OMEGAlpes Documentation

Release 0.0.1

B. DELINCHANT, S. HODENCQ, Y. MARECHAL, L. MORRIET, C. PA

Mar 23, 2020

Contents

CHAPTER 1

Introduction to OMEGAlpes

OMEGAlpes stands for Generation of Optimization Models As Linear Programming for Energy Systems. It aims to be an energy systems modelling tool for linear optimisation (LP, MILP). It is currently based on the LP modeler PuLP.

It is an Open Source project located on GitLab at https://gricad-gitlab.univ-grenoble-alpes.fr/omegalpes/omegalpes

CHAPTER 2

Contents

2.1 OMEGAlpes Installation

2.1.1 Python 3.6.0

Please use Python 3.6.0 for the project interpreter. https://www.python.org/downloads/release/python-360/

2.1.2 pip install omegalpes

Please install OMEGAlpes Lib with pip using the command prompt

If you are admin on Windows or working on a virtual environment pip install omegalpes

If you want a local installation or you are not admin pip install –user omegalpes

If you are admin on Linux: sudo pip install omegalpes

Then, you can download (or clone) the OMEGAlpes Examples folder (repository) at : https://gricad-gitlab.univ-grenoble-alpes.fr/omegalpes/omegalpes-examples

Launch the examples (with Pycharm for instance) to understand how the OMEGAlpes Lib works. Remember that the examples are detailed at : https://omegalpes-examples.readthedocs.io/en/latest/

Enjoy your time using OMEGAlpes !

2.2 Other installation requirements

Please install the following packages to use OMEGAlpes if not already installed:

- *PuLP >= 1.6.8*
- *Matplotlib >= 2.2.2, <3.1*
- *Numpy >= 1.14.2, <1.16*
- *Pandas >= 0.22.0, <0.24*

2.2.1 PuLP >= 1.6.8

PuLP is an LP modeler written in python. PuLP can generate MPS or LP files and call GLPK, COIN CLP/CBC, CPLEX, and GUROBI to solve linear problems. https://github.com/coin-or/pulp

2.2.2 Matplotlib >= 2.2.2, <3.1

Matplotlib is a Python 2D plotting library. https://matplotlib.org/

2.2.3 Numpy >= 1.14.2, <1.16

NumPy is the fundamental package needed for scientific computing with Python. https://github.com/numpy/numpy

2.2.4 Pandas >= 0.22.0, <0.24

pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. https://pandas.pydata.org/pandas-docs/version/0.23.1/index.html

Help for library installation:

Command lover:

```
pip install <library_name>==version
```

If required, the command to upgrade the library is

```
pip install --upgrade <library_name>
```

Pycharm lover:

Install automatically the library using pip with Pycharm on "File", "settings…", "Project Interpreter", "+", and choosing the required library

2.3 Install OMEGAlpes as a developer

2.3.1 Installation as a developer and local branch creation

1. Create a new folder in the suitable path, name it as you wish for example : OMEGAlpes

2. Clone the OMEGAlpes library repository

   — Command lover:

   git clone https://gricad-gitlab.univ-grenoble-alpes.fr/omegalpes/omegalpes.git

   — Pycharm lover:

   Open Pycharm On the Pycharm window, click on "Check out from version control" then choose "Git". A "clone repository" window open. Copy the following link into the URL corresponding area:

   https://gricad-gitlab.univ-grenoble-alpes.fr/omegalpes/omegalpes.git

   Copy the path of the new folder created just before. Test if the connection to the git works and if it works click on "Clone". Once OMEGAlpes is cloned, you must be able to see the full OMEGAlpes library on Pycharm or on another development environment.

   If the connection does not work and if you are working with local protected network, please try again with the wifi.

3. First, choose or change your project interpreter

   — Pycharm lover:

   Click on the yellow warning link or go to "File", "settings…", "Project Interpreter"

   You can: - either select the "Python 3.6" project interpreter but you may change the version of some library that you could use for another application - either create a virtual environment in order to avoid this problem (recommended). Click on the star wheel near the project interpreter box. Click on "add…". Select "New environment" if it not selected. The location is pre-filled, if not fill it with the path of the folder as folder_path/venv Select "Python 3.6" as your base interpreter Then click on "Ok"

   —

4. You can install the library on developing mode using the following command in command prompt once your are located it on the former folder. If you are calling OMEGAlpes library in another project, the following command enables you to refer to the OMEGAlpes library you are developing

   python setup.py develop

5. If it is not already done, install the library requirements.

   — Command lover:

   pip install <library_name>

   If required, the command to upgrade the library is :

   pip install –upgrade <library_name>

   — Pycharm lover:

   You should still have a yellow warning. You can: - install automatically the libraries clicking on the yellow bar - install automatically the library using pip with Pycharm on "File", "settings…", "Project Interpreter", "+", and choosing the required library as indicated in the Library Installation Requirements part

6. Finally, you can create your own local development branch.

   — Command lover:

   git branch <branch_name>

— Pycharm lover:

By default you are on a local branch named master. Click on "Git: master" located on the bottom write of Pycharm Select "+ New Branch" Name the branch as you convenience for instance "dev_your_name"

7. Do not forget to "rebase" regularly to update your version of the library.

— Command lover:

git rebase origin

— Pycharm lover:

To do so, click on your branch name on the bottom write of the Pycharm window select "Origin/master" and click on "Rebase current onto selected"

If you want to have access to other examples and study cases, please have a look to the user's installation.

Enjoy your time developing OMEGAlpes!

2.4 OMEGAlpes structure

The models are bases on **general**, **energy** and **actor** classes. The structure of the library is described on the following class diagram:

