# Generating scenarios in GOSM for multiple sources using copulas

Simon Strotmann, David L. Woodruff

October 1, 2018

# Contents

# 1 Introduction

In this brief report, we are describing how we use copulas in a Markov Chain Random Walk to generate power scenarios for the next planning period for multiple sources. This method is part of the *GOSM* software package, which is part of the *prescient* software package itself.

All of the code for implementing this method can be found in the *prescient_gosm* repository in the directory *prescient_gosm/markov_chains/markov_populator.py* and in the *statdist* repository in the directory *statdist/base_distribution*.

# 2 How to generate scenarios with multiple sources

## 2.1 Option files

To run GOSM with multiple sources, you will need a couple of option files. The first one is a *populator-file*, which is used to run the *populator.py* script. You have to add the following options in the *populator-file*:

- **start-date & end-date**: These dates determine the period for which scenarios are created. Both must be in the format of YYYY-MM-DD-HH:MM.
- **load-scaling-factor**: A float value for the load scaling factor.
- **output-directory**: The path that stores the output.
- **scenario-creator-options-file**: The options file for the scenario creator.
- **sources-file**: The sourcelist file.
- **allow-multiprocessing**: Determines if multiprocessing is used or not.
- **traceback**

According to the populator file, you also have to provide an options file for the scenario creator as well. It has to contain the following options:

- **use-markov-chains**: Determines if the Markov Chain method is used for generating the scenarios. If this option is set, Markov Chains will be used. (This will be the default method in GOSM 3.0)
- **copula-random-walk**: Determines if copulas are used for the random walk. Can only be set, if use-markov-chains is also set. For multiple sources, this is a must.
- **planning-period-length**: The length for the planning or scenario period as an integer directly following by a capital letter, where "H" stands for hours and "T" stands for minutes.
- **sources-file**: The sourcelist file.
- **output-directory**: The path which stores the output.
- **scenario-template-file**: The scenario template file used for creating the scenarios. For more information look into the User Manual for GOSM 2.0.
- **tree-template-file**: The tree template file used for creating the scenarios. For more information look into the User Manual for GOSM 2.0.
- **reference-model-file**: The reference model which is used for creating the scenarios. For more information look into the User Manual for GOSM 2.0.
- **number-scenarios**: The number of scenarios generated for each day.

- **scenario-day**: If the scenario creator is run by itself and not by the populator, the scenarios are created for this day only.
- **wind-frac-nondispatch**: A float number.

Another useful option for the scenario creator option file is the **error_ tolerance** option. More about it is described in the last section.

## 2.2 Other files

**Sources-file**

The sources file includes every source used for generating scenarios (including "load"). Each source has its own entry. Every entry must start with the word "Source", after that the specific attributes of the source are enclosed in parenthesis. Each attribute is separated by a comma and each source is separated by a semicolon. Required attributes are:

- Name: the name of the source
- actuals-file: the file containing the actual data for that source
- forecasts-file: the file containing the forecast data for that source.
- source-type: the type of the source
- segmentation_file: a file containing information about what type of segmentation will be used
- time_step: the time step of the actual and forecast data.

Except for the name, every attribute is followed by a equal sign and the value of the attribute inside quotation marks. These attributes must require the directory where the respective file is located. The only source types that are supported are wind and solar. The time step must be an "H" for hours or a "T" for minutes with an optional prefixed number.

**Template-files and Reference-model**

Please read the GOSM User Manual for more information about how these files are created and how they look like. In the next section, some example files are provided.

## 2.3 How to run GOSM

If every required options are set and the paths for the files point into the right directory, GOSM is run by the following command:

```
runner.py populator_options.txt
```

# 3 Examples

## 3.1 How to run examples

To run some examples you have to navigate into the *prescient_gosm* repository. There you will find a folder called *examples*. Navigate into this folder as well. If you want to run examples with the CAISO data, run the command:

```
runner.py multiple_sources_gosm_test/CAISO/CAISO_populator_mc.txt
```

After that you will find the output in the directory *multiple_sources_gosm_test/CAISO/caiso_output*.

It is a little bit trickier to run some examples with BPA data. First of all, you will have to change some code. Inside the *prescient_gosm* repo, navigate to *prescient_gosm/markov_chains*. Then edit the script *markov_populator.py* in the following way: In the function *multiple_sources* search for the part introduced by the comments "Adding some noise for testing purposes. (If theres real data available for one source)". Right after this comment there is a for-loop, which is commented out with three quotation marks. Delete these before and after the for-loop.

```
# Adding some noise for testing purposes. (If theres real data available
# for one source)
"""
for i, rolling_window in enumerate(rolling_windows.values()):
    if i > 0:
        raw = rolling_window.get_column('errors')
        errors = [raw[j] + int(np.random.randint(-500, 500, 1)) for j in
                  range(len(raw))]
        rolling_window.data['errors'] = errors
"""


if args.copula_random_walk:
```

Figure 1: The part of the code where you have to delete the quotation marks.

