# Microgrid-bench Documentation

***Release 1***

**Bertrand Cornélusse**

**Oct 30, 2017**

# Contents:

CHAPTER 1

# What is Microgrid-bench

A microgrid is a small power system connecting devices that consume, generate and store electricity. Usually microgrids are able to operate in islanded mode (off-grid), but they can also be connected to the public grid. We are interested mostly in the latter case, because it offers many more valorization mechanisms.

## Operational planning

Operational planning is the process that controls the operation of a microgrid over a relatively long time horizon (from a few hours to several days) divided in periods during which quantities are assumed constant. A period can last from one minute to one hour. Hence, fast dynamics of the system are not considered.

## A test-bench to assess operational planning policies

Microgrid-bench is a python tool that aims at simulating the techno-economics of a microgrid, and in particular at quantifying the performance of an operational planning controller as a function of the random processes governing all the variables that impact the microgrid operation (e.g. consumption, renewable generation, market price).

## Functionalities

Microgrid-bench offers the following functionalities:

- To simulate an operational planning policy on real data
- Forecasters are automatically generated for all variables that have to be predicted
- New datasets can be easily integrated
- The microgrid topology can be easily configured
- Results are stored in the `results` folder

- Plots are automatically generated and can be regenerated from a set of existing results

So far, the following devices are available:

- Non-flexible loads
- Non-flexible generation
- Simple battery model: limited capacity, max (dis)charge rates, (dis)charge efficiencies

Regarding the pricing scheme:

- Variable purchase price
- Variable sales price
- Peak pricing: max monthly peak, 12 months rolling horizon.

CHAPTER 2

# Installation

1. Download the code from Github
2. We highly recommend to use an Anaconda distribution
    1. download and install Anaconda for Python 2.7 and your specific OS.
    2. Create one environement for this project

```
conda create --name microgrid --file conda-{platform}.txt
```

    where "{platform}" must match your OS. Checkout this reference for more information about how to manage Anaconda environments.

    3. Activate the environment

    For Windows:

```
activate microgrid
```

    For OSX and Linux,

```
source activate microgrid
```

CHAPTER 3

# Running the application

Please refer to the examples

## Examples

You can find these examples at the .

An example is described in a python file and relies on cases stored in the `data` folder.

Examples are better than precepts and the best is to get started with the following examples (with the simplest examples listed first)

### Toy example: running an idle controller on case 1

In the directory of the example, run

```
python run_idle_controller_on_case1.py
```

Results are automatically generated in the `results` folder (plots and data).

You can start from a copy of this example file if you want to change

- the simulation length
- the controller
- etc.

Description of cases:

## Case 1 description

Case 1 is a really simple microgrid made of three loads, one PV installation, and one storage device. The microgrid is grid-tied, and can thus exchange with the grid at a price indexed on the spot price. A peak pricing penalty is also applied and is implemented as follows:

- At the beginning of each month, the highest peak over the 11 preceding months is stored in $p_{past}$
- A cost proportional to $p_{past}$ is directly incurred
- Then, the highest monthly peak is recorded at each period $t$ in a variable $p_t$
- Every time $p_t > p_{past}$, a cost proportional to $(p_t - p_{past})$ is incurred, and $p_{past} := p_t$

CHAPTER 4

# Documentation

Read the doc for more information.

Alternatively you can generate the documentation yourself if you have sphinx installed:

```
cd <to the root of the project>
sphinx-apidoc -o docs/ microgrid/ -f --separate
cd docs; make html; cd ..
```

The html doc is in `_build/html`

CHAPTER 5

# microgrid

## microgrid package

### Subpackages

#### microgrid.control package

#### Submodules

#### microgrid.control.abstract_controller module

**class** microgrid.control.abstract_controller.**AbstractController**(*grid*)

Bases: object

**__init__**(*grid*)

**compute_actions**(*start_date*, *end_date*, *grid_state*, *horizon*, *debug=False*)

**Parameters**

- **start_date** – Start period for which actions is requested
- **end_date** – End period for which actions is requested
- **grid_state** – State of the grid at start_period
- **horizon** – optimization horizon
- **debug** – flag to (de)active debug information

