SWG Instruction Manual

Introduction

This manual describes the operation and design of the stochastic weather generator (SWG) developed by Bodeker Scientific for the SLMACC climate shock vulnerability project. While this document focuses on the technical elements of the project, the theoretical basis for the design is described in a complementary document. All code is written in the python programming language and only a single script, SWG_Final.py is required.

Operation

NOTE: The first time the code is run on your system the yaml file will need to be reconfigured to appropriate directories. For full detail on configuring the yaml file please read the following section.

Once the yaml file has been set up operation of the code is very simple. Just boot any terminal with python in the directory of the SWG_Final.py file and enter the following command:

$ python SWG_Final.py C:\EXAMPLE\DIRECTORY\YAML_FILE.yaml

Where the appropriate directory and yaml file name are used to identify the file. The terminal should regularly print output and should aid in the identification of any errors. Almost all of these errors should be resolvable by editing the yaml file.

Requirements to run code

There are several dataset/files required to run the SWG:

- Yaml Parameter File: To adjust the various parameters used in the code a yaml file is used. By using this file no modifications to the code base should be required, all changes instead should be applied through the yaml file.
- VCSN data: The VCSN data is an interpolated weather record over New Zealand. From this dataset we need each of the variables that will be simulated by the stochastic weather generator. For development, data from 1972 to 2019 for a particular subset was used but it would be possible to extend this range. All of the VCSN files should use the

naming convention “vcsn_YYYY0101-YYYY1231_for-Komanawa.nc” and exist in one folder named “SLMACC-Subset”. The code is only built to take entire years.
- Southern Hemisphere Model Temperature data: A long term temperature trend is needed to adjust the VCSN data. Currently, this is based on taking the average of several historic model runs and should be included with code as SHTemps.dat
- VCSN classifications: To simulate relevant storylines the stochastic weather generator requires a method to identify the VCSN data in a way compatible with the storylines. This should take the form of a csv file that includes the classifications of all of the VCSN data that is being used. Note this file needs to be formatted in a particular matter described in detail below. We have included a script (fix_csv.py) which can convert files to the appropriate format.
- VCSN Storyline: The storyline that will be simulated by the SWG. Ideally, this is provided as a CSV file.

Additionally, the following python libraries are required: numpy, netcdf4, OS, scipy, random, pandas, math, yaml, warnings, sys.

Yaml File

The Yaml file is made up of several different parameters. Every parameter must be given in a format that will result in a python dictionary ie. Variable_name: Variable_value with one entry per line. This file can simply be edited as if it were a text file to change the necessary parameters. This subsection describes what each parameter does:

- base_directory: The directory which contains the folder “SLMACC-Subset” which contains all the VCSN files. Note that this will also serve as a location where partially processed output will be saved which significantly speeds up repeat simulations
- SH_model_temp_file: The full filename (including directory!!!) of the Southern Hemisphere Model Temperature data file
- VCSN_Class_file: The full filename (including directory!!!) of the VCSN classifications data file
- story_line_filename: The full filename (including directory!!!) of the storyline that will be simulated
- lat_val: An float value which indicates the latitude of the location of the simulation
- lon_val: An float value which indicates the longitude of the location of the simulation
- number_of_simulations: The requested number of simulations of the given storyline
- month_scale_factor: An integer value which scales the lengths of each month. Useful for generating large amounts of data for statistical purposes otherwise set to 1.
- simulation_savename: The filename (including directory!!!) that serves as the basis for the model output save name. Flags indicating the simulation number will be added to this when the data is saved so do not include an extension.
- netcdf_save_flag: A boolean value that indicates if netcdf copies of the data should be saved.

