetical model that first appeared in Morway et al. (2016) and is shown in schematic form in Fig. 13. The model is patterned after an irrigated river valley governed by the prior appropriation doctrine (i.e., "first in time, first in right"). For this example, the net streamflow accretions and depletions as calculated by MODFLOW along every simulated stream reach are added to or subtracted from the corresponding MODSIM link. With this information, MODSIM recalculates the reservoir release and diversion rates based on the updated groundwater and surface-water exchanges. Next, the MODFLOW time step is rerun in order to update the accretions and depletions associated with

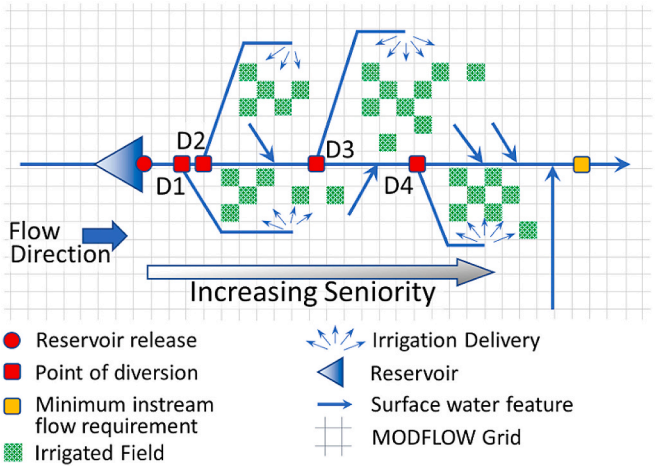


Fig. 13. Schematic of the hypothetical model used to test the integration of MODSIM with MODFLOW. MODSIM-calculated values include the reservoir release (red circle) and amount of water to divert (red squares) into four ditches used to deliver irrigation water. The BMI interface is used to retrieve the groundwater surface-water exchange volumes calculated by MODFLOW for overwriting user-specified groundwater gains and losses inside MODSIM.

the latest MODSIM operational decisions. The two models continue iterating within a time step until all the changes in shared values (reservoir releases, diversions, and surface-water groundwater interactions) satisfy the convergence criteria. In this way, the respective hydrologic and river operation solutions synchronize in time and space before moving on to the next time step, an important advancement in simulating conjunctive use systems.

The impact of integrating MODSIM with MODFLOW, as opposed to running MODFLOW by itself, is shown in Fig. 14. For an abbreviated simulation period of one year, reservoir storage is drawn down and delivered to specific ditches that "own" the stored water (Fig. 14A). For this example, the three junior ditches, D1, D2, and D3 (Fig. 13) are assigned storage accounts that provide water during the summer months when natural flows (as opposed to stored water) generally taper off. The first storage account to run out of water is D1, and without access to other sources of water, the diversion amounts are reduced to zero by MODSIM without any further input adjustments by the user (Fig. 14B). In the MODFLOW-only run, however, total monthly diversions remain inappropriately high in D1 throughout the year by virtue of its upstream location within the system. On its own, MODFLOW cannot simulate prior-appropriations within the surface-water irrigation system, and is therefore limited to a simplified set of rules available with the SFR Package (Langevin et al., 2017) for determining diversion rates that may or may not reflect management practices. D2 shows relatively good agreement between MODSIM–MODFLOW and MODFLOW-only total monthly diversion amounts owing to the fact that its reservoir storage persists throughout the relatively short one year simulation period (Fig. 14C). However, differences between the two simulations similar to Fig. 14B would occur were D2 to run out of storage water. Fig. 14D and E further highlight the difficulty in simulating river operations without a tool like MODSIM informing river diversions. Recognizing that the next downstream diversion, D4 (Fig. 13), has a senior water right, an appropriate rule was selected from the four rules available within the SFR Package that attempts to allow enough water by to satisfy the senior downstream water right. However, in so doing, no water is diverted by D3 even though it should have access to stored water that was released from the reservoir. Integration with MODSIM through the BMI correctly diverts the water released from the D3 storage account into the D3 ditch. Finally, because natural flows are the only source of water available to D4, MODSIM diverts a full allotment for D4 during the spring snowmelt season (commonly April–June; Fig. 14E) but reduces diversions in the