**Returns** grid actions to be applied to the microgrid, as a GridAction object

### microgrid.control.idle_controller module

**class** microgrid.control.idle_controller.**IdleController**(*grid*)
Bases: *microgrid.control.abstract_controller.AbstractController*

**__init__**(*grid*)
Controller that takes no action and thus returns zero for all devices and all the horizon.

**Parameters** **grid** – Cf. base class

**compute_actions**(*start_date*, *end_date*, *grid_state*, *horizon*, *debug=False*)

### Module contents

The control module provides an interface for designing microgrid operational planning controllers.

**class** microgrid.control.**IdleController**(*grid*)
Bases: *microgrid.control.abstract_controller.AbstractController*

**__init__**(*grid*)
Controller that takes no action and thus returns zero for all devices and all the horizon.

**Parameters** **grid** – Cf. base class

**compute_actions**(*start_date*, *end_date*, *grid_state*, *horizon*, *debug=False*)

### microgrid.forecast package

### Submodules

### microgrid.forecast.forecaster module

**class** microgrid.forecast.forecaster.**Forecaster**(*database*)

**__init__**(*database*)
A Forecaster object allows to generate forecast of any of the uncertain quantities referenced in the database.

**Parameters** **database** – A Database object used for training the forecaster

**forecast**(*column*, *dt_from*, *dt_to*)
Forecast an uncertain quantity over a specified time range with a hourly resolution. Each time a forecast is asked, a new forecaster is trained using all previous values of the quantity until dt_from.

**Parameters**

- **column** – Name of the series to forecast
- **dt_from** – A date_time object specifying the start of the prediction horizon
- **dt_to** – A date_time object specifying the end of the prediction horizon

**Returns** The forecast as a numpy array. The length of the array is equal to the number of hours between dt_from and dt_to, rounded down

### Module contents

The forecast module provides functions to generate forecasts fot the uncertain quantities impacting the operation of the microgrid.

**class** `microgrid.forecast.`**`Forecaster`**(*database*)

**`__init__`**(*database*)
A Forecaster object allows to generate forecast of any of the uncertain quantities referenced in the database.

**Parameters** **`database`** – A `Database` object used for training the forecaster

**`forecast`**(*column*, *dt_from*, *dt_to*)
Forecast an uncertain quantity over a specified time range with a hourly resolution. Each time a forecast is asked, a new forecaster is trained using all previous values of the quantity until dt_from.

**Parameters**

- **`column`** – Name of the series to forecast
- **`dt_from`** – A date_time object specifying the start of the prediction horizon
- **`dt_to`** – A date_time object specifying the end of the prediction horizon

**Returns** The forecast as a numpy array. The length of the array is equal to the number of hours between dt_from and dt_to, rounded down

## microgrid.history package

### Submodules

### microgrid.history.database module

**class** `microgrid.history.database.`**`Database`**(*path_to_csv*, *grid*)

**`__init__`**(*path_to_csv*, *grid*)
A Database objects holds the realized data of the microgrid in a pandas dataframe.

The CSV file values are separated by ';' and the first line must contain series names. It must contain

•a 'DateTime' column with values interpretable as python date time objects.

•a 'Price' column with values interpretable as floats.

•All the non-flexible quantities (load and generation) described in the microgrid configuration

Some new columns are generated from the DateTime column to indicate e.g. whether a datetime corresponds to a day of the week or not.

