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

Key Points:

- This paper provides a simple example of a reproducible hydrologic modeling study.
- All of the data, models and processing scripts used in this study are shared online.
- Customized data preparation scripts are typically required due to data heterogeneity.

*orcid.org/0000-0002-1373-2396

†orcid.org/0000-0002-6612-3439

‡orcid.org/0000-0002-8310-3261

§orcid.org/0000-0001-9585-8517

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

Abstract

Modern geoscientists have online access to an abundance of different data sets and models, but these resources differ from each other in myriad ways and this heterogeneity works against interoperability as well as reproducibility. The purpose of this paper is to illustrate the main issues and some best practices for addressing the challenge of reproducible science in the context of a relatively simple hydrologic modeling study for a small Arctic watershed near Fairbanks, Alaska. This study requires several different types of input data in addition to several, coupled model components. All data sets, model components and processing scripts (e.g. for preparation of data and figures, and for analysis of model output) are fully documented and made available online at persistent URLs. Similarly, all source code for the models and scripts is open-source, version controlled and made available online via GitHub. Each model component has a Basic Model Interface (BMI) to simplify coupling and its own HTML help page that includes a list of all equations and variables used. The set of all model components (TopoFlow) has also been made available as a Python package for easy installation. Three different graphical user interfaces for setting up TopoFlow runs are described, including one that allows model components to run and be coupled as web services.

1 Introduction

Observational data and predictive models are the two pillars of science. Physically-based, mathematical models summarize our current best knowledge of how physical systems operate and they are used to build computational models that predict future states from initial conditions. Data sets and computational models therefore represent two fundamental resource types that geoscientists use to work on problems of societal interest, such as watershed management and the effects of climate change. While modern geoscientists have online access to an abundance of different data sets and models, these resources differ from each other in myriad ways and this heterogeneity works against interoperability. Interoperability is typically necessary, however, because a single data set or model is seldom sufficient to tackle a nontrivial geoscience problem. For example, models typically require several different types of input data which they may obtain from reading input files or from other models. It is therefore often necessary to couple together many different heterogeneous resources into a computational workflow, and geoscientists spend a large fraction of their time just setting up and executing these workflows. An extensive,

empirical analysis of scientific workflows [*Garijo et al.*, 2013] (and references therein) suggests that geoscientists typically spend between 60% and 80% of their time on dealing with such issues, leaving the remainder as the time available for the science. The phrase *data friction* [*Edwards*, 2010; *Edwards et al.*, 2011] has also been used to describe this problem.

This challenge of interoperability leads to significant complexity and is one of the main barriers to *reproducibility*, that is, the ability to reproduce a scientific study or experiment. Reproducibility is considered to be one of the cornerstones of science, but complex workflows often cannot be replicated unless they are fully described and documented. In recent years this topic has been receiving increased attention. [*Hutton et al.*, 2016] cite examples from a broad array of scientific disciplines and then focus on this issue in the context of computational hydrology. Since most modern scientific workflows rely on numerous *digital resources* such as data sets, data preparation and analysis software and computational models, it is now recognized that simply describing a workflow is not enough — all of the digital resources used in a study should also be made available with persistent identifiers (i.e. URLs, DOIs).

This paper begins with a description of a spatial, hydrologic model called TopoFlow developed by the first author and colleagues over the last 16 years. This is followed by an example application to a small, Arctic watershed in the Caribou – Poker Creek Research Watershed (CPCRW). This application starts with a description of the study site, then walks through all of the steps involved in obtaining the required input data sets, preparing them for use by TopoFlow, setting up a model run, executing the model and then analyzing the results. By focusing on late summer rainfall events, the dominant hydrologic processes are rainfall and infiltration — in fact, a large percentage of the rainfall volume is lost to infiltration and does not contribute to the volume flow rate (discharge) at the basin outlet. A well-known, simplified, but physically-based model for the infiltration process, known as the Green-Ampt method, is used within the model and also in our interpretation of the results.

2 Overview of the TopoFlow Model Toolkit

TopoFlow is a spatially-distributed, process-based and open-source hydrologic model [*Peckham*, 2009a]. Development of TopoFlow started in 2000 through a collaboration be-

tween Peckham (University of Colorado, Boulder) and several colleagues at WERC, the Water and Environmental Research Center (University of Alaska, Fairbanks). Peckham had just developed a spatially-distributed rainfall-runoff model that used the D8 method for determining flow directions over topography given as a digital elevation model (DEM). That model distinguished between DEM grid cells on hillslopes and those containing channels, using overland flow for the former and open-channel flow hydraulics for the latter [*Henderson*, 1966]. Each channel grid cell contained a prismatic channel with its own trapezoidal cross-section and roughness parameters. It supported three different methods of channel flow routing, namely kinematic wave, diffusive wave and dynamic wave. It also provided the option of using either Manning's formula or the logarithmic law of the wall to model flow resistance. Hinzman's group at WERC had just published a paper on their ARHYTHM model [*Zhang et al.*, 2000], a spatially-distributed hydrologic model for use with Arctic watersheds with an advanced treatment of thermal processes. ARHYTHM used a computational grid of triangles and supported multiple treatments of snowmelt and evaporation, based on many years of prior work in the Arctic, e.g. *Hinzman et al.* [1996], *Hinzman et al.* [1998]. ARHYTHM supported energy-balance treatments of both snowmelt and evaporation in cases where shortwave and longwave radiation measurements were available. It also supported simplified treatments of these two processes that required less input data, namely the Degree-Day method for snowmelt and the Priestley-Taylor method for evaporation.

2.1 Single-application, IDL version of TopoFlow with GUI

This collaboration led to the merging of the two models into a single model called TopoFlow-IDL that supported multiple methods of modeling each hydrologic process and that also had a user-friendly graphical user interface (GUI). While ARHYTHM was written in Fortran and used triangular grid cells, TopoFlow was written in IDL (Interactive Data Language) and used rectangular grid cells with D8-based flow directions. The process treatments of snowmelt, evaporation and shallow subsurface flow from ARHYTHM were converted to IDL, and array-based best programming practices (e.g. avoidance of spatial loops) were used to ensure good runtime performance. A wizard-style GUI for TopoFlow-IDL was developed and from 2000 to 2007 support for many additional hydrologic processes were added. These additions can be summarized as follows:

Meteorology. The first major addition was a Meteorology module based on celestial mechanics with shortwave and longwave radiation calculators, using the approach outlined in *Dingman* [2002] (Appendix E). This allowed the energy-balance snowmelt and evaporation process modules to be used even when shortwave and longwave radiation measurements were unavailable. This module required measurements of precipitation rate, air temperature, relative humidity and wind velocity as its primary inputs, to be read from files, but computed many output variables from these.

Infiltration. Three different methods for modeling infiltration were added, based on the work of *Smith et al.* [2002], namely the well-known Green-Ampt, Smith-Parlange three-parameter method and the Richards equation (1D) method.

Diversions. Support for flow diversions were added (e.g. canals, tunnels) as well as support for sources and sinks that add or remove some or all of the flow that passes through a given grid cell. Flow diversions are relatively common and must sometimes be modeled to properly account for mass balance. (e.g. the 23.3-mile long Harold D. Roberts Tunnel under the Continental Divide from Dillon Reservoir to the South Platte River near Denver, Colorado). This capability has also been used to model flow in river delta distributaries [*Hannon et al.*, 2008].

D8 Toolkit. The D8 method [*Jenson*, 1985] allows flow direction (aspect) to be computed from a DEM. Given a grid of D8 flow direction codes, several additional geometric attributes can be computed including topographic slope, total contributing areas (TCA) and channel lengths. TCA can then be used to compute grid-based estimates of channel attributes including widths and roughness parameters as explained in *Peckham* [2009a]. This component contains all of the code necessary to compute D8-based attributes. (See detailed instructions below.)

All of the TopoFlow components are described in detail in *Peckham* [2009a] and also in HTML-based help files online that are listed in Appendix A. In addition to the hydrologic process components, TopoFlow-IDL includes a number of tools for preparing the input data required by the process components.

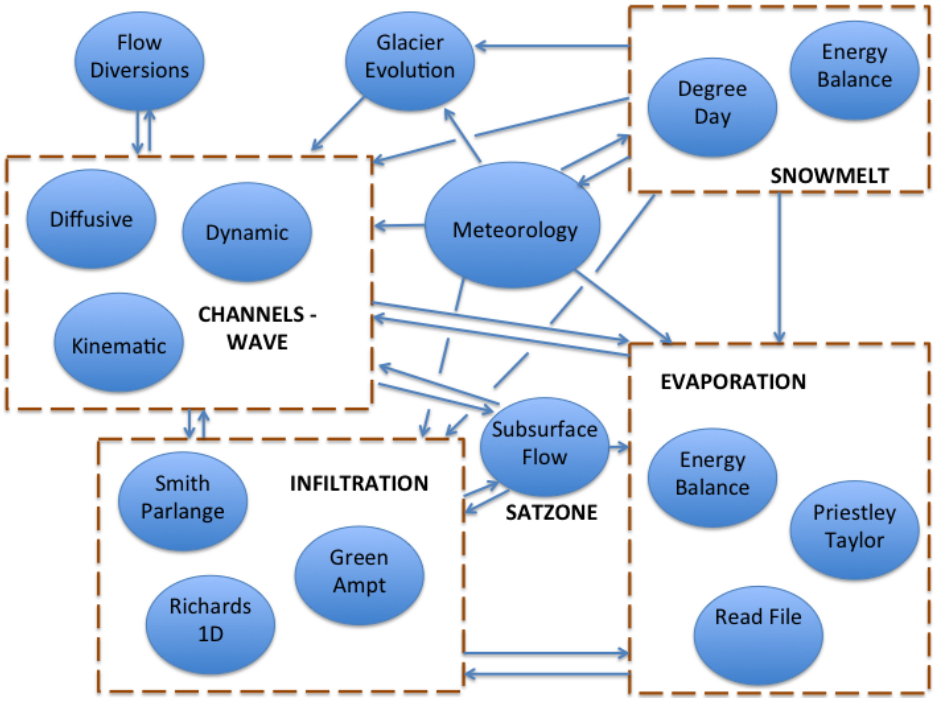
TopoFlow-IDL has been used in a number of studies, such as *Schramm* [2005], *Bolton* [2006], *Coe et al.* [2008], *Liljedahl* [2008] and *Pohl et al.* [2009] several of which involved Arctic hydrology.