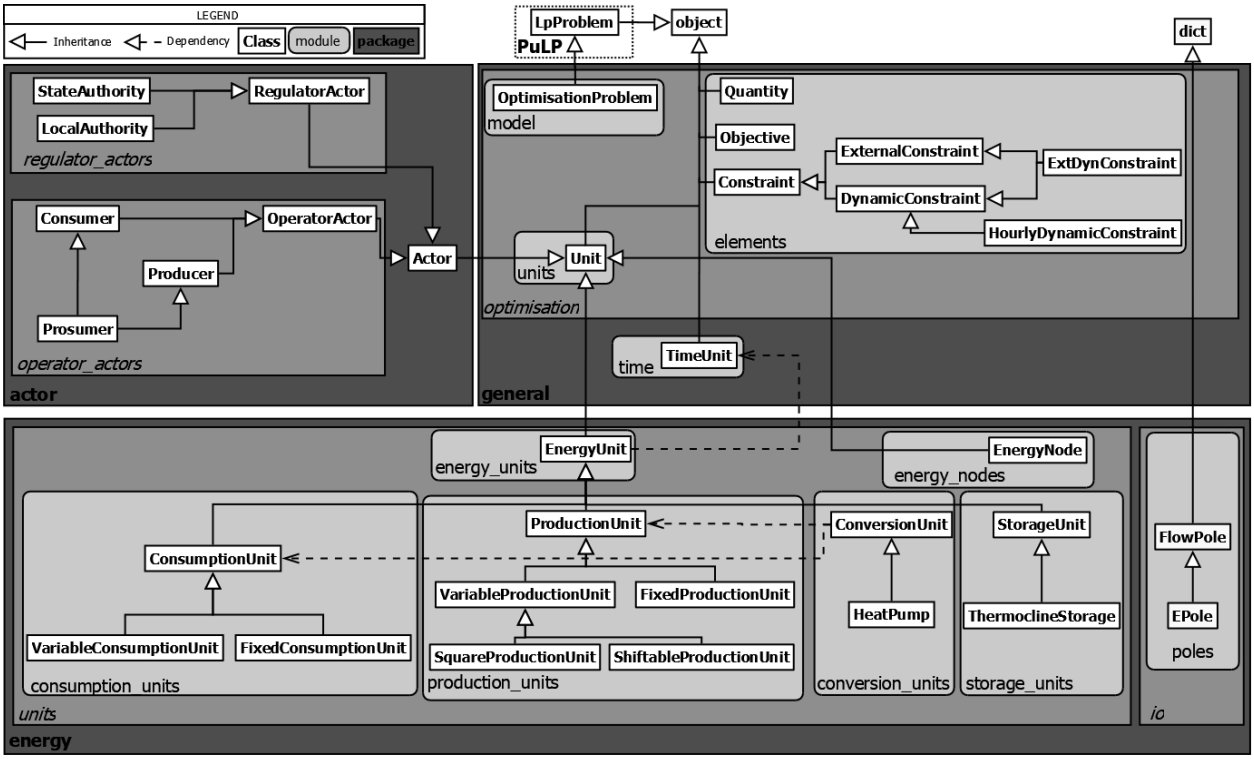

Fig. 1: *Figure: OMEGAlpes class diagram*

The energy units are detailed in the energy package

2.4.1 energy package

The energy package gathers the energy units, poles and nodes modules:

The energy units inherit from the *EnergyUnit* object which itself inherit from the Unit object.

Energy_units module

** This module defines the energy units of OMEGALPES. The production, consumption and storage unit will inherit from it. **

The energy_units module defines the basic attributes and methods of an energy unit in OMEGALPES.

It includes the following attributes and quantities:

- p : instantaneous power of the energy unit (kW)
- p_min : minimal power (kW)
- p_max : maximal power (kW)
- e_tot : total energy during the time period (kWh)
- e_min : minimal energy of the unit (kWh)
- e_max : maximal energy of the unit (kWh)
- u : binary describing if the unit is operating or not at t (delivering or consuming P)

class omegalpes.energy.units.energy_units.**EnergyUnit**(*time*, *name*, *flow_direction='in'*, *p=None*, *p_min=-10000.0*, *p_max=10000.0*, *e_min=-1000000.0*, *e_max=1000000.0*, *starting_cost=None*, *operating_cost=None*, *min_time_on=None*, *min_time_off=None*, *max_ramp_up=None*, *max_ramp_down=None*, *co2_out=None*, *availability_hours=None*, *energy_type=None*, *operator=None*, *verbose=False*)

Bases: omegalpes.general.optimisation.core.OptObject

** Description **

Module dedicated to the parent class (EnergyUnit) of :

- production units
- consumption units
- storage units

add_availability(*av_hours*)

add_co2_emissions(*co2_out*)

add_max_ramp_down(*max_ramp_down*)

add_max_ramp_up(*max_ramp_up*)

add_min_time_off(*min_time_off*)

add_min_time_on(*min_time_on*)

add_operating_cost(*operating_cost*)

add_starting_cost(*start_cost*)

minimize_CO2_emissions(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

Returns

minimize_costs(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

minimize_energy(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

minimize_operating_cost(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

minimize_starting_cost(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

minimize_time_of_use(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

set_energy_limits_on_time_period(*e_min=0*, *e_max=None*, *start='YYYY-MM-DD HH:MM:SS'*, *end='YYYY-MM-DD HH:MM:SS'*, *period_index=None*)

Parameters

- **e_min** – Minimal energy set during the time period (int or float)
- **e_max** – Maximal energy set during the time period (int or float)
- **start** – Date of start of the time period YYYY-MM-DD HH:MM:SS (str)
- **end** – Date of end of the time period YYYY-MM-DD HH:MM:SS (str)

class omegalpes.energy.units.energy_units.**FixedEnergyUnit**(*time*, *name: str*, *p: list*, *flow_direction='in'*, *starting_cost=None*, *operating_cost=None*, *co2_out=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.EnergyUnit*

Description

Energy unit with a fixed power profile.

Attributs

- p : instantaneous power known by advance (kW)
- energy_type : type of energy ('Electrical', 'Heat', ...)
- operator : stakeholder how owns the EnergyUnit

class omegalpes.energy.units.energy_units.**SawtoothEnergyUnit**(*time*, *name*, *flow_direction*, *p_peak*, *p_low*, *alpha_peak*, *t_triangle*, *t_sawtooth*, *mandatory=True*, *starting_cost=None*, *operating_cost=None*, *co2_out=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.ShiftableEnergyUnit*

class omegalpes.energy.units.energy_units.**SeveralEnergyUnit**(*time*, *name*, *fixed_power*, *pmin=1e-05*, *pmax=100000.0*, *imaginary=False*, *e_min=0*, *e_max=1000000.0*, *nb_unit_min=0*, *nb_unit_max=None*, *flow_direction='in'*, *starting_cost=None*, *operating_cost=None*, *max_ramp_up=None*, *max_ramp_down=None*, *co2_out=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.VariableEnergyUnit*

Description

Energy unit based on a fixed power curve enabling to multiply several times (nb_unit) the same power curve

Be careful, if imaginary == True, the solution may be imaginary as nb_unit can be continuous. The accurate number of the power unit should be calculated later

Attributs

- fixed_power : fixed power curve

class omegalpes.energy.units.energy_units.**ShiftableEnergyUnit**(*time*, *name: str*, *flow_direction*, *power_values*, *mandatory=True*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.VariableEnergyUnit*

Description

EnergyUnit with shiftable power profile.

Attributs

- power_values : power profile to shift (kW)
- **mandatory** [indicates if the power is mandatory][True] or not : False
- starting_cost : cost of the starting of the EnergyUnit
- operating_cost : cost of the operation (€/kW)
- energy_type : type of energy ('Electrical', 'Heat', ...)

- operator : stakeholder how owns the EnergyUnit

class omegalpes.energy.units.energy_units.**SquareEnergyUnit**(*time*, *name*, *p_square*, *n_square*, *t_between_sq*, *t_square=1*, *flow_direction='in'*, *starting_cost=None*, *operating_cost=None*, *co2_out=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.VariableEnergyUnit*

class omegalpes.energy.units.energy_units.**TriangleEnergyUnit**(*time*, *name*, *flow_direction*, *p_peak*, *alpha_peak*, *t_triangle*, *mandatory=True*, *starting_cost=None*, *operating_cost=None*, *co2_out=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.ShiftableEnergyUnit*

class omegalpes.energy.units.energy_units.**VariableEnergyUnit**(*time*, *name*, *flow_direction='in'*, *p_min=-10000.0*, *p_max=10000.0*, *e_min=-1000000.0*, *e_max=1000000.0*, *starting_cost=None*, *operating_cost=None*, *min_time_on=None*, *min_time_off=None*, *max_ramp_up=None*, *max_ramp_down=None*, *co2_out=None*, *availability_hours=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.EnergyUnit*

Consumption_units module

** This module defines the consumption units**

The consumption_units module defines various classes of consumption units, from generic to specific ones.

