Supplementary Information for: Reproducible, Component-based Modeling with TopoFlow, A Spatial Hydrologic Modeling Toolkit

S. D. Peckham [1*], **M. Stoica** [1†], **E. Jafarov** [2‡], **A. Endalamaw** [3§], **W. R. Bolton** [3¶]

[1]Institute of Arctic and Alpine Research, University of Colorado, Boulder, CO 80309
[2]Los Alamos National Lab, Los Alamos, NM, 87545
[3]International Arctic Research Center, University of Alaska, Fairbanks, AK 99775

Appendix A: Links to HTML Help Pages for TopoFlow Components

Each TopoFlow component has its own HTML help page on the CSDMS wiki that includes a listing of all equations and variables used by the component, along with their units. These pages also provide a Notes section with additional information for users, and a list of useful References. These help pages may also be accessed from within the CSDMS WMT application. Each blue section heading here is a link to the corresponding HTML help page. Source code for each component can be found in the *topoflow/components* folder of the TopoFlow Python package, with the indicated filenames. All **_base.py* components inherit from *topoflow/utils/BMI_base.py*.

Channel and Overland Flow Routing Components

These components model the lateral flow of surface water between grid cells in the DEM using D8 flow directions. Every grid cell in the DEM contains its own 1D channel segment — with a trapezoidal cross-section, roughness parameter and sinuosity — as part of the river network. All methods conserve mass.

Kinematic Wave Method. This is the simplest method for modeling flow in open channels. It assumes that all terms in the general momentum equation (pressure gradient, local acceleration and convective acceleration) are very small compared to the friction and gravity terms. Flow variables in channel segments vary in time (i.e. unsteady) but only as a result of mass conservation. (*channels_kinematic_wave.py* inherits from *channels_base.py*).

Diffusive Wave Method. Same as kinematic, except the pressure gradient term in the momentum equation is also retained. Water surface slope (due to water depth gradient) can differ from the bed slope. As a notable consequence, water is able to move across flat areas that have a bed slope of zero. (*channels_diffusive_wave.py* inherits from *channels_base.py*).

Dynamic Wave Method. The most comprehensive method for modeling channel flow, where all terms in the integral form of the general momentum equation are retained. This includes terms that allow momentum to accumulate within a grid cell (local acceleration) and for momentum to be transferred between adjacent grid cells (convective acceleration). (*channels_dynamic_wave.py* inherits from *channels_base.py*).

Evaporation Process Components

*orcid.org/0000-0002-1373-2396
†orcid.org/0000-0002-6612-3439
‡orcid.org/0000-0002-8310-3261
§orcid.org/0000-0001-9585-8517
¶orcid.org/0000-0003-0842-9867

Corresponding author: Scott D. Peckham, Scott.Peckham@colorado.edu

Priestley-Taylor Method. This component uses a semi-empirical equation to compute the potential evaporation rate throughout the watershed based on energy considerations [*Priestley and Taylor*, 1972]. It uses shortwave and longwave radiation as well as conduction energy flux from the soil. However, it uses a coefficient to parametrize sensible heat flux and can therefore be used when weather inputs such as relative humidity and wind speed are not available. (*evap_priestley_taylor.py* inherits from *evap_base.py*).

Energy Balance Method. This component uses a more complete energy-balance formulation to compute the potential evaporation rate throughout the watershed. It uses shortwave and longwave radiation, conduction energy flux from the soil and sensible heat flux. (*evap_energy_balance.py* inherits from *evap_base.py*).

Infiltration Process Components

These components compute the rate at which surface water infiltrates into soil due to capillary suction and gravity, in addition to other variables.

Green-Ampt Method. This method can be viewed as an approximate solution to Richards equation (see below) but also conserves mass. The soil moisture profile with depth is treated as piston flow, with saturated soil above the wetting front and the initial water content below [*Green and Ampt*, 1911]. (*infil_green_ampt.py* inherits from *infil_base.py*).

Smith-Parlange (3-parameter) Method. This method is similar to the Green-Ampt method but includes an additional parameter, γ, between 0 and 1. It reduces to the Green-Ampt method as γ approaches 0 [*Smith and Parlange*, 1978; *Parlange et al.*, 1982].
(*infil_smith_parlange.py* inherits from *infil_base.py*).

Richards Equation (1D) Method. This method solves the 1D Richards equation [*Richards*, 1931] for flow through unsaturated porous media. The nonlinear 1D partial differential equation is solved separately but simultaneously for every grid cell in the DEM to determine how soil moisture variables vary vertically with depth below the surface. Lateral flow between cells is assumed negligible in the unsaturated zone. Three soil layers with different properties are supported. The numerical method used may not converge for some parameter settings. (*infil_richards_1D.py* inherits from *infil_base.py*).

Snowmelt Process Components

These components compute variables that describe the accumulation and possible melting of snow throughout a watershed. This includes the rate of snowmelt, which contributes to runoff.