2.2 Component-based, Python version of TopoFlow

The NSF-funded CSDMS project began in 2007, with Peckham as its Chief Software Architect. The CSDMS modeling framework was designed to support flexible, component-based, "plug and play" modeling and to promote reuse and coupling of open-source models written by different authors [*Peckham et al.*, 2013]. For component-based modeling, one has to decide on an appropriate level of *granularity*, that is, the level of functionality that components should encapsulate. In any given geoscience modeling context, there are typically multiple *physical processes* that contribute to conservation of mass, momentum and energy, and there are typically multiple *methods* (from very simple to sophisticated) for modeling each process. Different *methods* may differ in various ways, such as (1) the set of equations and variables used to represent the process, (2) the numerical scheme for solving the equations, (3) the programming language, (4) the computational grid, or (5) the use of processors (e.g. parallel vs. serial). Several examples of hydrologic processes were described in the previous section. It is therefore natural to encapsulate individual physical process treatments in interchangeable components, and this sets an appropriate and useful granularity for plug-and-play modeling.

While TopoFlow-IDL already supported multiple methods for modeling each of several hydrologic processes — with users selecting methods from process droplists — all of these modules were dependent on the application to connect the selected modules. A user could only choose from modules that were already included in the application and could not easily replace some of them with modules written by others, nor use a TopoFlow process module outside of the TopoFlow-IDL app. This type of "all in one" model is typical, where the application basically serves as a self-contained framework that allows all of the process modules to interoperate.

As an efficient means of exploring different architectural designs to support plug-and-play modeling, much of the source code for TopoFlow-IDL was converted from IDL to Python/NumPy. TopoFlow was also decomposed into a set of independent process-level *model components* as shown in Figure (1). Each component then had its own time loop, its own configuration file (a text file read by the component at startup to set parameters, etc.), its own HTML help page and so on. Any variables that needed to be passed between model components were made available through a standardized component interface (i.e. set of functions), and many different interface prototypes were tested against numer-



171 **Figure 1.** A diagram of all TopoFlow components, where arrows between components indicate dynamic
172 coupling and the passing of one or more variables between components. Each component provides a method
173 of modeling a particular hydrologic process and the dashed boxes contain alternate methods for modeling the
174 same process, ranging from simple to complex. Users choose one process component from each dashed box.

168 ous design criteria. TopoFlow therefore became a vehicle for designing the CSDMS mod-
169 eling framework [*Peckham et al.*, 2013], including the Basic Model Interface [*CSDMS-*
170 *BMI*, 2016] and the CSDMS Standard Names [*Peckham*, 2014a].

175 The BMI enables simplified, plug-and-play reusability and coupling of model com-
176 ponents written by different people, even when those models differ in terms of program-
177 ming language, computational grid, time-stepping scheme, and the names, units and data
178 types of their input and output variables. The standardized set of functions in the BMI
179 provide a modeling framework with (1) fine-grained control of model execution, (2) de-
180 scriptive information needed for coupling and (3) variable getter and setter functions to
181 support exchanging the values of variables. Based on information retrieved from BMI
182 functions, the modeling framework is able to automatically call its built-in *mediators* (e.g.
183 regridders, unit converters) to deal with differences between the coupled models. A unique
184 feature of BMI is that it also addresses the problem of *semantic mediation* that arises from

each model using its own set of names for input and output variables (its own *internal vocabulary*). A BMI implementation includes a simple mapping from a model's internal variable names to a set of standard variable names called the CSDMS Standard Names (CSN). This mapping allows the framework to perform *automatic* semantic mediation. The CSN are a systematic, unambiguous, cross-domain set of variable naming conventions which have evolved into a formal ontology called the Geoscience Standard Names [*GSN*, 2017].

Prior to 2012, CSDMS had developed a web-based graphical tool called the Component Modeling Tool (CMT) that CSDMS members could download as a lightweight Java application. CMT allowed users to select components from a palette and drag them into an arena to become part of a composite model. Once in the arena, CMT provided a tabbed-dialog GUI for *each model component* to collect the user settings necessary to automatically create a configuration file from a model-specific template. The GUI also provided standardized, HTML help pages for each component. Users could create new models from components, configure them, then run them on a high-performance cluster and also monitor their progress through the CMT. CMT has since been superseded by the CSDMS Web Modeling Tool (WMT), which is completely browser-based as well as more robust and efficient. TopoFlow was available in the CMT and is now available in the WMT, [*CSDMS-WMT*, 2016] by choosing *wmt-hydrology*. The WMT allows models to be composed and configured without logging into the high-performance cluster until they are ready for execution. Note that both CMT and WMT are simply graphical *front ends* to the CSDMS modeling framework, which itself is a software stack running on a cluster. They collect and store information in a framework configuration file that is passed to the underlying framework. However, these configuration files may also be created with a text editor.

As a result of deconstructing TopoFlow into separate process-level model components, the Python version of TopoFlow required the presence of a model coupling framework like CSDMS in order to be used. However, some TopoFlow users wanted to be able to run TopoFlow on their own computers, without relying on a cluster with the full CSDMS software stack. This led to the development of a lightweight, experimental modeling framework written entirely in Python called EMELI (Experimental Modeling Environment for Linking and Interoperability) [*Peckham*, 2014b]. Users list which model components they want to use in a *provider file* and then EMELI 1.0 automatically connects each com-

ponent to other components in the set that are able to provide the input variables they require, based on semantic matching with CSDMS Standard Names. EMELI requires each model component to have an implementation of BMI with Python bindings. EMELI also includes two mediators, a time interpolator and a unit converter, that are automatically invoked when model components use different time steps or units.

All of the TopoFlow model components, utilities and a few example data sets are now available in a single, stand-alone Python package, bundled with EMELI [*Peckham*, 2016]. Source code for EMELI is in the *framework* folder of this package. Source code for TopoFlow model components is in the *components* folder, and each has a BMI interface and uses CSDMS Standard Names. This includes a component called *d8_global.py* with a complete D8 toolkit for computing D8 flow direction grids and associated grids (e.g. total contributing area grids). Also included is a D8-based, fluvial landscape evolution model called Erode (*erode_d8_global.py*). Source code for a variety of shared, low-level TopoFlow utilities is in the *utils* folder, and these are used by all components. Several complete sets of input files for testing are included in the *examples* folder of the Python package. Altogether, the TopoFlow package (version 3.5) consists of over 71,000 lines of Python code, including comments.

As part of an NSF EarthCube building block project called *GeoSemantics*, an alternate version of EMELI, *EMELI-Web* [2016], has been developed that can couple TopoFlow model components running on different servers as web services [*Jiang et al.*, 2017]. Values of variables that must be passed between components running on different servers are bundled in NetCDF files for transmission. It also provides a browser-based GUI, similar to WMT, for configuring each model component prior to a model run.

TopoFlow continues to be used in different contexts and in support of different cyberinfrastructure projects. For example, TopoFlow is used by CSDMS staff for teaching. Individual TopoFlow components can also be run within an iPython notebook, which makes it possible to follow every detail of model execution. TopoFlow is also being used in the NSF EarthCube building block project *OntoSoft* to help drive the development of standardized metadata and ontologies for describing geoscience models [*OntoSoft-CSDMS*, 2016].

2.3 The Typical TopoFlow Workflow

With physically-based, spatially-distributed hydrologic models like TopoFlow, most of the work in using them has to do with acquiring and preparing the input data for the study site, configuring the model and then running the model a large number of times with different parameter settings. (Without using a standard component interface like BMI, coupling process models can also be very time-consuming.) Digital elevation data, which plays a very important role in this type of modeling, is generally easy to find and download for any region of interest, at a variety of resolutions, thanks to the efforts of the U.S. Geological Survey and other agencies and projects. Many of the other types of input data, including meteorological, soil and snowpack data, may not exist for a basin of interest, may have only been measured at a station some distance away, or may be difficult to find with an online search. However, this information is generally available online for experimental watersheds such as LTER (Long Term Ecological Research) sites. Data at the scale of individual river channels, such as their widths, depths and bed roughness, is generally not available and must be parameterized with known empirical relationships. Figure (2) provides a high-level illustration of the steps in a typical workflow. Details on how to perform Steps 2 through 4 are included in multiple appendices. Note that Steps 6 through 8 may be repeated many times with different choices of process components and different parameter settings in an effort to understand the physical processes at work in the basin and to make predictions that compare favorably with observations. For hydrologic modeling, one typically compares the predicted and observed *hydrographs* at the outlet of the basin of interest, which show the volume of water per unit time as a function of time (i.e. $Q(t)$) that flows through the outlet.

2.4 Preparing Input Files for TopoFlow

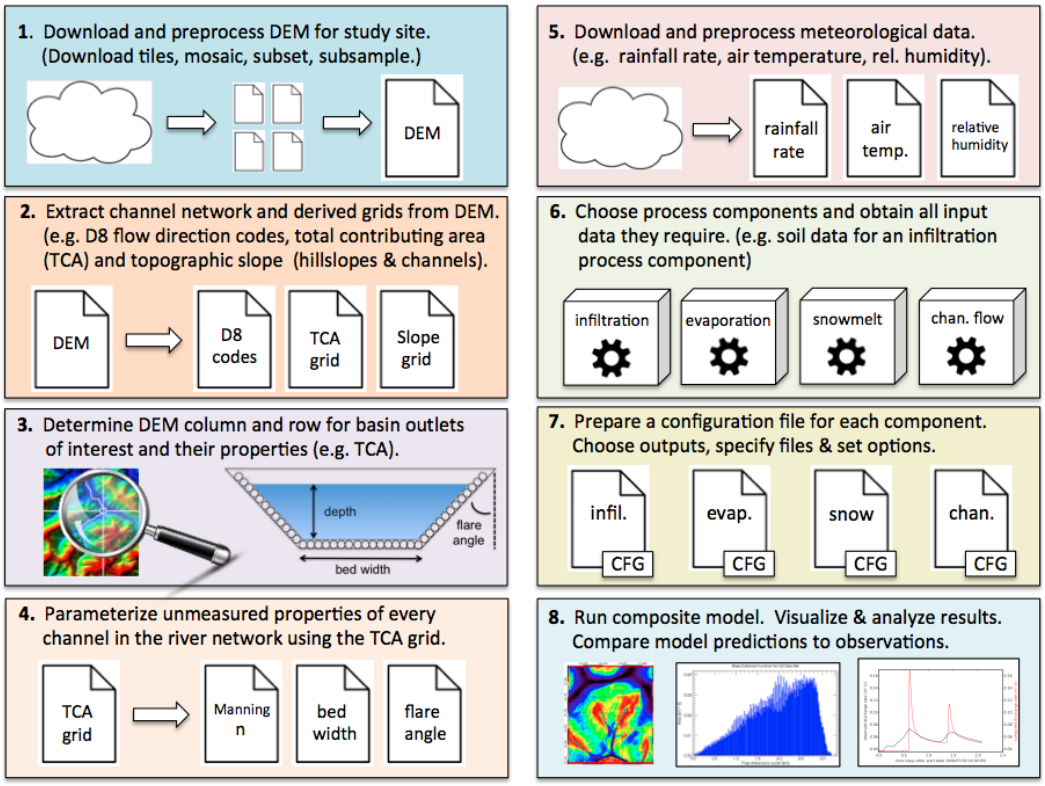
One of the unique features of TopoFlow is that, by design, almost any input variable that occurs in any of the process components can be provided as any of the following:

(1) a **scalar** (a single value that does not vary in space or time),

(2) a **time series** (a value that varies in time, but not in space),

(3) a **grid** (values that vary over space, but do not change over time) or

(4) a **grid stack** (values that are variable in both space and time).