It includes :

- ConsumptionUnit : simple consumption unit. It inherits from EnergyUnit, its power flow direction is always 'in'.

  3 Objectives are also available :

  – minimize consumption, maximize consumption and minimize

  consumption costs.

- FixedConsumptionUnit : consumption with a fixed load profile. It inherits from ConsumptionUnit.
- VariableConsumptionUnit : consumption unit allowing for a variation of power between pmin et pmax. It inherits from ConsumptionUnit.

class omegalpes.energy.units.consumption_units.**ConsumptionUnit**(*time*, *name*, *p=None*, *p_min=1e-05*, *p_max=100000.0*, *e_min=0*, *e_max=1000000.0*, *co2_out=None*, *starting_cost=None*, *consumption_cost=None*, *min_time_on=None*, *min_time_off=None*, *max_ramp_up=None*, *max_ramp_down=None*, *availability_hours=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.EnergyUnit*

Description

Simple Consumption unit

Attributes

- p : instantaneous power demand (kW)

- pmax : maximal instantaneous power demand (kW)
- pmin : minimal instantaneous power demand (kW)
- energy_type : type of energy ('Electrical', 'Heat', ...)
- consumption_cost : cost associated to the energy consumption
- operator : stakeholder how owns the consumption unit

maximize_consumption(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

minimize_consumption(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

minimize_consumption_cost(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

class omegalpes.energy.units.consumption_units.**FixedConsumptionUnit**(*time*, *name*, *p: list = None*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.FixedEnergyUnit*, *omegalpes.energy.units.consumption_units.ConsumptionUnit*

Description

Consumption unit with a fixed consumption profile.

Attributes

- p : instantaneous power demand known in advance (kW)
- energy_type : type of energy ('Electrical', 'Heat', ...)
- consumption_cost : cost associated to the energy consumption
- operator : stakeholder how owns the consumption unit

class omegalpes.energy.units.consumption_units.**SeveralConsumptionUnit**(*time*, *name*, *fixed_cons*, *pmin=1e-05*, *pmax=100000.0*, *e_min=0*, *e_max=1000000.0*, *nb_unit_min=0*, *nb_unit_max=None*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *max_ramp_up=None*, *max_ramp_down=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.consumption_units.VariableConsumptionUnit*, *omegalpes.energy.units.energy_units.SeveralEnergyUnit*

Description

Consumption unit based on a fixed consumption curve enabling to multiply several times (nb_unit) the same consumption curve

Attributs

- fixed_cons : fixed consumption curve

class omegalpes.energy.units.consumption_units.**SeveralImaginaryConsumptionUnit**(*time*, *name*, *fixed_cons*, *pmin=1e-05*, *pmax=100000.0*, *e_min=0*, *e_max=1000000*, *nb_unit_min=0*, *nb_unit_max=No*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *max_ramp_up=N*, *max_ramp_down*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.consumption_units.VariableConsumptionUnit*, *omegalpes.energy.units.energy_units.SeveralEnergyUnit*

Description

Consumption unit based on a fixed consumption curve enabling to multiply several times (nb_unit) the same consumption curve. Be careful, the solution may be imaginary as nb_unit can be continuous. The accurate number of the consumption unit should be calculated later

Attributs

- fixed_cons : fixed consumption curve

class omegalpes.energy.units.consumption_units.**ShiftableConsumptionUnit**(*time*, *name: str*, *power_values*, *mandatory=True*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.ShiftableEnergyUnit*, *omegalpes.energy.units.consumption_units.VariableConsumptionUnit*

Description

Consumption unit with shiftable consumption profile.

Attributs

- power_values : consumption profile to shift (kW)
- **mandatory** [indicates if the consumption is mandatory][True] or not : False
- starting_cost : cost of the starting of the consumption
- operating_cost : cost of the operation (€/kW)
- energy_type : type of energy ('Electrical', 'Heat', ...)
- operator : stakeholder how owns the consumption unit

class omegalpes.energy.units.consumption_units.**SquareConsumptionUnit**(*time*, *name*, *p_square*, *duration*, *n_square*, *t_between_sq*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.SquareEnergyUnit*, *omegalpes.energy.units.consumption_units.VariableConsumptionUnit*

Description

Consumption unit with a fixed value and fixed duration. >> Only the time of beginning can be modified >> Operation can be mandatory or not

Attributs

- p : instantaneous power consumption (kW)
- duration : duration of the power delivery (hours)
- mandatory : indicates if the power delivery is mandatory or not
- energy_type : type of energy ('Electrical', 'Heat', ...)
- consumption_cost : cost associated to the energy consumption
- operator : stakeholder how owns the consumption unit

class omegalpes.energy.units.consumption_units.**VariableConsumptionUnit**(*time*, *name*, *pmin=1e-05*, *pmax=100000.0*, *e_min=0*, *e_max=1000000.0*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *min_time_on=None*, *min_time_off=None*, *max_ramp_up=None*, *max_ramp_down=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.VariableEnergyUnit*, *omegalpes.energy.units.consumption_units.ConsumptionUnit*

Description

Consumption unit with a variation of power between pmin et pmax.

Attributs

- pmax : maximal instantaneous power consumption (kW)
- pmin : minimal instantaneous power consumption (kW)
- energy_type : type of energy ('Electrical', 'Heat', ...)
- operator : stakeholder how owns the consumption unit

Production_units module

** This module defines the production units**

The production_units module defines various kinds of production units with associated attributes and methods.

It includes :

- ProductionUnit : simple production unit inheriting from EnergyUnit and with an outer flow direction. The outside co2 emissions, the starting cost, the operating cost, the minimal operating time, the minimal non-operating time, the maximal increasing ramp and the maximal decreasing ramp can be filled.

  Objectives are also available :

  - minimize starting cost, operating cost, total cost
  - minimize production, co2_emissions, time of use
  - maximize production

- FixedProductionUnit : Production unit with a fixed production profile.
- VariableProductionUnit : Production unit with a variation of power between pmin et pmax.

And also :

- SeveralProductionUnit: Production unit based on a fixed production curve enabling to multiply several times (nb_unit) the same production curve
- SeveralImaginaryProductionUnit: Production unit based on a fixed production curve enabling to multiply several times (nb_unit) the same production curve. Be careful, the solution may be imaginary as nb_unit can be continuous. The accurate number of the production unit should be calculated later
- SquareProductionUnit: Production unit with a fixed value and fixed duration.
- ShiftableProductionUnit: Production unit with shiftable production profile.

class omegalpes.energy.units.production_units.**FixedProductionUnit**(*time*, *name: str*, *p: list = None*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.FixedEnergyUnit*, *omegalpes.energy.units.production_units.ProductionUnit*

Description

Production unit with a fixed production profile.