Degree-Day Method. This method assumes that the rate of snowmelt is directly proportional to the difference between the mean air temperature at the surface and a threshold temperature (e.g. melting point of snow). The coefficient of proportionality is dependent on season, location, snow density, and wind speed.
(*snow_degree_day.py* inherits from *snow_base.py*).

Energy Balance Method. This method is much more sophisticated than the Degree-Day method, but also requires many more input variables. It accounts for shortwave and longwave radiation, sensible and latent heat flux and the cold content of the snowpack. It incorporates meteorologic variables such as air temperature, wind speed, relative humidity and vapor pressure as well as surface properties. (*snow_energy_balance.py* inherits from *snow_base.py*).

Meteorology Components

Meteorology. This component computes variables such as shortwave and longwave radiation, vapor pressure and emissivity throughout a watershed using the equations given by

Dingman [2002] (Appendix E) and a formula for optical mass. It includes shortwave and longwave radiation calculators based on celestial mechanics, latitude, longitude and topography (slope and aspect). It reads inputs such as precipitation rate, air temperature, wind speed and relative humidity from files. (*met_base.py*).

Saturated Zone Components

Darcy Layers Method. This component provides a simple treatment of saturated, subsurface flow through shallow soil that is underlain by an impermeable boundary (e.g. permafrost or bedrock). Darcy's Law is used and the gradient of the water table is assumed to be the same as the land surface gradient. Multiple soil layers with different properties are supported. (*satzone_darcy_layers.py* inherits from *satzone_base.py*).

Additional Components

Data–HIS Component. This component lets you create a query by entering a search keyword, bounding box, start date, stop date, etc. Once data has been downloaded from the web service, it can either be saved to files or accessed directly by other components through the "Data" port, which has a standard CSDMS BMI interface. See *Peckham and Goodall* [2013] for more information. (**Note:** Written for CMT. Not yet updated for use in WMT.) (*HIS_base.py*).

Flow Diversions Component. TopoFlow supports three different types of flow diversions: sources, sinks and canals. **Sources** are locations such as natural springs where water enters the watershed at a point by some process other than those that are otherwise modeled. Similarly, **sinks** are locations where water leaves the watershed at a point. **Canals** are generally man-made reaches such as tunnels or irrigation ditches that transport water from one point to another, typically without following the natural gradient of the terrain that is indicated by the DEM. The upstream end is essentially a sink and the downstream end a source. (*diversions_fractions_method.py* inherits from *diversions_base.py*).

Erode (Fluvial Landscape Evolution Model). While included in the TopoFlow Python package, with a BMI interface and runnable with EMELI, this component is not really part of TopoFlow. However, it could easily be modified to provide a sediment or contaminant transport model component for use with TopoFlow. It is a robust, fluvial landscape evolution model (LEM). Unlike most or all other LEMs, Erode does not fill pits in the initial DEM artificially at the start. Instead, local depressions are filled naturally by the sediment transport process itself over time. Movies of the D8 area grid evolving over time show what looks like avulsions during this filling process In addition, Erode includes a robust numerical stability condition and uses adaptive time stepping for optimum performance. (*erode_d8_global.py* inherits from *erode_base.py*).

GC2D (Valley Glacier Evolution and Icemelt). GC2D is a two-dimensional finite difference numerical model that is driven by a calculations of glacier mass balance (snow precipitation - melt rate). The model calculates ice surface elevations above a two-dimensional terrain by solving equations for ice flux and mass conservation using explicit methods. The original version was written in MatLab by Mark Kessler, but TopoFlow includes a version converted to Python and provided with a BMI interface. While GC2D can simulate valley glacier evolution over time (with large time steps), when coupled to other TopoFlow components it mainly provides meltwater to a river system.
(*ice_base.py* calls *gc2d.py*).

DEM Profile Smoother Tool. This is a pre-processing tool that can be applied to a DEM to create a new DEM with smoother and more realistic channel slopes. Well-defined and smoothly-varying slopes along channel streamlines is important when using the kinematic wave method of flow routing. (*smooth_dem.py*).

Appendix B: Reading and Writing Binary Grid Data Files

A *binary grid* is one of the most common, simple and efficient ways to store a grid of data values in a file. Here *grid* refers to a 2D array of numbers associated with some discretization of space. A binary grid stores the values in a 2D array by starting on the left side of the top row and writing them row by row, similar to numbers in a calendar. This is called *row-major order*, and the first/top row usually corresponds to the most northern edge of a geographic bounding box for a region of interest. Each number in the 2D array is encoded using a fixed number of bytes according to the *IEEE* [2008] standard, using either a little-endian or big-endian byte ordering. (Macs and PCs with Intel processors use little-endian, but others may use big-endian.) Floating-point numbers are typically stored using either 4 or 8 bytes per data value (for single or double precision), and integers are typically stored using 1, 2 or 4 bytes per data value (byte, int, long). (More bytes are needed for a greater range of possible values.) Floating-point numbers are stored differently than integers, and the IEEE standard also has provisions for storing some special values such as NaN (Not a Number) and Infinity. This method of storing a 2D array of numbers in a file is dramatically more efficient — in terms of both the required disk space and the time to read values from a file into RAM — than text files that store each number as a collection of ASCII characters.

