https://doi.org/10.5194/gmd-2019-179
Preprint. Discussion started: 9 August 2019

MetSim v2.0.0: A flexible and extensible framework for the estimation and disaggregation of meteorological data

Andrew R. Bennett[1], Joseph J. Hamman[2], and Bart Nijssen[1]
[1]Department of Civil and Environmental Engineering, University of Washington
[2]Climate and Global Dynamics Laboratory, National Center for Atmospheric Research
Correspondence to: Bart Nijssen (nijssen@uw.edu)

Abstract. MetSim is a freely available, open source Python based model for simulation and disaggregation of meteorological variables with applications in the environmental and Earth sciences. MetSim can be used to generate spatially distributed sub-daily timeseries of incoming shortwave radiation, outgoing longwave radiation, air pressure, specific humidity, relative humidity, vapor pressure, precipitation, and air temperature given daily timeseries of minimum temperature, maximum temperature, and precipitation. Based on previously developed algorithms, we demonstrate that MetSim is able to closely reproduce their results while providing a number of advantages and improvements. We implemented automated testing to decrease errors during development and to improve reproducibility, modularized the algorithms to allow for extensions and modifications, and implemented robust single and multi-node parallelism. We describe the overall architecture, algorithms, and capabilities of MetSim by describing its four major modules. These are split into a model driver, solar geometry module, meteorological simulation module, and temporal disaggregation module. We also describe the available options and parameters that MetSim exposes to its users and analyze MetSim's scalability for large datasets.

1 Introduction

Hydrometeorological modeling is concerned with domains that have uncertain or unknown boundary conditions (Beven and Cloke, 2012). For example, incoming shortwave radiation, longwave radiation, and humidity have sparse and irregular sensor locations with varying record lengths and observation intervals. Further, even when such quantities are measured it is often at a daily resolution, while many environmental models require finer temporal resolution for simulation. To provide closure to the model equations in such circumstances we must be able to provide estimates for these quantities at the appropriate temporal resolution.

In this paper we describe MetSim v2.0.0 (hereafter just MetSim), a software package and computational tool that provides functonality to address both estimating missing quantities as well as providing finer temporal resolutions. We refer to the problem of estimating quantities of interest as forcing generation, simulation, or estimation. We refer to the problem of taking daily values and estimating subdaily values as temporal disaggregation or forcing disaggregation.

MetSim is a Python-based meteorological simulator and forcing disaggregator designed for hydrological modeling and climate applications, although its functionality is relevant to other applications as well. We have based MetSim on methods from the Mountain Microclimate Simulator (MTCLIM) and the forcing preprocessor that was built into the Variable Infiltration Capacity (VIC) hydrological model version 4 (Bohn et al., 2013; Thornton and Running, 1999; Liang et al., 1994). MetSim provides a modern workflow, building upon previous tools by improving performance, adding new IO routines, allowing for exact restarts, and providing an extensible architecture which can incorporate new features.

Before the release of VIC model version 5 (VIC-5; Hamman et al., 2018), the VIC model (Liang et al., 1994; Cherkauer et al., 2003) allowed model setups in which the user provided only a small subset of the required input forcing data (daily minimum temperature, daily maximum temperature, daily precipitation, as well as daily wind speed) and then used internal routines to generate additional forcing variables (shortwave radiation, longwave radiation, humidity, and surface air pressure). These algorithms were based on MTCLIM (Thornton and Running, 1999) and others (e.g., Prata, 1996; Deardorff, 1978). Further, VIC allowed for the input of this subset of forcing variables at a daily time step even if the model was configured to run at a shorter time step. The VIC internal preprocessor automatically disaggregated the daily inputs to the required subdaily model time step.

Decoupling this internal preprocessor functionality from VIC has several advantages. MetSim's functionality can easily be adapted to estimate forcings for other modeling frameworks and for other environmental applications. To facilitate a more general use, we have implemented the ability to run on arbitrary spatial configurations, as opposed to the forced latitude-longitude grid that was used in the VIC preprocessor. With this in mind we adopt the term "cell" for a single spatial location to be processed (following the commonly used "grid cell").

Another advantage of making MetSim a standalone tool was the opportunity to rewrite it in Python, rather than the C implementation of the VIC preprocessor. By doing so, we are able to take advantage of high level tools which allow us to ignore the implementation details of many complex operations such as the management of parallelism, IO, and optimization (e.g., Rocklin, 2015; Hoyer and Hamman, 2017; Lam et al., 2015).

Furthermore, the dynamic nature of Python makes it easier to both decompose and extend MetSim. All of the individual components and functions that MetSim provides can easily be imported into other Python programs and used directly or interactively. MetSim provides both a command-line and programmatic interface.