Attributes

- p : instantaneous power production known by advance (kW)

- energy_type : type of energy ('Electrical', 'Heat', ...)
- operator : stakeholder how owns the production unit

class omegalpes.energy.units.production_units.**ProductionUnit**(*time*, *name*, *p=None*, *p_min=1e-05*, *p_max=100000.0*, *e_min=0*, *e_max=1000000.0*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *min_time_on=None*, *min_time_off=None*, *max_ramp_up=None*, *max_ramp_down=None*, *availability_hours=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.EnergyUnit*

Description

Simple Production unit

Attributes

- co2_out: outside co2 emissions
- starting_cost: the starting cost
- operating_cost: the operating cost
- min_time_on : the minimal operating time
- min_time_off : the minimal non-operating time
- max_ramp_up : the maximal increasing ramp
- max_ramp_down ; the maximal decreasing ramp

maximize_production(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

minimize_production(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

class omegalpes.energy.units.production_units.**SeveralImaginaryProductionUnit**(*time*, *name*, *fixed_prod*, *pmin=1e-05*, *pmax=100000.0*, *e_min=0*, *e_max=1000000.0*, *nb_unit_min=0*, *nb_unit_max=None*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *max_ramp_up=None*, *max_ramp_down=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.production_units.VariableProductionUnit*, *omegalpes.energy.units.energy_units.SeveralEnergyUnit*

Description

Production unit based on a fixed production curve enabling to multiply several times (nb_unit) the same production curve. Be careful, the solution may be imaginary as nb_unit can be continuous. The accurate number of the production unit should be calculated later

Attributes

- fixed_prod : fixed production curve

class omegalpes.energy.units.production_units.**SeveralProductionUnit**(*time*, *name*, *fixed_prod*, *pmin=1e-05*, *pmax=100000.0*, *e_min=0*, *e_max=1000000.0*, *nb_unit_min=0*, *nb_unit_max=None*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *max_ramp_up=None*, *max_ramp_down=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.production_units.VariableProductionUnit*, *omegalpes.energy.units.energy_units.SeveralEnergyUnit*

Description

Production unit based on a fixed production curve enabling to multiply several times (nb_unit) the same production curve nb_unit is an integer variable

Attributes

- fixed_prod : fixed production curve

class omegalpes.energy.units.production_units.**ShiftableProductionUnit**(*time*, *name: str*, *power_values*, *mandatory=True*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.ShiftableEnergyUnit*, *omegalpes.energy.units.production_units.VariableProductionUnit*

Description

Production unit with shiftable production profile.

Attributes

- power_values : production profile to shift (kW)
- mandatory : indicates if the production is mandatory : True or not : False
- starting_cost : cost of the starting of the production
- operating_cost : cost of the operation (€/kW)
- energy_type : type of energy ('Electrical', 'Heat', ...)
- operator : stakeholder how owns the production unit

class omegalpes.energy.units.production_units.**SquareProductionUnit**(*time*, *name*, *p_square*, *duration*, *n_square*, *t_between_sq*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.SquareEnergyUnit*, *omegalpes.energy.units.production_units.VariableProductionUnit*

Description

Production unit with a fixed value and fixed duration. Only the time of beginning can be modified Operation can be mandatory or not

Attributes

- p : instantaneous power production (kW)
- duration : duration of the power delivery (hours)
- mandatory : indicates if the power delivery is mandatory or not
- energy_type : type of energy ('Electrical', 'Heat', ...)
- operator : stakeholder how owns the production unit

class omegalpes.energy.units.production_units.**VariableProductionUnit**(*time*, *name*, *pmin=1e-05*, *pmax=100000.0*, *e_min=0*, *e_max=1000000.0*, *co2_out=None*, *starting_cost=None*, *operating_cost=None*, *min_time_on=None*, *min_time_off=None*, *max_ramp_up=None*, *max_ramp_down=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.VariableEnergyUnit*, *omegalpes.energy.units.production_units.ProductionUnit*

Description

Production unit with a variation of power between pmin et pmax.

Attributes

- pmax : maximal instantaneous power production (kW)
- pmin : minimal instantaneous power production (kW)
- energy_type : type of energy ('Electrical', 'Heat', ...)
- operator : stakeholder how owns the production unit

Conversion_units module

** This module defines the conversion units, with at least a production unit and a consumption unit and using one or several energy types**

The conversion_units module defines various classes of conversion units, from generic to specific ones.

It includes :

- ConversionUnit : simple conversion unit. It inherits from OptObject.
- ElectricalToHeatConversionUnit : Electrical to heat Conversion unit with an electricity consumption and a heat production linked by and electrical to heat ratio. It inherits from ConversionUnit
- HeatPump : Simple Heat Pump with an electricity consumption, a heat production and a heat consumption. It has a theoretical coefficient of performance COP and inherits from ConversionUnit.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class omegalpes.energy.units.conversion_units.**ConversionUnit**(*time*, *name*, *prod_units=None*, *cons_units=None*, *operator=None*)

Bases: omegalpes.general.optimisation.core.OptObject

Description

Simple Conversion unit

Attributes

- time : TimeUnit describing the studied time period
- prod_units : list of the production units
- cons_units : list of the consumption units
- operator : stakeholder who owns the conversion unit
- poles : dictionary of the poles of the conversion unit

class omegalpes.energy.units.conversion_units.**ElectricalToHeatConversionUnit**(*time*, *name*, *pmin_in_elec=1e-05*, *pmax_in_elec=1000*, *p_in_elec=None*, *pmin_out_heat=1e-05*, *pmax_out_heat=100*, *p_out_heat=None*, *elec_to_heat_ratio=*, *operator=None*)

Bases: *omegalpes.energy.units.conversion_units.ConversionUnit*

Description

Electrical to heat Conversion unit with an electricity consumption and a heat production

Attributes

- heat_production_unit : heat production unit (heat output)
- elec_consumption_unit : electricity consumption unit (electrical input)
- conversion : Dynamic Constraint linking the electrical input to the heat output through the electrical to heat ratio

class omegalpes.energy.units.conversion_units.**HeatPump**(*time*, *name*, *pmin_in_elec=1e-05*, *pmax_in_elec=100000.0*, *p_in_elec=None*, *pmin_in_heat=1e-05*, *pmax_in_heat=100000.0*, *p_in_heat=None*, *pmin_out_heat=1e-05*, *pmax_out_heat=100000.0*, *p_out_heat=None*, *cop=3*, *losses=0*, *operator=None*)

Bases: *omegalpes.energy.units.conversion_units.ConversionUnit*

Description

Simple Heat Pump with an electricity consumption, a heat production and a heat consumption. It has a theoretical coefficient of performance COP and inherits from ConversionUnit.

Attributes

- heat_production_unit : heat production unit (condenser)
- elec_consumption_unit : electricity consumption unit (electrical input)
- heat_consumption_unit : heay consumption unit (evaporator)
- COP : Quantity describing the coefficient of performance of the heat pump
- conversion : Dynamic Constraint linking the electrical input to the heat output through the electrical to heat ratio
- power_flow : Dynamic constraint linking the heat output to the electrical and heat inputs in relation to the losses.

Storage_units module

** This module defines the storage units**

The storage_units module defines various kinds of storage units with associated attributes and methods, from simple to specific ones.

