swiflow Documentation

Release 0.3.0

Micah Johnson

Mar 23, 2020

Contents:

1 Welcome to swiflow **1**
- 1.1 Usage . . . 1
- 1.2 Setting up a new project . . . 1
- 1.3 Credits . . . 2

2 Installation **3**
- 2.1 Stable release . . . 3
- 2.2 From sources . . . 3

3 Usage **5**

4 Catchment Models **7**
- 4.1 Comola Catchment Model . . . 7

5 Routing Methods **9**
- 5.1 Direct . . . 9
- 5.2 Muskingum-Cunge (Under Development) . . . 9

6 Calibration Theory **11**
- 6.1 Max Infiltration Theory . . . 11
- 6.2 Residence Time Theory . . . 12
- 6.3 Fair Warning . . . 12

7 Configuration File Reference **13**
- 7.1 setup . . . 13
- 7.2 model . . . 14
- 7.3 output . . . 15
- 7.4 system . . . 16
- 7.5 analysis . . . 16

8 Contributing **17**
- 8.1 Types of Contributions . . . 17
- 8.2 Adding Models . . . 18
- 8.3 Get Started! . . . 18
- 8.4 Pull Request Guidelines . . . 19
- 8.5 Tips . . . 19
- 8.6 Deploying . . . 19

9 Credits **21**
9.1 Development Lead 21
9.2 Contributors 21

10 History **23**
10.1 0.1.0 (2019-05-23) 23
10.2 0.2.0 (2019-06-05) 23
10.3 0.3.0 (2019-07-17) 23

11 Indices and tables **25**

CHAPTER 1

Welcome to swiflow

WARNING: Swiflow is under active development on its master branch.

A python package for modeling streamflow using surface water input from iSnobal

- Free software: MIT license
- Documentation: https://swiflow.readthedocs.io

1.1 Usage

Once installed, swiflow is ran as simply as:

```
swiflow config.ini
```

1.2 Setting up a new project

To setup a project you must have a vector for each subbasin representing water landing on the ground surface. In most cases this project uses aggregated Surface Water Input (SWI) from iSnobal typically generated using AWSM .

1. To setup a basin consider using delineate with the *–streamflow* flag. THis will create all the static files you need for your basin.
2. SWI is usually outputted as an image. To aggregate SWI to a vector, SWIFlow has a function to manage this which is used like the code below. This will produce a CSV containing the vectorized surface water input (SWI) by looping through the watersheds and the em.nc file.

```
prep_swi em.nc watersheds.shp SWI
```

3. Setup a config.ini that looks similar to the one in *./examples/catchment_csv/config.ini* which should point to the new files you just created. For more info on the config simply use inicheck:

```
inicheck -m swiflow --details <section>
```

4. Once you have started a configuration file, attempt to determine the max_infiltration setting in your config file by running the following command over a whole water year. To do this you will also need provide a path to observed_streamflow in the config file.

```
solve_rmax config.ini
```

   This will print out the estimated max_infiltration.

5. Once a max_infiltration value is determined. Use the calibrate_swi tool to estimate the upper_residence_time and lower_residence_time by running the following command over a whole water year and setting your timestep to 4 hours to speed it up:

```
calibrate_swi config.ini
```

   The calibration process is still experimental so you may need to adjust values still. Attempt to use a value set that was listed in the print out from the calibration scheme will help narrow it down.

6. Once you have your parameters set, set the dates of interest and run:

```
swiflow config.ini
```

1.3 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install swiflow, run this command in your terminal:

```
$ pip install swiflow
```

This is the preferred method to install swiflow, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for swiflow can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/USDA-ARS-NWRC/swiflow
```

Or download the tarball:

```
$ curl  -OL https://github.com/USDA-ARS-NWRC/swiflow/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use swiflow simply create a config file point to all the necessary files and

```
swiflow config.ini
```

CHAPTER 4

Catchment Models

Catchment models describe how the inputted water leaves the subcatchment. Currently only one is implemented:

- Comola

4.1 Comola Catchment Model

SWIFlow is a two layer streamflow model. There is an upper storage and a lower storage that interact. The model uses three state variables to model streamflow. The are storage of a layer, infiltration rate into the layer, and the flowrate from the layer.