You have to do this, because there is only one real data set available for BPA, so we have to add some noise to this set to generate a different data set which can be used as the second, third, etc, source. After this is done, you have to navigate again in the directory *prescient_gosm/examples* and run the following command:

```
runner.py multiple_sources_gosm_test/BPA/BPA_populator_mc.txt
```

After that you will find the output in the directory *multiple_sources_gosm_test/BPA/bpa_output*. **Do not forget to add the quotation marks inside the code before you are running the script with CAISO data!**

## 3.2 Example files

### CAISO

```
command/exec populator.py

--start-date 2014-10-03-15:00
--end-date 2014-10-03-15:00

--load-scaling-factor 0.045

--output-directory multiple_sources_gosm_test/CAISO/caiso_output
--scenario-creator-options-file multiple_sources_gosm_test/CAISO/CAISO_scenario_creator_mc.txt
--sources-file multiple_sources_gosm_test/CAISO/caiso_sourcelist.txt
--allow-multiprocessing 0
--traceback
```

Figure 2: CAISO_populator_mc.txt

```
command/exec scenario_creator.py

--use-markov-chains
--copula-random-walk
--planning-period-length 10H
# Options regarding file in- and output:
--sources-file multiple_sources_gosm_test/CAISO/caiso_sourcelist.txt
--output-directory multiple_sources_gosm_test/CAISO/output_scenario_creator
--scenario-template-file multiple_sources_gosm_test/CAISO/caiso_simple_nostorage_skeleton.dat
--tree-template-file multiple_sources_gosm_test/TreeTemplate.dat
--reference-model-file ../models/knueven/ReferenceModel.py
--number-scenarios 1

# General options:
--scenario-day 2015-06-30

--wind-frac-nondispatch=0.50
```

Figure 3: CAISO_scenario_creator_mc.txt

```
Source(Wind_NP15,
        actuals_file="multiple_sources_gosm_test/CAISO/data/Wind/wind_forecasts_actuals_NP15_070114_063015.csv",
        forecasts_file="multiple_sources_gosm_test/CAISO/data/Wind/wind_forecasts_actuals_NP15_070114_063015.csv",
        source_type="wind",
        segmentation_file="multiple_sources_gosm_test/segment_bpa.txt",
        time_step="H");
Source(Wind_SP15,
        actuals_file="multiple_sources_gosm_test/CAISO/data/Wind/wind_forecasts_actuals_SP15_070114_063015.csv",
        forecasts_file="multiple_sources_gosm_test/CAISO/data/Wind/wind_forecasts_actuals_SP15_070114_063015.csv",
        source_type="wind",
        segmentation_file="multiple_sources_gosm_test/segment_bpa.txt",
        time_step="H");
Source(Bus1,
        actuals_file="multiple_sources_gosm_test/CAISO/data/Load/caiso_load_070114_063015.csv",
        forecasts_file="multiple_sources_gosm_test/CAISO/data/Load/caiso_load_070114_063015.csv",
        source_type="load",
        segmentation_file="multiple_sources_gosm_test/no_segmentation.txt",
        time_step="H");
```

Figure 4: caiso_sourcelist.txt

**BPA**

```
command/exec populator.py

--start-date 2013-02-03-03:00
--end-date 2013-02-03-03:00

--load-scaling-factor 0.045

--output-directory multiple_sources_gosm_test/BPA/bpa_output
--scenario-creator-options-file multiple_sources_gosm_test/BPA/BPA_scenario_creator_mc.txt
--sources-file multiple_sources_gosm_test/BPA/bpa_sourcelist_bus1.txt

--allow-multiprocessing 0

--traceback
```

Figure 5: BPA_populator_mc.txt

```
command/exec scenario_creator.py

--use-markov-chains
--copula-random-walk
--planning-period-length 10H

# Options regarding file in- and output:
--sources-file multiple_sources_gosm_test/BPA/bpa_sourcelist_bus1.txt
--output-directory multiple_sources_gosm_test/BPA/output_scenario_creator
--scenario-template-file multiple_sources_gosm_test/BPA/simple_nostorage_skeleton.dat
--tree-template-file multiple_sources_gosm_test/TreeTemplate.dat
--reference-model-file ../models/knueven/ReferenceModel.py
--number-scenarios 1

# General options:
--scenario-day 2015-06-30

--wind-frac-nondispatch=0.50
```