It includes :

- StorageUnit : simple storage unit inheriting from EnergyUnit, with storage specific attributes. It includes the objective "minimize capacity".
- Thermocline storage : a thermal storage that need to cycle (i.e. reach SOC_max) every period of Tcycle

class omegalpes.energy.units.storage_units.**StorageUnit**(*time*, *name*, *pc_min=1e-05*, *pc_max=100000.0*, *pd_min=1e-05*, *pd_max=100000.0*, *capacity=None*, *e_0=None*, *e_f=None*, *soc_min=0*, *soc_max=1*, *eff_c=1*, *eff_d=1*, *self_disch=0*, *self_disch_t=0*, *ef_is_e0=False*, *cycles=None*, *energy_type=None*, *operator=None*)

Bases: *omegalpes.energy.units.energy_units.VariableEnergyUnit*

Description

Simple Storage unit

Attributes

- capacity (Quantity): maximal energy that can be stored
- e (Quantity): energy at time t in the storage
- set_soc_min (DynamicConstraint): constraining the energy to be above the value : soc_min*capacity
- set_soc_max (DynamicConstraint): constraining the energy to be below the value : soc_max*capacity
- pc (Quantity) : charging power
- pd (Quantity) : discharging power
- u_c (Quantity) : binary variable describing the charge of the storage unit : 0 : Not charging & 1 : charging
- calc_e (DynamicConstraint) : energy calculation at time t ; relation power/energy
- calc_p (DynamicConstraint) : power calculation at time t ; power flow equals charging power minus discharging power
- on_off_stor (DynamicConstraint) : making u[t] matching with storage modes (on/off)
- def_max_charging (DynamicConstraint) : defining the max charging power, avoiding charging and discharging at the same time
- def_max_discharging (DynamicConstraint) : defining the max discharging power, avoiding charging and discharging at the same time
- def_min_charging (DynamicConstraint) : defining the min charging power, avoiding charging and discharging at the same time
- def_min_discharging (DynamicConstraint) : defining the min discharging power, avoiding charging and discharging at the same time
- set_e_0 (ExternalConstraint) : set the energy state for t=0
- e_f (Quantity) : energy in the storage at the end of the time horizon, i.e. after the last time step
- e_f_min (Constraint) : e_f value is constrained above soc_min*capacity
- e_f_max (Constraint) : e_f value is constrained below soc_max*capacity
- set_e_f (Constraint) : when e_f is given, it is set in the same way the energy is, but after the last time step
- calc_e_f (Constraint) : when e_f is not given, it is calculated in the same way the energy is, but after the last time step

- ef_is_e0 (ExternalConstraint) : Imposing ef=e0 on the time period.
- cycles (ExternalDynamicConstraint) : setting a cycle constraint e[t] = e[t+cycles/dt]

minimize_capacity(*weight=1*)

Parameters **weight** – Weight coefficient for the objective

class omegalpes.energy.units.storage_units.**ThermoclineStorage**(*time*, *name*, *pc_min=1e-05*, *pc_max=100000.0*, *pd_min=1e-05*, *pd_max=100000.0*, *capacity=None*, *e_0=None*, *e_f=None*, *soc_min=0*, *soc_max=1*, *eff_c=1*, *eff_d=1*, *self_disch=0*, *Tcycl=120*, *ef_is_e0=False*, *operator=None*)

Bases: *omegalpes.energy.units.storage_units.StorageUnit*

Description

Class ThermoclineStorage : class defining a thermocline heat storage, inheriting from StorageUnit.

Attributes

- is_soc_max (Quantity) : indicating if the storage is fully charged 0:No 1:Yes
- def_is_soc_max_inf (DynamicConstraint) : setting the right value for is_soc_max
- def_is_soc_max_sup (DynamicConstraint) : setting the right value for is_soc_max
- force_soc_max (ExtDynConstraint) : The energy has to be at least once at its maximal value during the period Tcycl.

Energy_nodes module

** This module defines the energy nodes that will allow energy transmission between the various energy units and conversion units **

The energy_node module includes the EnergyNode class for energy transmission between production, consumption, conversion and storage. Defining several energy nodes and exporting/importing energy between them can also allow for a better demarcation of the energy system.

class omegalpes.energy.energy_nodes.**EnergyNode**(*time*, *name*, *energy_type=None*, *operator=None*)

Bases: omegalpes.general.optimisation.core.OptObject

This class defines an energy node.

add_connected_energy_unit(*unit*)
Add an EnergyUnit to the connected_units list

add_pole(*pole: omegalpes.energy.io.poles.Epole*) → None
Add an energy pole to the poles_list

Parameters **pole** – Epole

connect_units(**units*)
Connecting all EnergyUnit to the EnergyNode

Parameters **units** (*list*) – EnergyUnits connected to the EnergyNode

create_export(*node*, *export_min*, *export_max*)
Create the export from the EnergyNode (self) to the EnergyNode (node)

Parameters

- **node** – EnergyNode to whom power can be exported
- **export_min** – Minimal value of exported power when there is export
- **export_max** – Maximal value of exported power when there is export

Returns Quantity that defines the power exported

export_to_node(*node*, *export_min=1e-05*, *export_max=100000.0*)
Add an export of power from the node to another node

Parameters

- **node** – EnergyNode to whom power can be exported
- **export_min** – Minimal value of exported power when there is export
- **export_max** – Maximal value of exported power when there is export

get_connected_energy_units
Return the list of connected EnergyUnits in the EnergyNode

get_exports
Return the list of exports to the EnergyNode

get_flows
Get all the power flows of the energy node :rtype: list :return: list of power flows

get_imports
Return the list of imports to the EnergyNode

get_poles
Return the list of energy poles in the EnergyNode

import_from_node(*node*, *import_min=1e-05*, *import_max=100000.0*)

Parameters

- **node** – EnergyNode from whom power can be imported
- **import_min** – Minimal value of imported power when there is import
- **import_max** – Maximal value of imported power when there is import

is_export_flow(*flow*)
Get if the power flow is an export or not

is_import_flow(*flow*)
Get if the power flow is an import or not

set_power_balance()
Set the power balance equation for the EnergyNode

Poles module

** This module defines inputs and outputs of as poles**

The poles module includes :

- FlowPole : this class defines a pole with a directed flow (in or out)
- EPole : this class define an energy pole

class omegalpes.energy.io.poles.**Epole**(*p*, *direction*, *energy_type=None*)
Bases: *omegalpes.energy.io.poles.FlowPole*

** Description **

Definition of an energetic pole, power and power flow direction convention 'in' or 'out'

class omegalpes.energy.io.poles.**FlowPole**(*flow='flow'*, *direction='in'*)
Bases: dict

Interface for basics flux poles

The **actor classes** enables to build the energy model considering pre-defined stakeholders' constraints and objectives

2.4.2 actor package

The actor modelling is based on a main actor class defined on the actor module. Then, the actors are divided in two categories: the "operator actors" who operates energy units and the "regulator actors" who unable to create regulation constraints.

Actor module

** This modules define the basic Actor object **

Few methods are available:

- add_external_constraint
- add_external_dynamic_constraint
- add_objective

class omegalpes.actor.actor.**Actor**(*name*)

Bases: omegalpes.general.optimisation.core.OptObject

Description

Actor class is the basic class to model an actor. The basic actor is defined by its name and description. An actor is then defined by its constraints and objectives.

