RAPIDpy Documentation

Release 2.5.1

Alan D. Snow

Aug 30, 2017

Contents

RAPIDpy is a python interface for RAPID that assists to prepare inputs, runs the RAPID program, and provides post-processing utilities. More information about installation and the input parameters for RAPID can be found at http://rapid-hub.org. The source code for RAPID is located at https://github.com/c-h-david/rapid.

CHAPTER 1

Publications:

Tavakoly, A. A., A. D. Snow, C. H. David, M. L. Follum, D. R. Maidment, and Z.-L. Yang, (2016) "Continental-Scale River Flow Modeling of the Mississippi River Basin Using High-Resolution NHDPlus Dataset", Journal of the American Water Resources Association (JAWRA) 1-22. DOI: 10.1111/1752-1688.12456

CHAPTER 2

Datasets:

Ahmad A Tavakoly. (2017). RAPID input files corresponding to the Mississippi River Basin using the NHDPlus v2 Dataset [Data set]. Zenodo. http://doi.org/10.5281/zenodo.322886

Contents:

2.1 Installation

2.1.1 Step 1: Install RAPID

Before Installation Steps:

Ubuntu:

```
$ sudo apt-get install gfortran g++
```

RedHat/CentOS:

```
$ sudo yum install gcc-c++ gcc-gfortran
```

Windows with Cygwin:

Downloaded Cygwin (64-bit) (https://www.cygwin.com/) with these dependencies:

- gcc-core
- gcc-fortran
- gcc-g++

- gdb
- git
- make
- time
- wget

Installation Steps:

Manual:

- See: http://rapid-hub.org

Bash:

1. Clone RAPID repository:

```
$ git clone https://github.com/c-h-david/rapid.git
```

2. Install Prereqs:

```
$ cd rapid
$ chmod u+x rapid_install_prereqs.sh
$ ./rapid_install_prereqs.sh
```

3. Append *source rapid_specify_varpath.sh* to the ~/.bashrc or ~/.bash_profile:

```
source /path/to/cloned/rapid/rapid_specify_varpath.sh
```

4. Restart Terminal
5. Build RAPID:

```
$ cd rapid/src
$ make rapid
```

2.1.2 Step 2: Install RAPIDpy

Due to the dependencies required, we recommend using Anaconda or Miniconda. They can be downloaded from https://www.continuum.io/downloads or from https://conda.io/miniconda.html.

After installing Anaconda or Miniconda:

```
$ conda install -c conda-forge rapidpy
```

Developer Installation

This is how you get the most up-to-date version of the code.

See: https://github.com/erdc-cm/RAPIDpy/blob/master/.travis.yml for a more detailed list of installation steps.

Note: If you don't have git, you can download the code from https://github.com/erdc-cm/RAPIDpy

```
$ git clone https://github.com/erdc-cm/RAPIDpy.git
$ cd RAPIDpy
$ python setup.py install
```

To develop on the latest version:

```
$ git clone https://github.com/erdc-cm/RAPIDpy.git
$ cd RAPIDpy
$ python setup.py develop
```

2.2 Generating Stream Network

2.2.1 Using ArcHydro to Generate Stream Network

See:

- https://github.com/Esri/python-toolbox-for-rapid
- https://github.com/erdc-cm/python-toolbox-for-rapid

2.2.2 Using TauDEM to Generate Stream Network

For more information about taudem, see: http://hydrology.usu.edu/taudem/taudem5/index.html

2.2.3 Installation

Step 1: Install TauDEM

```
$ cd /path/to/scripts
$ git clone https://github.com/dtarb/TauDEM.git
$ cd TauDEM/src
$ make
```

Step 2: Install RAPIDpy with GIS Dependencies

See: *Installation*

2.2.4 How To Use

Initialize TauDEM Manager

class RAPIDpy.gis.taudem.**TauDEM**(*taudem_exe_path=”*, *num_processors=1*, *use_all_processors=False*, *mpiexec_path='mpiexec'*)

TauDEM process manager.

taudem_exe_path
Optional[str] – Path to TauDEM directory containing executables. This is requred to use TauDEM functionality.

num_processors
Optional[int] – Number of proessors to use with TauDEM. It only works if use_all_processors=False.

use_all_processors
Optional[bool] – If True, the TauDEM processes will use all avaialble processors.

mpiexec_path
Optional[str] – Path to mpiexec command. Default is 'mpiexec'.

Initialization Example:

```
from RAPIDpy.gis.taudem import TauDEM

td = TauDEM("/path/to/scripts/TauDEM", use_all_processors=True)
```

Generate network from DEM

TauDEM.**demToStreamNetwork**(*output_directory*, *raw_elevation_dem=""*, *pit_filled_elevation_grid=""*, *flow_dir_grid_d8=""*, *contributing_area_grid_d8=""*, *flow_dir_grid_dinf=""*, *contributing_area_grid_dinf=""*, *use_dinf=False*, *threshold=1000*, *delineate=False*)

This function will run all of the TauDEM processes to generate a stream network from an elevation raster.

Note: For information about the stream reach and watershed process: http://hydrology.usu.edu/taudem/taudem5/help53/StreamReachAndWatershed.html

Note: For information about the *threshold* parameter, see: http://hydrology.usu.edu/taudem/taudem5/help53/StreamDefinitionByThreshold.html

Parameters

- **output_directory** (*str*) – Path to output generated files to.
- **raw_elevation_dem** (*Optional[str]*) – Path to original elevation DEM file. Required if *pit_filled_elevation_grid* is not used.
- **pit_filled_elevation_grid** (*Optional[str]*) – Path to pit filled elevation DEM file. Required if *raw_elevation_dem* is not used.
- **flow_dir_grid_d8** (*Optional[str]*) – Path to flow direction grid generated using TauDEM's D8 method.
- **contributing_area_grid_d8** (*Optional[str]*) – Path to contributing area grid generated using TauDEM's D8 method.
- **flow_dir_grid_dinf** (*Optional[str]*) – Path to flow direction grid generated using TauDEM's D-Infinity method (EXPERIMENTAL).
- **contributing_area_grid_dinf** (*Optional[str]*) – Path to contributing area grid generated using TauDEM's D-Infinity method (EXPERIMENTAL).

- **use_dinf** (*Optional[bool]*) – Use the D-Infinity method to get stream definition (EXPERIMENTAL).
- **threshold** (*Optional[int]*) – The stream threshold or maximum number of upstream grid cells. See above note.
- **delineate** (*Optional[bool]*) – If True, this will use the delineate option for theis method using TauDEM. Default is False.

Here is an example of how to use this:

```
from RAPIDpy.gis.taudem import TauDEM

td = TauDEM("/path/to/scripts/TauDEM")

elevation_dem = '/path/to/dem.tif'
output_directory = '/path/to/output/files'
td.demToStreamNetwork(output_directory,
                      elevation_dem,
                      threshold=1000)
```

Add Length in meters attribute

TauDEM.**addLengthMeters**(*stream_network*)

Adds length field in meters to network (The added field name will be 'LENGTH_M').

Note: This may be needed for generating the kfac file depending on the units of your raster. See: *RAPID GIS Tools*.

Parameters **stream_network** (*str*) – Path to stream network file.

Here is an example of how to use this:

```
import os
from RAPIDpy.gis.taudem import TauDEM

td = TauDEM()

output_directory = '/path/to/output/files'
td.addLengthMeters(os.path.join(output_directory,"stream_reach_file.shp"))
```

Extract Sub Network

STEP 1: Extract sub network from stream network

There are two options to do this.

1. Choose your own outlet point: *extractSubNetwork()*
2. Or let the code find the larges network: *extractLargestSubNetwork()*.

TauDEM.**extractSubNetwork**(*network_file*, *out_subset_network_file*, *outlet_ids*, *river_id_field*, *next_down_id_field*, *river_magnitude_field*, *safe_mode=True*)

Extracts a subset river network from the main river network based on the outlet IDs.

Parameters

- **network_file** (*str*) – Path to the stream network shapefile.
- **out_subset_network_file** (*str*) – Path to the output subset stream network shapefile.
- **outlet_ids** (*list*) – List if integers reperesenting the outlet IDs to be included in the subset stream network.
- **river_id_field** (*str*) – Name of the river ID field in the stream network shapefile.
- **next_down_id_field** (*str*) – Name if the field with the river ID of the next downstream river segment in the stream network shapefile.
- **river_magnitude_field** (*str*) – Name of the river magnitude field in the stream network shapefile.
- **safe_mode** (*Optional[bool]*) – If True, it will kill the simulation early before over taxing your computer. If you are confident your computer can handle it, set it to False.