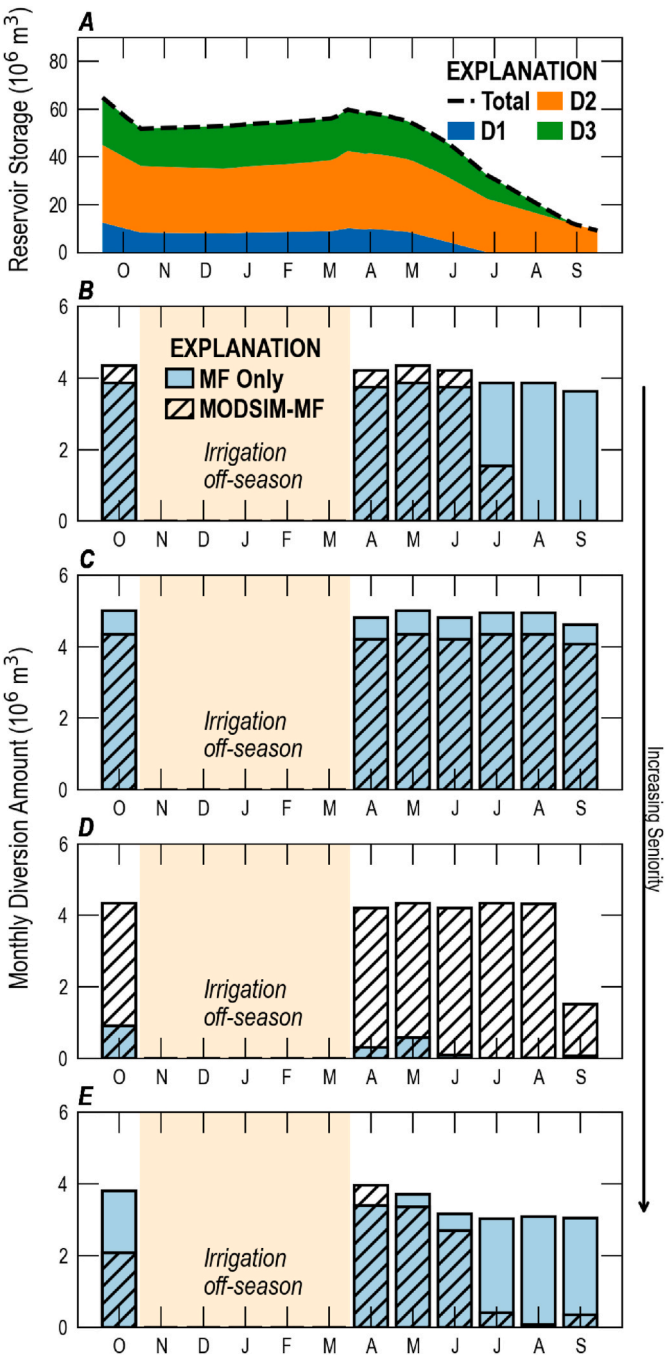


Fig. 14. (A) Change in total reservoir storage through time is the same in both models; however, volumes of water stored by individual storage accounts are maintained by MODSIM. (B)–(E) Total monthly diversion amounts for the ditches D1, D2, D3, and D4 shown in Fig. 13, respectively. Results highlight the differences in diversion amounts for a MODFLOW-only run versus a MODSIM–MODFLOW run whereby MODSIM honors relative priorities among the four ditches.

other months. Although this is hypothetical example, it highlights the value of the MODFLOW API; without it, it would be impossible to properly simulate the surface-water operations, and in connected stream-aquifer systems, if the surface hydrology is wrong, then accurate simulation of the groundwater system remains unattained.