271 **Figure 2.** Typical steps in a TopoFlow modeling workflow.

For example, a spatially uniform, steady rainfall rate would be provided as a *scalar* (to be used for all grid cells and all times), while a space-time rainfall field would be provided as a *grid stack*. A grid stack is essentially a time series of grids. Implementation of this feature is simplified because both IDL and Python are *dynamically typed* programming languages, and *upcasting* therefore occurs automatically in expressions that contain a mixture of scalar values and grids. So functions in TopoFlow usually don't need to check whether arguments are scalar values or grids and can handle mixtures. (*Note:* The term *scalar* is used in TopoFlow to mean *a single numerical value*; this could be confusing because a grid of values can be called a *scalar field*.)

For consistency, simplicity and runtime performance, when TopoFlow reads the values of input variables from files, it expects (and requires) that:

(1) any *time series* is stored in a single-column text file (one value per line),
(2) any *grid* is stored in a IEEE binary grid file, in row-major order
(3) any *grid stack* is stored as a succession of binary grids, stored in a single file.

Binary grid files are a very common and computationally efficient file format. Appendix B describes binary grid files in more detail and explains how to read and write them with Python. Appendix C describes the binary grids that are needed to run TopoFlow. Appendix D explains how to prepare D8-based, binary grid input files for TopoFlow from a DEM with the D8 Global component. (The DEM is also assumed to be stored in a binary grid file.)

While all of the hydrologic process components in TopoFlow are optional and can be disabled with a setting in their configuration file, there are very few simulations that can be performed without using a channel flow component such as the *Kinematic Wave* component. (One example is snow depth accumulation using a snow component and meteorology component.) However, channel flow components require input files that describe both the channel geometry and bed roughness as spatial grids with the same dimensions as the DEM, as well as a D8 channel slope grid. This type of data is not readily available online, so TopoFlow users must prepare these grids. As explained in *Peckham* [2009a], these can be estimated by applying power-law functions of the form $V = c\,(A + b)^p$ to a D8 total contributing area (TCA) grid, because both channel width and bed roughness vary with discharge (and therefore with TCA). However, the power-law function parameters should be chosen carefully to obtain reasonable results. The IDL version of TopoFlow

(TopoFlow-IDL) has a *Create* menu in its GUI that includes dialogs for computing these grids. However, one could also use Python commands similar to those in Appendix C and D to read a TCA grid into a variable, *A*, apply a power-law function, and then write the resulting grid to a binary grid file. The compound filename extensions *_chan-w.rtg*, *_chan-a.rtg* and *_chan-n.rtg* are commonly used in TopoFlow for grids of channel widths, channel trapezoidal bank angles (i.e. flare angles) and the channel roughness parameter, Manning's n, respectively.

Many of the other TopoFlow input variables represent either *initial conditions* – such as depth of water or snow, soil water content and position of the water table – or *forcing variables* (or drivers), such as precipitation rate, air temperature, relative humidity, shortwave and longwave radiation fluxes and wind speed [*Peckham*, 2009a]. These can be acquired from a variety of sources, including federal agencies, but must be either downloaded as or converted to binary grid files (or single-column text files for time series data). They may also need to be clipped or resampled to have the same dimensions and spatial extent as the DEM, which can be done using GIS software.

For output files, TopoFlow users can choose to save a *time series*, *profile series*, *grid stack*, or *cube stack* of data values to a NetCDF file, or to binary grid files. A "cube stack" is a succession of 3D arrays indexed by time, while a "profile series" is a succession of 1D arrays indexed by time, such as a soil-moisture profile that varies over time. These four types of output files include *0D*, *1D*, *2D* or *3D* in their filenames, respectively. The 1D and 3D options are currently only used for subsurface flow variables. Users set flags in TopoFlow configuration files to choose which computed variables to save, using one of these four types of files. Often, one is interested in saving the values of some output variable (e.g. discharge, flow depth or flow speed) at one or more specific locations (e.g. basin outlets) as a time series. This option can be turned on with a setting in a component's configuration file, but requires the user to first create an *outlet file* – a simple text file with the following format:

```
------------------------------------------------------------
 Monitored Grid Cell (Outlet) Information
------------------------------------------------------------
    Column       Row     Area [km^2]      Relief [m]
------------------------------------------------------------
```

```
124        101        4.83503              385.0
```

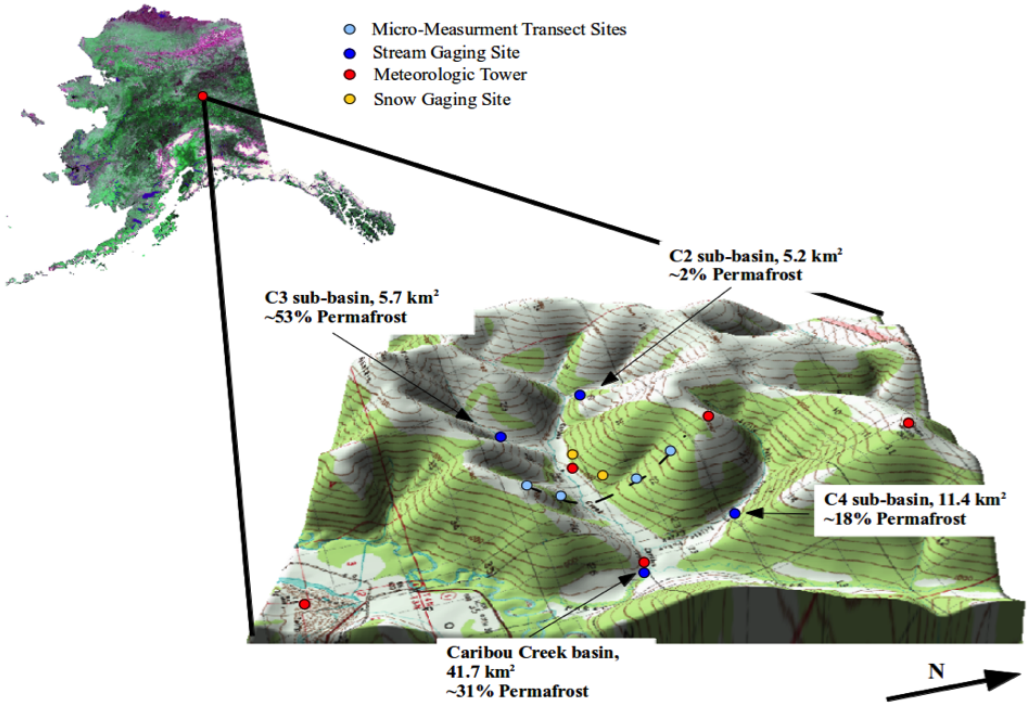
The Column and Row values are essential and must be obtained with GIS software, such as RiverTools, but the Area and Relief values are nonessential. Multiple grid cells can be indicated for monitoring by adding rows.

Many applications provide a graphical user interface (GUI) that can make it much easier to prepare, process, edit, analyze and visualize binary grid input files. For example, Appendix E explains how RiverTools and TopoFlow-IDL can be used. Additional information on preparing input files for TopoFlow can be found in *Peckham* [2009a], and in the TopoFlow Online Tutorial.

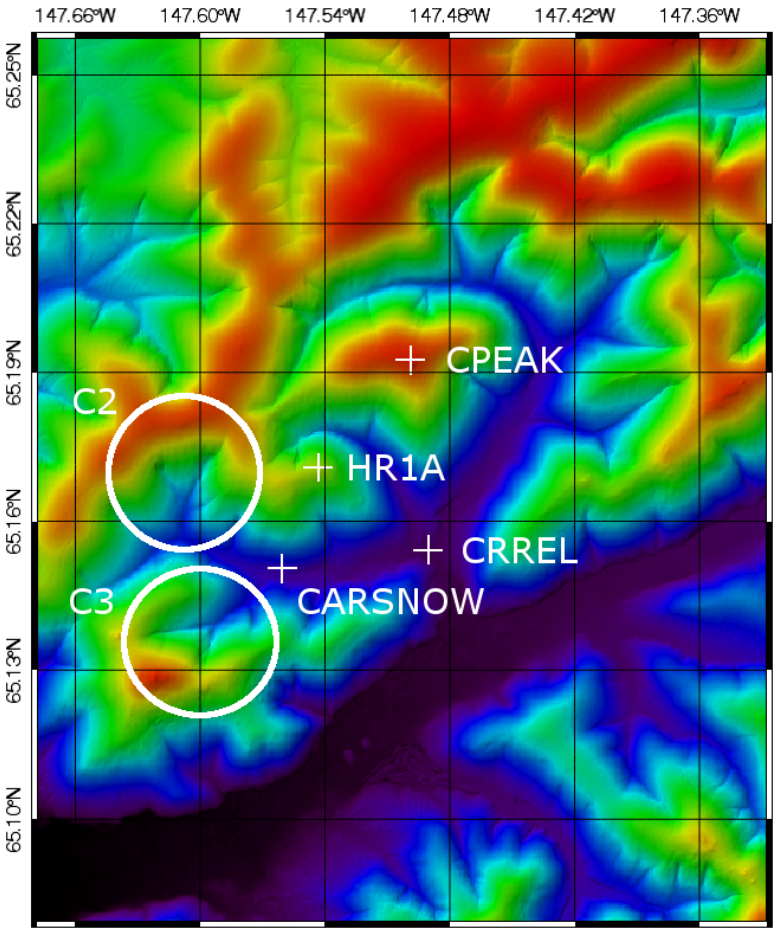
3 Example Application – Caribou Poker Creek Research Watershed

3.1 Description of the Watershed

To illustrate the workflow of the models setup and run we use the data from the Caribou Poker Creek Research Watershed (CPCRW) located 48 km north of Fairbanks N 65° 10′ 147° 30′ Alaska. The CPCRW site is part of the LTER (Long Term Ecological Research) network. Parts of this watershed are underlain by permafrost, where the maximum seasonal thaw depth thickness is about 0.52 [m] at a low elevation point near the confluence of Poker and Caribou Creeks [*Bolton et al.*, 2000, 2004; *Bolton*, 2006]. Black spruce is generally found along poorly drained north-facing slopes and valley bottoms. Aspen, birch, alder and sporadic white spruce are found on the well-drained, south-facing slopes. Tussock tundra, feather moss, and sphagnum mosses are also found along valley bottoms [*Bolton*, 2006]. The watershed encompasses an area of 101.5 [km^2] as shown in Figure (3). CPCRW is located within the boreal forest area. The watershed has six sub-watersheds, where three of them (C2, C3, and C4) have been continuously monitored over the last few decades. We chose to model the C2 sub-watershed within the CPCRW because it is south-facing and has almost no permafrost. South-facing slopes usually correspond to a warmer micro-climate, have a thinner organic layer and well-drained soils. The snowmelt at this site usually happens in a span of one or two weeks at the end of the spring season. Previous studies (e.g. *Bolton et al.* [2004]) have compared results for the C2 basin to those of the north-facing, C3 basin, which has about the same basin area but has 54% permafrost vs. 4% for C2. See Figure (4).