A binary grid typically contains only a 2D array of numbers, although it is possible to reserve some number of bytes at the beginning of the binary file — called a header — for storing additional, descriptive information such as the number of rows and columns in the 2D array. The filesize of a binary grid with no header is therefore simply: filesize = (ncols x nrows x BPE), where ncols and nrows are the number of columns and rows in the 2D array and BPE is the number of Bytes Per Element used to store a single numerical value. Note that several items of metadata are necessary in order to interpret the numbers in the binary file, such as the number of rows and columns, the x (east-west) and y (north-south) dimensions of each grid cell, the geographic coordinates of the lower-left corner of the grid and measurement units associated with the data values themselves and the grid cell dimensions. For simple binary grids, this metadata (i.e. georeferencing info) is often stored in a small, separate and human-readable text file that is meant to accompany the binary grid file, and typically called a *header file*. Many other file formats for storing gridded data, such as NetCDF and GeoTIFF, bundle the data values (again as IEEE binary numbers) as well as the metadata into a single file; these are referred to as *self-describing* file formats. An advantage of doing this is it prevents misplacing the essential header file.

Many commercial software packages that act on spatial, gridded data make use of the same simple, IEEE binary grid format, although they typically have their own type of header file for storing the metadata. For example, GridFloat is one of the ESRI grid formats, where the binary grid file has the filename extension ".flt" and the header file has extension ".hdr". GridFloat files always have BPE = 4. (They use additional small text files called *world files* and *projection files* for additional metadata.) Many data providers, such as the USGS, provide the option of downloading data in GridFloat format. RiverTools uses the same, simple binary grid files (but supports any BPE) and calls them RTG files (RiverTools Grid), with filename extension ".rtg". The corresponding header file with metadata is called an RTI file (RiverTools Information), and has filename extension ".rti". ENVI, a popular image-processing application is similar, but uses ".img" as the filename extension for the binary grid file and ".hdr" as the extension for the header file. BOV (Block of Value) files are yet another version of this common pattern.

Reading and writing binary grid files is simple in most programming languages. For example, here are the Python/NumPy commands for writing a 2D array of values (stored in a NumPy *ndarray*) from RAM to a binary grid

```
import numpy as np
```

```
nx = 8
ny = 4
grid = np.arange( nx * ny ).reshape(ny, nx)
grid = np.float32( grid)
print type(grid)
print grid.dtype, grid.shape
new_grid_file = 'my_grid_file.rtg'
grid_unit = open( new_grid_file, 'wb' )
if (SWAP_ENDIAN): grid.byteswap( True )  # (optional byte swapping)
grid.tofile( grid_unit )
grid_unit.close()
```

And here are the commands for reading values from a binary grid into a 2D array in RAM:

```
import numpy as np
nx = 8
ny = 4
n_values = nx * ny
dtype = 'float32'
grid_file = 'my_grid_file.rtg'
grid_unit = open( grid_file, 'rb' )
grid = np.fromfile( grid_unit, count=n_values, dtype=dtype )
grid = grid.reshape( ny, nx )
grid_unit.close()
if (SWAP_ENDIAN): grid.byteswap( True )
print type( grid )
print grid.dtype, grid.shape
```

The TopoFlow Python package contains files called *rtg_files.py* and *rti_files.py* in its *utils* folder that provide APIs for working with RTG and RTI files, respectively. Note that header files of any type are easily created by opening an existing header file of that type in a text editor and editing the information.

Appendix C: Description of Binary Grids Needed to Run TopoFlow

The following set of binary grids are required by the Channel Flow components in TopoFlow (i.e. Kinematic Wave, Diffusive Wave and Dynamic Wave). Each grid name is followed by (1) the name of the variable that it contains, (2) the measurement units and (3) the number of Bytes Per Element (BPE), which defines the data type. While TopoFlow allows most other variables to be specified as a single, scalar value to be used for all grid cells, those listed here should be specified as binary grids. Appendix B describes the format of these files and how to read and write them. Each grid should have the same dimensions (ncols and nrows) as the DEM.

Digital Elevation Model (DEM), *elevation*, [m], 4-byte float or 2-byte integer.

D8 Area Grid, *total contributing area*, [km^2], 4-byte float.

D8 Slope Grid, *topographic slope*, [m/m], 4-byte float

D8 Flow Grid, *D8 flow direction code*, [none], 1-byte integer

Channel Width Grid, *bottom widths of all channels*, [m], 4-byte float (For channel grid cells, this width will be less than the grid cell size, but for hillslope grid cells it will be the projected width of the grid cell. Channels are assumed to be prismatic with a trapezoidal cross-sections.)

Manning N Grid, *Manning's N parameter*, [s $m^{-1/3}$], 4-byte float (For channel grid cells, typical values are between 0.025 and 0.05, and for hillslope grid cells (overland flow) a value about 10 times larger should be used.)