**Parameters**

- **`path_to_csv`** – Path to csv containing realized data
- **`grid`** – A Grid object describing the configuration of the microgrid

**`get_column`**(*column_indexer*, *dt_from*, *dt_to*)

**Parameters**

- **`column_indexer`** – The name of a column

- **dt_from** – A start datetime
- **dt_to** – An end datetime

**Returns** A list of values of the column_indexer series between dt_from and dt_to

**get_columns**(*column_indexer*, *time_indexer*)

**Parameters**

- **column_indexer** – The name of a column
- **time_indexer** – A datetime

**Returns** The realized value of the series column_indexer at time time_indexer

**get_times**(*time_indexer*)

**Parameters** **time_indexer** – A date time

**Returns** A list containing the value of all the series at time time_indexer

**read_data**(*path*)

Read data and generate new columns based on the DateTime column.

**Parameters** **path** – Path to the csv data file

**Returns** A pandas dataframe

### Module contents

The history module gathers all functionalities related to the organization of time series data that are used by the simulator and by the forecasters.

**class** microgrid.history.**Database**(*path_to_csv*, *grid*)

**__init__**(*path_to_csv*, *grid*)

A Database objects holds the realized data of the microgrid in a pandas dataframe.

The CSV file values are separated by ';' and the first line must contain series names. It must contain

•a 'DateTime' column with values interpretable as python date time objects.

•a 'Price' column with values interpretable as floats.

•All the non-flexible quantities (load and generation) described in the microgrid configuration

Some new columns are generated from the DateTime column to indicate e.g. whether a datetime corresponds to a day of the week or not.

**Parameters**

- **path_to_csv** – Path to csv containing realized data
- **grid** – A Grid object describing the configuration of the microgrid

**get_column**(*column_indexer*, *dt_from*, *dt_to*)

**Parameters**

- **column_indexer** – The name of a column
- **dt_from** – A start datetime
- **dt_to** – An end datetime

**Returns** A list of values of the column_indexer series between dt_from and dt_to

**get_columns**(*column_indexer*, *time_indexer*)

**Parameters**

- **column_indexer** – The name of a column
- **time_indexer** – A datetime

**Returns** The realized value of the series column_indexer at time time_indexer

**get_times**(*time_indexer*)

**Parameters** **time_indexer** – A date time

**Returns** A list containing the value of all the series at time time_indexer

**read_data**(*path*)
Read data and generate new columns based on the DateTime column.

**Parameters** **path** – Path to the csv data file

**Returns** A pandas dataframe

### microgrid.model package

### Submodules

### microgrid.model.device module

**class** microgrid.model.device.**Device**(*name*)
Bases: object

**__init__**(*name*)
Base class for all devices.

**Parameters** **name** – Name of the devices, used as a reference for access to realized data or forecasts

### microgrid.model.generator module

**class** microgrid.model.generator.**Generator**(*name*, *params*)
Bases: *microgrid.model.device.Device*

**__init__**(*name*, *params*)

**Parameters**

- **name** – Cf. parent class
- **params** – dictionary of params, must include a capacity value , a steerable flag, and a min_stable_generation value

### microgrid.model.grid module

**class** microgrid.model.grid.**Grid**(*data*)

**__init__**(*data*)
A microgrid is represented by its devices which are either loads, generators or storage devices, and additional information such as prices. The period duration of the simulation is also stored at this level, although it is more part of the configuration of the simulation.

**Parameters** **data** – A json type dictionary containing a description of the microgrid.

**base_purchase_price**

**get_non_flexible_device_names**()

**Returns** The list of names of all non-flexible loads and generators for which there must be an entry in the data history

**peak_price**

**period_duration**

**price_margin**

**purchase_price**(*energy_prices*)

**Parameters** **energy_prices** – A list of energy prices (i.e. a time series), in EUR/MWh

**Returns** The actual purchase price taking into account all components, in EUR/kWh

**sale_price**(*energy_prices*)

**Parameters** **energy_prices** – A list of energy prices (i.e. a time series), in EUR/MWh

**Returns** The actual sale price taking into account all components, in EUR/kWh

### microgrid.model.load module

**class** microgrid.model.load.**Load**(*name*, *capacity*)
Bases: *microgrid.model.device.Device*

**__init__**(*name*, *capacity*)

**Parameters**

- **name** – Cf. parent class
- **capacity** – Max rated power of the load.