373 **Figure 3.** A map of the present measurement sensors in Caribou Poker Creek Research Watershed study
374 site.



375 **Figure 4.** Locations of the C2 and C2 subbasins and the four met stations within the Caribou – Poker Creek
376 Research Watershed. North is at the top of the image and the circles have a diameter of about 3 [km].

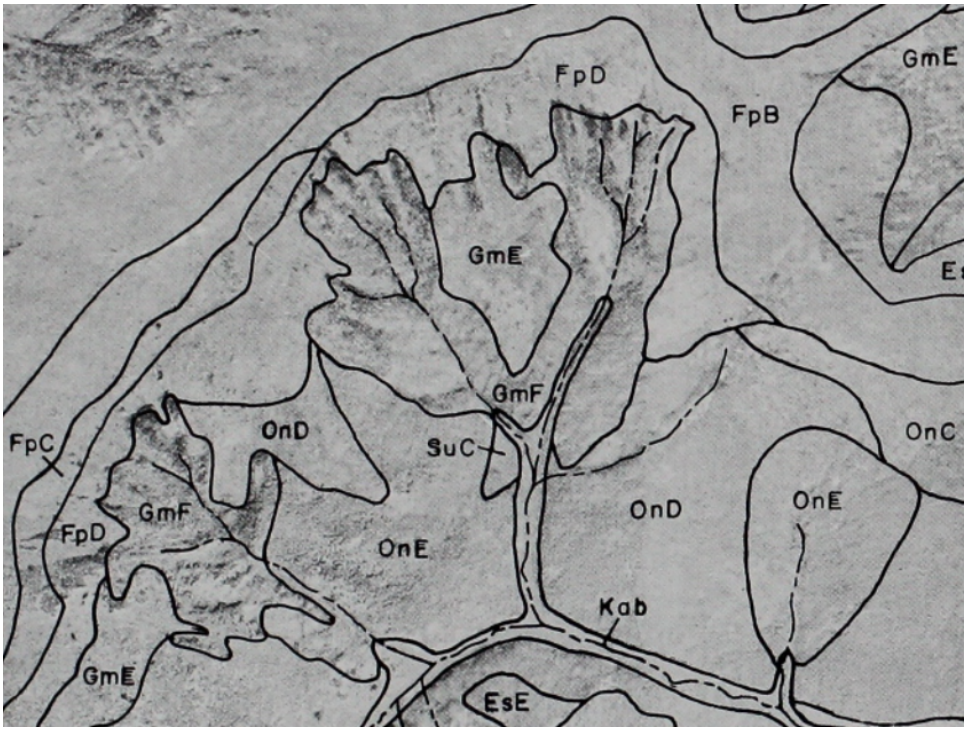


Figure 5. Soil map for part of the Caribou – Poker Creek Research Watershed that includes the C2 basin, from *Rieger et al.* [1972]. Soil types are all specific types of silt loam and corresponding slope ranges, as indicated with the following symbols: *Bradway*: **Br**; *Ester*: **EsD** (12 to 20%), **EsE** (20 to 30%), **EsF** (30 to 40%), *Fairplay*: **FpB** (3 to 7%), **FpC** (7 to 12%), **FpD** (12 to 20%), **FpE** (20 to 30%); *Gilmore*: **GmD** (12 to 20%), **GmE** (20 to 30%), **GmF** (30 to 45%); *Karshner*: **KaB** (3 to 7%), **KaC** (7 to 12%), *Olnes*: **OnB** (3 to 7%), **OnC** (7 to 12%), **OnD** (12 to 20%), **OnE** (20 to 30%), **OnF** (30 to 45%); *Saulich*: **SuB** (3 to 7%), **SuC** (7 to 12%), **SuD** (12 to 20%).

Figure (5) is a soil map for the vicinity of the C2 basin, clipped from a larger soil map contained in *Rieger et al.* [1972] (p. 14). It shows that the soil in the C2 basin is mostly Gilmore silt loam, with Fairplay silt loam near the drainage divide. Table 2 in *Rieger et al.* [1972] (p. 13) provides estimates of K_s (which they call *permeability*, as explained on p. 12), with values between 0.6 and 2.0 [in/hr], or 4.23×10^{-6} to 1.41×10^{-5} [m/s].

3.2 Acquiring a Digital Elevation Model for the Study Site

The TopoFlow model relies on a digital elevation model (DEM) in order to determine the flow directions and slopes that it uses to route water across the landscape. We

used the following steps to obtain a DEM for our study site. While we think it is illustrative to describe these steps, we note that they are specific to a browser-based, graphical user interface (GUI) provided by the U.S. Geological Survey (USGS) as it exists at the time of writing — one that is likely to change in the future. While GUIs like this are typical and fairly easy to use, a hydrologist may need to use many different ones in order to acquire all of the input needed for a modeling study. This illustrates another challenge for reproducibility that is partially alleviated when data providers make their data available for download via a web service with a standardized API.

Step 1. The USGS provides an online tool called The National Map (TNM) for downloading data. In the navigation section on the left side of the page, check the box labeled *Elevation Products (3DEP)* in the *Data* section. This displays additional check boxes with different horizontal resolutions. We used 1 arc-second DEMs for this study, so we checked the box with this label. In the section labeled *File Format*, we checked the radio button labeled *GridFloat*. This is a simple, nonproprietary and efficient format that stores elevation values in row-major order (IEEE 4-byte, floating-point binary values), with the filename extension".flt". Metadata — such as number of rows and columns, bounding box and grid cell dimensions — is saved in a small, companion text file with filename extension ".hdr".

Step 2. Using the TNM browser-based map, we zoomed into the region labeled *Yukon Flats National Wildlife Refuge*, in the area southwest of (and downstream from) the confluence of the Porcupine River and the Yukon River. This is about 38 km north and slightly east of Fairbanks. The Caribou Poker Creek Research Watershed is contained within the 1 degree by 1 degree tile with its southwest corner at 66 degrees North latitude and 148 degrees West longitude. We used the plus button to zoom into this region and clicked the radio button at the top labeled *Current Extent*. We then clicked on the blue button labeled *Find Products*. This resulted in a list of 1-degree DEM tiles, and we then selected the one labeled *USGS NED 1 arc-second n66w148 1 x 1 degree GridFloat 2016* by clicking on the shopping cart plus button to the right. The size of this file (one tile) is 42.99 MB. (A higher-resolution DEM for the same tile, with a grid spacing of 1/3 arc-second and file-size of 377.98 MB is also available.) Note that 1 arc-second of latitude is always roughly 93.6 meters, while 1 arc-second of longitude decreases with latitude as 93.6 $\cos(lat)$ and is significantly less than 93.6 meters for these Arctic latitudes.

3.3 Acquiring Data from the Bonanza Creek LTER Station

The Institute of Arctic Biology at the University of Alaska, Fairbanks maintains a Long-Term Ecological Research (LTER) site (funded by NSF) called the Bonanza Creek LTER. The Caribou-Poker Creeks Research Watershed (CPCRW) is one of the study sites for this LTER project where long-term monitoring data is collected and made available online. Clicking on *Access Data* > *Study Sites Catalog* in the *Data* menu of this website brings up a search filter page for the Study Sites Catalog. Typing *Caribou* in the text box labeled *Name, Description, History* and clicking on the *Submit* button generates a listing of available data and a locator map. For this paper, we selected the C2 subbasin within the Caribou Creek watershed. Data for 4 separate subbasins of Caribou Creek is/are available, namely C1, C2, C3 and C4. Note that each subbasin name begins with the letter "C" for Caribou. There are four weather stations (or "met stations") in the vicinity of Caribou Creek, designated as CARSNOW, CPEAK, CRREL and HR1A. The longitudes and latitudes of these stations are given by

```
CPEAK    -147.4990579,  65.19275149
CRREL    -147.4903787,  65.15425986
CARSNOW  -147.5606703,  65.15065772
HR1A     -147.5435743,  65.17091866
```

Rainfall rates, measured with a tipping bucket, are stored in text files where the header and first line of data for the CARSNOW station look like this:

```
site_id,date,hour,measurement,value,unit,flag
CARSNOW,2006-10-04,1400,Tipping Rain Bucket,0.000,mm,G
```

Similarly, discharge measurements for the C2, C3 and C4 subbasins are available in text files where the header and first line of data look like this:

```
"Watershed","Date-Time","Flow","Units","Flag"
"C2",7/14/1978 7:00:00,29.19,"L/s","G"
```

Volume flow rates (discharges) at the C2 basin outlet have been measured every summer since 1979. The measurement frequency was hourly until summer 2001, after which it was measured every 15 minutes. This long record presents an opportunity for investigating

the possible effects of climate change. Additional metadata is available in a file called *README.rtf.doc* that comes with the data.

3.4 Preparing LTER Data for Model Use

Discharge and precipitation data was cleaned with the preprocessing scripts that can be found on GitHub. The time-date format for both sets of data was converted to seconds with respect to a common reference date so that data could be properly aligned. Discharge data for the three basins (C2, C3, C4) was interleaved, so we selected the desired data using the data frame manipulation utilities in the Pandas Python package. A sample of the data was selected based on the desired date-time interval values. Both data sets contained outliers that were mislabeled with a 'G' (to indicate that the data is good), so these were detected using histograms and then filtered out. Details and analysis of the data sets can be found at the GitHub link presented above.