In section 2 we describe the overall architecture, algorithms, and options implemented in MetSim. Following this, we describe the various options and parameters that MetSim exposes to the user (section 3). In section 4 we describe our use of modern software development workflows to improve reproducibility and automated testing procedures that reduce the risk of introducing new bugs into MetSim during the development lifecycle. We demonstrate some of MetSim's capabilities by comparing its output to the VIC version 4 preprocessor output and quantify performance characteristics via a scaling study in section 5.

https://doi.org/10.5194/gmd-2019-179
Preprint. Discussion started: 9 August 2019

2 Architecture

MetSim's overall structure follows the common design of a model driver which coordinates high level operations and which delegates computation to several modules to do the bulk of the work. MetSim has three of these top-level modules for solar geometry, meteorological simulation, and temporal disaggregation. We discuss each of these three modules along with the model driver and parallel implementation here.

2.1 Model Driver

The model driver is the main interface that users interact with. It is referenced as the `MetSim` object and provides the entry point to both command-line and scripting usage. In both cases four sources of input are used to set up the driver for simulation. These are a plain text configuration file (in INI format), a file describing the spatial extent of the simulation, a file providing the model state at the start of the simulation to enable exact restarts, and finally, the input data files. A schematic of the interaction between the `MetSim` object, the input, and modules is shown in Figure 1.

The configuration is specified by a plain text file with a simple format which contains the options, parameters, and file paths required to run the driver. As a part of the offial MetSim code base and documentation we provide several sample configuration files. We will describe the available options and their respective impacts in section 3.

The remainder of the inputs required by MetSim are data sources. We refer to them as the domain file, state file, and input forcing file(s). The domain file contains information about the spatial extent of the simulation, elevation information for each cell (used for the calculation of surface air pressure), and a mask variable that defines which cells to run, allowing users to run subsets of the given input forcing files. Additional spatially-varying parameters for the simulation can be provided in the domain file, depending on the options that are chosen.

The state file contains a 90-day record of the daily minimum temperature, maximum temperature, and precipitation for the dates preceeding the start of the simulation. This history is used to calculate seasonal averages of precipitation and daily temperature range, which are used by the MTCLIM algorithms (Thornton and Running, 1999) as part of simulation portion of the MetSim run. The seasonal daily temperature range is used to estimate cloud cover, while seasonal precipitation is used to estimate dew point temperature.

The input forcing file contains the meteorological data, which must include the daily minimum temperature, maximum temperature, and precipitation for each cell that is to be run (specified from the mask variable of the domain file). Optionally a wind component can be provided for disaggregation. MetSim can use NetCDF (Rew and Davis, 1990), ASCII, or VIC4 binary formatted input forcing files, and writes NetCDF files with associated metadata. The separation of the IO routines makes it relatively straightforward for additional IO formats to be implemented as extensions.

The domain file is read first and is used to specify the spatial arrangement of the state and input forcing datasets. The domain is then split into chunks, whose sizes are specified by the user (see section 3 for more information). Additionally there are command line options for splitting the run into temporal chunks, similar to how chunking in space occurs. Users can specify whether they want to read in the entire forcing record at once or to group by fixed time intervals such as year by year. The way in which these individual jobs are processed depends on how the user chooses to handle parallelism. MetSim implements several options for parallelism, which are primarily managed by the Dask library (Rocklin, 2015). These different parallelism options are described in section 2.4. No matter which parallelism option is chosen, the jobs are processed in a similar fashion.

For each job, the input forcings are read into memory and merged with the elevation data from the domain file. Finally, the state file is read and seasonal statistics are calculated. MetSim then proceeds to simulate the remaining meteorological quantities, and if required disaggregates them to the specified time resolution.

During the meteorologic simulation or, if specified, temporal disaggregation, output data from each processed cell is written to disk. We have implemented IO locks that are managed by the main model driver to coordinate write operations by each of the parallel workers. The meteorologic simulation process is carried out in three steps: solar geometry, meteorologic simulation, and (optionally) temporal disaggregation.

2.2 Solar Geometry and Meteorologic Simulation

The meteorologic simulation process starts with the calculation of daily potential radiation, daylength, and transmittance using the algorithms described in Whiteman and Allwine (1986) as implemented in MTCLIM (Thornton and Running, 1999). Additionally, the fraction of daily radiation that is received at the top of atmosphere during each 30 second interval is calculated for each day-of-year. These values are later used in the disaggregation routines. The default implementation of the solar geometry calculations does not account for slope or aspect. That is, we assume that each cell can be represented as a flat, horizontal plane.

The data from the solar geometry module is fed to the meteorology simulation module along with the input forcings. MetSim implements the estimation methods discussed in Bohn et al. (2013), as originally implemented in MTCLIM (Thornton and Running, 1999). We refer the reader directly to these earlier publications for specific estimation details, which are not the focus of this paper. From this data, the values for daily mean temperature, shortwave radiation, vapor pressure, and potential evapotranspiration are estimated.

2.3 Temporal Disaggregation

If disaggregation to shorter time steps is configured, the data is passed from the meteorology simulation module to the disaggregation module. MetSim implements several variable-specific disaggregation routines. Bohn et al. (2013) provides a further description and evaluation of these algorithms. Here we briefly mention the disaggregation procedures for completeness, but no substantial changes were made to the earlier algorithms.

