Unit Commitment Scenario Maker Quickstart

Ravishdeep Singh

February 2019

1 Introduction

This class was built so that users of GOSM can easily get scenarios for their experiments with an easy to use interface. First we will go through a quick tutorial to show how to interact with the interface. Then there is a quick description of what sampling procedures you should use and at the end of this document is a detail description about the implementation of the scenario maker.

2 Getting Started

2.1 Intialization

The first thing to do is to import the class:

```
import prescient_gosm.gosm_scnmk as scnmk
```

To initialize the class you call an instance of **Unitcommit_scenario_maker()** initializer.

```
scmaker = scnmk.Unitcommit_scenario_maker()
```

This implicit initilization will create two subdirectories **In_Sample** and **Out_of_Sample** which will contain in sample scenarios and out of sample scenarios respectively.

You can also explicitly name the folders of where these samples will be placed by adding extra arguments to the intializer.

```
scnmk.Unitcommit_scenario_maker(in_sample_scenarios='Folder1', out_of_sample_scenarios='Folder2')
```

2.2 Creating Experiments

An **experiment** in this case is just a set of scenarios that a user will test with. We initialize a certain experiment with three parameters:

- A **name** to specify how to identify this set of scenarios. Note you cannot use the same name for two experiments.

- A **sampling method** that will be used for the experiment. There are only two sampling methods **"resampling"** and **"augmentation"** to choose from. More info about this property is found in Section 3.
- A **seed offset** to initialize the seedlist and provide a set of scenarios you can test each time that will act accordingly. This parameter is optional but we recommend to use this value.

An example of creating an experiment with name "Experiment1," an augmentation sampling method, and a seed offset of 56 is shown below.

```
scmaker.create_experiment("Experiment1", "augmentation", seed_offset=56)
```

You can create as many experiments you want and they won't interfere with each other, as long as they have different names. The Unitcommit_scenario_maker class just keeps track of each experiment created by the name provided.

```
scmaker.create_experiment("Experiment2", "resampling", seed_offset=4)
```

2.3 Setting Parameters

Before creating the scenarios in an experiments there needs to be parameters passed in. In order to pass in parameters the user will create a dictionary where the key is the parameter and the value is the value we want the parameter to have. An example is pictured below.

```
params = {"reference_model_file": "ReferenceModel.py", "start_date": "2013-01-01","end_date": "2013-01-03", "sources_file": "bpa_sourcelist.txt", "tree_template_file": "TreeTemplate.dat", "scenario_template_file": "simple_nostorage_skeleton.dat"}
```

The parmeteres to know for creating scenarios are listed here:

- reference_model_file (string): Location of ReferenceModel.py.
- start_date (string): Start date for the range for which the scenarios will be constructed in(YYY-MM-DD format)
- end_date (string): End date for the range for which the scenarios will be constructed in(YYY-MM-DD format)
- sources_file (string): Location of source file.
- tree_template_file (string): Location of tree template file.
- scenario_template_file (string): Location of scenario template file.
- planning_period_length (string): The length of time for the scenario period. The format is an integer followed by a capital letter, with "H" for hours and "T" for minutes. For example the default is "23H" representing a period for 24 hours.
- solvername (string): The name of the mathematical solver to use

In order to actually set the parameters for an experiment we pass the dictionary into the method **set_parameters**. We also make sure to name which experiment we want to set the parametrs for.

```
scmaker.set_parameters("Experiment1", params)
```

2.4 Running Experiments

To run an experiment we use the **run_experiment** where the parameters are the name of the experiment we want to run, the number of scenarios that we want to be created, and the location where we want the scenarios to be located in. The location is based on whether we want to send it to the directory where **"in_sample"** scenarios or **"out_of_sample"** scenarios are.

```
scmaker.run_experiment("Experiment1", 2, "in_sample")
```

If the user just created the experiment it will initialize the experiment with two scenarios. Every subsequent call to **run_experiment** to this certain experiment will add on to the already existing scenarios. For example if the command is run below after the command above then the total amount of scenarios for this experiment is 6.

```
scmaker.run_experiment("Experiment1", 4, "in_sample")
```

(In actuallity the second command might actually create 6 scenarios instead of 4 if the sampling method was resampling but more about that is detailed in Section 3).

2.5 Solving Experiments

To solve the scenarios in an experiment we use the **solve_experiment** method which takes in the name of the experiment and the location we want the output to be located in. just like **run_experiment** we have a choice of **"in_sample"** or **"out_of_sample"**.

```
scmaker.solve_experiment('Experiment1', 'in_sample')
```

3 Sampling Scenarios and Solving

When initializing a set of scenarios ie an "experiment" you are required to enter a sampling method. A sampling method specifies how **sequential sampling** is processed with the set of scenarios. This process just allows us to increase the amount of scenarios ("samples") in an "experiment" by a certain amount. We will now go into quick descriptions of how both sampling methods work:

- **augmentation**: When adding additional scenarios to an experiment the previous scenarios will be kept and we just simply add a new number of scenarios to the experiment.
- **resampling**: When adding additional scenarios to an experiment the previous scenarios will be erased and we start over redoing the previous scenarios plus the new amount of scenarios to be created.

4 Behind the Scenes

In the file *gosm_scnmk.py* you will see the code for the Unitcommit_scenario_maker class. Along with it there are two other classes UC_params and Unitcommit_experiment. Here are a brief descriptions of what each class does:

a) **UC_params**: This is a container class for the parameters passed into scenario maker. It takes the a dictionary as input which contains the values of specific parameters. Each valid parameter found is initialized with the value in the dictionary and every other parameter not found will be set to **None**. Some of the methods in this class include setter functions for each of the parameters and method **check_valid_args()** which should check if the parameters are valid for an instance of GOSM to be run on them.

b) **Unitcommit_experiment**: This is a class which contains information about a certain instance of a GOSM experiment run. Therefore it contains information about the certain set of scenarios made in the experiment such as the sampling procedure and seedlist used. It also allows options to solve the set of scenarios. There are two important methods for this class: 1) **run_experiment(self, output_name, num_scenarios, additional_seeds=[])** which basically runs an iteration of the sampling procedure for the GOSM experiment. The parameters passed in: **output_name** is the name for the location of the where the Scenarios folder will be, **num_scenarios** is the number of scenarios to be created from this iteration, and **additional_seeds** is an optional argument where the user can pass in their own seeds to be tested in the environment. 2) **solve_scenarios(self, name, output_location)** which solves the scenarios for this experiment. **output_name** is the same as the argument for run_experiment and name is the name of the experiment.

c) **Unitcommit_scenario_maker** This is the actual class a potential user will interact with. Using this class the user can create different GOSM experiments each with its own set of scenarios. The user can specify the sampling procedure used in each of the experiments and the location of **In Sample and Out of Sample** Scenarios. Other than the methods mentioned above in Section 1, a useful method is **check_experiment_exists(name)** which checks if a certain experiment exists based on the name.