3.5 Observed Response of the C2 Basin to a Late Summer Rainfall Event

Figure (6) shows rainfall rates that were measured with a tipping bucket at one-hour time intervals at the CPEAK met station for a rainfall event that took place over a two-day period from July 28-30, 2008. Figure (7) shows the corresponding volume flow rates (discharges) measured at the outlet of the C2 basin. The C2 hydrograph shows that there is a baseflow discharge of approximately $0.036\,[m^3/s]$ prior to the rainfall event. (In our model runs we start with dry channels and then add this amount of baseflow to the resulting hydrograph.) The four main rainfall peak values in the measured rainfall time series for the July rainfall event at the CPEAK met station and the three corresponding peak discharges at the C2 basin outlet are given in Table (1). As shown in Figure (8), observed rainfall rates at the other met stations display a similar temporal intensity pattern, which supports treating rainfall as spatially uniform over the C2 basin as a first approximation. However, methods such as *inverse distance weighting* could be used to create a grid sequence of rainfall rates to be used as input to TopoFlow.

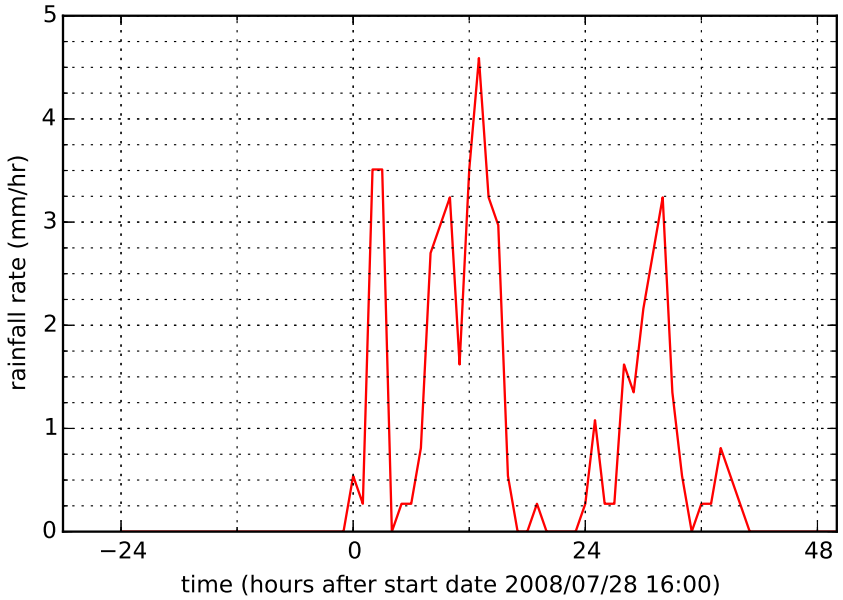


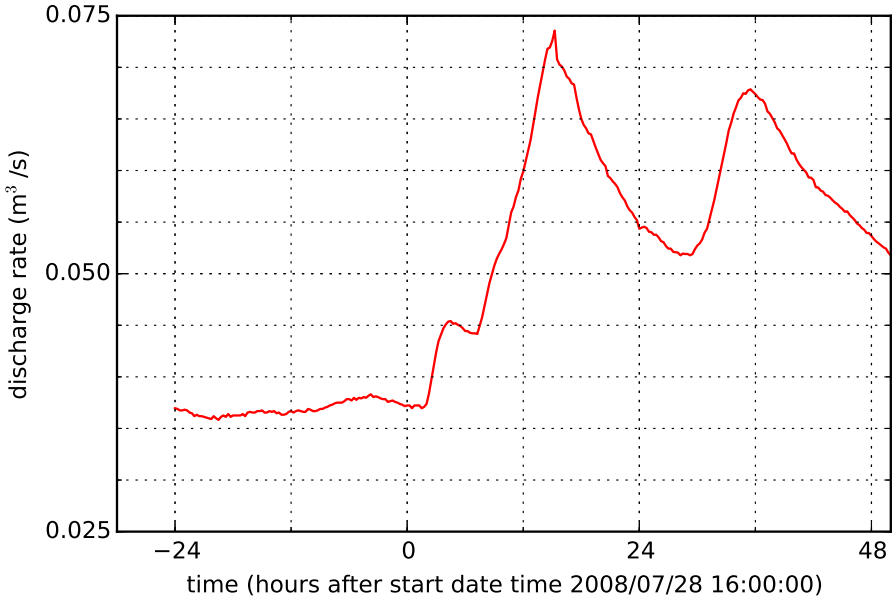
Figure 6. Measured rainfall rates at the CPEAK met station for a summer rainfall event: July 28-30, 2008.

Time [min]	Rainfall Rates [mm/h] ([m/s])	Time [min]	C2 Discharges [L/s]
120 – 240	$P_1 = 3.51 \quad (9.75 \times 10^{-7})$	255 – 285	$Q_1 = 45.4$
600 – 660	$P_{2a} = 3.24 \quad (9.00 \times 10^{-7})$	—	—
780 – 840	$P_{2b} = 4.59 \quad (1.28 \times 10^{-6})$	915 – 930	$Q_2 = 73.6$
1920 – 1980	$P_3 = 3.24 \quad (9.00 \times 10^{-7})$	2130 – 2145	$Q_3 = 67.9$

Table 1. Observed peak rainfall rates at CPEAK and corresponding peak discharges at the C2 basin outlet.

3.6 Model Component Selection and Setup

While TopoFlow includes numerous hydrologic process components, we deliberately chose to model a relatively simple situation where the dominant processes are surface flow, rainfall and infiltration. We chose a late summer rainfall event so that contributions from snowmelt could be neglected. We also chose the C2 subbasin, which is south-facing and almost free of permafrost. The near absence of permafrost, the relatively uniform soil type (Gilmore silt loam) and the selection of a rainfall event preceded by several days of no rainfall mean that the assumptions of the Green-Ampt infiltration model should be ap-



480 **Figure 7.** Volume flow rates measured at the outlet of the C2 basin for a summer rainfall event: July 28-30,
481 2008.

495 proximately satisfied. In addition, evaporation is considered to be negligible compared
496 to the rainfall and infiltration rates. Finally, the relatively steep slopes throughout the C2
497 basin mean that the kinematic wave method of channel flow routing should be appropriate
498 (i.e. no backwater effects, etc.) With these simplifications, we can focus on surface flow,
499 as described by Manning's formula, and the infiltration process physics included in the
500 Green-Ampt model.

501 ### 3.7 Choosing a Soil Water Retention Model for the Infiltration Process

502 In infiltration theory, there are four inter-related variables of interest that represent
503 3D scalar fields below the land surface, namely K, the *hydraulic conductivity*, v, the *ver-*
504 *tical component of the Darcy velocity*, θ, the *soil water content* and ψ the *pressure head*. In
505 order to create a mathematical model that can solve for these four variables, four equa-
506 tions are needed. Two equations are *conservation of mass* and *Darcy's Law*, and com-
507 bining them results in the well-known *Richards equation* for modeling the flow of water
508 through a porous medium (e.g. soil), driven by gravity as well as capillary suction. How-
509 ever, two additional equations are needed, and these are empirical relations of the form

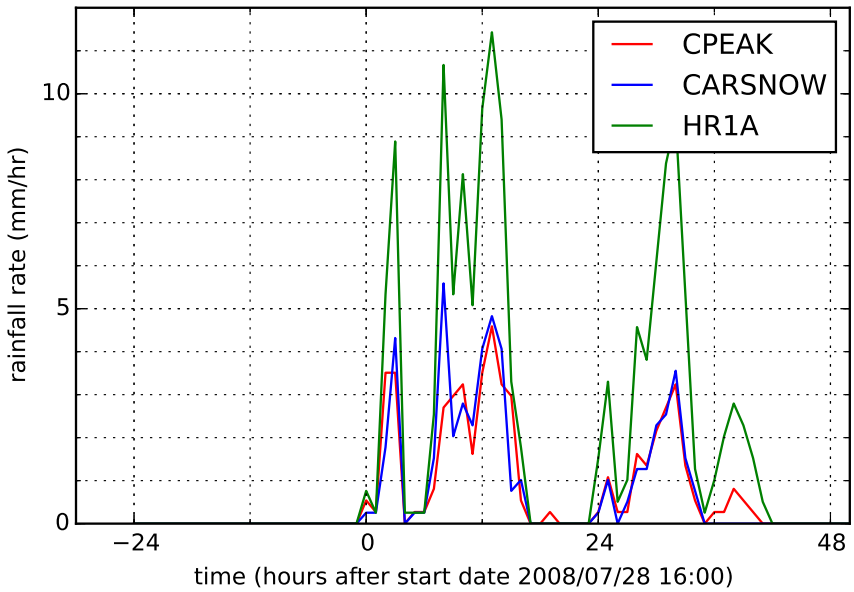


Figure 8. Measured rainfall rates at the CPEAK, CARSNOW and HR1A met stations for a summer rainfall event: July 28-30, 2008.

$K(\theta)$ and $\psi(\theta)$ known as *soil characteristic relations*, that allow K and ψ to be computed as functions of θ. *Brooks and Corey* [1964] proposed functional forms for $K(\theta)$ and $\psi(\theta)$ that depend on three parameters, namely $\psi_B < 0$, the *bubbling pressure*, and two model parameters η and λ, where $\eta = 2 + 3\,\lambda$. The parameters $\eta > 0$ and $\lambda > 0$ can be set to different values in order to provide good fits to observational data for the flow dynamics of different soil texture types. However, while their model has $K = K_s$ at saturation ($\theta = \theta_s$), it yields $\psi = \psi_B < 0$ instead of $\psi = 0$ at saturation. van Genuchten (1980) proposed an alternate pair of relations that also depend on three parameters, namely $\alpha_g < 0$, $m > 0$ and $n > 0$. While this model has $\psi = 0$ at saturation, its parameters are less physically meaningful and it has a complicated functional form for $K(\psi)$ [*Smith et al.*, 2002] (p. 21) that is difficult to integrate. (This makes it difficult to compute the parameter G in (8).)