### microgrid.model.storage module

**class** microgrid.model.storage.**Storage**(*name*, *params*)
Bases: *microgrid.model.device.Device*

**__init__**(*name*, *params*)

**Parameters**

- **name** – Cf. parent class
- **params** – dictionary of params, must include a capacity value , a max_charge_rate value, a max_max_discharge_rate value, a charge_charge_efficiency value and a discharge_charge_efficiency value.

### Module contents

The model package defines all classes that are used to represent a microgrid and its devices. It mainly contains data and a few useful methods.

**class** microgrid.model.**Grid**(*data*)

**__init__**(*data*)
A microgrid is represented by its devices which are either loads, generators or storage devices, and additional information such as prices. The period duration of the simulation is also stored at this level, although it is more part of the configuration of the simulation.

**Parameters** **data** – A json type dictionary containing a description of the microgrid.

**base_purchase_price**

**get_non_flexible_device_names**()

**Returns** The list of names of all non-flexible loads and generators for which there must be an entry in the data history

**peak_price**

**period_duration**

**price_margin**

**purchase_price**(*energy_prices*)

**Parameters** **energy_prices** – A list of energy prices (i.e. a time series), in EUR/MWh

**Returns** The actual purchase price taking into account all components, in EUR/kWh

**sale_price**(*energy_prices*)

**Parameters** **energy_prices** – A list of energy prices (i.e. a time series), in EUR/MWh

**Returns** The actual sale price taking into account all components, in EUR/kWh

**class** microgrid.model.**Device**(*name*)
Bases: object

**__init__**(*name*)
Base class for all devices.

**Parameters** **name** – Name of the devices, used as a reference for access to realized data or forecasts

**class** microgrid.model.**Generator**(*name*, *params*)
Bases: *microgrid.model.device.Device*

**__init__**(*name*, *params*)

**Parameters**

- **name** – Cf. parent class
- **params** – dictionary of params, must include a capacity value , a steerable flag, and a min_stable_generation value

**class** microgrid.model.**Load**(*name*, *capacity*)
Bases: *microgrid.model.device.Device*

**__init__**(*name*, *capacity*)

**Parameters**

- **name** – Cf. parent class
- **capacity** – Max rated power of the load.

**class** microgrid.model.**Storage**(*name*, *params*)
Bases: *microgrid.model.device.Device*

**__init__**(*name*, *params*)

**Parameters**

- **name** – Cf. parent class
- **params** – dictionary of params, must include a capacity value , a max_charge_rate value, a max_max_discharge_rate value, a charge_charge_efficiency value and a discharge_charge_efficiency value.

### microgrid.plot package

### Submodules

### microgrid.plot.plot_results module

### Module contents

### microgrid.simulate package

### Submodules

### microgrid.simulate.gridaction module

**class** microgrid.simulate.gridaction.**GridAction**(*grid_import*, *grid_export*, *production*, *consumption*, *state_of_charge*, *charge*, *discharge*, *peak*, *peak_increase*)

**__init__**(*grid_import*, *grid_export*, *production*, *consumption*, *state_of_charge*, *charge*, *discharge*, *peak*, *peak_increase*)

Action taken by the controller. Actually, the action is only a subset of the parameters and other members represent auxiliary information that may be used for reporting purposes.

Each action is defined per device, then per period of the optimization horizon. Each member is defined as a list or as nested lists.

**Parameters**

- **grid_import** – Auxiliary variable.
- **grid_export** – Auxiliary variable.
- **production** – Auxiliary variable.
- **consumption** – Auxiliary variable.
- **state_of_charge** – Auxiliary variable.
- **charge** – Action to charge storage devices.
- **discharge** – Action to discharge storage devices.
- **peak** – Auxiliary variable.

- **peak_increase** – Auxiliary variable.