Shortwave is disaggregated by multiplying the total daily shortwave by the fraction of radiation received in a given timestep (provided by the solar geometry module). This calculation is corrected for cloud cover by assuming constant transmissivity throughout the day (which is calculated in the meteorological simulation module). Temperature is disaggregated by estimating the time at which the daily maximum and daily minimum temperatures occur. These are chosen so that the daily minimum temperature occurs at sunrise and the daily maximum temperature occurs at a fixed time during the day (which is configurable by the user as a parameter in the configuration file if desired). Then a Hermite polynomial interpolation is used to obtain the full

https://doi.org/10.5194/gmd-2019-179
Preprint. Discussion started: 9 August 2019

temperature timeseries at sub-daily time steps. Vapor pressure is disaggregated by linearly interpolating between the saturation vapor pressure values calculated based on the daily minimum temperature and that are assumed to occur at the time of the daily minimum temperature. An additional correction is made to ensure that the vapor pressure at any given time step does not exceed the saturation vapor pressure, which is calculated directly from the disaggregated temperature timeseries. Air pressure is disaggregated by using the disaggregated temperature as well as the elevation data provided by the domain file. Both specific and relative humidity are then disaggregated using the disaggregated temperature and air pressure time series. If provided, wind speed is disaggregated, but is assumed to be constant throughout the day.

As part of the model configuration, the user can select from a number of different algorithms to estimate longwave radiation. Sub-daily values are calculated with the selected method using the disaggregated values for vapor pressure and temperature. MetSim provies six methods for longwave estimation which can be configured at runtime (Tennessee Valley Authority, 1972; Anderson, 1954; Brutsaert, 1975; Satterlund, 1979; Idso, 1981; Prata, 1996). For more information about the available options see section 3.

Precipitation can be disaggregated in one of two ways. The first and simplest way is to evenly spread the daily precipitation across the sub-daily time steps. A second method has been implemented by Bohn et al. (2019) in MetSim as an extension to the methods that were originally available as part of the VIC preprocessor. The method requires two additional parameters to be specified in the domain file to represent the average precipitation duration and the time of peak precipitation for each cell. The method then disaggregates precipitation by constructing a triangular kernel with total area equal to the daily precipitation centered at the time which is specified as the time of peak precipitation.

2.4 Parallelism

As mentioned earlier, the management of parallelism in MetSim is handled by the Dask Python library. Two complementary options are provided for configuring parallelism. A simple command line option is used to specify the number of processors to use. The way that the pool of available processors are split up is managed by Dask through an object called the scheduler. Users can specify one of three schedulers for managing the pool of processors. Supported options include `distributed`, `threading`, and a custom scheduler for use in multi-node runs or other custom situations. The last option is selected through the specification of a scheduler file. The `distributed` scheduler is the default setting and is the desired setting for most users. It allows for simple and efficient single-machine parallelism with a high degree of stability. The `threading` scheduler will render almost all of the execution to happen in serial due to Python's global interpreter lock and can be useful for debugging purposes. It will always process cells in the same order, as well as provide more complete error messaging in the event of a crash. However, use of the `threading` scheduler is not recommended for general usage because it slows down model execution. Finally, the option of providing a scheduler file allows for more custom usage. A scheduler file can be written by an active Dask Scheduler process, which MetSim uses to connect to the scheduler node to manage parallel computations. This is the mechanism we use in Section 5 to run multi-node jobs by using a Dask interface to the SLURM job scheduler on the cluster used for computation. Dask provides a number of abstractions for different machine architectures, job scheduling systems, and cloud infrastuctures which make this task relatively straightforward in almost all use cases. The specifics of how this is accomplished are machine dependent, so users should refer to the Dask documentation for more information (Dask Development Team, 2016).

When MetSim runs in a parallel mode we have implemented IO locks to manage the coordination of the process pool. Currently we only support parallel IO when reading multiple input files. Currently MetSim does not support parallelized writes.

2.5 Extensibility

The three main computational modules (solar geometry, meteorology simulation, and disaggregation) are separated from the model driver so that each module can be independently changed. This clean separation makes it much easier to modify each respective module.

To extend the meteorological simulation process, a module containing the simulation functions would be added to the code base. This module is required to include a top level `run` function which takes input forcings and the parameter set from the `MetSim` object. Then the new module would be registered with the model driver by adding an entry to the `methods` dictionary which associates a name with the module. After these changes are made, the new simulation approach can be invoked by setting the `method` variable in the configuration to the name given in the `methods` dictionary.

Additional disaggregation methods can be implemented as new functions within the disaggregation module. New entries can be added to the `MetSim` object's parameter set (which is simply a Python dictionary) to allow for selection of the correct disaggregation method. As described in section 2.3, Bohn et al. (2019) provided an example of how MetSim can be extended by developing an alternative precipitation disaggregation that represents the diurnal cycle in a more realistic fashion. Bohn et al. (2019) added the `prec_type` parameter to select between different precipitation disaggregation options.