Smith [1990] introduced a third pair of relations based on the Brooks-Corey model, which he called the *transitional Brooks-Corey* (TBC) model, that combines the benefits of the Brooks-Corey and van Genuchten models. *This is the soil water retention model that is used by every infiltration component in TopoFlow.* It introduces two new parameters $c > 0$ and ψ_A, such that the Brooks-Corey model is obtained in the limit as $c \to \infty$. For Smith's

TBC model, the soil characteristic relations are given by

$$K(\Theta_e) = K_s \, \Theta_e^{\left(\frac{\eta}{\lambda}\right)} \tag{1}$$

$$\psi(\Theta_e) = \psi_B \left[\Theta_e^{\frac{-c}{\lambda}} - 1\right]^{\frac{1}{c}} - \psi_A \tag{2}$$

At saturation, it has $K = K_s$ and $\psi = -\psi_A$, and one typically sets $\psi_A = 0$. The parameters η, λ and c are not independent, and in fact $\eta = 2 + 3\,\lambda$ and $c = \eta/\lambda = 2/\lambda + 3$ in this model. Since $\lambda > 0$, this implies that $\eta > 2$ and $c > 3$. Here, Θ_e is the *normalized soil water content*, defined as:

$$\Theta_e = \left(\frac{\theta - \theta_r}{\theta_s - \theta_r}\right) \tag{3}$$

and θ_r is the *residual water content*. Note that $\Theta_e(\theta_r) = 0$ and $\Theta_e(\theta_s) = 1$. For the TBC soil model, it is also possible to solve for $K(\psi)$ as

$$K(\psi) = K_s \, \{1 + [(\psi + \psi_A)/\psi_B]^c\}^{\left(\frac{-\eta}{c}\right)} \tag{4}$$

3.8 Green-Ampt Infiltration Model Component – Theory

The Green-Ampt infiltration model [*Green and Ampt*, 1911; *Smith et al.*, 2002] can be derived as a physically-based approximation to *Richards Equation*, which in turn is considered the best-available mathematical model for the process of infiltration. This approximation conserves the mass of water and incorporates a soil model (e.g. Brooks-Corey) but assumes that the initial soil moisture profile is uniform with depth and that lateral flow in the unsaturated zone can be neglected. It also assumes that there is a single, deep soil layer with uniform properties. Unlike the Richards equation model, the Green-Ampt model treats the variation of soil water content, θ, with depth below the land surface, z, as simple "piston flow" with a sharp wetting front — that is, with $\theta = \theta_s$ above the wetting front and $\theta = \theta_i$ below the wetting front.

Both the Green-Ampt and Smith-Parlange (three parameter) models of infiltration make use of what *Smith et al.* [2002] calls the *Infiltrability-Depth Approximation* or IDA. Instead of expressing the infiltration rate at the surface, v_0, as a function of time, t, the IDA instead expresses v_0 as a function of the *cumulative infiltration depth*, F, given by

$$F(t) = \int_0^t v_0(\tau)\, d\tau. \tag{5}$$

Note that $v_0 = dF/dt$. This change of independent variable (from t to F) provides a more robust treatment of the relevant boundary conditions and the transition between them. Ac-

cording to the Green-Ampt model, v_0 is given by

$$v_0 = \begin{cases} \min(P, f_c), & P > K_s \\ P, & P < K_s. \end{cases} \tag{6}$$

where f_c is called the *infiltrability* (or *infiltration capacity*) and represents the maximum possible infiltration rate that the soil will allow, given by

$$\begin{aligned} f_c &= K_i + [(K_s - K_i)(F + J)/F] \\ &= K_s + (J/F)(K_s - K_i). \end{aligned} \tag{7}$$

Here, $J = G(\theta_s - \theta_i)$, and G is the *capillary length scale* [*Smith et al.*, 2002], defined as

$$G = \frac{1}{K_s} \int_{-\infty}^{0} K(\psi)\, d\psi. \tag{8}$$

The Green-Ampt parameter, G, characterizes an initially dry soil with $K_i \ll K_s$.

For the TBC soil model used by TopoFlow, G can be computed by inserting (4) into (8), and for the typical case where $\psi_A = 0$, *Peckham* [2010] found the following closed-form expression for the resulting integral

$$G = -\psi_B \left[\frac{\Gamma(1 + 1/c)\, \Gamma[(\eta - 1)/c]}{\Gamma(\eta/c)} \right]. \tag{9}$$

Here $\Gamma(x)$ is the *Gamma function* and recall that $\psi_B < 0$. Since $\eta > 2$ and $c > 3$, it can be shown that $0 < G < -2\psi_B$. While *Smith et al.* [2002] (p. 71) gave a closed-form expression for G in the case of the standard Brooks-Corey model, namely $G = -\psi_B \eta/(\eta - 1)$, this result for the TBC model appears to be new. It also yields the Brooks-Corey expression for G in the limit as $c \to \infty$, but can give very different values for smaller values of c. Note that while Mathematica has powerful symbolic integration capabilities and can evaluate the G integral for several specific values of c (e.g. $c = 1$, $c = 3/2$, $c = 2$), it does not provide the general expression in (9). Mathematica can also be used to check (9) by computing the G integral numerically for arbitrary choices of c and η.

3.9 Matching Rainfall Peaks to Hydrograph Peaks with Green-Ampt

Assuming no other gains from snowmelt or baseflow, and no losses from evaporation, the runoff available to generate discharge is given by $R = (P - v_0)$, the difference between the rainfall rate and the surface infiltration rate. In the special case where the soil is saturated at the start of a rainfall event, we clearly have $K_i = K_s$, as well as $\theta_i = \theta_s$, in view of (1). Equation (7) then reduces to $f_c = K_s$ and equation (6) simplifies to

$$v_0 = \begin{cases} K_s, & \text{if } P > K_s \\ P, & \text{if } P < K_s. \end{cases} \tag{10}$$

574 In this case, the Green-Ampt model only allows rainfall rates to generate nonzero runoff
575 when $K_s < P$, that is

$$R = \begin{cases} P - K_s, & \text{if } P > K_s \\ 0, & \text{if } P < K_s. \end{cases} \tag{11}$$

576 In the more general case when $K_i \ll K_s$, Green-Ampt predicts that the runoff rate
577 associated with a given rainfall peak (at any point in the basin) should be given by (see
578 (7))

$$R_k = \max\left\{P_k - \left(K_s + \frac{C}{F_k}\right), 0\right\}, \tag{12}$$

579 where $C = G(\theta_s - \theta_i)(K_s - K_i)$, and $F_k < F_{k+1}$ are the values of the cumulative infil-
580 trated depth at the time a rainfall peak occurs. The fact that infiltration rates are higher at
581 the beginning of a rainfall event (and F is a nondecreasing function) means that a rain-
582 fall peak of a given magnitude will be less strongly reflected in the basin hydrograph if
583 it occurs toward the beginning of a rainfall event than if it instead occurs at a later time.
584 This can be clearly seen in the C2 measured hydrograph, where the first and fourth rain-
585 fall peaks are comparable in magnitude but the first hydrograph peak is much smaller than
586 the third.

587 In addition, the total contributing area of the C2 basin is $A = 4.84\,[\text{km}^2]$ and the
588 longest channel in the C2 basin is 2.93 [km]. Assuming a mean flow velocity of roughly
589 1 [m/s] in the channels, then once water reaches a channel, most water should reach the
590 basin outlet in less than half an hour.

591 At a given point in the C2 basin, runoff will not occur unless the rainfall rate, P, ex-
592 ceeds the infiltrability, $f_c = K_s + C/F$, and therefore it must exceed K_s. It follows that in
593 order for the hydrograph peaks to have been generated by the corresponding rainfall peaks
594 (although routed to the basin outlet by the channel network), we must have $P_1 > K_s$, $P_2 >$
595 K_s and $P_3 > K_s$. To satisfy all three inequalities, we must have $K_s < 9.0 \times 10^{-7}$ [m/s].
596 Note, however, that the rainfall rates were measured at one-hour intervals while the model
597 timesteps for both the infiltration and channel routing components were 2 seconds and the
598 hydrograph at the outlet was measured at a time interval of 15 minutes. The actual rainfall
599 peak values were likely higher (and no less than) the recorded peak values. If the range
600 estimate of $4.23 \times 10^{-6} < K_s < 1.41 \times 10^{-5}$ [m/s] given by *Rieger et al.* [1972] is ac-
601 curate, then the instantaneous rainfall rates must have been higher than the low end of this
602 range. But since these higher values weren't measured, it would appear to be necessary

to use values of K_s in the model at least *10 times smaller* in order to match the observed hydrograph.

3.10 Model Results and Analysis

We use Equation (7) to guide our search for a good combination of parameters for the watershed. K_s is the offset that determines which pulses in the precipitation data are completely suppressed, since when $P < K_s$, the model indicates that all water is infiltrated. It is important to note, then, that the sampling rate for the precipitation, which in the case of this data set is one hour, is critical to being able to utilize a K_s value that reflects the observed value. Since the precipitation sampling rate is low, we need to utilize a much lower K_s value than the reported value to preserve the peaks. Figure (9) shows the discharge at the tracked outlet in C2 for different values of K_s when $\theta_i = \theta_s$ so that $f_c = 0$. We notice that it is the second peak that is attenuated the most by an increase in K_s. Noting that there is a shoulder in the observed discharge output in Figure (7) indicates that the second peak is not attenuated and in fact looks to have approximately the same amplitude as the first peak. Therefore, selecting a K_s value for which a 1:1 ratio may be maintained between these two peaks is desirable; a K_s value above 8.0×10^{-7} seems to completely suppress the second peak and is thus not desirable.

For any given K_s value, introducing the second term in Equation (7) by decreasing θ_i will further attenuate the peaks. The amount of attenuation and which peaks are attenuated most is determined by the weighting factor, G. Figure (10) illustrates how increasing G preferentially attenuates to first peaks more than latter peaks. The first three peaks all experience attenuation while the fourth peak is relatively unaffected.

With this understanding of parameter adjustment on modeled output, it is possible to find an optimum combination of parameters that will yield a reasonable output volume. Attempting to achieve the correct volume output while maintaining all the peaks results in incorrect peak response matching with the third peak being much larger than the other three peaks. Allowing suppression of the first two peaks and attempting to optimize the ratio of the third and fourth peak heights as well as the total volume discharge yields the result in Figure (11). In order to obtain this simulated output, several adjustments were made to the input data. First, the precipitation data was resampled at 2 second intervals (from the original 1 hour), so that instead of a single pulse of precipitation at the begin-

ning of each hour, the precipitation was added to the model in smaller increments. This resulted in a better overall precipitation volume estimate (within 100 m^3) in comparison to that modeled with the hour-interval precipitation (off by more than 6000 m^3). Additionally, the default surface drag effect from the Manning n coefficient was increased by a factor of 6. Without this drag effect, each pulse response occurs sooner and decays faster.

Figure (12) illustrates the way the model responds to a given input. It can be seen that the model behaves like a simple integrator upon being fed an impulse. It is the amount of cumulative precipitation during a short period of time that determines whether the model responds to a rain event, not the number of peaks during an event. Effectively, the precipitation event mimics a sequence of two step functions and the outputs show how the motion of water through the landscape behaves similarly to the discharging of a capacitor in an integrator circuit.

Lastly, in Figure (13), we show an example of the influence of increased surface drag and 4-peak response on the simulated output. From this example, we can see the importance of these two effects in smoothing out the discharge curve.