Figure 6: BPA_scenario_creator_mc.txt

```
Source(Wind,
        actuals_file="multiple_sources_gosm_test/BPA/data/2012-2013_BPA_forecasts_actuals.csv",
        forecasts_file="multiple_sources_gosm_test/BPA/data/2012-2013_BPA_forecasts_actuals.csv",
        source_type="wind",
        segmentation_file="multiple_sources_gosm_test/segment_bpa.txt",
        capacity_file="multiple_sources_gosm_test/BPA/data/manual_ub.dat",
        time_step="H");
Source(Wind2,
        actuals_file="multiple_sources_gosm_test/BPA/data/2012-2013_BPA_forecasts_actuals.csv",
        forecasts_file="multiple_sources_gosm_test/BPA/data/2012-2013_BPA_forecasts_actuals.csv",
        source_type="wind",
        segmentation_file="multiple_sources_gosm_test/segment_bpa.txt",
        capacity_file="multiple_sources_gosm_test/BPA/data/manual_ub.dat",
        time_step="H");
#Source(Wind3,
#       actuals_file="multiple_sources_gosm_test/BPA/data/2012-2013_BPA_forecasts_actuals.csv",
#       forecasts_file="multiple_sources_gosm_test/BPA/data/2012-2013_BPA_forecasts_actuals.csv",
#       source_type="wind",
#       segmentation_file="multiple_sources_gosm_test/segment_bpa.txt",
#       capacity_file="multiple_sources_gosm_test/BPA/data/manual_ub.dat",
#       time_step="H");
Source(Bus1,
        actuals_file="multiple_sources_gosm_test/BPA/data/CAiso-TAC_demand_12-15.csv",
        forecasts_file="multiple_sources_gosm_test/BPA/data/CAiso-TAC_demand_12-15.csv",
        source_type="load",
        segmentation_file="multiple_sources_gosm_test/no_segmentation.txt",
        time_step="H");
```

Figure 7: bpa_sourcelist_bus1.txt

# 4 Theory

In this section we will explain the theory behind the Markov Chain Random Walk and using copulas for multiple sources. The purpose of this Markov Chain Random Walk is to generate power production scenarios for the next planning period. A scenario is a vector of power values for each specific time point in the planning period for each power source (i.e. if you have 3 different power sources, the scenario will contain 3 vectors of power values). These vectors are computed by the Markov Chain Random Walk through the space of errors. Random Walk refers to the fact that we are going randomly from one state to the next one. In our case a state is an error value for each power source at one time point in the planning period. The following basic algorithm shows how the Markov Chain Random Walk looks like:

1. Get a start state.
2. Set the start state as the current state.
3. Fit a marginal distribution (i.e. one dimensional distribution) to the historic error data of each source for the time point of the current state and for the time point of the next state. That means, if there are 2 sources, there will be 4 marginal distributions. If there are 3 sources, there will be 6 marginals and so on.
4. Use the marginal's cdf to transform the error data of each dimension into the unit interval $[0, 1]$. After that, fit a copula to these transformed data sets. (The transformation must be done, because a copula C is defined as a distribution function $C : [0, 1]^n \to [0, 1]$ )
5. Sample a next state from the conditional distribution, which is defined by the copula conditioned on being in the current state (i.e. the transition probability for the random walk is the probability to get into a specific next state while being in the current state).
6. Add this next state to the list of states of the random walk. Set computed next state as the current state and return to 3.

Repeat this algorithm until a state for every time point in the planning period is computed.

## 4.1 How to sample from a Multidimensional distribution

The general idea of this method is to divide every dimension into slices. After that a one dimensional distribution is layered over the slices. Through inverse sampling of this one dimensional distribution, it is determined which slice is chosen to continue this process (i.e. this slice was divided into slices as well, so you can repeat the method). For a better understanding we recommend reading this online blog[1], but we also provide a description of the method for 2 dimensions.