3 Configuration

One motivation for implementing MetSim as a stand-alone tool was that MetSim options could be exposed more easily, enabling the user to make more informed decisions as to the suitability of a specific option for their application. Table 1 contains a listing of the various configuration options. Most options are used for specification of input and output locations and formats.

MetSim contains several options to select between alternate process representations or represent global parameters that are used as constants in the calculations. The `lw_cloud` and `lw_type` options affect the estimation of the longwave radiation component in the meteorological simulation module. Likewise, the `prec_type` option switches between the precipitation disaggregation options discussed in section 2.3

Time stamps can be interpreted as local time or UTC by configuring the `utc_offset` option. If the `utc_offset` option is set to `False` then the time stamps are interpreted as local time for all cells. For example, if set to `False` then all of the cells in a simulation would experience solar noon at the same time index value. For large, spatially-distributed runs this option should be set to `True` so that the time stamps for all cells are interpreted as UTC.

https://doi.org/10.5194/gmd-2019-179
Preprint. Discussion started: 9 August 2019
© Author(s) 2019. CC BY 4.0 License.

The `sw_prec_thresh` and `rain_scalar` parameters are used together to adjust the amount of shortwave radiation estimated during precipition events (Thornton et al., 2000). `sw_prec_thresh` is the threshold amount of precipitation it takes to apply the `rain_scalar` multiplier to the estimated shortwave radiation. The default values are taken from the MTCLIM 4.2 codebase.

The `tday_coef` parameter (denoted mathematically as T_{day}) is used to calculate the daylight average temperature, with a default value as specified by Thornton and Running (1999). This is calculated as

$$T_{avg} = (1 + T_{day}) \cdot \frac{(T_{max} + T_{min})}{2},$$

where T_{avg} is the daylight average temperature, T_{max} is the daily maximum temperature, and T_{min} is the daily minimum temperature.

The configuration file contains several other required sections. The first is the specification of the "chunking" method. Chunking is the process of decomposing the spatial domain into grouped batches which are run separately. MetSim is able to spatially chunk in a very flexible manner because cells are run independently (that is, there are no spatial interactions between neighboring cells). Running MetSim on a single cell is a computationally simple task, which means that it is often more performant to run multiple cells at a time to minimize the amount of time spent reading from disk or memory.

Chunking can be especially important in parallel runs, where communication across nodes is costly. As a rule of thumb, it is desirable to set the chunk size such that the number of jobs is evenly divided by the number of parallel processes used for the run. This ensures that no idle processors end up waiting at the end of the run.

An example of this section for running 100 cell chunks (which are 10 by 10 cells in shape) of a latitude-longitude gridded dataset (with spatial dimensions named `lat` and `lon`, respectively) would look like:

```
[chunks]
lat = 10
lon = 10
```

The `[chunks]` section also specifies the spatial arrangement for the run. In the previous example we show how this section would be set for a latitude-longitude gridded dataset. This section can also be used to specify running on unstructured grids, such as hydologic response units (HRU). An example for running 100 cell chunks on an HRU discretization (with spatial dimension denoted `hru`) would be:

```
[chunks]
hru = 100
```

The other required sections are the `[forcing_vars]`, `[state_vars]`, and `[domain_vars]` sections. All three provide mappings between the names of the variables in the input datasets and the names which MetSim requires in the format of `metsim_varname = given_varname`. We provide example configuration files in the MetSim repository and documentation.

https://doi.org/10.5194/gmd-2019-179
Preprint. Discussion started: 9 August 2019

4 Testing/Continuous integration

A core part of MetSim's development was the implementation of a test suite that can be run in a continuous integration environment. All tests are executed every time a change to the MetSim's source code repository (i.e., each time a commit is made). This helps to ensure reproducibility and reduce the number of bugs during the development process.

The tests that have been implemented to ensure that each of the software components and high level interfaces work in the intended fashion. As a part of this process we test the various IO formats, several of the options, and the command-line arguments. We also test that the output that MetSim generates matches a previously vetted example output from the VIC4.2 preprocessor. This ensures that the core functionality is not altered without careful consideration. The MetSim v2.0 release of MetSim very closely reproduces the VIC4.2 preprocessor's output when using the default settings. For this test we compute the normalized root mean square error (NRMSE) for all output variables at a sample site. The NRMSE for a variable Y, is given by

$$NRMSE = \frac{\sqrt{\frac{1}{N} \cdot \sum_i^N (Y^i_{metsim} - Y^i_{vic})^2}}{\max(\mathbf{Y}_{metsim}, \mathbf{Y}_{vic}) - \min(\mathbf{Y}_{metsim}, \mathbf{Y}_{vic})},$$

where superscripts denote model timestep, subscripts denote the model name, and boldface denote vectors containing the entire timeseries, and N is the number of timesteps in the model output. In order for the test to be considered successful we check that the NRMSE is less than 0.02 for all variables for a sample site. We demonstrate this in section 5, along with some further discussion of the test case domain and NRMSE values.