A detailed, reproducible workflow of how the results in this section were obtained is available on GitHub as an iPython notebook.

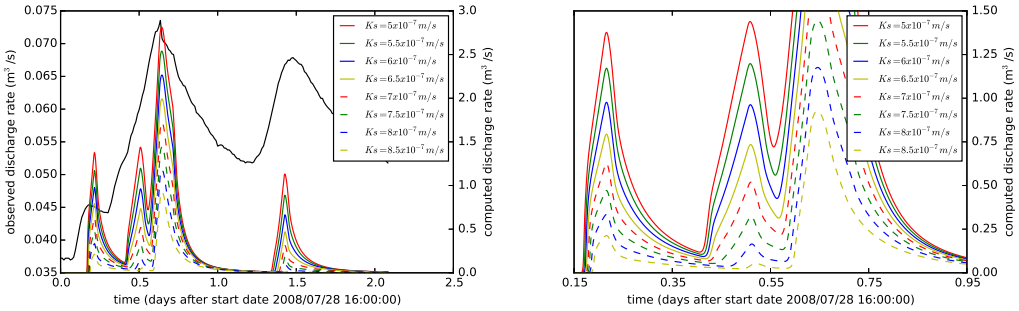


Figure 9. Effect of varying K_s with $\theta_i = \theta_s$ on precipitation peak response suppression. Notice that the observed and modeled hydrographs have different y-axes.

4 Conclusions and Recommendations

This paper has attempted to document the entire workflow of a spatial hydrologic modeling study to the extent that it can be easily reproduced and extended by other scien-

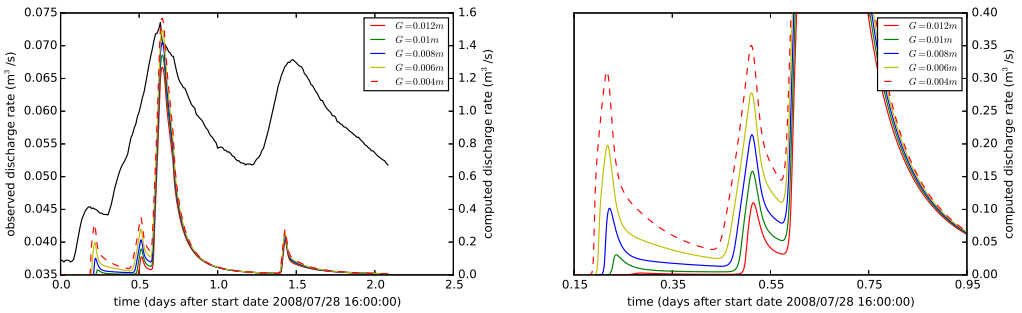


Figure 10. Effect of varying G with $K_s = 7.7 \times 10^{-7}$ m/s and $\theta_i = 0.01$ on precipitation peak response suppression. Notice that the observed and modeled hydrographs have different y-axes.

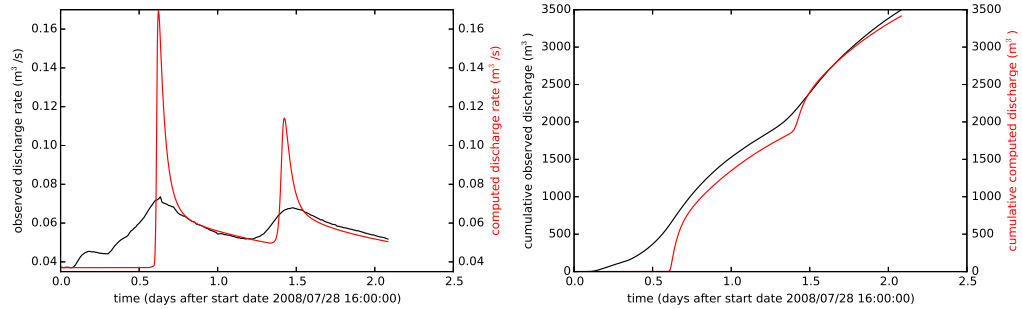


Figure 11. Best match result in terms of total volume discharge when $K_s = 4.5 \times 10^{-7}$ m/s, $\theta_i = 0.17$, $G = 1.1$ m. The precipitation data was resampled for 2 second intervals using a uniform distribution, and the drag factor due to the Manning n coefficient was multiplied by a factor of 6.

tists. In support of reproducibility, the entire TopoFlow modeling toolkit, including components, utilities and framework is:

- open source and accessible on GitHub (MIT license)
- version controlled
- easily installed as a Python package
- citable with a DOI
- extensible by adding new components
- runnable with the CSDMS, EMELI and EMELI-Web frameworks

In addition, every TopoFlow model component has:

- object-oriented source code that takes advantage of inheritance

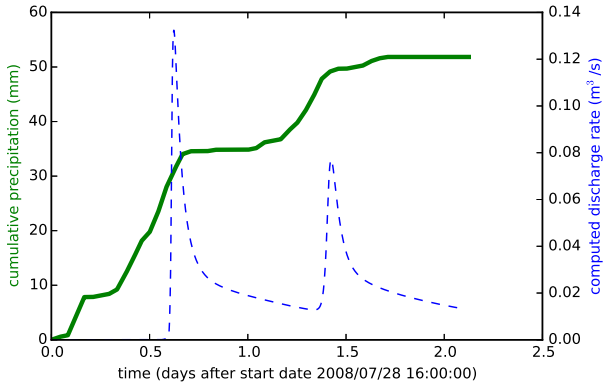


Figure 12. Best match result in terms of total volume discharge when $K_s = 4.5 \times 10^{-7}$ m/s, $\theta_i = 0.17$, $G =$ 1.1 m plotted against the cumulative precipitation.

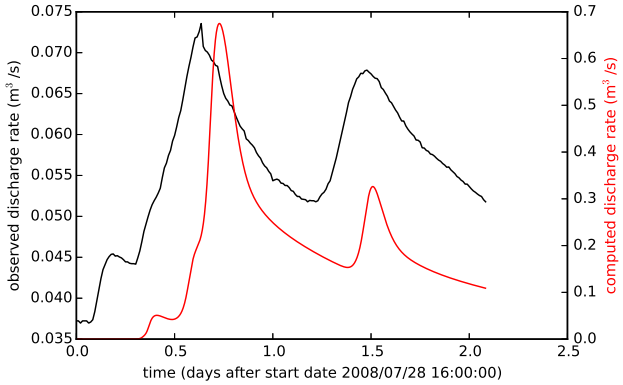


Figure 13. Illustrative example of the effect of increased surface drag (factor of 30 vs. 6) on the curve shape of the modeled discharge. This model result yielded a volume discharge ten times higher than observed, but shows the features that could be captured for higher precipitation data sampling rates. Notice that the observed and modeled hydrographs have different y-axes.

- a Basic Model Interface (BMI) to support plug-and-play reuse in frameworks
- all input and output variables mapped to CSDMS Standard Names
- additional, standardized metadata at OntoSoft portal (CSDMS section)
- its own HTML help page that describes all variables and all equations used
- its own easy-to-read and edit configuration file (read at startup)
- a graphical user interface (GUI): i.e. WMT for Python & TopoFlow-IDL for IDL
- the ability to run as a web service with EMELI-Web

- the ability to save its output to standard-format NetCDF files or generic binary grids. (These NetCDF files can be viewed with many visualization software toolkits, such as *VisIt* [2016].)

New components can easily be created by copying and then editing the Python source code of existing components. Components of a given process type can also inherit many capabilities from existing base classes. In addition, *BMI-to-Framework-X* adapters are under development by the EarthCube Earth System Bridge project that will allow components to run in several other modeling frameworks. These adapters are available in the *BMI-Forum* [2016] on GitHub.

For the hydrologic modeling study, the procedure for obtaining input data was fully described, and open-source Python scripts for preparing input data for TopoFlow were provided. This included links to iPython notebooks that can be used to: (1) prepare input data, (2) reproduce the model analysis and (3) recreate many of the figures.

While used mainly as a vehicle for highlighting the issue of reproducibility and best practices, and while only tapping a small fraction of the capabilities of the TopoFlow model toolkit, the hydrologic modeling study for the C2 watershed led to interesting results that could be pursued further in several different directions. We showed how the features of a hydrograph resulting from a late summer storm could be interpreted using the Green-Ampt infiltration model. However, this study also clearly illustrated how insufficient temporal resolution in rainfall rate measurements suppresses the magnitudes of rainfall rate peaks, and how this prevents the model from matching observations unless parameters such as K_s are adjusted away from observed values. In this analysis, a robust, but lesser-known soil water retention model used by TopoFlow — known as transitional Brooks-Corey — was also highlighted, and a new, closed-form expression for G was provided.

Acknowledgments

We would like to thank Matt Nolan, who initiated the collaboration that led to TopoFlow, Larry Hinzman, who helped to guide and provide funding for its development, as well as the entire Arctic hydrology team at WERC (Water and Environmental Research Center), at the University of Alaska, Fairbanks. S. Peckham is the primary author of both TopoFlow, all versions of which are free and open-source, and the product RiverTools 4.0, which is

sold commercially by Rivix, LLC. To avoid any perceived conflict of interest, this paper has described workflows that do not require RiverTools. This work was partially funded by NSF grants ICER 1440332 (GeoSoft), 1440333 (GeoSemantics) and PLR 1503559.

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 uses, 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

Kinematic Wave Method. Basic method for modeling steady, uniform flow in open channels. Mass conservation equation and momentum calculation includes friction and gravity terms only. Surface, bed and energy slope are equal.
(*channels_kinematic_wave.py* inherits from *channels_base.py*).

Diffusive Wave Method. Same as kinematic, except uses the pressure gradient that is induced by a water-depth gradient in the downstream direction. Benefit: 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. Most comprehensive method for modeling channel flow. Includes momentum flux terms, as well as friction, gravity and pressure gradient.
(*channels_dynamic_wave.py* inherits from *channels_base.py*).

Evaporation Process Components

Priestley-Taylor Method. Empirical equation for evapo-transpiration. Useful 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. Describe the component here.
(*evap_energy_balance.py* inherits from *evap_base.py*).

Infiltration Process Components

Green-Ampt Method. Infiltration modeled as a wetting front that moves down into the soil, separating saturated and dry soil with a sharp boundary. Matric suction and gravity pull water into soil. (*infil_green_ampt.py* inherits from *infil_base.py*).

Smith-Parlange (3-parameter) Method. Potential infiltration rate is a function of effective saturated hydraulic conductivity, effective net capillary drive and cumulative infiltration depth. (*infil_smith_parlange.py* inherits from *infil_base.py*).

