**Overview for hurricane analysis project**

# **Summary**

The *hurricane\_analysis* repository contains Python modules written to tabularize river gauge data, extract statistics from before and after a hurricane impact period and produce graphical representation of the extracted statistics. This document provides an overview of the functionality and design rational for the code. For specifics on using each module, refer to the docstrings and comments within the module.

# **Main Modules**

The master module is *full.py*, which encapsulates the functionality of the four main modules. These four modules are described below in the order that they are meant to be run. The call lines for modules that that come early in the sequence may be commented out in *full.py* if their output is acceptable. Each subsequent module uses the output from the previous module as input, and each module contains several variables that can be modified at the head of the file if desired.

Also note that there are several supporting modules that define functions used in the main module but do not execute any code themselves.

## processing.py

This module takes in csv formatted gauge data as well as a folder containing a hierarchy of PRISM rain data. The gauge data is optionally detrended (either for linear or sinusoidal signals, depending on parameter specification) and then joined to rain gauge data before being written out as a csv.

## analysis.py

This module performs several functions.

1. Calculates the “true” date of the onset of a storm
   1. Though the landfall of each storm is explicitly defined, the actual date that the storm reaches a gauge is not. The module calculates the date of the storm onset as the first date that the daily rainfall exceeds a defined threshold within a user-defined window around landfall. If the rainfall never exceeds the minimum threshold or if there is no rain data, it is assumed that the effect is the date of landfall.
2. Calculates the typical standard deviation for each parameter for each gauge
   1. This “typical” standard deviation is used to adjust the sensitivity pre-effect window splitting. This functionality is encapsulated in the *typical\_stddev()* function, defined in the *read.py* module.
3. Calculates the pre-effect window
   1. Finds the period of time before storm impact that the parameter is relatively stable. This is done by breaking a chunk of data before the storm impact into linear segments. The linear segment right before the storm onset is the pre-effect window. This functionality is encapsulated in the *get\_preeffect\_window()* function, defined in the *read.py* module.
4. Calculates the mean and standard deviation of the pre-effect window
   1. The mean +/- the stddev is used as the range of values which is considered to be within normalcy for the time period before the storm. If the values after the storm rise above or below (depending on the parameter) then the storm is considered to have perturbed the system. This functionality is encapsulated in the *analyze\_window()* function, defined in the *read.py* module.

Once these functions have been performed, the module writes the results as a single csv for each parameter. Each entry in a csv relates a gauge and storm to that storm’s impact at the gauge, including the number of days that the system was perturbed.

## get\_effect\_period.py

This module calculates the length of a storm’s impact on each parameter at each gauge. By using the range of “normal” values as calculated in analysis.py, the module checks each parameter to see if the parameter is perturbed outside of the normal/expected range after the onset of the storm. This functionality is encapsulated in the *get­­\_effect()* function, defined in the *effect\_tools.py* module. The rules for calculating the length of effect are highly nuanced and parameterized; please refer to the documentation for *get\_effect()* to understand how it is calculated. However, overall the function works by checking to see when a parameter exceeds the expected range and the counts how many days it takes to fall back within the expected range. Parameterization affects how edge cases are handled, such as if a parameter seems to be falling back to the normal range but it perturbed by another storm or if the effect takes too long to occur.

## resolve.py

This module visualizes the output of the previous modules as line plots.