5 Analysis

5.1 Comparison to VIC 4.2

To demonstrate that MetSim replicates previous implementations of the same algorithms, we show that MetSim closely reproduces the behavior of the VIC 4.2 preprocessor. We do this using a test case that uses one of the domains included in the VIC sample data. This test case covers a single cell located near Stehekin, Washington, USA, and creates meteorological forcings for January 1949. We compare the output of both MetSim and the VIC 4.2 preprocessor at an hourly timestep for temperature, shortwave radiation, longwave radiation, vapor pressure, air pressure, relative humidity, specific humidity, precipiation, and wind speed. To evaluate the simulations, we compute the NRMSE for each output variable. The NRMSE values for the test case are shown in table 2. We can see that MetSim provides very similar results to VIC 4.2.

5.2 Parallelism and scaling performance

We explore MetSim's computational performance by conducting two scaling experiments. Strong scaling experiments test how the total runtime is affected by adding processors for a fixed overall problem size. Weak scaling experiments test how the total runtime is affected by adding processors proportional to the overall problem size. All of the times reported for the scaling

https://doi.org/10.5194/gmd-2019-179
Preprint. Discussion started: 9 August 2019

experiments were for a single year run at an hourly time step with default parameter and output settings except for changing the `[chunks]` section of the configuration.

The `[chunks]` section allows some performance tuning for parallel runs. We adjusted the chunking strategy in both experiments so that the number of chunks to process was roughly equal to, but less than, two times the number of processers given. That is, each processor ran at most two jobs over the course of each run. By choosing chunks so that the number of jobs is as close to an integer multiple of the number of processors used (without going over), we minimize the chances that a small number of jobs would constitute a disproportionate amount of the execution time. For both experiments we ran MetSim for one year at an hourly timestep over a section of the Pacific Northwestern United states. The input forcings were provided by Livneh et al. (2013) and ran for the year 1915.

For the strong scaling experiment we ran MetSim for one year at an hourly timestep over a domain of 6333 cells. We ran this domain using 2, 4, 8, 16, 32, 64, and 128 processors and measure the time to complete each run. Results are shown in Figure 2 panel a. Runs were conducted on nodes with 16 processors, so the runs with 16, 32, 64, and 128 processors included some inter-node communication. The IO locks were managed by the master process that runs the model driver. We found that there is no additional scaling penalty to running on multiple nodes. The results show that scaling is nearly log-linear with the number of processors.

In the weak scaling experiment we ran MetSim using 2, 4, 8, 16, 32, 64, and 128 processors while varying the number of cells in the run to maintain a constant workload per processor. We ran 125 cells per 2 processors, resulting in runs of 125, 250, 500, 1000, 2000, 4000, and 8000 cells, respectively.

The results of the weak scaling experiment are shown in panel b in Figure 2. Similarly to the strong scaling experiment, we see increasing penalties for adding additional processors. Generally, these scaling results are indicative of a bottleneck due to some portion of the run that must be completed serially. We found that writing the output is one of the main bottlenecks in increasing the number of processors. As we increase the number of processors we see an increase in runtime due to jobs which have completed computation and must wait for the write lock to be released by other jobs.

The architecture and hardware of the machine that MetSim is running on will affect the the amount of IO overhead that the user will experience. Generally speaking, systems with spinning disk hard drives will show the greatest overhead, while solid state drives will show less overhead; likewise, systems with parallel IO systems will also show less overhead. For both experiments we wrote out the standard set of output variables in NetCDF format while using NetCDF for the input format. The results of these experiments would also change with different numbers of output variables and IO formats.

6 Conclusions

We have described MetSim, a software package and standalone tool for the estimation of sparsely observed meteorological quantities at variable temporal resolutions. MetSim is able to provide more features than previous tools which served similar purposes. MetSim's architecture makes extensive use of modern capabilities provided by the Python scientific computing ecosystem. We have described how this modern ecosystem and approach to model development allowed for us to easily expose options and allowed extensions to be developed. As a part of the model development process we have adhered to modern software engineering practices which allowed for new users to be able to contribute to the code base to add new features. MetSim can be used by a broader community than just hydrologic modellers, which was the main target of it's predecessor, the VIC 4.2 preprocessor.

We have shown how the modular nature of MetSim allows for extensions to be written on top of the base algorithms and helps to expose options and parameters that were previously hidden from the user. Part of our motivation to develop MetSim in this fashion is to build a tool which the community can easily extend and improve. We have already accepted modifications from community members for a number of features to both the infrastructure as well as new algorithms for disaggregation.

We demonstrated that MetSim provides nearly identical results as the VIC 4.2 preprocessor while adding additional features such as additional IO formats, parallelism, and additional disaggregation methods for precipitation. We found MetSim's ability to run computationally costly jobs in reasonable amounts of time by showing that it can scale up to 128 processes with only moderate overhead. This makes MetSim usable across a wide range of modelling applications.