Attributes

- description : description as an Actor OptObject

add_external_constraint(*cst_name*, *exp*)

Enable to add an external constraint linked with an actor

Parameters

- **cst_name** – name of the constraint
- **exp** – expression of the constraint

add_external_dynamic_constraint(*cst_name*, *exp_t*, *t_range='for t in time.I'*)

Enable to add an external dynamic constraint linked with an actor. A dynamic constraint changes over time

Parameters

- **cst_name** – name of the constraint
- **exp** – expression of the constraint depending on the time
- **t_range** – expression of time for the constraint

add_objective(*obj_name*, *exp*)

Enable to add an objective linked with an actor

Parameters

- **obj_name** – name of the objective

- **exp** – expression of the objective

Operator_actors module

** This module defines the operator_actor and its scope of responsibility **

class omegalpes.actor.operator_actors.operator_actors.**OperatorActor**(*name*, *operated_unit_type_tuple*, *operated_unit_list=None*, *operated_node_list=None*)

Bases: *omegalpes.actor.actor.Actor*

Description

OperatorActor class inherits from the the basic class Actor. It enables one to model an actor who operates energy units which is part of its scope of responsibility. An operator actor has objectives and constraints which are linked to the energy units he operates.

Attributes

- name : name of the actor
- operated_unit_list: list of the energy units operated by the actor or more precisely in its scope of responsibility

Consumer_actors module

** This module describes the Consumer actor **

Few objectives and constraints are available. Objectives :

- maximize_consumption
- minimize_consumption
- minimize_co2_consumption
- minimize_consumption_costs

Constraints :

- energy_consumption_minimum
- energy_consumption_maximum
- power_consumption_minimum
- power_consumption_maximum

class omegalpes.actor.operator_actors.consumer_actors.**Consumer**(*name*, *operated_unit_list*, *operated_node_list=None*)

Bases: *omegalpes.actor.operator_actors.operator_actors.OperatorActor*

Description

Consumer class inherits from the the class OperatorActor. It enables one to model a consumer actor.

energy_consumption_maximum(*max_e_tot*, *cst_operated_unit_list=None*)

To create the actor constraint of a maximum of energy consumption.

Parameters

- **max_e_tot** – Maximum of the total energy consumption over the study period
- **cst_operated_unit_list** – List of units on which the constraint will be applied. Might be empty.

energy_consumption_minimum(*min_e_tot*, *cst_operated_unit_list=None*)

To create the actor constraint of a minimum of energy consumption.

Parameters

- **min_e_tot** – Minimum of the total energy consumption over the study period
- **cst_operated_unit_list** – List of units on which the constraint will be applied. Might be empty.

maximize_consumption(*obj_operated_unit_list=None*, *weight=1*)

To create the objective in order to maximize the consumption of the consumer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied. Might be empty
- **weight** – Weight coefficient for the objective

minimize_co2_consumption(*obj_operated_unit_list=None*, *weight=1*)

To create the objective in order to minimize the co2 emissions due to the consumer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied. Might be empty
- **weight** – Weight coefficient for the objective

minimize_consumption(*obj_operated_unit_list=None*, *weight=1*)

To create the objective in order to minimize the consumption of the consumer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied. Might be empty
- **weight** – Weight coefficient for the objective

minimize_consumption_cost(*obj_operated_unit_list=None*, *weight=1*)
To create the objective in order to minimize the expenses due to the consumer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied. Might be empty
- **weight** – Weight coefficient for the objective

power_consumption_maximum(*max_p*, *time*, *cst_operated_unit_list=None*)
To create the actor constraint of a maximum of power consumption.

Parameters

- **max_p** – Maximum of the power consumption. May be an int, float or a list with the size of the period study
- **time** – period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied. Might be empty.

power_consumption_minimum(*min_p*, *time*, *cst_operated_unit_list=None*)
To create the actor constraint of a minimum of power consumption.

Parameters

- **min_p** – Minimum of the power consumption. May be an int, float or a list with the size of the period study
- **time** – period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied. Might be empty.

Producer_actors module

** This module describes the Producer actor **

Few objectives and constraints are available. Objectives :

- maximize_production
- minimize_production
- minimize_time_of_use
- minimize_co2_emissions
- minimize_costs
- minimize_operating_cost
- minimize_starting_cost

Constraints :

- energy_production_minimum

- energy_production_maximum
- power_production_minimum
- power_production_maximum

class omegalpes.actor.operator_actors.producer_actors.**Producer**(*name*, *operated_unit_list*, *operated_node_list=None*)

Bases: *omegalpes.actor.operator_actors.operator_actors.OperatorActor*

Description Producer class inherits from the the class OperatorActor. It enables one to model an energy producer actor.

energy_production_maximum(*max_e_tot*, *cst_operated_unit_list=None*)

To create the actor constraint of a maximum of energy production.

Parameters

- **max_e_tot** – Maximum of the total energy production over the period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied. Might be empty.

energy_production_minimum(*min_e_tot*, *cst_operated_unit_list=None*)

To create the actor constraint of a minimum of energy production.

Parameters

- **min_e_tot** – Minimum of the total energy production over the period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied. Might be empty.

maximize_production(*obj_operated_unit_list=None*, *weight=1*)

To create the objective in order to maximize the production of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied. Might be empty.
- **weight** – Weight coefficient for the objective

minimize_co2_emissions(*obj_operated_unit_list=None*, *weight=1*)

To create the objective in order to minimize the co2 emissions of the producer's units (all or part of them). based on the quantity "co2_emission"

Parameters

- **obj_operated_unit_list** – list of the operated energy units on which the objective will be applied

- **weight** – weight of the objective

minimize_costs(*obj_operated_unit_list=None*, *weight=1*)
To create the objective in order to minimize the cost of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – list of the operated energy units on which the objective will be applied
- **weight** – weight of the objective

minimize_operating_cost(*obj_operated_unit_list=None*, *weight=1*)
To create the objective in order to minimize the operating costs of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – list of the operated energy units on which the objective will be applied
- **weight** – weight of the objective

minimize_production(*obj_operated_unit_list=None*, *weight=1*)
To create the objective in order to minimize the production of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied. Might be empty.
- **weight** – Weight coefficient for the objective

minimize_starting_cost(*obj_operated_unit_list=None*, *weight=1*)
To create the objective in order to minimize the starting costs of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – list of the operated energy units on which the objective will be applied
- **weight** – weight of the objective

minimize_time_of_use(*obj_operated_unit_list=None*, *weight=1*)
To create the objective in order to minimize the time of use of the producer's units (all or part of them).

Parameters

- **obj_operated_unit_list** – List of units on which the objective will be applied. Might be empty.
- **weight** – Weight coefficient for the objective

power_production_maximum(*max_p*, *time*, *cst_operated_unit_list=None*)
To create the actor constraint of a maximum of power production.

Parameters

- **max_p** – Minimum of the power production. May be an int, float or a list with the size of the period study
- **time** – period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied. Might be empty.

power_production_minimum(*min_p*, *time*, *cst_operated_unit_list=None*)
To create the constraint of a minimum of power production.

Parameters

- **min_p** – Minimum of the power production. May be an int, float or a list with the size of the period study
- **time** – period of the study
- **cst_operated_unit_list** – List of units on which the constraint will be applied. Might be empty.

Consumer_producer_actors module

** This module describes the Prosumer (producer and consumer) actor **

class omegalpes.actor.operator_actors.consumer_producer_actors.**Prosumer**(*name*, *operated_consumption_unit_list*, *operated_production_unit_list*, *operated_node_list=None*)

Bases: *omegalpes.actor.operator_actors.consumer_actors.Consumer*, *omegalpes.actor.operator_actors.producer_actors.Producer*

Description Prosumer class inherits from the the class OperatorActor, Consumer and Producer. It enables one to model an actor which is at the same time an energy producer and consumer

maximize_conso_prod_match(*time*, *obj_operated_consumption_unit_list=None*, *obj_operated_production_unit_list=None*, *weight=1*)

To create the objective in order to match at each time the consumption with the local production of the prosumer's units (all or part of them).