Appendix D: Preparing D8-based, Binary Grid Input Files for TopoFlow with D8 Global

The *components* folder in the TopoFlow Python package contains a powerful D8 toolkit called *d8_global.py*, which inherits from a base class defined in *d8_base.py*. Like all other TopoFlow components, this one can be configured by editing a configuration file (this one ending in *_d8_global.cfg*. Note that the *components* folder also contains a fluvial landscape evolution model model component called *erode_d8_global.py*, which inherits from a base class defined in *erode_d8_base.py*. This component calls the D8 Global component after each of its time steps to update the D8 grids after the elevation grid (DEM) has changed due to erosion. The source code for the Erode D8 Global component therefore illustrates how to call the D8 Global component.

The following commands show how to use TopoFlow's D8 toolkit to compute (1) a D8 flow direction grid (with *Jenson* [1985] flow codes), (2) a D8 topographic slope grid, and (3) a D8 total contributing area grid. All of these grids have the same dimensions as the source DEM they are derived from. Note that TopoFlow uses two types of filename prefix to help organize files — a *site prefix* is used for files that describe the study site and therefore don't change between model runs (e.g. the DEM), while a *case prefix* is used for files that result from running the model for a particular scenario or *case* (e.g. response to a particular storm). Before running this code, you should create a directory in your home directory, such as *"/Users/peckham/TF_Tests/C2_runs"* and copy a DEM as a binary grid along with an RTI header file into the new directory as well as a CFG file for the D8 Global component with extension *"_d8_global.cfg"*. Then *cd* to this directory and use it for both *in_directory* and *out_directory* in the following. (Allowing these two directories to be different provides maximum flexibility, e.g. several users can share data in the same input directory but save results in their own output directory, and different components can use different output directories.)

```
import topoflow
from topoflow.components import d8_global

d8 = d8_global.d8_component()

d8.DEBUG  = False
SILENT = False
REPORT = True

in_directory = '/Users/peckham/TF_Tests/C2_runs/'
site_prefix  = 'C2_basin'
filename     = site_prefix + '_d8_global.cfg'
cfg_file     = in_directory + filename    # This is the config file.
time = 0.0

d8.initialize( cfg_file=cfg_file, SILENT=SILENT, REPORT=REPORT )
d8.update( time, SILENT=SILENT, REPORT=REPORT )
```

Once the above set of D8-based grids have be computed for a given DEM, they can be saved into binary grid files (see Appendix B) with the following additional commands.

```
from topoflow.utils import rtg_files
from topoflow.utils import rti_files

out_directory = '/Users/peckham/TF_Tests/C2_runs/'
#  site_prefix = 'C2_basin'
```

```
header_file = (out_directory + site_prefix + '.rti')
grid_info = rti_files.read_info( header_file, REPORT=True)

# Save D8 flow direction grid (flow codes, not aspect)
d8_code_file = (out_directory + site_prefix + '_flow.rtg')
rtg_files.write_grid( d8.d8_grid, d8_code_file, grid_info, RTG_type='BYTE')

# Save D8 contributing area grid
d8_area_file = (out_directory + site_prefix + '_d8-area.rtg')
rtg_files.write_grid( d8.A, d8_area_file, grid_info, RTG_type='FLOAT')

# Compute the D8 slope grid (not available in d8 object yet)
pIDs   = d8.parent_IDs
d8_slope = (d8.DEM - d8.DEM[pIDs]) / d8.ds

# Save the D8 slope grid
d8_slope_file = (out_directory + site_prefix + '_d8-slope.rtg')
rtg_files.write_grid( d8_slope, d8_slope_file, grid_info, RTG_type='FLOAT')
```

Appendix E: Installing the TopoFlow Python Package and Dependencies

Step 1. Install Python 2.7 and commonly-used Python packages. One of the easiest ways to do this is to install *Anaconda*, a complete, open-source Python platform. Anaconda supports MacOS, Linux and Windows. You can download the installers from: *https://www.continuum.io/downloads*. The installation includes over 100 Python packages that support scientific work. This includes all but one of the packages needed by TopoFlow, including: *NumPy, SciPy, setuptools, pip and h5py*. It also includes *Matplotlib, Jupyter, Pandas, curl, wheel* and many others. Anaconda also includes a package and dependency manager called *conda*, which makes it easy to install any of 620 other Python packages (e.g. *netCDF4*).

Step 2. Install the netCDF4 module. TopoFlow uses this module to write model output to standardized netCDF files. The netCDF4 module relies on the *h5py* package that is included with Anaconda.

```
$ conda install netCDF4
```

You can check whether the package was installed correctly by typing *python* in a terminal window (to start a Python session) and then typing *import netCDF4* at the Python prompt.

Step 3. Download the TopoFlow Python package (v. 3.5) from GitHub.
Download it from: *https://github.com/peckhams/topoflow* as a zip file and unzip it.