Code and data availability. MetSim is publicly available on GitHub at https://github.com/UW-Hydro/MetSim.

Additionally we provide an interactive tutorial at https://github.com/UW-Hydro/MetSim-tutorial.

The DOI for the version of MetSim described in this paper is 10.5281/zenodo.3247806.

Author contributions. AB designed and ran the experiments. AB and JH developed the model code. AB and BN prepared the manuscript with contributions from JH

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. This work was supported in part by grant 80NSSC17K0541 from the NASA AIST program. We would like to thank Elizabeth Clark, Kostas Andreadis, Theodore Bohn, and Kristen Whitney for their contributions to the MetSim code base.

References

Anderson, E.: Energy budget studies, water loss investigations: lake Hefner studies, U.S. Geological Survey Professional Papers, 269, 71–119, 1954.

Beven, K. J. and Cloke, H. L.: Comment on "Hyperresolution global land surface modeling: Meeting a grand challenge for monitoring Earth's terrestrial water" by Eric F. Wood et al., Water Resources Research, 48, n/a–n/a, doi:10.1029/2011WR010982, http://dx.doi.org/10.1029/2011WR010982, w01801, 2012.

Bohn, T. J., Livneh, B., Oyler, J. W., Running, S. W., Nijssen, B., and Lettenmaier, D. P.: Global evaluation of {MTCLIM} and related algorithms for forcing of ecological and hydrological models, Agricultural and Forest Meteorology, 176, 38 – 49, doi:10.1016/j.agrformet.2013.03.003, https://doi.org/10.1016/j.agrformet.2013.03.003, 2013.

Bohn, T. J., Whitney, K. M., Mascaro, G., and Vivoni, E. R.: A Deterministic Approach for Approximating the Diurnal Cycle of Precipitation for Use in Large-Scale Hydrological Modeling, Journal of Hydrometeorology, 0, null, doi:10.1175/JHM-D-18-0203.1, https://doi.org/10.1175/JHM-D-18-0203.1, 2019.

Brutsaert, W.: On a derivable formula for long-wave radiation from clear skies, Water Resources Research, 11, 742–744, doi:10.1029/WR011i005p00742, https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/WR011i005p00742, 1975.

Cherkauer, K. A., Bowling, L. C., and Lettenmaier, D. P.: Variable infiltration capacity cold land process model updates, Global and Planetary Change, 38, 151–159, 2003.

Dask Development Team: Dask: Library for dynamic task scheduling, https://dask.org, 2016.

Deardorff, J. W.: Efficient prediction of ground surface temperature and moisture, with inclusion of a layer of vegetation, Journal of Geophysical Research: Oceans, 83, 1889–1903, doi:10.1029/JC083iC04p01889, https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/JC083iC04p01889, 1978.

Hamman, J. J., Nijssen, B., Bohn, T. J., Gergel, D. R., and Mao, Y.: The Variable Infiltration Capacity model version 5 (VIC-5): infrastructure improvements for new applications and reproducibility, Geoscientific Model Development, 11, 3481–3496, doi:10.5194/gmd-11-3481-2018, https://www.geosci-model-dev.net/11/3481/2018/, 2018.

Hoyer, S. and Hamman, J.: xarray: N-D labeled arrays and datasets in Python, Journal of Open Research Software, 5, doi:10.5334/jors.148, http://doi.org/10.5334/jors.148, 2017.

Idso, S. B.: A set of equations for full spectrum and 8- to 14-μ m and 10.5- to 12.5-μ m thermal radiation from cloudless skies, Water Resources Research, 17, 295–304, doi:10.1029/WR017i002p00295, https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/WR017i002p00295, 1981.

Lam, S. K., Pitrou, A., and Seibert, S.: Numba: A LLVM-based Python JIT Compiler, in: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, LLVM '15, pp. 7:1–7:6, ACM, New York, NY, USA, doi:10.1145/2833157.2833162, http://doi.acm.org/10.1145/2833157.2833162, 2015.

Liang, X., Lettenmaier, D. P., Wood, E. F., and Burges, S. J.: A simple hydrologically based model of land surface water and energy fluxes for general circulation models, Journal of Geophysical Research: Atmospheres, 99, 14 415–14 428, doi:10.1029/94JD00483, http://dx.doi.org/10.1029/94JD00483, 1994.

Livneh, B., Rosenberg, E. A., Lin, C., Nijssen, B., Mishra, V., Andreadis, K. M., Maurer, E. P., and Lettenmaier, D. P.: A Long-Term Hydrologically Based Dataset of Land Surface Fluxes and States for the Conterminous United States: Update and Extensions, Journal of Climate, 26, 9384–9392, doi:10.1175/JCLI-D-12-00508.1, https://doi.org/10.1175/JCLI-D-12-00508.1, 2013.

Prata, A. J.: A new long-wave formula for estimating downward clear-sky radiation at the surface, Quarterly Journal of the Royal Meteorological Society, 122, 1127–1151, doi:10.1002/qj.49712253306, https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/qj.49712253306, 1996.

