`markov_populator.py` Documentation

September 14, 2017

1 Introduction

This document serves to explain the algorithm behind how the script `markov_populator.py` functions and how a user may use it to produce scenarios. This script lives within the `GOSM` library and the proper installation of the most recent version of `GOSM` will correctly install this script. Note it may be necessary to run `python setup.py install` again if you do not have the most recent version installed.

2 Algorithm

This section will serve as a brief description how the script actually produces scenarios. It functions by first classifying given historic data on forecasts, actuals, and errors, each historic date-time into a specific state S_i. The state will generally just be an interval that a certain field is in. For example if we have that the error at a given time is 125, then the corresponding state might be $[100, 150]$. It might also be preferred to specify the state by quantiles in which case the state might be $[0.2, 0.4]$. In the simplest case, the state will just be an error interval, but in separate tests, it may be useful to include the forecast interval or the forecast derivative interval in the state. Then given this mapping of date-times to states produces a transition matrix A where A_{ij} is the frequency with which state S_j follows state S_i. Empirically we take this as the probability with which one state will proceed another.

For each scenario on a specific day, we choose a start state for the Markov Chain. To do this, we categorize the first hour of that day based on the forecast into a certain state and pick a random sample from the historically observed states which fall into the same forecast category. Then using this start state, we transition from state to state based on the probabilities in the transition matrix until we have a state for all 24 hours of the day.

Using this sequence of states, we then can convert this into an 24-error vector by sampling from an empirical distribution on the error interval of the state. This error is then added to the daily forecast which then is truncated at 0 and the capacity of the source to complete the scenario.

3 Input Files

This script only requires two input files, a data file and an options file. The data file is structured the same as is used for `prescient` and is described here for good measure. These files are described in turn.

Once you have both of these, with an appropriately structured options file, the script can be executed with the command:

```
runner.py <options_file>
```

where `<options_file>` is replaced by the name of the file.

3.1 Data Files

Before the data can be used in the scripts, it must be processed and formatted in a manner the programs expect. This entails creating two separate csv files, one of which contains actual power data and the other one forecast data (also historic forecast data). These files will from now on be referred to as the actuals data file and forecasts data file, respectively. Both of these files must contain as the first column datetimes in the 'YYYY-MM-DD HH:MM' format (note that other formats may be recognized as well) with an appropriate header, e.g. 'datetimes'. The columns containing the data must be named accordingly to the type of data (i.e. 'forecasts' or 'actuals', respectively). Any additional columns will be ignored.

In general, a user will have multiple data files to work with which can contain load data, solar generation data, or wind generation data. All of these files should be formatted in the manner described above. An example snippet from a properly formatted file is shown below.

```
datetimes,forecasts,actuals
2016-07-01 06:00,160.15,150
2016-07-01 07:00,710.8,629
2016-07-01 08:00,1536.78,1241
2016-07-01 09:00,2103.23,2020
```

Figure 1: Example file structure of data file

3.2 Options File

The options file specifies how you may configure behavior of the script to generate scenarios differently. Any option's description can be found using the command `python markov_populator.py -h`. We discuss the most common options here. We begin by presenting the most basic example of an options file in Figure 3.2.

```
command/exec markov_populator.py
--power-source wind.csv
--start-date 2013-01-01
--end-date 2013-01-31
--output-directory markov_output
--number-of-scenarios 100
```

Figure 2: A basic example of an options file

The first thing we note is that all options file used for executing this script must begin with the following line verbatim:

```
command/exec markov_populator.py
```

This specifies to the script `runner.py` which program to actually run.

Also contained in this script are required options for every run of `markov_populator.py`. Namely, these are `--power-source` which specifies the data file which contains forecasts and actuals, `--start-date` and `--end-date` which specify the date range for which you would like to produce scenarios, and `--output-directory` which specifies where to store all the output files.