This model was introduced by Comola et. al 2015 and implemented the same in swiflow as it was originally described.

4.1.1 Equations

Below are the definitions used in the model equations:

- Q - Flowrate in $\frac{m^3}{s}$
- S - Storage in units of meters of water height over subbasin area.
- I - Infiltration rate in units of $\frac{m}{s}$
- $\bar{\tau}$ - Residence time, (calibrated term) in units of seconds
- R_{max} - Maximum infiltration rate to the lower layer (calibrated)
- A - Surface area of either the subbasin or the total watershed in m^2.

The model is described by the following equations:

$$I_{res_L} = min(I_{swi}, R_{max})$$
$$I_{res_U} = I_{swi} - I_{res_L}$$

For change in storage in the upper and lower layer:

$$\frac{dS_{res_{L,U}}}{dt} = I_{res_{L,U}} - \frac{Q}{A_{subbasin}}$$

The Residence time for each layer for each subbasin is:

$$\tau = \left(\frac{A_{subbasin}}{A_{total}}\right)^{\frac{1}{3}} \bar{\tau}$$

The flowrate out of a layer is defined by:

$$Q_{res_{L,U}} = A_{subbasin} \frac{S_{res_{L,U}}}{\tau_{res_{L,U}}}$$
$$Q_{subbasin} = Q_{res_U} + Q_{res_L}$$

4.1.2 Discretization

Spatially the model is split up by subwatersheds or sometimes called HRUs prior to running SWIFlow. This can be performed by basin_setup . Temporally, the change in storage equation is split up explicitly which is implemented as follows:

$$S_{res_{t+1}} = \left(I_{res_t} - \frac{Q_t}{A_{subbasin}}\right) \Delta t + S_{res_t}$$

4.1.3 Algorithm

At the beginning of the run, SWIFlow the initial conditions for $Q_{res_{L,U}}$ and $S_{res_{L,U}}$, S are all zero. SWIFlow computes the state variables in the following order for each timestep.

1. Calculate lower **then** upper infiltration rates ($I_{res_{L,U}}$)
2. Calculate upper and lower flowrates ($Q_{res_{L,U}}$)
3. Calculate future storage using discretized change in storage equation ($S_{res_{t+1}}$)

CHAPTER 5

Routing Methods

SWIFlow has two routing methods developed in it.

- Direct
- Muskingum-Cunge (1969)

5.1 Direct

Direct routing assumes that a bulk of the time water is spent on the hill slope. This is only valid for small catchments.

5.2 Muskingum-Cunge (Under Development)

Description of technique first provided by Cunge 1969. Numerical discretization taken from Gallice et. al 2016 and re-transcribed below.

Variables

- n - Time index [unitless]
- i - Discretize stream reach index [unitless]
- t - Time [Seconds]
- Q - Flowrate [M^3/S]
- l_i - Stream reach discretized length for stream i [m]
- w - Stream segment width [m]
- S_0 - Local bed slope []
- C_r - Wave celerity [m/s]
- Q_r - Representative discharge [m^3/s]

- n_m - Mannings coefficient (0.03 - 0.10 for small natural streams) [s/m^-1/3]

Assumptions

- C_r is derived from Q_r assuming rectangular channel cross section

$$Q_i^{n+1} = C_1 Q_{i-1}^n + C_2 Q_{i-1}^{n+1} + C_3 Q_i^n$$

Where

$$C_1 = \frac{k_i x_i + 0.5\Delta t}{k_i(1 - x_i) + 0.5\Delta t}$$
$$C_2 = \frac{-k_i x_i + 0.5\Delta t}{k_i(1 - x_i) + 0.5\Delta t}$$
$$C_3 = \frac{k_i(1 - x_i) - 0.5\Delta t}{k_i(1 - x_i) + 0.5\Delta t}$$

Where

$$k_i = \frac{l_i}{c_r}$$
$$c_r = \frac{5}{3}\left(\frac{S_0}{n_m^2}\right)^{\frac{3}{10}}\left(\frac{Q_r}{w}\right)^{\frac{2}{5}}$$
$$x_i = 0.5 * min\left(1, 1 - \frac{Q_r}{c_r w S_0 l_i}\right)$$
$$Q_r = \frac{Q_{i-1}^n + Q_{i-1}^{n+1} + Q_i^n}{3}$$
$$h_i^{n+1} = \left(\frac{n_m Q_i^{n+1}}{w S_0}\right)^{\frac{3}{5}}$$