Rew, R. and Davis, G.: NetCDF: an interface for scientific data access, IEEE Computer Graphics and Applications, 10, 76–82, doi:10.1109/38.56302, 1990.

Rocklin, M.: Dask: Parallel Computation with Blocked algorithms and Task Scheduling, in: Proceedings of the 14th Python in Science Conference, edited by Huff, K. and Bergstra, J., pp. 130 – 136, 2015.

Satterlund, D. R.: An improved equation for estimating long-wave radiation from the atmosphere, Water Resources Research, 15, 1649–1650, doi:10.1029/WR015i006p01649, https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/WR015i006p01649, 1979.

Tennessee Valley Authority: Heat and mass transfer between a water surface and the atmosphere., Water resources research report, 14, 1972.

Thornton, P. E. and Running, S. W.: An improved algorithm for estimating incident daily solar radiation from measurements of temperature, humidity, and precipitation, Agricultural and Forest Meteorology, 93, 211 – 228, doi:10.1016/S0168-1923(98)00126-9, http://dx.doi.org/10.1016/S0168-1923(98)00126-9, 1999.

Thornton, P. E., Hasenauer, H., and White, M. A.: Simultaneous estimation of daily solar radiation and humidity from observed temperature and precipitation: an application over complex terrain in Austria, Agricultural and Forest Meteorology, 104, 255 – 271, doi:https://doi.org/10.1016/S0168-1923(00)00170-2, http://www.sciencedirect.com/science/article/pii/S0168192300001702, 2000.

Whiteman, C. and Allwine, K.: Extraterrestrial solar radiation on inclined surfaces, Environmental Software, 1, 164 – 169, doi:10.1016/0266-9838(86)90020-1, http://dx.doi.org/10.1016/0266-9838(86)90020-1, 1986.