Parameters

- **obj_operated_consumption_unit_list** – List of consumption units on which the objective will be applied. Might be empty.
- **obj_operated_production_unit_list** – List of production units on which the objective will be applied. Might be empty.
- **weight** – Weight coefficient for the objective

maximize_selfconsumption_rate(*time*, *obj_operated_production_unit_list=None*, *obj_operated_consumption_unit_list=None*, *obj_operated_selfconsummed_production_export_list=None*, *obj_operated_selfconsummed_production_unit_list=None*, *weight=1*)

To create the objective in order to maximize the selfconsumption rate of the prosumer's units (all or part of them) WHILE maximizing the load matching selfconsummed production is calculated with the export nodes.

Selfconsumption rate = selfconsummed production / total production

Parameters

- **obj_operated_production_unit_list** – List of production units on which the objective will be applied. Might be empty.
- **obj_operated_selfconsummed_production_export_list** – List of production exports from nodes on which the objective will be applied. Might be empty.
- **obj_operated_selfconsummed_production_unit_list** – List of production units on which the objective will be applied. Might be empty.
- **weight** – Weight coefficient for the objective

maximize_selfproduction_rate(*time*, *obj_operated_consumption_unit_list=None*, *obj_operated_production_unit_list=None*, *obj_operated_selfproduced_consumption_export_list=None*, *obj_operated_selfproduced_consumption_unit_list=None*, *weight=1*)

To create the objective in order to maximize the selfproduction rate of the prosumer's units (all or part of them) WHILE maximizing the load matching selfproduced consumption may required export nodes

Selfproduction rate = selfproduced consumption / total consumption

Parameters

- **obj_operated_consumption_unit_list** – List of consumption units on which the objective will be applied. Might be empty.
- **obj_operated_selfproduced_consumption_export_list** – List of production exports from nodes on which the objective will be applied. Might be empty.
- **obj_operated_selfproduced_consumption_unit_list** – List of production units on which the objective will be applied. Might be empty.
- **weight** – Weight coefficient for the objective

class omegalpes.actor.operator_actors.consumer_producer_actors.**Supplier**(*name*, *operated_consumption_unit_list*, *operated_production_unit_list*)

Bases: *omegalpes.actor.operator_actors.consumer_actors.Consumer*, *omegalpes.actor.operator_actors.producer_actors.Producer*

Description Supplier class inherits from the the class OperatorActor, Consumer and Producer. It enables one to model a supplier.

Regulator_actors module

**** This module defines the operator_actor and its scope of responsibility ****

One constraint is available :

- co2_emission_maximum

class omegalpes.actor.regulator_actors.regulator_actors.**LocalAuthorities**(*name*)
Bases: *omegalpes.actor.regulator_actors.regulator_actors.RegulatorActor*

Description LocalAuthorities class inherits from the basic class RegulatorActor. It focuses on local Authorities constraints

class omegalpes.actor.regulator_actors.regulator_actors.**PublicAuthorities**(*name*)
Bases: *omegalpes.actor.regulator_actors.regulator_actors.RegulatorActor*

Description PublicAuthorities class inherits from the basic class RegulatorActor. It focuses on local Authorities constraints

class omegalpes.actor.regulator_actors.regulator_actors.**RegulatorActor**(*name*)
Bases: *omegalpes.actor.actor.Actor*

Description

RegulatorActor class inherits from the the basic class Actor. It enables one to model an actor who can add constraints on all energy units of the study case.

Attributes

- name : name of the actor

co2_emission_maximum(*max_co2*, *time*, *cst_production_list*)
To create the actor constraint of a maximum of CO2 emission.

Parameters

- **max_co2** – Minimum of the CO2 emission. May be an int, float or a list with the size of the period study
- **time** – period of the study
- **cst_unit_list** – List of units on which the constraint will be applied.

The **general classes** helps to build the units, the model and to plot the results

2.4.3 general package

Please, have a look to the following general modules

Elements module

This module includes the optimization elements (quantities, constraints and objectives) formulated in LP or MILP :

- Quantity : related to the decision variable or parameter
- Constraint : related to the optimization problem constraints
- Objective : related to the objective function

class omegalpes.general.optimisation.elements.**Constraint**(*exp*, *name='CST0'*, *description=''*, *active=True*, *parent=None*)

Bases: `object`

Description

Class that defines a constraint object

Attributes

- name: name of the constraint
- description: a description of the constraint
- active: False = non-active constraint; True = active constraint
- exp: (str) : expression of the constraint
- parent: (unit) : this constraint belongs to this unit

Note: Make sure that all the modifications on Constraints are made before adding the unit to the Model (OptimisationModel.addUnit()).

---

class omegalpes.general.optimisation.elements.**DynamicConstraint**(*exp_t*, *t_range='for t in time.I'*, *name='DCST0'*, *description='dynamic constraint'*, *active=True*, *parent=None*)

Bases: *omegalpes.general.optimisation.elements.Constraint*

** Description **

Defining a constraint depending on the time. NB : Mandatory for PuLP

class omegalpes.general.optimisation.elements.**ExtDynConstraint**(*exp_t*, *t_range='for t in time.I'*, *name='EDCST0'*, *active=True*, *description='Non-physical and dynamic constraint'*, *parent=None*)

Bases: *omegalpes.general.optimisation.elements.DynamicConstraint*, *omegalpes.general.optimisation.elements.ExternalConstraint*

** Description **

Defining a constraint both external and dynamic (see: DynamicConstraint, ExternalConstraint)

class omegalpes.general.optimisation.elements.**ExternalConstraint**(*exp*, *name='ExCST0'*, *description=''*, *active=True*, *parent=None*)

Bases: *omegalpes.general.optimisation.elements.Constraint*

** Description **

Defining a special type of constraint [the external constraint]

- This constraint does not translate a physical constraint
- This constraint defines an external constraint, which could be relaxed

deactivate_constraint()

An external constraint can be deactivated : - To compare scenarios - To try a less constrained problem

class omegalpes.general.optimisation.elements.**HourlyDynamicConstraint**(*exp_t*, *time*, *init_h: int = 0*, *final_h: int = 24*, *name='HDCST0'*, *description='hourly dynamic constraint'*, *active=True*, *parent=None*)

Bases: *omegalpes.general.optimisation.elements.DynamicConstraint*

** Description **

Class that defines an dynamic contraint for a time range

Ex : Constraint applying between 7am and 10pm

ex_cst = HourlyDynamicConstraint(exp_t, time, init_h=7, final_h=22, name='ex_cst')

Attributes

- name (str) : name of the constraint
- exp_t (str) : expression of the constraint
- init_h (int) : hour of beginning of the constraint [0-23]
- final_h (int) : hour of end of the constraint [1-24]
- description (str) : description of the constraint
- active (bool) : defines if the constraint is active or not
- parent (OptObject) : parent of the constraint

class omegalpes.general.optimisation.elements.**Objective**(*exp*, *name='OBJ0'*, *description=''*, *active=True*, *weight=1*, *unit='s.u.'*, *parent=None*)