Here is an example of how to use this:

```
import os
from RAPIDpy.gis.taudem import TauDEM

td = TauDEM()

output_directory = '/path/to/output/files'
td.extractSubNetwork(network_file=os.path.join(output_directory,"stream_reach_
↪file.shp"),
                     out_subset_network_file=os.path.join(output_directory,
↪"stream_reach_file_subset.shp"),
                     outlet_ids=[60830],
                     river_id_field="LINKNO",
                     next_down_id_field="DSLINKNO",
                     river_magnitude_field="Magnitude",
                     )
```

TauDEM.**extractLargestSubNetwork**(*network_file*, *out_subset_network_file*, *river_id_field*, *next_down_id_field*, *river_magnitude_field*, *safe_mode=True*)

Extracts the larges sub network from the watershed based on the magnitude parameter.

Parameters

- **network_file** (*str*) – Path to the stream network shapefile.
- **out_subset_network_file** (*str*) – Path to the output subset stream network shapefile.
- **river_id_field** (*str*) – Name of the river ID field in the stream network shapefile.
- **next_down_id_field** (*str*) – Name if the field with the river ID of the next downstream river segment in the stream network shapefile.
- **river_magnitude_field** (*str*) – Name of the river magnitude field in the stream network shapefile.
- **safe_mode** (*Optional[bool]*) – If True, it will kill the simulation early before over taxing your computer. If you are confident your computer can handle it, set it to False.

Here is an example of how to use this:

```
import os
from RAPIDpy.gis.taudem import TauDEM

td = TauDEM()

output_directory = '/path/to/output/files'
td.extractLargestSubNetwork(network_file=os.path.join(output_directory,"stream_
↪reach_file.shp"),
                            out_subset_network_file=os.path.join(output_directory,
↪"stream_reach_file_subset.shp"),
                            river_id_field="LINKNO",
                            next_down_id_field="DSLINKNO",
                            river_magnitude_field="Magnitude",
                            )
```

STEP 2: Extract sub network from catchments

`TauDEM.`**`extractSubsetFromWatershed`**(*subset_network_file*, *subset_network_river_id_field*, *watershed_file*, *watershed_network_river_id_field*, *out_watershed_subset_file*)

Extract catchment by using subset network file. Use this after using either `extractSubNetwork()` or `extractLargestSubNetwork()`.

Parameters

- **network_file** (*str*) – Path to the stream network shapefile.
- **out_subset_network_file** (*str*) – Path to the output subset stream network shapefile.
- **river_id_field** (*str*) – Name of the river ID field in the stream network shapefile.
- **next_down_id_field** (*str*) – Name if the field with the river ID of the next downstream river segment in the stream network shapefile.
- **river_magnitude_field** (*str*) – Name of the river magnitude field in the stream network shapefile.
- **safe_mode** (*Optional[bool]*) – If True, it will kill the simulation early before over taxing your computer. If you are confident your computer can handle it, set it to False.

Here is an example of how to use this:

```
import os
from RAPIDpy.gis.taudem import TauDEM

td = TauDEM()

output_directory = '/path/to/output/files'
td.extractSubsetFromWatershed(subset_network_file=os.path.join(output_directory,
↪"stream_reach_file_subset.shp"),
                              subset_network_river_id_field="LINKNO",
                              watershed_file=os.path.join(output_directory,
↪"watershed_shapefile.shp"),
                              watershed_network_river_id_field="LINKNO",
                              out_watershed_subset_file=os.path.join(output_
↪directory,"watershed_shapefile_subset.shp"))
```

2.3 RAPID GIS Tools

These tools generate the RAPID input files and weight table files from the GIS stream networks.

Note: To generate your own network from a DEM see *Generating Stream Network*

Note: For these tools to work, you need GIS dependencies installed (See *Installation*).

There are also tools by Esri for ArcMap located here:

- https://github.com/Esri/python-toolbox-for-rapid
- https://github.com/erdc-cm/python-toolbox-for-rapid

2.3.1 Workflows

Static RAPID Files

Weight Table Files

Static RAPID Files and Weight Table Files

2.3.2 Individual Tools

Static RAPID Files

Weight Tables

2.3.3 Utilities

2.3.4 How it works:

Snow, Alan D., Scott D. Christensen, Nathan R. Swain, E. James Nelson, Daniel P. Ames, Norman L. Jones, Deng Ding, Nawajish S. Noman, Cedric H. David, Florian Pappenberger, and Ervin Zsoter, 2016. A High-Resolution National-Scale Hydrologic Forecast System from a Global Ensemble Land Surface Model. *Journal of the American Water Resources Association (JAWRA)* 1-15, DOI: 10.1111/1752-1688.12434

Snow, Alan Dee, "A New Global Forecasting Model to Produce High-Resolution Stream Forecasts" (2015). All Theses and Dissertations. Paper 5272. http://scholarsarchive.byu.edu/etd/5272

2.4 Running RAPID

2.4.1 Tutorial

Step 1: Initialize the RAPID manager class.

- First, add the path to you rapid executable location.

- Next, you need to either tell it how many processors to use using the *num_processors* input variable or to use all available processors set *use_all_processors* to true.
- After that, add any other parameters you would like to use that would normally be in the rapid namelist file (this is case sensitive).

class RAPIDpy.rapid.**RAPID**(*rapid_executable_location=''*, *num_processors=1*, *use_all_processors=False*, *cygwin_bin_location=''*, *mpiexec_command='mpiexec'*, *ksp_type='richardson'*, ***kwargs*)

This class is designed to prepare the rapid_namelist file and run the RAPID program. There are also other utilities added.

rapid_executable_location
Optional[str] – Path to the RAPID executable location.

num_processors
Optional[int] – Number of procesors to use. Default is 1. Overridden if *use_all_processors* is True.

use_all_processors
Optional[bool] – If set to True, the RAPID program will use all available processors. Default is False.

cygwin_bin_location
Optional[str] – If using Windows, this is the path to the Cygwin 'bin' directory.

mpiexec_command
Optional[str] – This is the mpi execute commmand. Default is "mpiexec".

ksp_type
Optional[str] – This is the solver type. Default is "richardson".

****kwargs**
Optional[str] – Keyword arguments matching the input parameters in the RAPID namelist.

Linux Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(rapid_executable_location='~/work/rapid/run/rapid'
                      use_all_processors=True,
                      ZS_TauR=24*3600, #duration of routing procedure (time step␣
↪of runoff data)
                      ZS_dtR=15*60, #internal routing time step
                      ZS_TauM=365*24*3600, #total simulation time
                      ZS_dtM=24*3600 #input time step
                     )
```

Windows with Cygwin Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(rapid_executable_location='C:/cygwin64/home/username/work/
↪rapid/run/rapid',
                      cygwin_bin_location='C:/cygwin64/bin',
                      use_all_processors=True,
                      ZS_TauR=24*3600, #duration of routing procedure (time step␣
↪of runoff data)
                      ZS_dtR=15*60, #internal routing time step
                      ZS_TauM=365*24*3600, #total simulation time
                      ZS_dtM=24*3600 #input time step
                     )
```

Step 2 (optional): Add/update additional namelist parameters later

RAPID.**update_parameters**(***kwargs*)

You can add or update rapid namelist parameters by using the name of the variable in the rapid namelist file (this is case sensitive).

Parameters ****kwargs** (*Optional[str]*) – Keyword arguments matching the input parameters in the RAPID namelist.

Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(
                        #ADD PARAMETERS
                     )

rapid_manager.update_parameters(rapid_connect_file='../rapid-io/input/rapid_
↪connect.csv',
                                Vlat_file='../rapid-io/input/m3_riv.nc',
                                riv_bas_id_file='../rapid-io/input/riv_bas_id.csv
↪',
                                k_file='../rapid-io/input/k.csv',
                                x_file='../rapid-io/input/x.csv',
                                Qout_file='../rapid-io/output/Qout.nc',
                                )
```

Step 3 (optional): Update reach number data

RAPID.**update_reach_number_data**()

Update the reach number data for the namelist based on input files.

> **Warning:** You need to make sure you set *rapid_connect_file* and *riv_bas_id_file* before running this function.

Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(
                      #ADD PARAMETERS
                      rapid_connect_file='../rapid-io/input/rapid_connect.csv',
                      riv_bas_id_file='../rapid-io/input/riv_bas_id.csv',
                     )

rapid_manager.update_reach_number_data()
```

Example with forcing data:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(
                      #ADD PARAMETERS
                      rapid_connect_file='../rapid-io/input/rapid_connect.csv',
                      riv_bas_id_file='../rapid-io/input/riv_bas_id.csv',
```

```
                        )

rapid_manager.update_parameters(Qfor_file=qfor_file,
                                    for_tot_id_file=for_tot_id_file,
                                    for_use_id_file=for_use_id_file,
                                    ZS_dtF=3*60*60,
                                    BS_opt_for=True)

rapid_manager.update_reach_number_data()
```

Step 4 (optional): Update simulation runtime data

RAPID.**update_simulation_runtime**()

Updates the total simulation duration from the m3 file (Vlat_file) and the time step (ZS_TauR).

Warning: You need to set the m3 file (Vlat_file) and the time step (ZS_TauR) before runnning this function.

Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(
                         #ADD PARAMETERS
                         Vlat_file='../rapid-io/input/m3_riv.csv',
                         ZS_TauR=3*3600,
                        )

rapid_manager.update_simulation_runtime()
```

Step 5: Run RAPID

RAPID.**run**(*rapid_namelist_file=''*)

Run RAPID program and generate file based on inputs This will generate your rapid_namelist file and run RAPID from wherever you call this script (your working directory).