In addition, this specifies the number of scenarios to generate with the `--number-of-scenarios` option. All other options are left to defaults. This means that the exact specification for what a state is will simply be which interval the recorded error is in. Put more precisely, the separate error states will be intervals of length 100 and the entire set is given by $\{[0, 100], [100, 200], \ldots\}$. This specific length can be adjusted with the `--error-bin-size` argument. Note that the limits of the intervals will always be multiples of this size option.

An example where the user expresses more control over the what a state actually is can be seen in Figure 3. Note that here, the user specifies that they would like to use quantile intervals instead of raw energy intervals with the `--use-error-quantiles` option. The user also specifies the exact width of the intervals with the `--error-quantile-size` option. This means the states will be quantile intervals $\{[0, 0.1], [0.1, 0.2], \ldots, [0.9, 1]\}$. If the user wants more control over the exact quantiles that determine the state, then he may use the `explicit-error-quantiles` option with a string specifying the quantiles in the form `0,0.1,0.9,1`.

```
command/exec markov_populator.py
--power-source wind.csv
--start-date 2013-01-01
--end-date 2013-01-31
--output-directory markov_output
--use-error-quantiles
--error-quantile-size 0.1
--consider-forecasts
--forecast-bin-size 100
--number-of-scenarios 100
--source-name wind
--allow-multiprocessing
```

Figure 3: A more involved example of an options file

Note that the user also includes forecasts in the state description with the option `--consider-forecasts`. This just includes the interval the forecast is in (each of size 100) in the state information. After generating the state walk, this will not use the forecast state though to compute

the scenario, only the error state. The forecast state is just used for computing the random walk itself. There are options `--forecast-bin-size`, `--use-forecast-quantiles`, `--forecast-quantile-size`, `--explicit-forecast-quantiles` and similar options for derivatives (replace 'forecast' with 'derivative' in the option) for more control over what a state is.

This options file also specifies the name of the source with `--source-name`. This has no effect on the computation and only affects names on the plots and data files generated.

The option `--allow-multiprocessing` enables parallelism over the days of scenario generation.

A final example of an options file is shown in Figure 4. The only new option used here is the `--capacity` option which specifies an upper bound at which the scenarios will be truncated.

```
command/exec markov_populator.py
--power-source wind.csv
--start-date 2013-01-01
--end-date 2013-01-31
--output-directory markov_output
--use-error-quantiles
--error-quantile-size 0.1
--consider-derivatives
--explicit-derivative-quantiles 0,0.5,1
--number-of-scenarios 100
--capacity 4000
--source-name wind
--allow-multiprocessing
```