4. Software and data availability

The MODFLOW API, XMI, xmipy, and the modflowapi (see Fig. 1) were collaboratively developed by the U.S. Geological Survey and Deltares. MODFLOW is available from the U.S. Geologic Survey at https://www.usgs.gov/software/modflow-6-usgs-modular-hydrologic-model and as an open-source code repository from https://github.com/MODFLOW-USGS/modflow6. The xmipy and modflowapi Python packages are available as open-source code repositories from https://github.com/Deltares/xmipy and https://github.com/MODFLOW-USGS/modflowapi, respectively, and both can be installed from PyPI. The PRMS soil and surface components, with the Basic Model Interface, are available from the csdms-stack conda channel or from PyPI.

Jupyter notebooks and model application datasets that are not built within the notebook using FloPy (Bakker et al., 2016) are available as a release asset from https://github.com/MODFLOW-USGS/modflowapi/releases/tag/esmdatasets. Each Jupyter notebook includes software installation instructions and use limitations. Several additional Jupyter notebooks that demonstrate use of the modflowapi Python package are available in the modflowapi repository.

5. Discussion and conclusions

The MODFLOW API was developed to allow external programs control of a MODFLOW simulation and access to internal variables while the simulation is running. These external programs may consist of another physical process model to be coupled with MODFLOW, a machine-learning model representing a physical process, or a Python script that needs control and access of MODFLOW while it is running. The MODFLOW API is based on the established BMI convention for software interoperability. There are many benefits to be gained from the MODFLOW API as demonstrated in the five examples and described in the following list.

- The MODFLOW API provides a sustainable way to design and maintain software that relies on MODFLOW as a component. Software designed in this manner can take advantage of new MODFLOW features without having to modify and update the source code.
- The MODFLOW API allows other process models to be coupled with MODFLOW. This coupling can be performed sequentially for each time step or the models can be tightly coupled at the matrix solution level using the XMI extension to the BMI. This tight coupling approach may allow for larger time steps and more accurate mass-conserved solutions for complex problems.
- Users have new flexibility to make custom MODFLOW packages and prototype new capabilities without having to compile MODFLOW or make any changes to the underlying source code. These new user packages can be developed with popular languages, such as Python, which are familiar and easy to use.
- There are many different MODFLOW variants, including those that represent parameter estimation, adjoint state calculations, farm processes, and chemical reactions, for example. Including all possible relevant processes into MODFLOW is not sustainable or practical. The API allows the MODFLOW program to remain focused on its intended scope as a hydrologic simulator, while enabling extension to other domains as necessary.
- MODFLOW reads and writes text files and binary files that are well documented, but not familiar to scientists in other domains. Through the data access routines afforded by the MODFLOW API, users can inject data read from alternative files, such as HDF and netCDF files. Similarly, model output could also be written to alternative file formats at runtime.
- Software capabilities are rapidly evolving and being made available to scientists and engineers in the form of new packages and modules. For example, sophisticated machine learning tools are being developed and released faster than they could be implemented in MODFLOW. The MODFLOW API allows users to innovate with new software tools that may not be accessible otherwise.

- MODFLOW is supported by several commercial graphical user interfaces (GUIs) that allow users to develop, run, and post-process groundwater models. The MODFLOW API provides a new way for these GUIs to run a simulation, monitor its progress, even at the iteration level, and provide instantaneous feedback to the user.