Parameters **rapid_namelist_file** (*Optional (str)*) – Path of namelist file to use in the simulation. It will be updated with any parameters added to the RAPID manager.

Linux Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(rapid_executable_location='~/work/rapid/src/rapid'
                         use_all_processors=True,
                        )

rapid_manager.update_parameters(rapid_connect_file='../rapid-io/input/rapid_
↪connect.csv',
                                    Vlat_file='../rapid-io/input/m3_riv.nc',
                                    riv_bas_id_file='../rapid-io/input/riv_bas_id.csv
↪',
                                    k_file='../rapid-io/input/k.csv',
                                    x_file='../rapid-io/input/x.csv',
```

```
                                    Qout_file='../rapid-io/output/Qout.nc',
                                    )

rapid_manager.update_reach_number_data()
rapid_manager.update_simulation_runtime()
rapid_manager.run(rapid_namelist_file='../rapid-io/input/rapid_namelist')
```

Linux Reservoir Forcing Flows Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(rapid_executable_location='~/work/rapid/src/rapid',
                      num_processors=4,
                      IS_for_tot=4,
                      IS_for_use=4,
                      for_tot_id_file='../rapid-io/input/dam_id.csv',
                      for_use_id_file='../rapid-io/input/dam_id.csv',
                      Qfor_file='../rapid-io/input/qout_dams.csv',
                      ZS_dtF=86400,
                      BS_opt_for=True,
                      )

rapid_manager.run(rapid_namelist_file='../rapid-io/input/rapid_namelist_regular_
↪run')
```

Windows with Cygwin Example:

```
from RAPIDpy import RAPID
from os import path

rapid_manager = RAPID(rapid_executable_location='C:/cygwin64/home/username/work/
↪rapid/run/rapid',
                      cygwin_bin_location='C:/cygwin64/bin',
                      use_all_processors=True,
                      ZS_TauR=24*3600,
                      ZS_dtR=15*60,
                      ZS_TauM=365*24*3600,
                      ZS_dtM=24*3600
                     )

rapid_input_folder = 'C:/cygwin64/home/username/work/rapid-io/input'
rapid_output_folder = 'C:/cygwin64/home/username/work/rapid-io/output'
rapid_manager.update_parameters(rapid_connect_file=path.join(rapid_input_folder,
↪'rapid_connect.csv'),
                                Vlat_file=path.join(rapid_input_folder, 'm3_riv.nc
↪'),
                                riv_bas_id_file=path.join(rapid_input_folder,
↪'riv_bas_id.csv'),
                                k_file=path.join(rapid_input_folder, 'k.csv'),
                                x_file=path.join(rapid_input_folder, 'x.csv'),
                                Qout_file=path.join(rapid_output_folder, 'Qout.nc
↪'),
                                )

rapid_manager.update_reach_number_data()
rapid_manager.update_simulation_runtime()
rapid_manager.run()
```

Step 6 (optional): Convert RAPID output to be CF Compliant

`RAPID.`**`make_output_CF_compliant`**(*simulation_start_datetime*, *comid_lat_lon_z_file=''*, *project_name='Normal RAPID project'*)

This function converts the RAPID output to be CF compliant. This will require a *comid_lat_lon_z.csv* file (See: `FlowlineToPoint()` to generate the file).

Note: It prepends time an initial flow to your simulation from the *qinit_file*. If no qinit file is given, an initial value of zero is added.

Warning: This will delete your original Qout file.

Parameters

- **simulation_start_datetime** (*datetime*) – Datetime object with the start date of the simulation.
- **comid_lat_lon_z_file** (*Optional[str]*) – Path to the *comid_lat_lon_z.csv* file. If none given, spatial information will be skipped.
- **project_name** (*Optional[str]*) – Name of project to add to the RAPID output file.

Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(rapid_executable_location='~/work/rapid/run/rapid'
                      use_all_processors=True,
                      ZS_TauR=24*3600,
                      ZS_dtR=15*60,
                      ZS_TauM=365*24*3600,
                      ZS_dtM=24*3600
                      rapid_connect_file='../rapid-io/input/rapid_connect.csv',
                      Vlat_file='../rapid-io/input/m3_riv.nc',
                      riv_bas_id_file='../rapid-io/input/riv_bas_id.csv',
                      k_file='../rapid-io/input/k.csv',
                      x_file='../rapid-io/input/x.csv',
                      Qout_file='../rapid-io/output/Qout.nc',
                     )

rapid_manager.run()

rapid_manager.make_output_CF_compliant(simulation_start_datetime=datetime.
↪datetime(1980, 1, 1),
                                       comid_lat_lon_z_file='../rapid-io/input/
↪comid_lat_lon_z.csv',
                                       project_name="ERA Interim Historical flows␣
↪by US Army ERDC")
```

2.4.2 Full API Description

class RAPIDpy.rapid.**RAPID**(*rapid_executable_location=”*, *num_processors=1*, *use_all_processors=False*, *cygwin_bin_location=”*, *mpiexec_command='mpiexec'*, *ksp_type='richardson'*, ***kwargs*)

This class is designed to prepare the rapid_namelist file and run the RAPID program. There are also other utilities added.

rapid_executable_location
Optional[str] – Path to the RAPID executable location.

num_processors
Optional[int] – Number of procesors to use. Default is 1. Overridden if *use_all_processors* is True.

use_all_processors
Optional[bool] – If set to True, the RAPID program will use all available processors. Default is False.

cygwin_bin_location
Optional[str] – If using Windows, this is the path to the Cygwin 'bin' directory.

mpiexec_command
Optional[str] – This is the mpi execute commmand. Default is “mpiexec”.

ksp_type
Optional[str] – This is the solver type. Default is “richardson”.

****kwargs**
Optional[str] – Keyword arguments matching the input parameters in the RAPID namelist.

Linux Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(rapid_executable_location='~/work/rapid/run/rapid'
                      use_all_processors=True,
                      ZS_TauR=24*3600, #duration of routing procedure (time step␣
↪of runoff data)
                      ZS_dtR=15*60, #internal routing time step
                      ZS_TauM=365*24*3600, #total simulation time
                      ZS_dtM=24*3600 #input time step
                     )
```

Windows with Cygwin Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(rapid_executable_location='C:/cygwin64/home/username/work/
↪rapid/run/rapid',
                      cygwin_bin_location='C:/cygwin64/bin',
                      use_all_processors=True,
                      ZS_TauR=24*3600, #duration of routing procedure (time step␣
↪of runoff data)
                      ZS_dtR=15*60, #internal routing time step
                      ZS_TauM=365*24*3600, #total simulation time
                      ZS_dtM=24*3600 #input time step
                     )
```

generate_namelist_file(*rapid_namelist_file*)
Generate rapid_namelist file.

Parameters **rapid_namelist_file** (*str*) – Path of namelist file to generate from parameters added to the RAPID manager.

generate_qinit_from_past_qout (*qinit_file*, *time_index=-1*, *out_datetime=None*)
Generate qinit from a RAPID qout file

Parameters

- **qinit_file** (*str*) – Path to output qinit_file.
- **time_index** (*Optional[int]*) – Index of simulation to generate initial flow file. Default is the end.
- **out_datetime** (*Optional[datetime]*) – Datetime object containing time of initialization.

Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(Qout_file='/output_mississippi-nfie/Qout_k2v1_
↪2005to2009.nc',
                      rapid_connect_file='/input_mississippi_nfie/rapid_
↪connect_ECMWF.csv'
                     )

rapid_manager.generate_qinit_from_past_qout(qinit_file='/input_mississippi_
↪nfie/Qinit_2008_flood.csv',
                                            time_index=10162)
```

generate_seasonal_intitialization (*qinit_file*, *datetime_start_initialization=datetime.datetime(2017, 8, 30, 14, 34, 27, 819137)*)
This creates a seasonal qinit file from a RAPID qout file. This requires a simulation Qout file with a longer time period of record and to be CF compliant. It takes the average of the current date +- 3 days and goes back as far as possible.