https://doi.org/10.5194/gmd-2019-179
Preprint. Discussion started: 9 August 2019

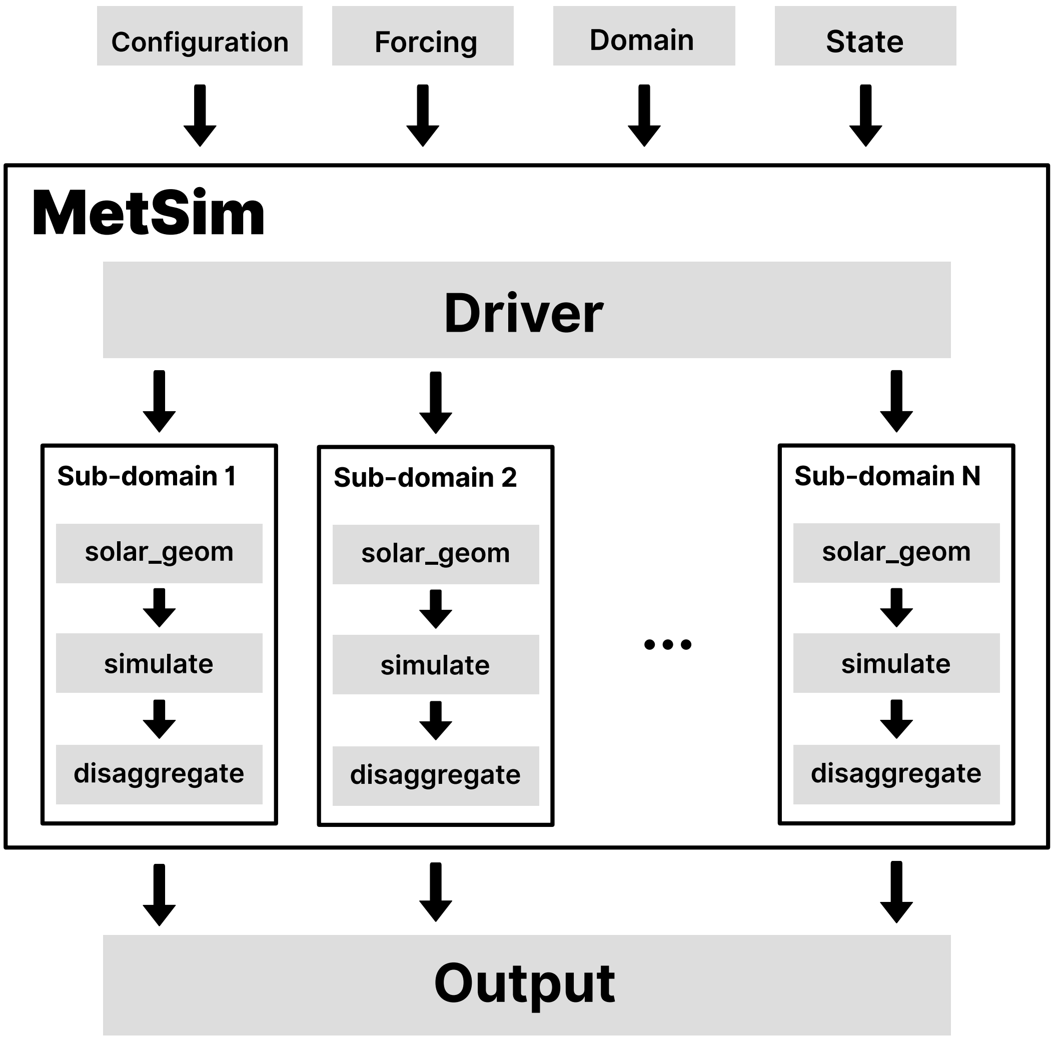

Figure 1. A schematic representation of the MetSim software flow

https://doi.org/10.5194/gmd-2019-179
Preprint. Discussion started: 9 August 2019

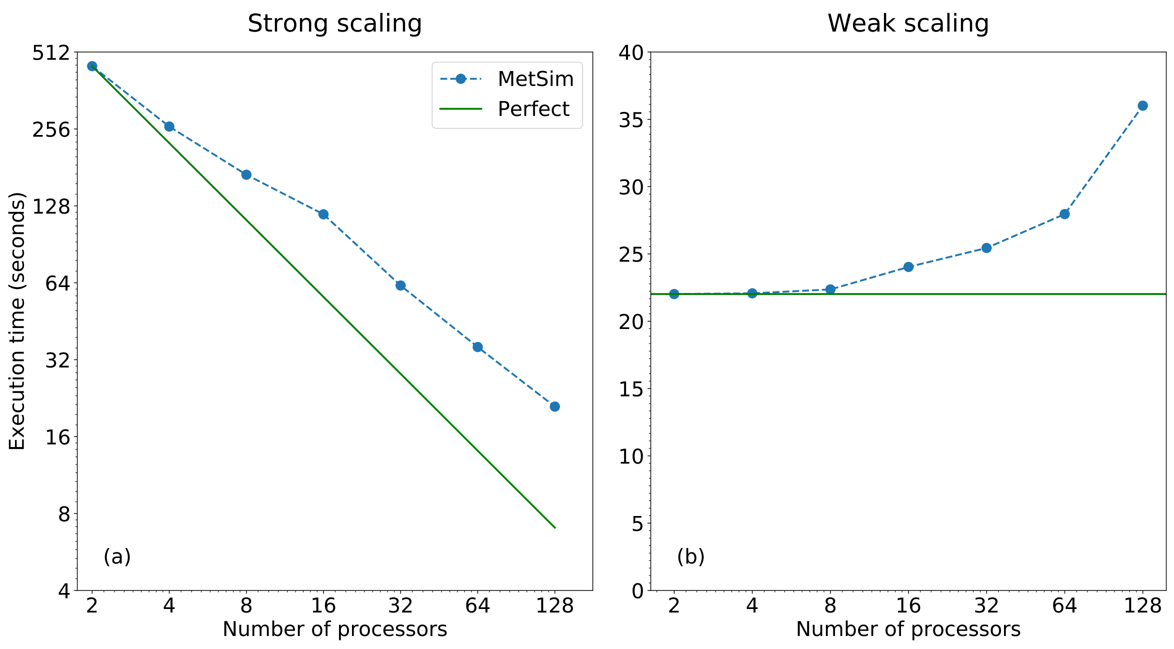

Figure 2. MetSim scaling performance per process

Table 1. Listing of options provided in MetSim

Option Name	Description	Available selections	Default value
out_precision	Precision of output data	f4, f8	f4
lw_cloud	Parameterization to use for calculating cloud effect on longwave radiation.	tva, cloud_deardorff	cloud_deardorff
lw_type	Parameterization to use for calculating longwave radiation	tva,anderson, brutsaert, satterlund, idso, prata	prata
tday_coef	Weight for calculating daily mean temperature	0-1	0.45
tmax_daylength_fraction	Weight for calculation of time of daily maximum temperature	0-1	0.67
sw_prec_thresh	Threshold amount of precipitation to require before applying the rain_scalar multiplier	0-1	0
rain_scalar	Multiplier to apply to shortwave when precipitation > sw_prec_thresh	0-1	0.75
lapse_rate	Temperature decrease with elevation ($^{\circ}C/m$)	0.004-0.009 (generally)	0.0065
utc_offset	Whether to apply local time correction	True/False	False
prec_type	Type of precipitation disaggregation to use	Uniform, Triangle, Mixed	Uniform

https://doi.org/10.5194/gmd-2019-179
Preprint. Discussion started: 9 August 2019

Table 2. Normalized root mean square errors of MetSim output compared to VIC output

Variable	**NRMSE**
Precipitation	4.4035e-06
Wind speed	9.2625e-09
Temperature	0.0072
Shortwave Radiation	0.0024
Longwave Radiation	0.0072
Vapor Pressure	0.0074
Specific Humiditiy	0.0086
Relative Humidity	0.0179
Air Pressure	0.0095