Numerical stability guidance provided by the following for lumped systems as opposed to gridded discretized reaches:

$$2k_i x_i <= \Delta t <= 2k_i(1 - x_i)$$

CHAPTER 6

Calibration Theory

In the SWIFlow configuration file, *max_infiltration* (R_{max}), *lower_residence_time* ($\bar{\tau_l}$), and *upper_residence_time* ($\bar{\tau_u}$) are all model Parameters to be set by the user. These values unfortunately are specific to a basin and thus need calibration.

Most of the following theory depends on the ability to separate the flow into two components of the observed flowrate. The current method utilizes an approach described by Eckhardt 2005 in a paper titled *"How to construct recursive digital filters for baseflow separation to be used for calibration."*

6.1 Max Infiltration Theory

Solving for R_{max} means that the scope of the following derivation should only be performed on the lower layer equations. The storage equation is simplified by assuming the change in storage is zero on an annual basis. Using the base component of the separate flow from Eckhardt's method our storage and infiltration equation becomes:

$$0 = \sum_{t=0}^{T} I_{res_L} - \sum_{t=0}^{T} \frac{Q_L}{A_{subbasin}}$$
$$I_{res_L} = min(I_{swi}, R_{max})$$

Substituting the infiltration equation into the storage equation, it then becomes:

$$0 = \sum_{t=0}^{T} min(I_{swi}, R_{max}) - \sum_{t=0}^{T} \frac{Q_L}{A_{subbasin}}$$

R_{max} is a constant value so using the above equation allows us to iteratively estimate the value. This is performed by bisection where the solution is driving the residual to 0 in the following manner where A is always less than B:

$$C = \frac{A+B}{2}$$

$$residual = \sum_{t=0}^{T} min(I_{swi}, C) - \sum_{t=0}^{T} \frac{Q_L}{A_{subbasin}}$$

$$B = \begin{cases} C, residual > 0 \\ B \end{cases} \qquad A = \begin{cases} C, residual < 0 \\ A \end{cases}$$

The steps to solve for max infiltration:

1. Separate the flow into direct and base flow using Eckhardt's method
2. Calculate the cumulative volumes from the Inputted SWI and the observed flowrate
3. Calculate the derivative of the difference between SWI and Observed volumes
4. Calculate the mean of this timeseries to estimate the infiltration value

TIP: The configuration file sets the time period for analysis. Use a time period of most importance to weight the average towards a high volume period

6.2 Residence Time Theory

After estimating R_{max}, the next step is to solve for residence times for the upper and lower layers ($\bar{\tau}_L$ and $\bar{\tau}_U$). This is performed in a similar fashion as in max infiltration.

The steps to estimate the upper and low residence times

1. Separate the flow into direct and base flow using Eckhardt's method
2. Insert the separated components into the model's states $Q_{base} \rightarrow Q_L$ and $Q_{direct} \rightarrow Q_U$
3. Run the model, skipping solving for Q
4. Solve for $\bar{\tau}_L$ and $\bar{\tau}_U$
5. Use the min/max to bound an iterative solution via bisection. e.g. $A = min(\bar{\tau}_L)$ $B = max(\bar{\tau}_L)$
6. Assume $\bar{\tau}_U$ is constant, and run model as in step #3 with $\bar{\tau_C} = \frac{A+B}{2}$
7. Rerun the model as in step #3.
8. Iterate on the solution picking the new bounds as described above.
9. Repeat Steps 1-8 leaving the new $\bar{\tau_L}$ constant and iterate on $\bar{\tau_U}$

6.3 Fair Warning

The calibration process is in still in development. Estimating for R_{max}, $\bar{\tau}_L$ and $\bar{\tau}_U$ is highly interconnected and thus in these early stages of development some user intuition made need to be applied and even iterated upon. E.g. Calibrate, adjust, calibrate.

CHAPTER 7

Configuration File Reference

The SWIFlow configuration file is described in detail below. This information is all based on the CoreConfig file stored under the top level of the package.