Bases: `object`

Description

Class that defines an optimisation objective

Attributes

- name (str) :
- description (str) :
- active (bool) :

- exp (str) :
- weight (float) : weighted factor of the objective
- parent (unit)
- unit (str) : unit of the cost expression

Note: Make sure that all the modifications on Objectives are made before adding the unit to the Optimisation-Model, otherwise, it won't be taken into account

---

class omegalpes.general.optimisation.elements.**Quantity**(*name='var0'*, *opt=True*, *unit='s.u'*, *vlen=None*, *value=None*, *description=''*, *vtype='Continuous'*, *lb=None*, *ub=None*, *parent=None*)

Bases: object

" **Description**

Class that defines what is a quantity. A quantity can wether be a decision variable or a parameter, depending on the opt parameter

Attributes

- name (str) : the name of the quantity
- description (str) : a description of the meaning of the quantity
- **vtype (PuLP)** [the variable type, depending on PuLP :]
  - LpBinary (binary variable)
  - LpInteger (integer variable)
  - LpContinuous (continuous variable)
- vlen (int) : size of the variable
- unit (str) : unit of the quantity
- **opt (binary) :**
  - True: this is an optimization variable
  - False: this is a constant - a parameter
- value (float, list, dict) : value (unused if opt=True)
- ub, lb : upper and lower bounds
- parent (OptObject) : the quantity belongs to this unit

Note: Make sure that all the modifications on Quantity are made before adding the unit to the Model

---

Model module

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class omegalpes.general.optimisation.model.**OptimisationModel**(*time*, *name='optimisation_model'*)

Bases: pulp.pulp.LpProblem

** Description **

This module includes the optimization model formulated in LP or MILP based on the package PuLP (LpProblem)

add_nodes(**nodes*)
Add nodes and all connected units to the model Check that the time is the same for the model and all the units

Parameters **nodes** – EnergyNode

add_nodes_and_actors(**nodes_or_actors*)
Add nodes, actors and all connected units to the model Check that the time is the same for the model and all the units

Parameters **nodes_or_actors** – EnergyNode or Actor type

get_model_constraints_list()
Get constraints of the model

get_model_constraints_name_list()
Get the names of the constraints of the model

get_model_objectives_list()
Get objectives of the model

get_model_objectives_name_list()
Get the names of the objectives of the model

get_model_quantities_list()
Get quantities of the model

get_model_quantities_name_list()
Get the names of the quantities of the model

solve_and_update(*solver: pulp.solvers.LpSolver = None*) → None
Solves the optimization model and updates all variables values.

Parameters **solver** (*LpSolver*) – Optimization solver

update_units()
Updates all units values with optimization results

omegalpes.general.optimisation.model.**check_if_unit_could_have_parent**(*unit*)
Check if the unit has an associated parent

:param : unit

omegalpes.general.optimisation.model.**compute_gurobi_IIS**(*gurobi_exe_path='C:\\gurobi800\\win64\\bin'*, *opt_model=None*, *MPS_model=None*)

Parameters

- **gurobi_exe_path** – Path to the gurobi solver "gurobi_cl.exe"
- **opt_model** – OptimisationModel to whom compute IIS
- **MPS_model** – name of the mps model

Units module

Plots module

This module includes the display tools

omegalpes.general.plots.**plot_energy_mix**(*node*)

omegalpes.general.plots.**plot_node_energetic_flows**(*node*)
** Description ** This function allows to plot the energy flows through an EnergyNode

The display is realized :

- with histograms for production and storage flow
- with dashed curve for consumption flow

Parameters **node** – EnergyNode

omegalpes.general.plots.**plot_quantity**(*time*, *q*, *fig=None*, *ax=None*, *color=None*)

Description

Function that plots a OMEGALPES.general.optimisation.elements.Quantity

Attributes

- q is the Quantity
- **fig could be None, a matplotlib.pyplot.Figure or Axes for multiple** plots

Returns

- arg1 the matplotlib.pyplot.Figure handle object
- arg2 the matplotlib.pyplot.Axes handle object
- arg3 the matplotlib.pyplot.Line2D handle object

Time module

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

class omegalpes.general.time.**TimeUnit**(*start='01/01/2018'*, *end=None*, *periods=None*, *dt=1*)
Bases: object

Description

Class defining the studied time period.

Attributes

- DATES : dated list of simulation steps
- DT : delta t between values in hours (int or float), i.e. 1/6 will be 10 minutes.
- LEN : number of simulation steps (length of DATES)
- I : index of time ([0 : LEN])

get_date_for_index(*index*)
Getting a date for a given index

Parameters **index** – int value for the index of the wanted dated, between 0 and LEN (it must be in the studied period)

get_days
Getting days for the studied period

Return all_days list of days of the studied period

get_index_for_date(*date='YYYY-MM-DD HH:MM:SS'*)
Getting the index associated with a date

Parameters **date** – date the index of is wanted. Format YYYY-MM-DD HH:MM:SS, must be within the studied period and consistent with the timestep value

get_index_for_date_range(*starting_date='YYYY-MM-DD HH:MM:SS'*, *end=None*, *periods=None*)
Getting a list of index for a date range

Parameters

- **starting_date** – starting date of the wanted index
- **end** – ending date of the wanted index
- **periods** – number of periods from the starting_date of the wanted index

Return index_list list of indexes for the given dates

get_non_working_dates(*month_range=range(0, 12)*, *hour_range=range(0, 24)*, *country='France'*)

get_non_working_days(*country='France'*)

get_working_dates(*month_range=range(0, 12)*, *hour_range=range(0, 24)*, *country='France'*)

get_working_days(*country='France'*)

print_studied_period()

omegalpes.general.time.**convert_european_format**(*date*)

Converting a date with an european format DD/MM/YYYY into a datetime format YYYY-MM-DD or return

Parameters **date** – date in european format

Returns date in format datetime

Utils module

Description

This module includes the utils

- To save results
- To define easily absolute value for quantity

omegalpes.general.utils.**def_abs_value**(*quantity*, *q_min*, *q_max*)

Parameters

- **quantity** – Quantity whose absolute value is wanted
- **q_min** – Minimal value of the quantity (negative value)
- **q_max** – Maximal value of the quantity (positive value)

Returns A new Quantity whose values equal absolute values of the initial Quantity

omegalpes.general.utils.**save_energy_flows**(**nodes*, *file_name=None*, *sep='\t'*, *decimal_sep='.'*)

omegalpes.general.utils.**select_csv_file_between_dates**(*file_path=None*, *start='DD/MM/YYYY HH:MM'*, *end='DD/MM/YYYY HH:MM'*, *v_cols=[]*, *sep=';'*)

2.5 OMEGAlpes Examples

Please find OMEGAlpes examples on the following link: https://omegalpes-examples.readthedocs.io/

It gathers detailed examples to explain the steps in order to create a model and article study cases.

CHAPTER 3

Acknowledgments

Vincent Reinbold - Library For Linear Modeling of Energetic Systems : https://github.com/ReinboldV

Mathieu Brugeron

This work has been partially supported by the CDP Eco-SESA receiving fund from the French National Research Agency in the framework of the “Investissements d’avenir” program (ANR-15-IDEX-02) and the VALOCAL project (CNRS Interdisciplinary Mission and INSIS)

Python Module Index

o

Index

E

F

G

H

I

L

M

O

V