This paper describes the initial release of the MODFLOW API, and there are several limitations to mention. Although the API allows the user to interact with time steps individually they must be executed sequentially from the first to the last time step (i.e., a time step that has been finalized using update or finalize_time_step can not be rerun). Development of restart capability for MODFLOW is a high priority but being able to rerun a time step and correctly synchronize file based input and output during run time is challenging. Another limitation is, the current version of the MODFLOW API requires that the user has intimate knowledge of the underlying MODFLOW program, variables, and order of operations. For coupling another model with MODFLOW, the user must be familiar with both programs, have a firm understanding of variable names, which variables to exchange, when to exchange information, the scientific units used in the models, and so forth. Work is ongoing to better document the internal functioning of MODFLOW in order to make it easier to use the MODFLOW API. For example, it may be possible to use standard names for common variables, as suggested by Peckham et al. (2013) and Peckham (2014), and allow users to access data using these standard names. There is also ongoing work to allow model simulations to be created in memory, rather than being created from input files, as is required in the current version. These limitations will continue to be addressed by the MODFLOW development team as the MODFLOW API is used in practice and the latest developments will be available at the urls listed in Section 4.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The authors would like to acknowledge Mark Piper and Eric Hutton from CSDMS for their help in implementing the BMI standard in PRMS and for discussions during the development of the MODFLOW API. The authors would also like to acknowledge Michael Fienen from the U.S. Geological Survey for early discussions about the development of the MODFLOW API and for his review of the manuscript, Laura Schachter from the U.S. Geological Survey for her review of the manuscript, Joeri van Engelen from Deltares for his review of the manuscript, and four anonymous reviewers at the journal.

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

References

Ahlfeld, D.P., Barlow, P.M., Mulligan, A.E., 2005. GWM—a ground-water management process for the U.S. Geological Survey modular ground-water model (MODFLOW-2000). U.S. Geological Survey Open-File Report 2005–1072 130. https://doi.org/10.3133/ofr20051072.

Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., Smolinski, B., 1999. Toward a common component architecture for high-performance scientific computing. In: Proceedings. The Eighth International Symposium on High Performance Distributed Computing (Cat. No. 99TH8469), IEEE, pp. 115–124. https://doi.org/10.1109/HPDC.1999.805289.

Bakker, M., Post, V., Langevin, C.D., Hughes, J.D., White, J., Starn, J., Fienen, M.N., 2016. Scripting modflow model development using python and flopy. Groundwater 54, 733–739. https://doi.org/10.1111/gwat.12413.

Belete, G.F., Voinov, A., Laniak, G.F., 2017. An overview of the model integration process: from pre-integration assessment to testing. Environ. Model. Software 87, 49–63. https://doi.org/10.1016/j.envsoft.2016.10.013. URL: https://www.sciencedirect.com/science/article/pii/S1364815216308805.

Brookfield, A., Gnau, C., Wilson, B., 2017. Incorporating surface water operations in an integrated hydrologic model: model development and application to the lower republican river basin, United States. J. Hydrol. Eng. 22 https://doi.org/10.1061/(ASCE)HE.1943-5584.0001486, 04016065 – 15.

Chen, M., Voinov, A., Ames, D.P., Kettner, A.J., Goodall, J.L., Jakeman, A.J., Barton, M. C., Harpham, Q., Cuddy, S.M., DeLuca, C., Yue, S., Wang, J., Zhang, F., Wen, Y., Lu, G., 2020. Position paper: open web-distributed integrated geographic modelling and simulation to enable broader participation and applications. Earth Sci. Rev. 207, 103223. https://doi.org/10.1016/j.earscirev.2020.103223. URL: https://www.sciencedirect.com/science/article/pii/S0012825220302695.

Collins, N., Theurich, G., DeLuca, C., Suarez, M., Trayanov, A., Balaji, V., Li, P., Yang, W., Hill, C., Da Silva, A., 2005. Design and implementation of components in the Earth system modeling framework. Int. J. High Perform. Comput. Appl. 19, 341–350. https://doi.org/10.1177/1094342005056120.

Davis, P.R., 2001. ISGW—the integrated hydrologic model coupling HSPF and MODFLOW. In: Panigrahi, B.K., Singh, U.P. (Eds.), Integrated Surface and Ground Water Management. American Society of Civil Engineers, pp. 100–109. https://doi.org/10.1061/40562(267)11.