Parameters

- **qinit_file** (*str*) – Path to output qinit_file.
- **datetime_start_initialization** (*Optional[datetime]*) – Datetime object with date of simulation to go back through the years and get a running average to generate streamflow initialization. Default is utcnow.

This example shows how to use it:

```
from RAPIDpy.rapid import RAPID

rapid_manager = RAPID(Qout_file='/output_mississippi-nfie/Qout_2000to2015.nc',
                      rapid_connect_file='/input_mississippi_nfie/rapid_
↪connect_ECMWF.csv'
                      )

rapid_manager.generate_seasonal_intitialization(qinit_file='/input_
↪mississippi_nfie/Qinit_seasonal_avg_jan_1.csv')
```

generate_usgs_avg_daily_flows_opt (*reach_id_gage_id_file*, *start_datetime*, *end_datetime*, *out_streamflow_file*, *out_stream_id_file*)
Generate daily streamflow file and stream id file required for calibration or for substituting flows based on USGS gage ids associated with stream ids.

Parameters

- **reach_id_gage_id_file** (*str*) – Path to reach_id_gage_id file.
- **start_datetime** (*datetime*) – A datetime object with the start date to download data.
- **end_datetime** (*datetime*) – A datetime object with the end date to download data.
- **out_streamflow_file** (*str*) – The path to output the streamflow file for RAPID.
- **out_stream_id_file** (*str*) – The path to output the stream ID file associated with the streamflow file for RAPID.

Example *reach_id_gage_id_file*:

```
COMID, USGS_GAGE_ID
2000, 503944
...
```

Warning: Overuse will get you blocked from downloading data from USGS.

Warning: This code does not clean the data in any way. Thus, you are likely to run into issues if you simply use the raw data.

Warning: The code skips gages that do not have data for the entire time period.

Simple Example:

```
import datetime
from os.path import join
from RAPIDpy import RAPID

main_path = "/home/username/data"

rapid_manager = RAPID()
rapid_manager.generate_usgs_avg_daily_flows_opt(reach_id_gage_id_
↪file=join(main_path,"mississippi_usgsgage_id_comid.csv"),
                                                start_datetime=datetime.
↪datetime(2000,1,1),
                                                end_datetime=datetime.
↪datetime(2014,12,31),
                                                out_streamflow_file=join(main_
↪path,"streamflow_2000_2014.csv"),
                                                out_stream_id_file=join(main_
↪path,"streamid_2000_2014.csv"))
```

Complex Example:

```
import datetime
from os.path import join
from RAPIDpy import RAPID

main_path = "/home/username/data"
```

```
rapid_manager = RAPID(rapid_executable_location='~/work/rapid/run/rapid'
                      use_all_processors=True,
                      ZS_TauR=24*3600,
                      ZS_dtR=15*60,
                      ZS_TauM=365*24*3600,
                      ZS_dtM=24*3600
                     )

rapid_manager.update_parameters(rapid_connect_file='../rapid-io/input/rapid_
↪connect.csv',
                                Vlat_file='../rapid-io/input/m3_riv.nc',
                                riv_bas_id_file='../rapid-io/input/riv_bas_id.
↪csv',
                                k_file='../rapid-io/input/k.csv',
                                x_file='../rapid-io/input/x.csv',
                                Qout_file='../rapid-io/output/Qout.nc',
                                )

rapid_manager.update_reach_number_data()
rapid_manager.update_simulation_runtime()
rapid_manager.generate_usgs_avg_daily_flows_opt(reach_id_gage_id_
↪file=join(main_path,"mississippi_usgsgage_id_comid.csv"),
                                                start_datetime=datetime.
↪datetime(2000,1,1),
                                                end_datetime=datetime.
↪datetime(2014,12,31),
                                                out_streamflow_file=join(main_
↪path,"streamflow_2000_2014.csv"),
                                                out_stream_id_file=join(main_
↪path,"streamid_2000_2014.csv"))
rapid_manager.run()
```

`make_output_CF_compliant` (*simulation_start_datetime*, *comid_lat_lon_z_file=''*, *project_name='Normal RAPID project'*)

This function converts the RAPID output to be CF compliant. This will require a *comid_lat_lon_z.csv* file (See: `FlowlineToPoint()` to generate the file).

Note: It prepends time an initial flow to your simulation from the *qinit_file*. If no qinit file is given, an initial value of zero is added.

Warning: This will delete your original Qout file.

Parameters

- **`simulation_start_datetime`** (*`datetime`*) – Datetime object with the start date of the simulation.
- **`comid_lat_lon_z_file`** (*`Optional[str]`*) – Path to the *comid_lat_lon_z.csv* file. If none given, spatial information will be skipped.
- **`project_name`** (*`Optional[str]`*) – Name of project to add to the RAPID output file.

Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(rapid_executable_location='~/work/rapid/run/rapid'
                      use_all_processors=True,
                      ZS_TauR=24*3600,
                      ZS_dtR=15*60,
                      ZS_TauM=365*24*3600,
                      ZS_dtM=24*3600
                      rapid_connect_file='../rapid-io/input/rapid_connect.csv
↪',
                      Vlat_file='../rapid-io/input/m3_riv.nc',
                      riv_bas_id_file='../rapid-io/input/riv_bas_id.csv',
                      k_file='../rapid-io/input/k.csv',
                      x_file='../rapid-io/input/x.csv',
                      Qout_file='../rapid-io/output/Qout.nc',
                     )

rapid_manager.run()

rapid_manager.make_output_CF_compliant(simulation_start_datetime=datetime.
↪datetime(1980, 1, 1),
                                       comid_lat_lon_z_file='../rapid-io/
↪input/comid_lat_lon_z.csv',
                                       project_name="ERA Interim Historical␣
↪flows by US Army ERDC")
```

run(*rapid_namelist_file=''*)

Run RAPID program and generate file based on inputs This will generate your rapid_namelist file and run RAPID from wherever you call this script (your working directory).

Parameters **rapid_namelist_file** (`Optional (str)`) – Path of namelist file to use in the simulation. It will be updated with any parameters added to the RAPID manager.

Linux Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(rapid_executable_location='~/work/rapid/src/rapid'
                      use_all_processors=True,
                     )

rapid_manager.update_parameters(rapid_connect_file='../rapid-io/input/rapid_
↪connect.csv',
                                Vlat_file='../rapid-io/input/m3_riv.nc',
                                riv_bas_id_file='../rapid-io/input/riv_bas_id.
↪csv',
                                k_file='../rapid-io/input/k.csv',
                                x_file='../rapid-io/input/x.csv',
                                Qout_file='../rapid-io/output/Qout.nc',
                                )

rapid_manager.update_reach_number_data()
rapid_manager.update_simulation_runtime()
rapid_manager.run(rapid_namelist_file='../rapid-io/input/rapid_namelist')
```

Linux Reservoir Forcing Flows Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(rapid_executable_location='~/work/rapid/src/rapid',
                      num_processors=4,
                      IS_for_tot=4,
                      IS_for_use=4,
                      for_tot_id_file='../rapid-io/input/dam_id.csv',
                      for_use_id_file='../rapid-io/input/dam_id.csv',
                      Qfor_file='../rapid-io/input/qout_dams.csv',
                      ZS_dtF=86400,
                      BS_opt_for=True,
                      )

rapid_manager.run(rapid_namelist_file='../rapid-io/input/rapid_namelist_
↪regular_run')
```

Windows with Cygwin Example:

```
from RAPIDpy import RAPID
from os import path

rapid_manager = RAPID(rapid_executable_location='C:/cygwin64/home/username/
↪work/rapid/run/rapid',
                      cygwin_bin_location='C:/cygwin64/bin',
                      use_all_processors=True,
                      ZS_TauR=24*3600,
                      ZS_dtR=15*60,
                      ZS_TauM=365*24*3600,
                      ZS_dtM=24*3600
                     )

rapid_input_folder = 'C:/cygwin64/home/username/work/rapid-io/input'
rapid_output_folder = 'C:/cygwin64/home/username/work/rapid-io/output'
rapid_manager.update_parameters(rapid_connect_file=path.join(rapid_input_
↪folder, 'rapid_connect.csv'),
                                Vlat_file=path.join(rapid_input_folder, 'm3_
↪riv.nc'),
                                riv_bas_id_file=path.join(rapid_input_folder,
↪'riv_bas_id.csv'),
                                k_file=path.join(rapid_input_folder, 'k.csv'),
                                x_file=path.join(rapid_input_folder, 'x.csv'),
                                Qout_file=path.join(rapid_output_folder,
↪'Qout.nc'),
                                )

rapid_manager.update_reach_number_data()
rapid_manager.update_simulation_runtime()
rapid_manager.run()
```

update_namelist_file(*rapid_namelist_file*, *new_namelist_file=None*)

Update existing namelist file with new parameters

Parameters

- **rapid_namelist_file** (*str*) – Path of namelist file to use in the simulation. It will be updated with any parameters added to the RAPID manager.
- **new_namelist_file** (*Optional[str]*) – Path to output the updated namelist file.

update_parameters (***kwargs*)

You can add or update rapid namelist parameters by using the name of the variable in the rapid namelist file (this is case sensitive).

Parameters ****kwargs** (*Optional[str]*) – Keyword arguments matching the input parameters in the RAPID namelist.

Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(
                        #ADD PARAMETERS
                     )

rapid_manager.update_parameters(rapid_connect_file='../rapid-io/input/rapid_
↪connect.csv',
                                Vlat_file='../rapid-io/input/m3_riv.nc',
                                riv_bas_id_file='../rapid-io/input/riv_bas_id.
↪csv',
                                k_file='../rapid-io/input/k.csv',
                                x_file='../rapid-io/input/x.csv',
                                Qout_file='../rapid-io/output/Qout.nc',
                                )
```

update_reach_number_data ()

Update the reach number data for the namelist based on input files.

> **Warning:** You need to make sure you set *rapid_connect_file* and *riv_bas_id_file* before running this function.

Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(
                      #ADD PARAMETERS
                      rapid_connect_file='../rapid-io/input/rapid_connect.csv
↪',
                      riv_bas_id_file='../rapid-io/input/riv_bas_id.csv',
                     )

rapid_manager.update_reach_number_data()
```

Example with forcing data:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(
                      #ADD PARAMETERS
                      rapid_connect_file='../rapid-io/input/rapid_connect.csv
↪',
                      riv_bas_id_file='../rapid-io/input/riv_bas_id.csv',
                     )

rapid_manager.update_parameters(Qfor_file=qfor_file,
                                for_tot_id_file=for_tot_id_file,
```

```
                                        for_use_id_file=for_use_id_file,
                                        ZS_dtF=3*60*60,
                                        BS_opt_for=True)

rapid_manager.update_reach_number_data()
```

update_simulation_runtime()

Updates the total simulation duration from the m3 file (Vlat_file) and the time step (ZS_TauR).

> **Warning:** You need to set the m3 file (Vlat_file) and the time step (ZS_TauR) before runnning this function.

Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(
                          #ADD PARAMETERS
                          Vlat_file='../rapid-io/input/m3_riv.csv',
                          ZS_TauR=3*3600,
                         )

rapid_manager.update_simulation_runtime()
```

2.5 Inflow from Land Surface Models

Code to use to prepare input data for RAPID from Land Surface Models (LSM) such as:

- ECMWF's ERA Interim Data
- NASA's GLDAS/NLDAS/LIS Data
- CMIP5 Data (daily VIC data available from 1950 to 2099)

2.5.1 Step 1: Retrieve Land Surface Model Runoff Output

Download the data into a local directory.

- http://apps.ecmwf.int/datasets
- http://ldas.gsfc.nasa.gov/index.php
- ftp://gdo-dcp.ucllnl.org/pub/dcp/archive/cmip5/hydro/BCSD_daily_VIC_nc/

2.5.2 Step 2: Create folders for RAPID input and output

In this instance:

```
$ cd $HOME
$ mkdir -p rapid-io/input rapid-io/output
```

2.5.3 Step 3: Create script using LSM process

Here is the API for the function to use. Follow the example to create a script to use the code such as in ~/run_lsm.py.

2.5.4 Step 4: Add RAPID files to the rapid-io/input directory

Make sure the directory is in the format [watershed name]-[subbasin name] with lowercase letters, numbers, and underscores only. No spaces!

Example:

```
$ ls /rapid-io/input
nfie_texas_gulf_region-huc_2_12

$ ls /rapid-io/input/nfie_texas_gulf_region-huc_2_12
comid_lat_lon_z.csv
k.csv
rapid_connect.csv
riv_bas_id.csv
weight_era_t511.csv
weight_nldas.csv
weight_gldas.csv
weight_lis.csv
weight_wrf.csv
weight_cmip5.csv
x.csv
```

If you have not generated these files yet, see *RAPID GIS Tools*

2.5.5 Step 5: Run the code

```
$ python ~/run_lsm.py
```

2.6 RAPIDDataset

This is a wrapper for the RAPID Qout netCDF file. Here are some basic examples for useage.

class RAPIDpy.dataset.**RAPIDDataset**(*filename*, *river_id_dimension=''*, *river_id_variable=''*, *streamflow_variable=''*, *datetime_simulation_start=None*, *simulation_time_step_seconds=None*, *out_tzinfo=None*)

This class is designed to access data from the RAPID Qout NetCDF file.

filename
str – Path to the RAPID Qout NetCDF file.

river_id_dimension
Optional[str] – Name of the river ID dimension. Default is to search through a standard list.

river_id_variable
Optional[str] – Name of the river ID variable. Default is to search through a standard list.

streamflow_variable
Optional[str] – Name of the streamflow varaible. Default is to search through a standard list.

datetime_simulation_start
: *Optional[datetime]* – This is a datetime object with the date of the simulation start time.

simulation_time_step_seconds
: *Optional[integer]* – This is the time step of the simulation output in seconds.

out_tzinfo
: *Optional[tzinfo]* – Time zone to output data as. The dates will be converted from UTC to the time zone input. Default is UTC.

Example:

```
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    #USE FUNCTIONS TO ACCESS DATA HERE
```

get_qout(*river_id_array=None, date_search_start=None, date_search_end=None, time_index_start=None, time_index_end=None, time_index=None, time_index_array=None, daily=False, pd_filter=None, daily_mode='mean'*)

This method extracts streamflow data by a single river ID or by a river ID array. It has options to extract by date or by date index.

Parameters

- **river_id_array** (`Optional[list or int]`) – A single river ID or an array of river IDs.
- **date_search_start** (`Optional[datetime]`) – This is a datetime object with the date of the minimum date for starting.
- **date_search_end** (`Optional[datetime]`) – This is a datetime object with the date of the maximum date for ending.
- **time_index_start** (`Optional[int]`) – This is the index of the start of the time array subset. Useful for the old file version.
- **time_index_end** (`Optional[int]`) – This is the index of the end of the time array subset. Useful for the old file version.
- **time_index** (`Optional[int]`) – This is the index of time to return in the case that your code only wants one index. Used internally.
- **time_index_array** (`Optional[list or np.array]`) – This is used to extract the vales only for particular dates. This can be from the *get_time_index_range* function.
- **daily** (`Optional[bool]`) – If true, this will convert qout to daily average.
- **pd_filter** (`Optional[str]`) – This is a valid pandas resample frequency filter.
- **filter_mode** (`Optional[str]`) – You can get the daily average "mean" or the maximum "max". Default is "mean".

Returns This is a 1D or 2D array or a single value depending on your input search.

Return type numpy.array

This example demonstrates how to retrieve the streamflow associated with the reach you are interested in:

```
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
```

```
river_id = 500
with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    streamflow_array = qout_nc.get_qout(river_id)
```

This example demonstrates how to retrieve the streamflow within a date range associated with the reach you are interested in:

```
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
river_id = 500
with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    streamflow_array = qout_nc.get_qout(river_id,
                                        date_search_start=datetime(1985,1,1),
                                        date_search_end=datetime(1985,2,4))
```

get_river_id_array()

This method returns the river ID array for this file.

Returns An array of the river ID's

Return type numpy.array

Example:

```
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    river_ids = qout_nc.get_river_id_array()
```

get_river_index(*river_id*)

This method retrieves the river index in the netCDF dataset corresponding to the river ID.

Returns The index of the river ID's in the file

Return type int

Example:

```
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
river_id = 53458

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    river_index = qout_nc.get_river_index(river_id)
```

get_time_array(*datetime_simulation_start=None*, *simulation_time_step_seconds=None*, *return_datetime=False*, *time_index_array=None*)

This method extracts or generates an array of time. The new version of RAPID output has the time array stored. However, the old version requires the user to know when the simulation began and the time step of the output.

Parameters

- **return_datetime** (*Optional[boolean]*) – If true, it converts the data to a list of datetime objects. Default is False.
- **time_index_array** (*Optional[list or np.array]*) – This is used to extract the datetime vales. This can be from the *get_time_index_range* function.

Returns An array of integers representing seconds since Jan 1, 1970 UTC or datetime objects if return_datetime is set to True.

Return type list

These examples demonstrates how to retrieve or generate a time array to go along with your RAPID streamflow series.

CF-Compliant Qout File Example:

```
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    #retrieve integer timestamp array
    time_array = qout_nc.get_time_array()

    #or, to get datetime array
    time_datetime = qout_nc.get_time_array(return_datetime=True)
```

Legacy Qout File Example:

```
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
with RAPIDDataset(path_to_rapid_qout,
                  datetime_simulation_start=datetime_simulation_start,
                  simulation_time_step_seconds=simulation_time_step_seconds)␣
↪as qout_nc:

    #retrieve integer timestamp array
    time_array = qout_nc.get_time_array()

    #or, to get datetime array
    time_datetime = qout_nc.get_time_array(return_datetime=True)
```

get_time_index_range(*date_search_start=None*, *date_search_end=None*, *time_index_start=None*, *time_index_end=None*, *time_index=None*)

Generates a time index range based on time bounds given. This is useful for subset data extraction.

Parameters

- **date_search_start** (*Optional[datetime]*) – This is a datetime object with the date of the minimum date for starting.
- **date_search_end** (*Optional[datetime]*) – This is a datetime object with the date of the maximum date for ending.
- **time_index_start** (*Optional[int]*) – This is the index of the start of the time array subset. Useful for the old file version.
- **time_index_end** (*Optional[int]*) – This is the index of the end of the time array subset. Useful for the old file version.
- **time_index** (*Optional[int]*) – This is the index of time to return in the case that your code only wants one index. Used internally.

Returns This is an array used to extract a subset of data.

Return type index_array

CF-Compliant Qout File Example:

```python
from datetime import datetime
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    time_index_range = qout_nc.get_time_index_range(date_search_
↪start=datetime(1980, 1, 1),
                                                    date_search_
↪end=datetime(1980, 12, 11))
```

Legacy Qout File Example:

```python
from datetime import datetime
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
with RAPIDDataset(path_to_rapid_qout,
                  datetime_simulation_start=datetime(1980, 1, 1),
                  simulation_time_step_seconds=3600) as qout_nc:

    time_index_range = qout_nc.get_time_index_range(date_search_
↪start=datetime(1980, 1, 1),
                                                    date_search_
↪end=datetime(1980, 12, 11))
```

is_time_variable_valid()

This function returns whether or not the time variable is valid.

Returns True if the time variable is valid, otherwise false.

Return type boolean

Example:

```python
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    if qout_nc.is_time_variable_valid():
        #DO WORK HERE
```

write_flows_to_csv(*path_to_output_file*, *river_index=None*, *river_id=None*, *date_search_start=None*, *date_search_end=None*, *daily=False*, *mode='mean'*)

Write out RAPID output to CSV file.

Note: Need either *reach_id* or *reach_index* parameter, but either can be used.

Parameters

- **path_to_output_file** (*str*) – Path to the output csv file.
- **river_index** (*Optional[datetime]*) – This is the index of the river in the file you want the streamflow for.
- **river_id** (*Optional[datetime]*) – This is the river ID that you want the streamflow for.

- **date_search_start** (*Optional[datetime]*) – This is a datetime object with the date of the minimum date for starting.
- **date_search_end** (*Optional[datetime]*) – This is a datetime object with the date of the maximum date for ending.
- **daily** (*Optional[boolean]*) – If True and the file is CF-Compliant, write out daily flows.
- **mode** (*Optional[str]*) – You can get the daily average "mean" or the maximum "max". Defauls is "mean".

Example writing entire time series to file:

```
from RAPIDpy import RAPIDDataset

river_id = 3624735
path_to_rapid_qout = '/path/to/Qout.nc'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    #for writing entire time series to file
    qout_nc.write_flows_to_csv('/timeseries/Qout_3624735.csv',
                               river_id=river_id,
                               )


    #if file is CF compliant, you can write out daily average

    #NOTE: Getting the river index is not necessary
    #this is just an example of how to use this
    river_index = qout_nc.get_river_index(river_id)
    qout_nc.write_flows_to_csv('/timeseries/Qout_daily.csv',
                               river_index=river_index,
                               daily=True,
                               )
```

Example writing entire time series as daily average to file:

```
from RAPIDpy import RAPIDDataset

river_id = 3624735
path_to_rapid_qout = '/path/to/Qout.nc'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    #NOTE: Getting the river index is not necessary
    #this is just an example of how to use this
    river_index = qout_nc.get_river_index(river_id)

    #if file is CF compliant, you can write out daily average
    qout_nc.write_flows_to_csv('/timeseries/Qout_daily.csv',
                               river_index=river_index,
                               daily=True,
                               )
```

Example writing entire time series as daily average to file:

```
from datetime import datetime
from RAPIDpy import RAPIDDataset
```

```
river_id = 3624735
path_to_rapid_qout = '/path/to/Qout.nc'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    # if file is CF compliant, you can filter by date
    qout_nc.write_flows_to_csv('/timeseries/Qout_daily_date_filter.csv',
                               river_id=river_id,
                               daily=True,
                               date_search_start=datetime(2002, 8, 31),
                               date_search_end=datetime(2002, 9, 15),
                               mode="max"
                               )
```

write_flows_to_gssha_time_series_ihg(*path_to_output_file*, *connection_list_file*, *date_search_start=None*, *date_search_end=None*, *daily=False*, *mode='mean'*)

Write out RAPID output to GSSHA time series ihg file

Note: See: http://www.gsshawiki.com/Surface_Water_Routing:Introducing_Dischage/Constituent_Hydrographs

Note: GSSHA project card is CHAN_POINT_INPUT

Parameters

- **path_to_output_file** (`str`) – Path to the output xys file.
- **connection_list_file** (`list`) – CSV file with link_id, node_id, baseflow, and rapid_rivid header and rows with data.
- **date_search_start** (`Optional[datetime]`) – This is a datetime object with the date of the minimum date for starting.
- **date_search_end** (`Optional[datetime]`) – This is a datetime object with the date of the maximum date for ending.
- **out_tzinfo** (`Optional[tzinfo]`) – Timezone object with output time zone for GSSHA. Default is the native RAPID output timezone (UTC).
- **daily** (`Optional[boolean]`) – If True and the file is CF-Compliant, write out daily flows.
- **mode** (`Optional[str]`) – You can get the daily average "mean" or the maximum "max". Defauls is "mean".

Example connection list file:

```
link_id, node_id, baseflow, rapid_rivid
599, 1, 0.0, 80968
603, 1, 0.0, 80967
```

Example writing entire time series to file:

```
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
connection_list_file = '/path/to/connection_list_file.csv'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    #for writing entire time series to file
    qout_nc.write_flows_to_gssha_time_series_ihg('/timeseries/Qout_3624735.ihg
↪',
                                                 connection_list_file,
                                                 )
```

Example writing entire time series as daily average to file:

```
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
connection_list_file = '/path/to/connection_list_file.csv'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    # if file is CF compliant, you can write out daily average
    qout_nc.write_flows_to_gssha_time_series_ihg('/timeseries/Qout_3624735.ihg
↪',
                                                 connection_list_file,
                                                 daily=True,
                                                 )
```

Example writing subset of time series as daily maximum to file:

```
from datetime import datetime
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
connection_list_file = '/path/to/connection_list_file.csv'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    # if file is CF compliant, you can filter by date and get daily values
    qout_nc.write_flows_to_gssha_time_series_ihg('/timeseries/Qout_daily_date_
↪filter.ihg',
                                                 connection_list_file,
                                                 date_search_

↪start=datetime(2002, 8, 31),
                                                 date_search_