Step 4. Install the TopoFlow Python package. There are many advantages to installing TopoFlow as a Python package, but it is also helpful to retain the option of making changes to the TopoFlow source code without rebuilding the package. It is therefore recommended to install TopoFlow as an "editable install". This is done by copying the contents of the TopoFlow package folder someplace convenient (e.g. to a folder called TopoFlow_3.5 in your home directory). The package folder contains a file called *setup.py* used for installation. Then, in a terminal window, type the commands

```
$ cd TopoFlow_3.5
$ pip install --editable ./
```

A folder with extension *.egg-info* will be created in the TopoFlow_3.5 folder that allows it to be recognized as a Python package. (Note: A similar but slightly different method is to use the command: *python setup.py develop* instead of *python setup.py install* .)

Step 5. Perform a test run of TopoFlow with the default data set *Treynor*. TopoFlow allows you to specify different directories for model input and output files in the CFG files. The input files for the Treynor data set are in the *examples/Treynor_Iowa_30m* folder of the TopoFlow package. However, output files are written to a directory in your home directory called *TF_Output/Treynor*. So first, create this directory with the commands

```
$ cd;  mkdir TF_Output;  mkdir TF_Output/Treynor
```

Next, open a terminal window and type:

```
$ python -m topoflow
```

You can edit the EMELI *provider file* (with extension *providers.txt*) to specify different components to use for the various hydrologic processes. Each process component is configured with its own *configuration file*, or CFG file, which are text files with extension *.cfg*.

Step 6. Run TopoFlow with your own data sets. Acquire a DEM for your study site and create necessary input files as explained in Appendix B and C. You need a CFG file for every component you want to use in the CFG directory. You also need a *path info CFG file* (extension *path_info.cfg*), an *outlet file* (extension *outlets.txt*) and an EMELI *provider file* (extension *providers.txt*). You should start with copies from the Treynor example and edit them as needed, making sure their *comp_status* has been set to *Enabled*. All of these files have filenames that begin with the *cfg_prefix*, which is typically the *case_prefix* associated with a particular modeling scenario. To run TopoFlow for your own data set, open a terminal window and type:

```
$ python -m topoflow --cfg_prefix PREFIX --cfg_directory DIRECTORY
         --driver_comp_name DRIVER
```

Here, *driver_comp_name* defaults to "topoflow_driver", but can be any of the component names listed in the comment section of the provider file. With your own data set, you may need to use smaller time steps in the CFG files to achieve a numerically stable model run (i.e. that doesn't crash). Also, you should use the same time step in the CFG files for the meteorology and infiltration components. Be aware that grid stack files can be large (i.e. those ending with *.rts* or *2D-*.nc*) and can accumulate over multiple model runs.

Appendix F: How to Set up a Model Run with TopoFlow-IDL

TopoFlow-IDL 1.6 is the original version of the TopoFlow model, written in Interactive Data Language (IDL). It is free, open-source software and has a graphical, point-and-click user interface. It also includes a set of data preparation routines that have not yet been ported to the newer, component-based version of TopoFlow written in Python/NumPy. TopoFlow-IDL requires some version of IDL (a commercial product sold by Harris Geospatial Solutions) in order to run. One option is to obtain the free IDL Virtual Machine [*IDL VM*, 2016], which is able to launch programs saved as IDL SAV files, including *topoflow.sav*. TopoFlow-IDL 1.6 can also be launched from within an application called RiverTools 4.0, which has an embedded IDL license. RiverTools 4.0 (RT4) is a commercial software package for digital terrain and hydrographic analysis available from Rivix Software [*RiverTools*, 2016]. See *Peckham* [2009b] for more information. TopoFlow-IDL 1.6 is included with RiverTools 4.0 as an example plugin, and can be launched from the RT4 *User* menu. This section explains how to prepare input data for TopoFlow model runs using RiverTools 4.0 and TopoFlow-IDL 1.6.

Step 1. Obtain a DEM (digital elevation model) for the basin that you wish to model. If the DEM has dimensions greater than about 500 columns and 500 rows, then it is usually best to subsample the DEM (by averaging) to have dimensions in this range. Using larger DEMs will result in longer model runs and may result in RTS files (RiverTools Sequence) for which you do not have enough space on your hard drive. It is good to start with smaller DEMs and then to increase the size/resolution of your DEM for subsequent model runs if you determine that higher resolution is necessary and you have sufficient time and disk space. Tools for mosaicking, subsetting and subsampling DEMs are available in hydrologic GIS software such as RiverTools 4.0.

Step 2. Create a D8 flow grid, area grid, slope grid and Horton-Strahler order grid for your DEM using RiverTools 4.0 or a similar program. The flow grid should be converted, if necessary, to have the RiverTools flow codes (the standard ones introduced by *Jenson* [1985] and a data type of "byte" (1 byte per pixel). The area and slope grids should have a data type of "float" (4 bytes per pixel) and the units in the area grid should be square kilometers. The Horton-Strahler order grid should also have a data type of "byte".

Step 3. TopoFlow requires the column and row of the pixel (i.e. grid cell) or pixels in your basin for which you wish to monitor the modeled values. It also requires the area (in sq. km) and relief (in meters) for this pixel or pixels. One way to obtain these values is to simultaneously use the *Vector Zoom* and *Value Zoom* tools (which are linked) that are available in RiverTools 4.0. The *Vector Zoom* tool allows you to make sure that you are selecting a pixel that lies on the main water course and not, for example, on a nearby hillslope. The *Value Zoom* tool has a *Select Grid* option in its *Options* menu that lets you determine the contributing area and relief for the selected pixel.