De Lange, W.J., Prinsen, G.F., Hoogewoud, J.C., Veldhuizen, A.A., Verkaik, J., Oude Essink, G.H., van Walsum, P.E., Delsman, J.R., Hunink, J.C., Massop, H.T., Kroon, T., 2014. An operational, multi-scale, multi-model system for consensus-based, integrated water management and policy analysis: The Netherlands Hydrological Instrument. Environ. Model. Software 59, 98–108. https://doi.org/10.1016/j.envsoft.2014.05.009.

Dogrul, E.C., Kadir, T.N., Brush, C.F., Chung, F.I., 2016. Linking groundwater simulation and reservoir system analysis models: the case for California's Central Valley. Environ. Model. Software 77, 168–182. https://doi.org/10.1016/j.envsoft.2015.12.006.

Farthing, M.W., Ogden, F.L., 2017. Numerical solution of Richards' equation: a review of advances and challenges. Soil Sci. Soc. Am. J. 81, 1257–1269. https://doi.org/10.2136/sssaj2017.02.0058.

Ferziger, J.H., Peric, M., 1996. Computational Methods for Fluid Dynamics. https://doi.org/10.1007/978-3-642-56026-2.

Gregersen, J., Gijsbers, P., Westen, S., 2007. OpenMI: open modelling interface. J. Hydroinf. 9, 175–191. https://doi.org/10.2166/hydro.2007.023.

Harbaugh, A.W., 2005. MODFLOW-2005, the U.S. Geological Survey Modular Ground-Water Model—The Ground-Water Flow Process. U.S. Geological Survey Techniques and Methods, Book 6, A16. https://doi.org/10.3133/tm6A16.

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M.H., Brett, M., Haldane, A., Fernandez del Rio, J., Wiebe, M., Peterson, P., Gerard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T.E., 2020. Array programming with NumPy. Nature 585, 357–362. https://doi.org/10.1038/s41586-020-2649-2.

Hill, M.C., Cooley, R.L., Pollock, D.W., 1998. A controlled experiment in ground water flow model calibration. Groundwater 36, 520–535. https://doi.org/10.1111/j.1745-6584.1998.tb02824.x.

Hoch, J.M., Eilander, D., Ikeuchi, H., Baart, F., Winsemius, H.C., 2019. Evaluating the impact of model complexity on flood wave propagation and inundation extent with a hydrologic–hydrodynamic model coupling framework. Nat. Hazards Earth Syst. Sci. 19, 1723–1735. https://doi.org/10.5194/nhess-19-1723-2019.

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

Hughes, J.D., Russcher, M.J., Langevin, C.D., Hofer, J., 2021. Modflowapi–An Extension to Xmipy for the MODFLOW API Version 0.0.1 [Computer Software]. U.S. Geological Survey Software Release. https://doi.org/10.5281/zenodo.4753410.

Hutton, E.W., Piper, M.D., 2020. The python Modeling Toolkit (Version v1.0.0) [Computer Software]. https://doi.org/10.5281/zenodo.3644240.

Hutton, E.W., Piper, M.D., Tucker, G.E., 2020. The basic model interface 2.0: a standard interface for coupling numerical models in the geosciences. Journal of Open Source Software 5, 2317. https://doi.org/10.21105/joss.02317.

Kernighan, B.W., Ritchie, D.M., 1988. The C Programming Language, second ed.

Kim, N.W., Chung, I.M., Won, Y.S., Arnold, J.G., 2008. Development and application of the integrated SWAT–MODFLOW model. J. Hydrol. 356, 1–16. https://doi.org/10.1016/j.jhydrol.2008.02.024.

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

Labadie, J.W., Baldo, M.L., Larson, R., 2000. MODSIM: Decision Support System for River Basin Management: Documentation and User Manual. Technical Report. Colorado State University, Department of Civil Engineering.

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

Langevin, C.D., Panday, S., Provost, A.M., 2020. Hydraulic-head formulation for density-dependent flow and transport. Groundwater 58, 349–362.