↪end=datetime(2002, 9, 15),
                                                 daily=True,
                                                 mode="max"
                                                 )
```

write_flows_to_gssha_time_series_xys(*path_to_output_file*, *series_name*, *series_id*, *river_index=None*, *river_id=None*, *date_search_start=None*, *date_search_end=None*, *daily=False*, *mode='mean'*)

Write out RAPID output to GSSHA WMS time series xys file.

Parameters

- **path_to_output_file** (*str*) – Path to the output xys file.

- **series_name** (*str*) – The name for the series.
- **series_id** (*int*) – The ID to give the series.
- **river_index** (*Optional[datetime]*) – This is the index of the river in the file you want the streamflow for.
- **river_id** (*Optional[datetime]*) – This is the river ID that you want the streamflow for.
- **date_search_start** (*Optional[datetime]*) – This is a datetime object with the date of the minimum date for starting.
- **date_search_end** (*Optional[datetime]*) – This is a datetime object with the date of the maximum date for ending.
- **daily** (*Optional[boolean]*) – If True and the file is CF-Compliant, write out daily flows.
- **mode** (*Optional[str]*) – You can get the daily average "mean" or the maximum "max". Defauls is "mean".

Example writing entire time series to file:

```
from RAPIDpy import RAPIDDataset

river_id = 3624735
path_to_rapid_qout = '/path/to/Qout.nc'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    qout_nc.write_flows_to_gssha_time_series_xys('/timeseries/Qout_3624735.xys
↪',
                                                 series_name="RAPID_TO_GSSHA_
↪{0}".format(river_id),
                                                 series_id=34,
                                                 river_id=river_id,
                                                 )
```

Example writing entire time series as daily average to file:

```
from RAPIDpy import RAPIDDataset

river_id = 3624735
path_to_rapid_qout = '/path/to/Qout.nc'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    # NOTE: Getting the river index is not necessary
    # this is just an example of how to use this
    river_index = qout_nc.get_river_index(river_id)

    # if file is CF compliant, you can write out daily average
    qout_nc.write_flows_to_gssha_time_series_xys('/timeseries/Qout_daily.xys',
                                                 series_name="RAPID_TO_GSSHA_
↪{0}".format(river_id),
                                                 series_id=34,
                                                 river_index=river_index,
                                                 daily=True,
                                                 )
```

Example writing subset of time series as daily maximum to file:

```
from datetime import datetime
from RAPIDpy import RAPIDDataset

river_id = 3624735
path_to_rapid_qout = '/path/to/Qout.nc'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    # NOTE: Getting the river index is not necessary
    # this is just an example of how to use this
    river_index = qout_nc.get_river_index(river_id)

    # if file is CF compliant, you can filter by date and
    # get daily values
    qout_nc.write_flows_to_gssha_time_series_xys('/timeseries/Qout_daily_date_
↪filter.xys',
                                                 series_name="RAPID_TO_GSSHA_
↪{0}".format(river_id),
                                                 series_id=34,
                                                 river_index=river_index,
                                                 date_search_
↪start=datetime(2002, 8, 31),
                                                 date_search_
↪end=datetime(2002, 9, 15),
                                                 daily=True,
                                                 mode="max"
                                                 )
```

2.7 Postprocessing

2.7.1 Merge RAPID Output

class RAPIDpy.postprocess.**ConvertRAPIDOutputToCF**(*rapid_output_file*, *start_datetime*, *time_step*, *qinit_file=''*, *comid_lat_lon_z_file=''*, *rapid_connect_file=''*, *project_name='Default RAPID Project'*, *output_id_dim_name='rivid'*, *output_flow_var_name='Qout'*, *print_debug=False*)

Class to convert RAPID output to be CF compliant. You can also use this to combine consecutive RAPID output files into one file.

rapid_output_file
str or list – Path to a single RAPID Qout file or a list of RAPID Qout files.

start_datetime
datetime – Datetime object with the time of the start of the simulation.

time_step
int or list – Time step of simulation in seconds if single Qout file or a list of time steps corresponding to each Qout file in the *rapid_output_file*.

qinit_file
Optional[str] – Path to the Qinit file for the simulation. If used, it will use the values in the file for the flow

at simulation time zero.

comid_lat_lon_z_file
Optional[str] – Path to comid_lat_lon_z file. If included, the spatial information will be added to the output NetCDF file.

rapid_connect_file
Optional[str] – Path to RAPID connect file. This is required if *qinit_file* is added.

project_name
Optional[str] – Name of your project in the output file. Default is "Default RAPID Project".

output_id_dim_name
Optional[str] – Name of the output river ID dimension name. Default is 'rivid'.

output_flow_var_name
Optional[str] – Name of streamflow variable in output file, typically 'Qout' or 'm3_riv'. Default is 'Qout'.

print_debug
Optional[bool] – If True, the debug output will be printed to the console. Default is False.

> **Warning:** This code replaces the first file with the combined output and deletes the second file. BACK UP YOUR FILES!!!!

Example:

```
import datetime
from RAPIDpy.postprocess import ConvertRAPIDOutputToCF

file1 = "/path/to/Qout_1980to1981.nc"
file2 = "/path/to/Qout_1981to1982.nc"

cv = ConvertRAPIDOutputToCF(rapid_output_file=[file1, file2],
                            start_datetime=datetime.datetime(2005,1,1),
                            time_step=[3*3600, 3*3600],
                            project_name="NLDAS(VIC)-RAPID historical flows by US␣
↪Army ERDC",
                            )
cv.convert()
```

2.7.2 Generate qinit from past qout

RAPIDpy also creates a qinit file from a RAPID qout file. This example shows how.

RAPID.**generate_qinit_from_past_qout**(*qinit_file*, *time_index=-1*, *out_datetime=None*)
Generate qinit from a RAPID qout file

Parameters

- **qinit_file** (*str*) – Path to output qinit_file.
- **time_index** (*Optional[int]*) – Index of simulation to generate initial flow file. Default is the end.
- **out_datetime** (*Optional[datetime]*) – Datetime object containing time of initialization.

Example:

```
from RAPIDpy import RAPID

rapid_manager = RAPID(Qout_file='/output_mississippi-nfie/Qout_k2v1_2005to2009.nc
↪',
                      rapid_connect_file='/input_mississippi_nfie/rapid_connect_
↪ECMWF.csv'
                     )

rapid_manager.generate_qinit_from_past_qout(qinit_file='/input_mississippi_nfie/
↪Qinit_2008_flood.csv',
                                            time_index=10162)
```

2.7.3 Generate seasonal qinit from past qout

RAPID.**generate_seasonal_intitialization**(*qinit_file*, *datetime_start_initialization=datetime.datetime(2017, 8, 30, 14, 34, 27, 819137)*)

This creates a seasonal qinit file from a RAPID qout file. This requires a simulation Qout file with a longer time period of record and to be CF compliant. It takes the average of the current date +- 3 days and goes back as far as possible.

Parameters

- **qinit_file** (*str*) – Path to output qinit_file.
- **datetime_start_initialization** (*Optional[datetime]*) – Datetime object with date of simulation to go back through the years and get a running average to generate streamflow initialization. Default is utcnow.

This example shows how to use it:

```
from RAPIDpy.rapid import RAPID

rapid_manager = RAPID(Qout_file='/output_mississippi-nfie/Qout_2000to2015.nc',
                      rapid_connect_file='/input_mississippi_nfie/rapid_connect_
↪ECMWF.csv'
                      )

rapid_manager.generate_seasonal_intitialization(qinit_file='/input_mississippi_
↪nfie/Qinit_seasonal_avg_jan_1.csv')
```

2.7.4 Goodness of Fit

To check how well your simulation performed versus observations, these functions can help you.

RAPIDpy.postprocess.**find_goodness_of_fit_csv**(*observed_simulated_file*, *out_file=None*)

Finds the goodness of fit comparing observed and simulated flows In the file, the first column is the observed flows and the second column is the simulated flows.

Example:

```
33.5, 77.2
34.7, 73.0
```

Parameters

- **observed_simulated_file** (*str*) – Path to the csv file with the observed and simulated flows.
- **out_file** (*Optional[str]*) – Path to output file. If not provided, it will print to console.

Example:

```
from RAPIDpy.postprocess import find_goodness_of_fit_csv

find_goodness_of_fit_csv('/united_kingdom-thames/flows_kingston_gage_noah.csv')
```

RAPIDpy.postprocess.**find_goodness_of_fit**(*rapid_qout_file*, *reach_id_file*, *observed_file*, *out_analysis_file*, *daily=False*, *steps_per_group=1*)

Finds the goodness of fit comparing observed streamflow in a rapid Qout file with simulated flows in a csv file.

Parameters

- **rapid_qout_file** (*str*) – Path to the RAPID Qout file.
- **reach_id_file** (*str*) – Path to file with river reach ID's associate with the RAPID Qout file. It is in the format of the RAPID observed flows reach ID file.
- **observed_file** (*str*) – Path to input csv with with observed flows corresponding to the RAPID Qout. It is in the format of the RAPID observed flows file.
- **out_analysis_file** (*str*) – Path to the analysis output csv file.
- **daily** (*Optional[bool]*) – If True and the file is CF-Compliant, it will compare the *observed_file* with daily average flow from Qout. Default is False.

Example with CF-Compliant RAPID Qout file:

```
import os
from RAPIDpy.postprocess import find_goodness_of_fit

INPUT_DATA_PATH = '/path/to/data'
reach_id_file = os.path.join(INPUT_DATA_PATH, 'obs_reach_id.csv')
observed_file = os.path.join(INPUT_DATA_PATH, 'obs_flow.csv')

cf_input_qout_file = os.path.join(COMPARE_DATA_PATH, 'Qout_nasa_lis_3hr_20020830_
↪CF.nc')
cf_out_analysis_file = os.path.join(OUTPUT_DATA_PATH, 'cf_goodness_of_fit_results-
↪daily.csv')
find_goodness_of_fit(cf_input_qout_file, reach_id_file, observed_file,
                     cf_out_analysis_file, daily=True)
```

2.8 RAPID to GSSHA

It is possible to use RAPID streamflow as an overland flow boundary condition to the Gridded Surface Subsurface Hydrologic Analysis (GSSHA) model.

2.8.1 What is GSSHA?

GSSHA is a physically-based, distributed hydrologic model. GSSHA is developed and maintained by Coastal and Hydraulics Laboratory (CHL) which is a member of the Engineer Research & Development Center of the United

States Army Corps of Engineers (USACE).

Note: For more information about GSSHA please visit the the gsshawiki .

2.8.2 Tutorial

There are two ways to input RAPID as a boundary condition for GSSHA. One is to connect the GSSHA stream network link and node to the RAPID river ID and generate the IHG file. The other is to generate an XYS timeseries file and add it to the netork using WMS.

Method 1: Generate IHG File

Step 1.1: Look at Stream Network in WMS to find Link & Node

1. Open GSSHA project in WMS
   - Switch to 2-D Grid Module
   - In the top menu: *GSSHA -> Open Project File*
2. Turn on *Stream Link Numbers* Display
   - In the top menu: *Display -> Display Options*
   - Select *Map Data* in top left box
   - In the center box, make sure *Stream Link Numbers* is checked under the Arcs subsection.
3. Determine the Link ID by looking on model.

Step 1.2: Connect RAPID river ID to GSSHA Link & Node and Generate IHG

RAPIDDataset.**write_flows_to_gssha_time_series_ihg**(*path_to_output_file*, *connection_list_file*, *date_search_start=None*, *date_search_end=None*, *daily=False*, *mode='mean'*)

Write out RAPID output to GSSHA time series ihg file

Note: See: http://www.gsshawiki.com/Surface_Water_Routing:Introducing_Dischage/Constituent_Hydrographs

Note: GSSHA project card is CHAN_POINT_INPUT

Parameters

- **path_to_output_file** (*str*) – Path to the output xys file.
- **connection_list_file** (*list*) – CSV file with link_id, node_id, baseflow, and rapid_rivid header and rows with data.

- **date_search_start** (*Optional[datetime]*) – This is a datetime object with the date of the minimum date for starting.
- **date_search_end** (*Optional[datetime]*) – This is a datetime object with the date of the maximum date for ending.
- **out_tzinfo** (*Optional[tzinfo]*) – Timezone object with output time zone for GSSHA. Default is the native RAPID output timezone (UTC).
- **daily** (*Optional[boolean]*) – If True and the file is CF-Compliant, write out daily flows.
- **mode** (*Optional[str]*) – You can get the daily average "mean" or the maximum "max". Defauls is "mean".

Example connection list file:

```
link_id, node_id, baseflow, rapid_rivid
599, 1, 0.0, 80968
603, 1, 0.0, 80967
```

Example writing entire time series to file:

```
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
connection_list_file = '/path/to/connection_list_file.csv'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    #for writing entire time series to file
    qout_nc.write_flows_to_gssha_time_series_ihg('/timeseries/Qout_3624735.ihg',
                                                  connection_list_file,
                                                  )
```

Example writing entire time series as daily average to file:

```
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
connection_list_file = '/path/to/connection_list_file.csv'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    # if file is CF compliant, you can write out daily average
    qout_nc.write_flows_to_gssha_time_series_ihg('/timeseries/Qout_3624735.ihg',
                                                  connection_list_file,
                                                  daily=True,
                                                  )
```

Example writing subset of time series as daily maximum to file:

```
from datetime import datetime
from RAPIDpy import RAPIDDataset

path_to_rapid_qout = '/path/to/Qout.nc'
connection_list_file = '/path/to/connection_list_file.csv'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    # if file is CF compliant, you can filter by date and get daily values
    qout_nc.write_flows_to_gssha_time_series_ihg('/timeseries/Qout_daily_date_
↪filter.ihg',
```

```
                                                   connection_list_file,
                                                   date_search_start=datetime(2002,␣
↪8, 31),
                                                   date_search_end=datetime(2002, 9,
↪ 15),
                                                   daily=True,
                                                   mode="max"
                                                   )
```

Method 2: Generate XYS File

Step 2.1: Generate XYS File

RAPIDDataset.**write_flows_to_gssha_time_series_xys**(*path_to_output_file*, *series_name*, *series_id*, *river_index=None*, *river_id=None*, *date_search_start=None*, *date_search_end=None*, *daily=False*, *mode='mean'*)

Write out RAPID output to GSSHA WMS time series xys file.

Parameters

- **path_to_output_file** (*str*) – Path to the output xys file.
- **series_name** (*str*) – The name for the series.
- **series_id** (*int*) – The ID to give the series.
- **river_index** (*Optional[datetime]*) – This is the index of the river in the file you want the streamflow for.
- **river_id** (*Optional[datetime]*) – This is the river ID that you want the streamflow for.
- **date_search_start** (*Optional[datetime]*) – This is a datetime object with the date of the minimum date for starting.
- **date_search_end** (*Optional[datetime]*) – This is a datetime object with the date of the maximum date for ending.
- **daily** (*Optional[boolean]*) – If True and the file is CF-Compliant, write out daily flows.
- **mode** (*Optional[str]*) – You can get the daily average "mean" or the maximum "max". Defauls is "mean".

Example writing entire time series to file:

```
from RAPIDpy import RAPIDDataset

river_id = 3624735
path_to_rapid_qout = '/path/to/Qout.nc'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    qout_nc.write_flows_to_gssha_time_series_xys('/timeseries/Qout_3624735.xys',
                                                   series_name="RAPID_TO_GSSHA_{0}".
↪format(river_id),
                                                   series_id=34,
```

```
                                                    river_id=river_id,
                                                    )
```

Example writing entire time series as daily average to file:

```
from RAPIDpy import RAPIDDataset

river_id = 3624735
path_to_rapid_qout = '/path/to/Qout.nc'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    # NOTE: Getting the river index is not necessary
    # this is just an example of how to use this
    river_index = qout_nc.get_river_index(river_id)

    # if file is CF compliant, you can write out daily average
    qout_nc.write_flows_to_gssha_time_series_xys('/timeseries/Qout_daily.xys',
                                                  series_name="RAPID_TO_GSSHA_{0}".
↪format(river_id),
                                                  series_id=34,
                                                  river_index=river_index,
                                                  daily=True,
                                                  )
```

Example writing subset of time series as daily maximum to file:

```
from datetime import datetime
from RAPIDpy import RAPIDDataset

river_id = 3624735
path_to_rapid_qout = '/path/to/Qout.nc'

with RAPIDDataset(path_to_rapid_qout) as qout_nc:
    # NOTE: Getting the river index is not necessary
    # this is just an example of how to use this
    river_index = qout_nc.get_river_index(river_id)

    # if file is CF compliant, you can filter by date and
    # get daily values
    qout_nc.write_flows_to_gssha_time_series_xys('/timeseries/Qout_daily_date_
↪filter.xys',
                                                  series_name="RAPID_TO_GSSHA_{0}".
↪format(river_id),
                                                  series_id=34,
                                                  river_index=river_index,
                                                  date_search_start=datetime(2002,␣
↪8, 31),
                                                  date_search_end=datetime(2002, 9,
↪ 15),
                                                  daily=True,
                                                  mode="max"
                                                  )
```

Step 2.2: Add XYS File in WMS

In the Distributed Hydrology section, go to *Overland Flow Boundary Conditions in GSSHA (Tutorial 55)* at http://www.aquaveo.com/software/wms-learning-tutorials

Here is a direct link to the document: http://wmstutorials-10.1.aquaveo.com/55%20Gssha-Applications-OverlandBoundaryConditions.pdf

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Index

Q

R

S

T

U

W