- Extra_station_flag: A boolean value that indicates if the SWG is just running for the primary location or if additional points will also be simulated
- Extra_sim_savename: The filename (including directory!!!) that serves as the basis for the model output save name for all of the extra stations simulated. Flags indicating the simulation number and which extra point it is will be added to this when the data is saved so do not include an extension.
- Extra_site_lat: A list of float values which indicate each of the latitudes at which the additional simulations should be conducted
- Extra_site_lon: A list of float values which indicate each of the longitudes at which the additional simulations should be conducted

VCSN Classifications and Storylines

These two files control how the SWG handles the climate buckets. The classifications file is used to “train” the SWG and needs to be a CSV with the following columns: year, month, temp_class and precip_class. The column year contains integer values for the given year and the column month contains three letter strings describing the month (ie Jan, Feb ...). The temp_class and precip_class columns are short strings that describe the climate buckets and can have the values ‘H’ (Hot), ‘AT’ (Average) and ‘C’ (Cold) for temp_class and ‘W’ (Wet), ‘AP’ (Average) and ‘D’ (Dry) for precip class. Ultimately, the format should be as bellow:

year	month	temp_class	precip_class
1972	Jan	H	D
1972	Feb	AT	D

With one row for each historic month classified. Note that the files using -1,0 and 1 to classify this data can be converted to this format with the fix_csv.py script which produces a file in the same directory with a “_fixed” suffix added. This script can be run in the terminal with the following command:

> Python fix_csv.py C:\EXAMPLE\DIRECTORY\csv_file.csv

The VCSN Storylines file has a similar format. It should have three columns: month, temp_class and precip_class. The only major difference to the classification VCSN is that the value ‘all’ is accepted in all of the columns. This makes the data month/climate bin agnostic. For example using an ‘all’ value for a month means that instead of pulling values from a given month it pulls from the yearwide values. If ‘all’ is used with temp_class and precip_class it must be used for both at the same time.

Design Details

Overview

The code for the SWG is relatively complex and features many nested function calls. In this section I will describe as best I can the design structure of the code. This is to provide as best I can any useful information if anyone decides to edit or further develop the code as it currently stands. First I will briefly describe the overall structure of the code in the overview subsection and then I will move onto more specific descriptions of the model components as required. The overall structure of the code can be described as follows:

- First pull in the parameters from the yaml file
- Next load in the storyline files with the function *Storyline_loader*
- Begin the main simulation loop which iterates for each month simulated
    - Either do all of the required processing or load in preprocessed data through the function *data_load_or_make*
    - Run the stochastic weather generator to generate the month of output through the function *SWG*
- Save the generated stochastic weather output as a set of numpy arrays and if it is requested in the yaml file as a netcdf as well. Saving as netcdf is handled by *netcdf_saver*
- If any extra points for simulation have been identified, simulate those as well. This is done through a second set of loops that iterate first through all the extra points and then all the months of simulation
    - Load in earlier prepared data with the function *data_load_or_make*
    - Generate the required transition histograms with the function *Extra_site_loader*
    - Remaining calculations are handled in loop without any complex function calls
- Save the extra site data as a set of numpy arrays and if it is requested in the yaml file as a netcdf as well. Saving as netcdf is handled by *netcdf_saver*

While most of these functions are relatively simple to understand, two may require a more detailed explanation: *data_load_or_make* and *SWG*

For debugging purposes it may also be important to state that internally the code handles months and climate buckets with integer values as follows:

For months: 0='all' 1='Jan' 2='Feb' ..... 12='Dec'

For buckets: 0='all' 1= (C,W) 2= (C,A) 3= (C,D)
4= (A,W) 5= (A,A) 6= (A,D)
7= (H,W) 8= (H,A) 9= (H, D)

Data_load_or_make

The *Data_load_or_make* function either loads in a pre-processed file or handles all of the processing required to make the file. This processing takes the data from the VCSN netcdf files to the variables that are required to run the *SWG.* Additionally, the function *extra_site_loader* is based strongly on the design of *Data_load_or_make. Data_load_or_make* has the following structure:

- First checks if the requested data has already been calculated and saved to disk. If it has it is loaded and returned as the function output otherwise the function will generate the data.
- Next the function *base_data_detrender* is called. This is responsible for loading in the data, reducing it to the appropriate point and then removing the trend from the data. For the underlying explanation of the methodology, look at the companion document. The exact implementation is as follows:
    - First the VCSN netcdf files are loaded with *data_loader* and the appropriate variables are accessed with *var_stripper*
    - This data is then reduced to the single relevant point with *point_stripper*
    - Next the Southern Hemisphere Model Temperature data is loaded in with *SH_model_load_in*
    - The curve used to fit the data is calculated with *special_curve_fit* and then the data is adjusted with *Line_based_correction*
- The length of precipitation streaks in the data is then calculated using *PR_streak_calc*
- The data is then filtered down to the requested month and climate bucket using *Data_Filter*. This is a relatively complicated function so it is explained below:
    - First the data is reduced to month specific data using the function *Month_Filtering*
    - Next the VCSN classifications file is loaded and the relevant data extracted with *regime_data_loader*
    - The data is then reduced to the identified climate buckets using *regime_filter* and *Regime_filtering*
    - Note that due to various issues the precipitation data is handled independently to the other variables but using the same functions. Additionally, to avoid data quality issues the precipitation values are normally only filtered by precipitation regime and not by temperature.
- The required precipitation data for the stochastic weather generator is then calculated with the functions *SWG_Reg_Precipitation* and *PR_CDF_gen*
- The wet and dry CDFs for all of the other variables are then calculated with the function *cdf_builder*
- The A and B matrices are calculated with the function *A_B_builder*
- Finally the data is saved as an npz file that can be loaded in the future to avoid duplicate processing. This npz file is specific to the vcsn grid cell and the month/climate bin combination. This means that each grid cell - climate bin - month combination only needs to be processed once.

Note: The output npz files are saved to the folder Output in the specified base_directory from the yaml file. These are saved in subfolders specific to each VCSN gridcell. These files are potentially useful for further analysis based on the VCSN data. They include the detrended and filtered VCSN data, various CDFs, the A and B matrices, precipitation streaks and various other required observational data

SWG

The *SWG* function contains the simulation processes used to generate the stochastic weather generator output. This works following the process outlined in the companion document. It is implemented as follows:

- First the appropriate variables are created so that they can be updated each run
- Next the function *Does_it_change_state* is used to determine if the precipitation remains in the same state (raining or not raining)
- If it is determined to be raining, the rainfall amount is calculated by indexing into the precipitation cdf with a random number using the function *percent2cdf*
- Next the Z vector is calculated. This uses the Z values from t-1, the A and B matrices and the epsilon vector. The Z values from t-1 are either taken from the previous iteration or for the first day given as input to *SWG*. The A and B matrices are always given as input to *SWG.* The epsilon vector is calculated with the *numpy.random.multivariate_normal* function
- The Z-values are converted to output variables first by converting the z-value to a percentile with the function *zptile* and then indexing into the appropriate CDF with the function *percent2cdf*

Output

The files generated by the SWG are saved as either a .npz or .nc depending on the settings chosen in the YAML file. For analyzing the simulation output I would recommend saving the data as netcdf files. The netcdf files have the following variables:

- *day* : A vector of integer values that simply give the days since the first day of simulation
- *Month* : The numeric value for the month simulated i.e. 1 for Jan, 2 for Feb, etc...
- *Tmax* : Simulated daily maximum temperature values in kelvin
- *Tmin* : Simulated daily minimum temperature values in kelvin
- *PR_A* : Simulated precipitation values in kg m^{-2} s^{-1} (to match VCSN)
- *RSDS* : Simulated surface downwelling shortwave radiation values in W m^{-2}
- *PEV* : Simulated potential evapotranspiration values in kg m^{-2}

Limitations

Several established limitations exist with the current form of the SWG:

- When using the multipoint simulation mode, coherence is not guaranteed beyond two points. For more detail on this point read the companion document for a brief discussion.
- When using the multipoint simulation mode, there is currently no ability to feed each of the points an independent storyline or classification and determine how that changes the results. While this could be potentially useful, it was considered to provide too little benefit to be worth including.
- When using the multipoint simulation mode, the degree of variable coherence is less than for the primary station. It may be possible to address this by recalculating A and B matrices for each extra station and then including it in the calculation.
- Currently the simulations do not produce leap years.
- While it is possible to feed in storlines that skip over months, it may lead to some unusual behaviour in the first few days after the skip.
- There is no guarantee that the simulation output will meet the storyline definition of a given month. For instance if a H AP January is requested the simulation works stochastically based on prior H AP Januarys. Due to the stochastic nature of the simulation this can lead to a situation where the simulation produces less than 10 days with T_{max} greater than or equal to 25°C. This would mean that it is technically an AT month not a H month. For the majority of purposes this is fine but if any re-classification is used on model output the user should be aware of this bias.
- Due to the fact empirical CDFs are used in place of analytical fits, this approach is unable to produce values larger than the largest observed value.