[1] url: http://code-spot.co.za/2009/04/15/generating-random-points-from-arbitrary-distributions-for-2d-and-up/

**Method in 2D**

1. Create a grid over the support of the overlying pdf. In our case we are using a copula as the pdf, that means we have to create a grid over $[0,1]^2$. After that, you assign to each cell the probability mass of the pdf conditioned on the current state. It might look like this:

| | | | |
|---|---|---|---|
| 0.0625 | 0.03125 | 0.01875 | 0.025 |
| 0.03125 | 0.125 | 0.1875 | 0.00625 |
| 0.05 | 0.1875 | 0.125 | 0.0125 |
| 0.0125 | 0.03125 | 0.03125 | 0.0625 |

2. Accumulate each column of the grid. Our above grid will then look like:

| | | | |
|---|---|---|---|
| 0.0625 | 0.03125 | 0.01875 | 0.025 |
| 0.09375 | 0.15625 | 0.20625 | 0.03125 |
| 0.14375 | 0.34375 | 0.33125 | 0.04375 |
| 0.15625 | 0.375 | 0.3625 | 0.10625 |

3. Now we have to accumulate the last row of the above grid and add a zero in the front. This yields to:

| | | | | |
|---|---|---|---|---|
| 0 | 0.15625 | 0.53125 | 0.89375 | 1 |

4. We use these values as the y-values. To get some x-values, we are taking the right bound of each column/cell. These (x,y) value pairs are then used to create a linear interpolation (i.e. this is our one dimensional distribution we need).

5. We need to sample a random number from this distribution. Therefore we are using the inverse sampling method. So we need to draw a random number from a uniform distribution which support is $[0,1]$. Next, find the inverse value of this number of your distribution. This is the first dimension of the sample.

6. We also need to look in which interval this sampled value falls. This interval will give us the column, where we have to repeat our method. Lets assume the sampled number yields us to the third column. Add a 0 at the beginning and normalize all the values. In the end, our column looks like:

| |
|---|
| 0 |
| 0.0517 |
| 0.56897 |
| 0.9138 |
| 1 |

7. We start again with step 4, and repeat the interpolation and inverse sampling method to get a sample for the second dimension.

8. In the end you will have two samples, each for one dimension. This is our 2D sample from our conditional pdf. Don't forget to convert these sample into error values using the inverse cdf of the respective marginal distribution.

You can generalize this method to $n$ dimensions. Therefore you have to repeat the accumulation, interpolation and inverse sampling, until you have a sample for each dimension.

# 5 Runtime

Whenever using more than two sources for the scenario generation, there is considerable slowdown due to the preciseness of Scipy's integration function. The process encounters these integration processes when:

1. the computation of the marginal distribution of a joint cdf
2. when accumulating the grid over the pdf as described in the section above

The default value for both error tolerances are $1.49e-08$ according to the Scipy integrate.nquad documentation. We will allow for a parameter $--error_tolerance$that will increase the run time by increasing the absolute and relative error tolerance for the integration processes. The parameter should be an integer and will correspond the increase in power of tens to the error tolerance. For example, with $--error\_tolerance = 3$, the error tolerance will be $1.49e-05$. As you can imagine, the room for error in the integration increases as a drawback for the decreased run time.

We will show two tables that correspond to different scenarios for different power generations. We only analyzed data at one hour with a plannning period length of 5 hours, thus resulting in 10 integration processes due to both integration for each 5 steps.

Since the sum of the grid pdf integration should result in 1, from the theory of the section above, we take the error of the actual value of the integration and provide the average over these steps. We also record the average time for both integration processes. Note that this is a general trend that we found, run time differences can be attributed to many factors in your setup.

Our table for an hour in a day where there was mid-level power generated:

| Error Tolerance (T) | Average Error of Grid Integration (Difference from 1) | Average Time for Grid Integration | Average Time for Marginal Integration |
|---|---|---|---|
| $T = 0$ or Default | $7.372168 \times 10^{-11}$ | 6.08498 min | 6.779567 min |
| $T = 1$ or $\times 10^1$ | $1.151786 \times 10^{-9}$ | 4.632438 min | 3.069336 min |
| $T = 2$ or $\times 10^2$ | $5.430827 \times 10^{-9}$ | 2.4640618 min | 2.442976 min |
| $T = 3$ or $\times 10^3$ | $1.086908 \times 10^{-6}$ | 1.755044 min | 1.643237 min |

Our table for an hour in a day where there was low-level power generated:

| Error Tolerance (T) | Average Error of Grid Integration (Difference from 1) | Average Time for Grid Integration | Average Time for Marginal Integration |
|---|---|---|---|
| $T = 0$ or Default | $4.927122 \times 10^{-10}$ | 9.902274 min | 7.629800 min |
| $T = 1$ or $\times 10^1$ | $1.152122 \times 10^{-9}$ | 9.751140 min | 7.752153 min |
| $T = 2$ or $\times 10^2$ | $2.419252 \times 10^{-7}$ | 4.366983 min | 7.481410 min |
| $T = 3$ or $\times 10^3$ | $3.929535 \times 10^{-7}$ | 4.126274 min | 5.315102 min |