**to_json**()

### microgrid.simulate.gridstate module

**class** microgrid.simulate.gridstate.**GridState**(*grid*, *date_time*)

**__init__**(*grid*, *date_time*)
Representation of the state of the system in the simulator. The state includes the state of charge of storage devices plus information regarding past operation of the system.

**Parameters**

- **grid** – A Grid object
- **date_time** – The time at which the system in this state

### microgrid.simulate.simulator module

**class** microgrid.simulate.simulator.**Simulator**(*grid*, *controller*, *database*)

**__init__**(*grid*, *controller*, *database*)

**Parameters**

- **grid** – A description of the grid as a Grid object
- **controller** – tool that decides, based on a forecast, a grid state and a model, which decisions to apply to the system now, until the next reoptimization
- **database** – (true) evolution of the exogeneous quantities over the simulation period

**run**(*start_date*, *end_date*, *decision_horizon=1*, *optim_horizon=12*)
Run the simulation.

**Parameters**

- **start_date** – start period of the simulation
- **end_date** – end period of the simulation
- **decision_horizon** – resolution of the simulation, in hours
- **optim_horizon** – parameter passed to the controller in case the latter computes decisions over an optimization horizon longer than 1 period.

**Returns**

microgrid.simulate.simulator.**datetime_range**(*start*, *end*, *delta*)

### Module contents

The simulate module defines the simulator which, given a grid model, some realized data, and a controller, evaluates the decisions of the controller on the realized data in the microgrid.

**class** microgrid.simulate.**Simulator**(*grid*, *controller*, *database*)

**__init__**(*grid*, *controller*, *database*)

**Parameters**

- **grid** – A description of the grid as a Grid object
- **controller** – tool that decides, based on a forecast, a grid state and a model, which decisions to apply to the system now, until the next reoptimization
- **database** – (true) evolution of the exogeneous quantities over the simulation period

**run**(*start_date*, *end_date*, *decision_horizon=1*, *optim_horizon=12*)

Run the simulation.

**Parameters**

- **start_date** – start period of the simulation
- **end_date** – end period of the simulation
- **decision_horizon** – resolution of the simulation, in hours
- **optim_horizon** – parameter passed to the controller in case the latter computes decisions over an optimization horizon longer than 1 period.

**Returns**

**class** microgrid.simulate.**GridState**(*grid*, *date_time*)

**__init__**(*grid*, *date_time*)

Representation of the state of the system in the simulator. The state includes the state of charge of storage devices plus information regarding past operation of the system.

**Parameters**

- **grid** – A Grid object
- **date_time** – The time at which the system in this state

## Submodules

### microgrid.simulation module

**class** microgrid.simulation.**Simulation**(*case*, *start_date*, *end_date*, *config=<microgrid.simulation.SimulationConfiguration instance>*)

**__init__**(*case*, *start_date*, *end_date*, *config=<microgrid.simulation.SimulationConfiguration instance>*)

**Parameters**

- **case** – Case name, as a string
- **start_date** – Start of simulation, datetime
- **end_date** – End of simulation, datetime
- **config** – Simulation configuration options, instance of SimulationConfiguration

**run**(*controller*, *store_results=True*, *generate_plots=True*)

**Parameters**

- **controller** – Instance of a controller derived from AbstractController
- **store_results** – Boolean to trigger dump of results in results folder
- **generate_plots** – Boolean to trigger plot of results in results folder

**Returns**

**class** microgrid.simulation.**SimulationConfiguration**(*params=None*)

**DECISION_HORIZON**
Horizon over which decisions are applied

**OPTIMIZATION_HORIZON**
Horizon over which decisions are computed

**STORE_CONTROLLER_ACTIONS**
Shall controller actions be stored in the results file

**__init__**(*params=None*)

## Module contents

The microgrid package organizes the test-bench functionalities in subpackages.

CHAPTER 6

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## m

# Index

## Symbols



## A



## B



## C



## D



## F



## G

## I



## L



## M



## O



## P



## R



## S



## T