Step 4. Collect hydrologic parameters for the basin of interest. Frictional losses must be parameterized for every channel in the river network and the parameters usually vary in the downstream direction. The *Create* → *Channel Geometry Grids* dialogs in TopoFlow-IDL 1.6 allow you to create grids of "Manning's N", bed width and bank angle (for a trapezoidal cross-section) by parameterizing them in terms of contributing area or Horton-Strahler order (given as grids). In these parameterizations, free parameters should be chosen so as to reproduce "Manning's N" or channel width values at locations for which these are known, such as the basin outlet.

The variables mentioned in the last paragraph are needed to model the "channel process" but you will also need parameters for every other type of hydrologic process that you wish to model. This includes snow melt, evapotranspiration, infiltration and shallow subsurface flow. You will need estimates for soil properties such as hydraulic conductivity and porosity in order to model the infiltration and subsurface flow processes. There is

usually considerable uncertainty associated with these soil properties, since they typically vary spatially and can also depend on various aspects of the local geology.

Step 5. Start TopoFlow-IDL and click on the *New Model Run* button. The first panel in the wizard-style interface asks you for general information regarding the run, such as the working directory, data set prefix and model run prefix. The model run prefix allows you to make multiple model runs, with different parameter settings, for the same DEM data set. You can also enter comments into the *Run comments* text box to describe the run and these will be saved in the specified *Comment file* in the working directory. A log file is also created that contains a summary of most parameter settings and model output. This log file contains plain text and can be viewed with any text editor.

Step 6. Click on the *Next* button at the bottom of the panel. The next panel lets you choose the method that will be used to parameterize each physical process that you will be modeling. You can turn off a process entirely by selecting *None* from the droplist of methods. The droplist usually contains a simple method that requires just a few parameters, as well as a more complex and physically-correct method that necessarily requires more input data. As an example, the Snowmelt process has the simple *Degree-Day* method as well as the more rigorous *Energy-Balance* method. TopoFlow uses the hierarchy of Process, Method, Functions and Variables as a unifying model framework. Each method may be concisely defined in terms of a set of functions that relate input variables to output variables. TopoFlow-IDL is designed so that it is relatively simple for users to add their own methods. However, adding a new method does require some programming in IDL and is beyond the scope of this tutorial.

Once you have selected a method from the droplist of choices for a given physical process, clicking on the *In...* button opens a dialog for entering the parameters that the selected method requires. You can learn more about any selected method, its input variables and equations by clicking on the *Help* button at the bottom of the dialog. TopoFlow-IDL 1.6 uses an HTML help system which requires that the help files be installed in a standard place, such as "C:/Program Files/TopoFlow/help" on a PC running Windows, "/Applications/TopoFlow/help" on a Mac running Mac OS X, and "/usr/local/topoflow/help" for Linux/Unix computers. Clicking on the *Out...* button opens a dialog for choosing which of the modeled hydrologic variables will be saved and how they are to be saved. The two main output options are: (1) Save the variable as a sequence of spatial grids (as a RiverTools Sequence or RTS file), at a specified sampling rate and (2) Save the values of the variable as a time series (in a multi-column text file) for each of the pixels that will be monitored, at a specified sampling rate. The pixels to be monitored are specified in a subsequent wizard panel.

Step 7. For this tutorial we will select the default, *Uniform in space, given durations* for the *Precipitation* process. Click on the *In...* button to open the dialog for entering the input parameters that this method requires. If you have rain gauge data (and a sufficiently small watershed) you may enter that data into this table, taking note of the units for *Rate* and *Duration*. You may want to do a test run with the default parameters to get a sense of how long it will take for the model to run with your data. Note that increasing the duration can result in a much larger peak discharge and a longer model run. It is unrealistic for large rainrates to occur in a spatially uniform manner over basins larger than a relatively moderate size. It is also uncommon for them to have long durations. It may also be unrealistic to neglect some processes such as infiltration.

Once *Rate* and *Duration* values have been entered into this dialog's table, you will generally want to save them in a text file by clicking on the *Save table to file* button. One convention for the name of this text file is "00_Rain_Data#.txt", where "#" is a number used to distinguish between multiple rainfall tables that you may want to experiment with. The "00_" prefix ensures that all of your saved tables will sort to the beginning of your working directory so that they are grouped together and easily located. In a future model

run, you can quickly reload this table of values by clicking on the *Read table from file* button and selecting the file you just saved. We will see later in this tutorial that other parameter tables in TopoFlow can also be saved in this way. Several sample data sets are available for learning to use TopoFlow (e.g. Treynor, Small and Plane) and you can reload sample parameter tables for them in this manner.