For configuration file syntax information please visit http://inicheck.readthedocs.io/en/latest/

7.1 setup

basin_name

Name of the basin to better tag data

Default: None

Type: string

end

Datetime for where to end modeling runoff

Default: None

Type: datetimeorderedpair

start

Datetime for where to start modeling runoff

Default: None

Type: datetimeorderedpair

stream_network_shp

ShapeFile describing the stream reach and their association to the subbasin. File is produced by basin_setup during delineation.

Default: None

Type: criticalfilename

swi_data

Time series of swi data

Default: None

Type: criticalfilename

watersheds_shp

ShapeFile defining the subbasins. File is produced by basin_setup during delineation.

Default: None

Type: criticalfilename

7.2 model

catchment_model_type

Theoretical streamflow model

Default: comola

Type: string

Options: comola

estimated_efficiency

Ratio of SWI allowed into the system. Mostly implemented for use with the direct routing

Default: 1.0

Type: float

lower_residence_time

Time in days a unit of water will stay in a subbasins lower layer

Default: 100

Type: float

mannings_coefficient

Mannings coefficient to be used when routing_model_type is MuskingumCunge (n_m in docs)

Default: 0.03

Type: float

max_infiltration

Max infiltration in mm/day rate to the the lower layer in the Comola model

Default: 5

Type: float

routing_model_type

Model to use for routing available water from subcatchments to the basin outlet

Default: direct

Type: string

Options: direct muskingumcunge

stream_width

An assumed stream width for when the routing_model_type is MuskingumCunge in meters (w in docs)

Default: 1

Type: float

timestep_hours

Timestep to march through time in integer hours

Default: 24

Type: float

upper_residence_time

Time in days a unit of water will stay in a subbasins upper layer

Default: 10

Type: float

7.3 output

output_location

directory location to output the variables

Default: output

Type: directory

7.4 system

log_level

logging level to use when running a model

Default: debug

Type: string

Options: info debug

log_to_file

Output the log info to log.txt

Default: false

Type: bool

7.5 analysis

observed_name

Name of the columnb containin the actual values of the measured streamflow

Default: qcms

Type: string

observed_streamflow

filename containing a CSV of observed streamflow

Default: None

Type: criticalfilename

show_comparison

Show the plot of the observed and modeled timeseries overlaid

Default: False

Type: bool

CHAPTER 8

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

8.1 Types of Contributions

8.1.1 Report Bugs

Report bugs at https://github.com/USDA-ARS-NWRC/swiflow/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

8.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

8.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

8.2 Adding Models

Adding a catchment model or adding a routing model to swiflow is as simple as

New models can easily be added to swiflow by following the steps below:

1. Create a new class in the swiflow.<relevant module> module with a name containing a keyword
2. Inherit from the Base<Keyword> Class and define the functions in the base class left blank

3. Add your class name (minus the word model) to the options listed under <relevant module>_type 4. Use it.

All new routing models are added to swiflow.routing_models using the keyword Routing in the class name

All new catchment models are added to swiflow.catchment_models using the keyword Model in the class name

8.2.1 Write Documentation

swiflow could always use more documentation, whether as part of the official swiflow docs, in docstrings, or even on the web in blog posts, articles, and such.

8.2.2 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/USDA-ARS-NWRC/swiflow/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

8.3 Get Started!

Ready to contribute? Here's how to set up *swiflow* for local development.

1. Fork the *swiflow* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/swiflow.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv swiflow
$ cd swiflow/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 swiflow tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

8.4 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5 3.6, and 3.7. Check https://travis-ci.org/USDA-ARS-NWRC/swiflow/pull_requests and make sure that the tests pass for all supported Python versions.

8.5 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_swiflow
```

8.6 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

CHAPTER 9

Credits

9.1 Development Lead

- Ernesto Trujillo @etrujil
- Micah Johnson @micahjohnson150

9.2 Contributors

Want to be the first?

CHAPTER 10

History

10.1 0.1.0 (2019-05-23)

- First push to Github

10.2 0.2.0 (2019-06-05)

- First working model

10.3 0.3.0 (2019-07-17)

- Added in a calibration method
- Added in a conversion function to convert different inputs
- Added in an analysis and validation function
- Added in SWI aggregation script

CHAPTER 11

Indices and tables

- genindex
- modindex
- search