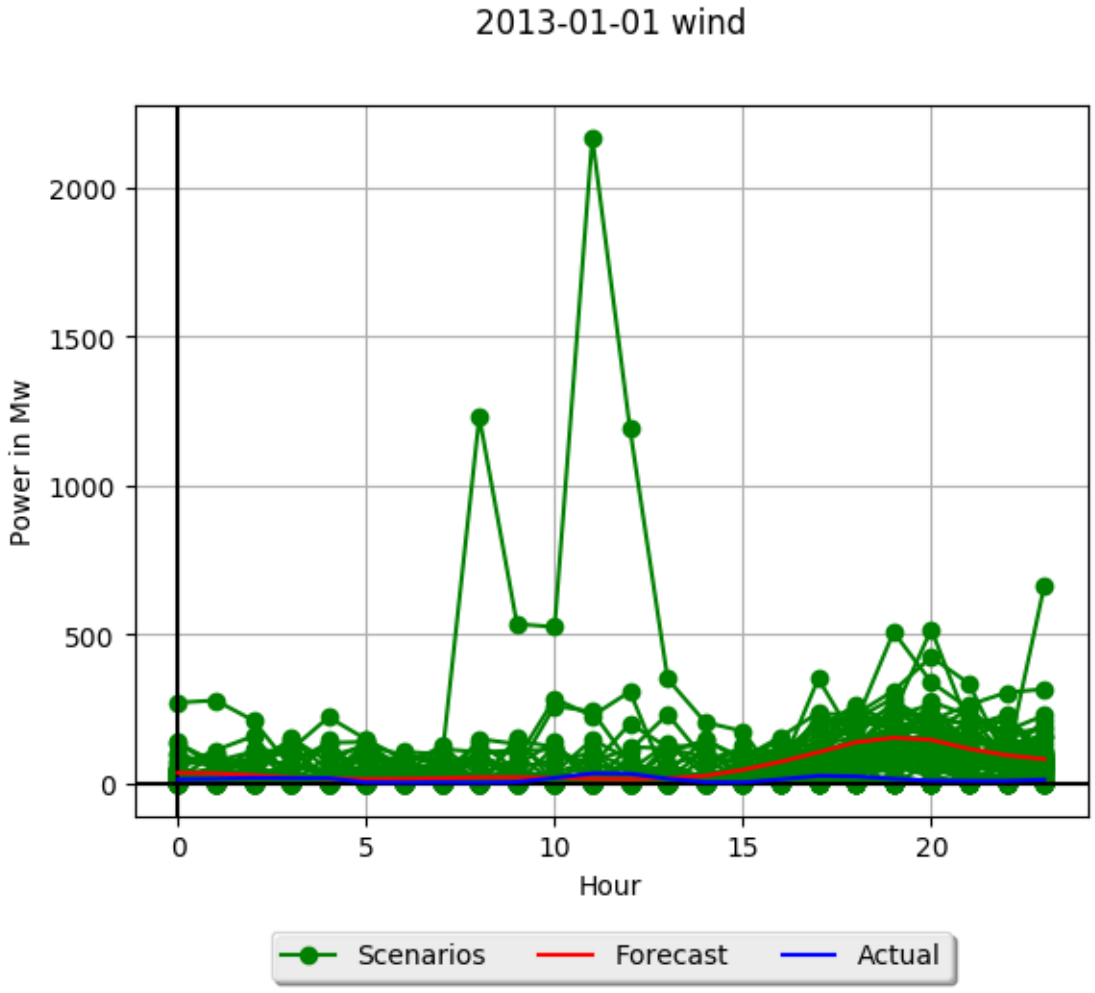
Figure 4: A third options file

4 Output Files

For each day of scenario generation, inside the output directory a directory will be created which will contain two files. The first of which is a csv file with all produced scenarios and the second a plot of scenarios.

Examples of each follow:

```
,wind: actuals,wind: forecasts,wind: Scenario 0,wind: Scenario 1
Probability,1,1,0.5,0.5
2013-01-01 00:00:00,13,35,7,0
2013-01-01 01:00:00,15,33,0,0
2013-01-01 02:00:00,18,27,0,0
2013-01-01 03:00:00,17,19,0,0
2013-01-01 04:00:00,17,16,0,0
2013-01-01 05:00:00,3,14,0,0
2013-01-01 06:00:00,2,15,0,0
2013-01-01 07:00:00,3,17,20,0
2013-01-01 08:00:00,2,20,0,0
2013-01-01 09:00:00,4,20,0,0
2013-01-01 10:00:00,18,18,0,0
2013-01-01 11:00:00,33,15,0,0
2013-01-01 12:00:00,32,14,0,0
2013-01-01 13:00:00,15,17,0,0
2013-01-01 14:00:00,4,25,0,0
2013-01-01 15:00:00,3,45,0,0
2013-01-01 16:00:00,13,72,25,7
2013-01-01 17:00:00,25,104,63,48
2013-01-01 18:00:00,23,138,95,47
2013-01-01 19:00:00,15,153,142,88
2013-01-01 20:00:00,9,146,128,96.4809592404
2013-01-01 21:00:00,8,116,84,8
2013-01-01 22:00:00,8,94,40.506517496,66
2013-01-01 23:00:00,12,81,52,47
```


2013-01-01 wind
Power in Mw
2000
1500
1000
500
0
0
5
10
15
20
Hour
Scenarios
Forecast
Actual