Step 8. For this tutorial we will select the default, *Kinematic Wave, Manning* method for the important *Channel flow* process. Click on the *In...* button to open the dialog for entering the input parameters that this method requires. This method parameterizes the downstream variation in channel parameters in terms of Horton-Strahler order. Since the flow grid effectively "channelizes" the entire DEM, including the hillslopes, orders 1 and 2 will most likely correspond to the hillslope pixels while the higher orders will correspond to channel pixels. However, if the pixel sizes for your subsampled DEM are larger than the hillslope scale, then it is possible that hillslopes are not resolved at all by your DEM and flow grid. How you set these parameters will depend on this distinction. If hillslope pixels are resolved by your DEM's pixel size (or grid spacing), then you should generally treat the order 1 and order 2 pixels as hillslope pixels and set their *Manning's N* and *Bed width* values accordingly. Overland flow on hillslopes tends to follow a Manning- like friction law, but with a "Manning's N" value that is around 0.30 instead of the typical value for open channels of about 0.03. The *Bed width* for a hillslope pixel should be set to the entire width of the pixel, since frictional loss of momentum will then occur over the entire surface of the pixel. Bank angles have no meaning for hillslope pixels, but for channel pixels can be set to a value that defines an appropriate trapezoidal channel cross-section. Once values have been entered into this dialog's table, you will generally want to save them in a text file by clicking on the *Save table to file* button. One convention for the filename of this text file is "00_Channel_Data#.txt", where "#" is a number used to distinguish between multiple tables that you may want to experiment with. Recall from Step 7 that precipitation parameters were previously saved in file called "00_Rain_Data.txt".

"Manning's N" parameters definitely have an effect on the resulting hydrograph, and can cause multiple peaks in a small basin's hydrograph to be either distinct or merged together. Tables of *Manning's N* values for typical channel types may be found in books on open channel flow. A typical, middle-range value for channels is 0.03. Note that the logarithmic law of the wall and Manning's formula are two different methods for parameterizing frictional loss of momentum but they agree quite closely as long as the relative roughness (water depth over typically roughness length scale) is in the range of 100 to 10000.

When you are finished entering values into this dialog, click on the OK button at the bottom of the dialog to accept and save the new settings. Note that if any of the required grid files (indicated toward the bottom of the dialog) are missing, a warning message will be issued. The "Timestep" at the bottom of the channel process method dialog is the timestep that controls the entire model, even though some of the other hydrologic processes may only be computed/updated according to their own, larger timestep. A Courant condition can be used to choose a timestep that matches your DEM's pixel size so as to ensure numerical stability. This condition dictates that the maximum distance travelled by water anywhere in the basin in one timestep $v_{max}\,\Delta t$ must be less than one pixel width, Δx. If all pixels have the same fixed width, Δx, then we require $\Delta t < (\Delta x / v_{max})$ for stability. The timestep, Δt, is typically reduced by an additional "factor of safety" of 2 or more. For DEMs with fixed-angle pixels the pixel size varies with latitude but the same principle applies. In the current version, TopoFlow automatically estimates an optimal timestep and uses it as the default in the *Timestep* text box. It is still possible, however, that the model run will require an even smaller timestep for numerical stability.

Step 9. Now click on the *Out...* button for the *Channel flow* process. This opens a dialog that lets you choose which of the modeled variables are to be saved and how they are to be saved. You will usually want to at least save the *Discharge grid* and the *Dis-*

charge values at your monitored pixels. It is a good idea to use the default filenames as a convention. Both the *Grids to save* and *Values to save* subsections of this dialog have their own sampling timestep. You may need to experiment with different timesteps to strike a balance between (1) ensuring that important details are resolved and (2) keeping the output file sizes from being larger than necessary. Note, however, that both of these sampling timesteps must be greater than the channel process timestep, and they should usually be many times larger (e.g. 60 times larger). Note that the units of the sampling timesteps in this dialog are minutes, while the units of the channel process timestep is specified in seconds.

Step 10. You will need to repeat the basic procedure described in Steps 7, 8 and 9 to set the parameters for all of the other physical processes that you wish to model. Note that there must be a runoff-generating process like Precipitation, Snowmelt, etc. in order for the model to operate.

Step 11. Once you have finished setting the parameters for all of the physical processes you wish to model, including both input and output variables, you can save them all in a special text file with the *File → Save Input Vars* option. The next time you start TopoFlow-IDL you can then reload all of these settings by selecting this same text file with the *File → Load Input Vars* option.

Step 12. Click on the *Next* button at the bottom of the*Physical Process* wizard panel to advance to the panel labeled *Info for monitored basins*. This is where you enter the values that you collected in Step 3. Once you have entered these values, you will generally want to save them for later use with the "Save table to file" button. One convention is to save them to a file called "00_Basin_Data.txt". Recall from Steps 7 and 8 that precipitation parameters and channel flow parameters were previously saved in files called "00_Rain_Data.txt" and "00_Channel_Data.txt". At the bottom of this wizard panel, there is a check box labeled "Check mass balance for basin 1?". If this option is checked, then you must specify an RTM (RiverTools mask) file that defines the set of grid cells that lie in the basin upstream of the first monitored grid cell in the list above. This option allows TopoFlow to compute a detailed mass-balance check which will be printed in the Output Log Window at the end of the model run. If you have RiverTools, you can create an RTM file for a basin with the *Extract → Mask → Subbasin Mask* tool.