Langevin, C.D., Thorne, D.T., Dausman, A.M., Sukop, M.C., Guo, W., 2008. SEAWAT Version 4: A Computer Program for Simulation of Multi-Species Solute and Heat Transport. U.S. Geological Survey Techniques and Methods, Book 6, A22, p. 39. https://doi.org/10.3133/tm6A22.

Makkink, G.F., 1957. Testing the penman formula by means of lysimeters. J. Inst. Water Eng. 11, 277–288.

Markstrom, S.L., Niswonger, R.G., Regan, R.S., Prudic, D.E., Barlow, P.M., 2008. GSFLOW-coupled Ground-Water and Surface-Water FLOW Model Based on the Integration of the Precipitation-Runoff Modeling System (PRMS) and the Modular Ground-Water Flow Model (MODFLOW-2005). Volume 6-D1 of U.S. Geological Survey Techniques and Methods, Book 6, D1, p. 240. https://doi.org/10.3133/tm6D1.

Markstrom, S.L., Regan, R.S., Hay, L.E., Viger, R.J., Webb, R.M., Payn, R.A., LaFontaine, J.H., 2015. PRMS-IV, the Precipitation-Runoff Modeling System, Version 4. U.S. Geological Survey Techniques and Methods, Book 6, B7, p. 158. https://doi.org/10.3133/tm6B7.

Markstrom, S.L., Regan, R.S., Niswonger, R.G., Prudic, D.E., Viger, R.J., 2006. GSFLOW–A basin scale model for coupled simulation of ground-water and surface-water–Part A. Concepts for modeling surface-water flow with the U.S. Geological Survey Precipitation-Runoff Modeling System. In: Proceedings of the Third Federal Interagency Hydrologic Modeling Conference, Reno, Nevada, USA. URL: https://acwi.gov/hydrology/mtsconfwkshops/conf_proceedings/3rdFIHMC/7F_Markstrom.pdf.

Metcalf, M., Reid, J., Cohen, M., 2018. Modern Fortran Explained. https://doi.org/10.1093/oso/9780198811893.001.0001.

Morway, E.D., Langevin, C.D., Hughes, J.D., 2021. Use of the MODFLOW 6 water mover package to represent natural and managed hydrologic connections. Groundwater. https://doi.org/10.1111/gwat.13117.

Morway, E.D., Niswonger, R.G., Triana, E., 2016. Toward improved simulation of river operations through integration with a hydrologic model. Environ. Model. Software 82, 255–274. https://doi.org/10.1016/j.envsoft.2016.04.018.

Panday, S., Langevin, C.D., Niswonger, R.G., Ibaraki, M., Hughes, J.D., 2013. MODFLOW-USG Version 1—An Unstructured Grid Version of MODFLOW for Simulating Groundwater Flow and Tightly Coupled Processes Using a Control Volume Finite-Difference Formulation. U.S. Geological Survey Techniques and Methods, Book 6, A45, p. 66. URL: https://pubs.usgs.gov/tm/06/a45/.

Peckham, S.D., 2014. The CSDMS standard names: cross-domain naming conventions for describing process models, data sets and their associated variables. In: Ames, D.P., T, Q.N.W., Rizzoli, A.E. (Eds.), 7th International Congress on Environmental Modelling and Software, International Environmental Modelling and Software Society (iEMSs), San Diego, California, USA. URL: https://csdms.colorado.edu/w/images/Peckham_2014_iEMSs.pdf.

Peckham, S.D., Hutton, E.W., Norris, B., 2013. A component-based approach to integrated modeling in the geosciences: the design of CSDMS. Comput. Geosci. 53, 3–12. https://doi.org/10.1016/j.cageo.2012.04.002.

Pérez, F., Granger, B.E., 2007. IPython: a system for interactive scientific computing. Comput. Sci. Eng. 9, 21–29. https://doi.org/10.1109/MCSE.2007.53.