Richards Equation (1D) Method. Nonlinear partial differential equation method that represents flow in unsaturated porous media. Obtained from Darcy's law and a continuity requirement. Caution: computationally expensive, may not converge. (*infil_richards_1D.py* inherits from *infil_base.py*).

Snowmelt Process Components

Degree-Day Method. Total daily melt is directly proportional to the temperature gradient between the mean ambient temperature and the base temperature. 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. Based on energy balance of energy absorbed and lost. Dependent on properties such as ambient and dew point temperatures, wind speed, snow albedo, rainfall, and incident solar radiation. (*snow_energy_balance.py* inherits from *snow_base.py*).

Meteorology Components

Meteorology. Meteorology variables computed using the equations given by *Dingman* [2002] (Appendix E), supplemented with an optical mass formulation. Includes shortwave and longwave radiation calculators based on celestial mechanics. (*met_base.py*).

Saturated Zone Components

Darcy Layers Method. Darcy's law for movement of fluid in a porous medium. Capability to handle multiple, shallow soil layers. Assumption: Water table has the same gradient as the land surface. (*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

808 smoothly-varying slopes along channel streamlines is important when using the kinematic
809 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 two 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* and *django*, but not *cfunits*).

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. Install the Unidata UDUNITS-2 library. This is needed by a Python package called *cfunits* that the EMELI framework uses to perform automatic unit conversion when needed. It can be downloaded from: *http://www.unidata.ucar.edu/downloads/udunits/index.jsp*. On MacOS and Linux, it is installed in */usr/local/bin/udunits2*. The easiest way to install it on a Mac is to first install the *homebrew* package manager from *https://brew.sh*, and then type *brew install udunits* in a terminal window. (Note that *brew* requires *xcodebuild*, which should already be installed.)

Step 4. Install the *cfunits* Python package. This package requires the *netCDF4* and *numpy* packages installed previously. First, download the package from: *https://bitbucket.org/cfpython/cfunits-python/downloads* and unzip it. Then, type

```
$ cd cfunits-1.1.4
$ python setup.py install
```

in a terminal window.

Step 5. 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 6. 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 entire TopoFlow package folder (TopoFlow_3.5) someplace convenient (e.g. your home directory or Dropbox folder). This 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 7. 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 8. To run TopoFlow, you need a CFG file for every component you want to use in the CFG directory. You also need an *outlet file* (extension *outlets.txt* and an EMELI provider file. 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
```

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*).

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 *Discharge 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

BMI-Forum (2016), Basic Model Interface (BMI) Forum on GitHub, https://github.com/bmi-forum.

Bolton, W. R. (2006), Dynamic modeling of the hydrologic processes in areas of discontinuous permafrost, Ph.D. thesis, University of Alaska, Fairbanks, Dept. of Civil Engineering, 163 pp.

Bolton, W. R., L. D. Hinzman, and K. Yoshikawa (2000), Stream flow studies in a watershed underlain by discontinuous permafrost, in *Water Resources in Extreme Environments, Proceedings*, edited by D. L. Kane, pp. 31–36, American Water Resources Association, Anchorage, AK.

Bolton, W. R., L. D. Hinzman, and K. Yoshikawa (2004), Water balance dynamics of three small catchments in a sub-arctic boreal forest, in *Northern Research Basins Water Balance Workshop Proceedings*, vol. 290, edited by D. L. Kane and D. Yang, pp. 213–223, IAHS Publication, Victoria, Canada.

Brooks, R., and A. Corey (1964), Hydraulic properties of porous media, *Hydrology Papers*, Colorado State University, Fort Collins, Colorado.

Coe, J. A., D. A. Kinner, and J. W. Godt (2008), Initiation conditions for debris flows generated by runoff at Chalk Cliffs, central Colorado, *Geomorphology*, *96*, 270–297, doi:10.1016/j.geomorph.2007.03.017, http://dx.doi.org/10.1016/j.geomorph.2007.03.017.

CSDMS-BMI (2016), The Basic Model Interface (BMI), Online documentation, CSDMS, http://bmi-python.readthedocs.io/en/latest/.

CSDMS-WMT (2016), Web Modeling Tool (WMT), https://csdms.colorado.edu/wmt/.

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

Edwards, P. N. (2010), *A Vast Machine: Computer Models, Climate Data, and the Politics of Global Warming*, MIT Press, Cambridge, MA.

Edwards, P. N., M. S. Mayernick, A. L. Batcheller, G. C. Bowker, and C. L. Borgman (2011), Science friction: Data, metadata, and collaboration, *Social Studies of Science*, *41*, 667, doi:10.1177/0306312711413314, http://dx.doi.org/10.1177/0306312711413314.

EMELI-Web (2016), Experimental Modeling Environment for Linking and Interoperability, Web service version, http://ecgs.ncsa.illinois.edu/emeli-web/.

Garijo, D., P. Alper, K. Belhajjame, O. Corcho, Y. Gil, and C. Goble (2013), Common motifs in scientific workflows: An empirical analysis, *Future Generation Computer Systems*, *36*, 338–351, doi:10.1016/j.future.2013.09.018, http://dx.doi.org/10.1016/j.future.2013.09.018.

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.

GSN (2017), Geoscience Standard Names (GSN) ontology, http://www.geostandardnames.org.

Hannon, M. T., J. P. M. Syvitski, and A. J. Kettner (2008), Hydrologic modeling of a tropical river delta by applying remote sensing data: The Niger Delta and its distributaries, *American Geophysical Union, Fall Meeting Abstract, #H53B-1050*.

Henderson, F. M. (1966), *Open Channel Flow*, Macmillan Publishing Co., New York.

Hinzman, L. D., D. L. K. DL, C. Benson, and K. Everett (1996), Chapter 6: Energy balance and hydrological processes in an Arctic watershed, in *Landscape Function and Disturbance in Arctic Tundra*, *Ecological Studies*, vol. 20, edited by J. Reynolds and J. Tenhunen, chap. 6, pp. 131–154, Springer-Verlag, Berlin, doi:10.1007/978-3-662-01145-4_6, http://dx.doi.org/10.1007/978-3-662-01145-4_6.

Hinzman, L. D., D. Goering, and D. L. Kane (1998), A distributed thermal model for calculating temperature profiles and depth of thaw in permafrost regions, *Journal of Geophysical Research — Atmospheres*, *103(D22)*, 28,975–28,991.

Hutton, C., T. Wagener, J. Freer, D. Han, C. Duffy, and B. Arheimer (2016), Most computational hydrology is not reproducible, so is it really science?, *Water Resources Research*, *52*, 7548–7555, doi:doi:10.1002/2016WR019285, http://dx.doi.org/10.1002/2016WR019285.

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 Knowl-*

edge, pp. 301–310, Auto-Carto VII, Washington, D.C., http://mapcontext.com/autocarto/proceedings/auto-carto-7/.

Jiang, P., M. Elag, P. Kumar, S. D. Peckham, L. Marini, and L. Rui (2017), A service-oriented architecture for coupling web service models using the basic model interface (bmi), *Environmental Modeling and Software (in press)*.

Liljedahl, A. (2008), Master's thesis, Master's thesis, University of Alaska, Fairbanks.

OntoSoft-CSDMS (2016), OntoSoft Software Repository for CSDMS, http://csdms.ontosoft.org/#list.

Peckham, S. D. (2009a), Chapter 25: Geomorphometry and spatial hydrologic modelling, in *Geomorphometry: Concepts, Software, Applications*, *Developments in Soil Science*, vol. 33, edited by T. Hengl and H. I. Reuter, pp. 579–602, Elsevier, http://dx.doi.org/10.1016/S0166-2481(08)00025-1.

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. (2010), TopoFlow Soil Properties Page, https://csdms.colorado.edu/wiki/Model_help:TopoFlow-Soil_Properties_Page.

Peckham, S. D. (2014a), The CSDMS Standard Names: Cross-domain naming conventions for describing process models, data sets and their associated variables, in *Proceedings of the 7th Intl. Congress on Env. Modelling and Software*, edited by D. P. Ames, N. W. T. Quinn, and A. E. Rizzoli, p. Paper 12, International Environmental Modelling and Software Society (iEMSs), San Diego, CA, http://scholarsarchive.byu.edu/iemssconference/2014/Stream-A/12/.

Peckham, S. D. (2014b), EMELI 1.0: An experimental smart modeling framework for automatic coupling of self-describing models, in *Proceedings of HIC 2014: 11th International Conference on Hydroinformatics*, CUNY Academic Works, New York, NY, http://academicworks.cuny.edu/cc_conf_hic/464/.

Peckham, S. D. (2016), TopoFlow Python package on GitHub, open-source, https://github.com/peckhams/topoflow.

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.

Peckham, S. D., E. W. H. Hutton, and B. Norris (2013), A component-based approach to integrated modeling in the geosciences: The design of CSDMS, *Computers & Geosciences, special issue: Modeling for Environmental Change*, *53*, 3–12, http://dx.doi.org/10.1016/j.cageo.2012.04.002.

Pohl, S., P. Marsh, C. Onclin, and M. Russell (2009), The summer hydrology of a small upland tundra thaw lake: implications to lake drainage, *Hydrological Processes*, *23*, 2536–2546, Special Issue: Thematic set of subsurface, surface and atmospheric processes in cold regions hydrology.

Rieger, S., C. E. Furbush, D. B. Schoephorster, H. Summerfield Jr., and L. C. Geiger (1972), Soils of the Caribou-Poker Creeks Research Watershed, Interior Alaska, *Tech. Rep. 236*, U.S. Army Corps of Engineers, Cold Regions Research and Engineering Lab (CRREL), Hanover, New Hampshire, prepared by the U.S. Department of Agriculture, Soil Conservation Service.

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

Schramm, I. (2005), Hydrologic modeling of an arctic watershed, Alaska, Ph.D. thesis, University of Potsdam, Germany.

Smith, R. E. (1990), Analysis of infiltration through a two-layer soil profile, *Soil Science Society of America Journal*, *54*(5), 1219–1227.

Smith, R. E., K. R. J. Smettem, P. Broadbridge, and D. A. Woolhiser (2002), *Infiltration Theory for Hydrologic Applications*, Water Resources Monograph 15, American Geophysical Union, doi:http://dx.doi.org/10.1029/WM105.

VisIt (2016), Visit visualization software, Lawrence Livermore National Laboratory, https://wci.llnl.gov/simulation/computer-codes/visit.

Zhang, Z., D. L. Kane, and L. D. Hinzman (2000), Development and application of a spatially-distributed arctic hydrological and thermal process model (ARHYTHM), *Hydrological Processes*, *14*(6), 1017–1044, doi:10.1002/(SICI)1099-1085(20000430)14:6<1017::AID-HYP982>3.0.CO;2-G.