Step 13. Click on the Next button at the bottom of the wizard panel to advance to the final panel of the *New Model Run* wizard. At the top of this dialog you will see a droplist labeled *Stopping criterion*. There are currently 3 different options in this droplist. The default is especially useful for modeling the hydrologic response due to a storm event and saves you from trying to guess how long it will take for the hydrograph to drop to a specified value. This method also works for hydrographs with multiple peaks. The model stops when a value equal to P% of the highest value encountered so far is reached. The default is 5 percent. You can change this value by clicking on the *Options* button. As with the *In...* and *Out...* buttons, you will get a different dialog when you click on this button depending on which *Stopping criterion* you have selected.

Step 14. You can use the *Back* button at the bottom of the wizard panels to go back and check or change any of the parameters that you entered previously. You can also get an estimate of the total space that will be required to save any output files you specified by clicking on the *Get Outfile Size* button at the bottom of the dialog. If there are messages in the *Output Log Window* from a previous run that you would like to delete, you can do this by clicking on the *Clear Window* button.

Step 15. When you are ready to start the model, click on the *Start Model Run* button at the bottom of the last wizard panel. Output messages will be displayed in the *Output log window* while the model is running. The hydrograph for the first pixel in your list of monitored pixels will be displayed dynamically in the small window on the left-hand side.

You can stop a model run at any time by pressing any key on your keyboard during the run. In most cases, TopoFlow-IDL should stop within 2 seconds of pressing a key and all output files should be closed properly.

Step 16. When a model run is finished, you will most likely want to plot some of the results. The *Plot → Function* dialog can be used to create a simple plot of numbers in a multi-column text file such as the hydrograph for the monitored pixels, which has the filename extension "_OUTQ.txt". Similarly, the *Plot → RTS File* option can be used to display a grid sequence (stored as an RTS file) as an animation. The *Plot → RTS to MPG* option can be used to create an MPEG file from a selected RTS file if you have a valid IDL license with the MPEG option enabled.

If you have access to RiverTools 4.0, you can use the *Display → Function* and *Display → Grid Sequence* dialogs to display your hydrographs and grid sequences. Each of these dialogs draws on the functionality of RiverTools 4.0 to offer numerous additional features, some of which are located in the *Options* and *Tools* menus of the display windows. The *Time Profile* and *Animated Profile* tools are particularly useful and you also have the option to save animations as movies in MP4 or AVI format.

References

Dingman, S. L. (2002), *Physical Hydrology, 2nd edition*, Prentice Hall.

Green, W. H., and G. A. Ampt (1911), Studies on soil physics: Part I. The flow of air and water through soils, *Journal of Agricultural Science*, *4*(1), 1–24.

IDL VM (2016), Interactive Data Language (IDL) Virtual Machine, http://www.harrisgeospatial.com/Support/HelpArticlesDetail/TabId/219/ArtMID/900/ArticleID/12395/The-IDL-Virtual-Machine.aspx.

IEEE (2008), IEEE 754-2008, Institute of Electrical and Electronics Engineers, https://standards.ieee.org/findstds/standard/754-2008.html.

Jenson, S. K. (1985), Automated derivation of hydrologic basin characteristics from digital elevation model data, in *Proceedings of the Digital Representations of Spatial Knowledge*, pp. 301–310, Auto-Carto VII, Washington, D.C., http://mapcontext.com/autocarto/proceedings/auto-carto-7/.

Parlange, J.-Y., I. G. Lisle, R. D. Braddock, and R. E. Smith (1982), The three-parameter infiltration equation, *Soil Science*, *133*(6), 337–341.

Peckham, S. D. (2009b), Chapter 18: Geomorphometry in RiverTools, in *Geomorphometry: Concepts, Software, Applications*, *Developments in Soil Science*, vol. 33, edited by T. Hengl and H. I. Reuter, pp. 411–430, Elsevier, http://dx.doi.org/10.1016/S0166-2481(08)00018-4.

Peckham, S. D., and J. L. Goodall (2013), Driving plug-and-play models with data from web-services: A demonstration of interoperability between CSDMS and CUAHSI-HIS, *Computers & Geosciences, special issue: Modeling for Environmental Change*, *53*, 154–161, doi:10.1016/j.cageo.2012.04.019, http://dx.doi.org/10.1016/j.cageo.2012.04.019.

Priestley, C. H. B., and R. J. Taylor (1972), On the assessment of surface heat flux and evaporation using large-scale parameters, *Mon. Weather Rev.*, *100*(2), 81–92.

Richards, L. A. (1931), Capillary conduction of liquids through porous mediums, *Journal of Applied Physics*, *1*(5), 318–333, doi:10.1063/1.1745010.

RiverTools (2016), RiverTools Home Page, Rivix Software LLC, http://www.rivertools.com.

Smith, R. E., and J.-Y. Parlange (1978), A parameter-efficient hydrologic infiltration model, *Water Resources Research*, *14*(3), 533–538.