Piper, M.D., 2020. The Fortran Specification, Created with Fortran 2003, for the CSDMS Basic Model Interface (BMI) (Version v2.0) [Computer Software]. https://doi.org/10.5281/zenodo.3637641.

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

Ratliff, K.M., Hutton, E.H., Murray, A.B., 2018. Exploring wave and sea-level rise effects on delta morphodynamics with a coupled river-ocean model. J. Geophys. Res.: Earth Surface 123, 2887–2900. https://doi.org/10.1029/2018JF004757.

Rodriguez, L.B., Cello, P.A., Vionnet, C.A., Goodrich, D., 2008. Fully conservative coupling of HEC-RAS with MODFLOW to simulate stream–aquifer interactions in a drainage basin. J. Hydrol. 353, 129–142. https://doi.org/10.1016/j.jhydrol.2008.02.002.

Russcher, M.J., Hofer, J., Hughes, J.D., 2020. xmipy–Python Bindings for the eXtended Model Interface Version 1.0.0 [Computer Software]. Deltares software release. https://doi.org/10.5281/zenodo.4740983.

Shoemaker, W.B., Kuniansky, E.L., Birk, S., Bauer, S., Swain, E.D., 2008. Documentation of a Conduit Flow Process (CFP) for MODFLOW-2005. U.S. Geological Survey Techniques and Methods, Book 6, A24, p. 50. https://doi.org/10.3133/tm6A55.

Swain, E.D., Wexler, E.J., 1996. A Coupled Surface-Water and Ground-Water Flow Model (MODBRANCH) for Simulation of Stream-Aquifer Interaction. U.S. Geological Survey Techniques of Water-Resources Investigations 06-A6, p. 125. https://doi.org/10.3133/twri06a6.

Twarakavi, N.K.C., Šimůnek, J., Seo, S., 2008. Evaluating interactions between groundwater and vadose zone using the HYDRUS-based flow package for MODFLOW. Vadose Zone J. 7, 757–768. https://doi.org/10.2136/vzj2007.0082.

Vermeulen, P., Roelofsen, F., Hunink, J., Janssen, G., Romero Verastegui, B., Van Engelen, J., Russcher, M., 2020. iMOD user manual version 5.2. Deltares.

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

van Walsum, P., Veldhuizen, A., 2011. Integration of models using shared state variables: implementation in the regional hydrologic modelling system SIMGRO. J. Hydrol. 409, 363–370. https://doi.org/10.1016/j.jhydrol.2011.08.036.

van Walsum, P.E.V., Groenendijk, P., 2008. Quasi steady-state simulation of the unsaturated zone in groundwater modeling of lowland regions. Vadose Zone J. 7, 769–781. https://doi.org/10.2136/vzj2007.0146.

Wang, J.D., Swain, E.D., Wolfert, M.A., Langevin, C.D., James, D.E., Telis, P.A., 2007. Application of FTLOADDS to Simulate Flow, Salinity, and Surface-Water Stage in the Southern Everglades, Florida. U.S. Geological Survey Scientific Investigations Report 2007-5010, p. 112. https://doi.org/10.3133/sir20075010.

White, J.T., Fienen, M.N., Barlow, P.M., Welter, D.E., 2018. A tool for efficient, model-independent management optimization under uncertainty. Environ. Model. Software 100, 213–221. https://doi.org/10.1016/j.envsoft.2017.11.019. URL: https://www.sciencedirect.com/science/article/pii/S1364815217306965.

Winston, R.B., Konikow, L.F., Hornberger, G.Z., 2018. Volume-weighted particle-tracking method for solute-transport modeling. In: Implementation in MODFLOW-GWT. U.S. Geological Survey Techniques and Methods, Book 6, A58, p. 44. https://doi.org/10.3133/tm6a58.

Winter, T.C., Harvey, J.W., Franke, O.L., Alley, W.M., 1998. Ground water and surface water: a single resource. US Geol. Surv. Circular 1139, 79. https://doi.org/10.3133/cir1139.