# EnerPyFlow

***Release 0.0.1***

**Hortense Ronzani**

**Oct 01, 2025**

# CONTENTS:

Add your content using `reStructuredText` syntax. See the reStructuredText documentation for details.

CHAPTER

# ONE

# COMPONENTS MODULE

**class** components.**Component**(*name*, *energy*, *environment*, *description*, *nb_of_timesteps=None*, *isHub=False*, *i=None*, *log=False*)

Bases: `object`

This class represents the base components and some fictional objects and is used to build the links (energy flows).

**name**

Name of the component.

**Type**
str

**energy**

Main energy type of the component.

**Type**
str

**environment**

Main environment of the component.

**Type**
str

**description**

Portion of the configuration file describing the characteristics of the links

**Type**
dict

**nb_of_timesteps**

Number of timesteps of the simulation.

**Type**
int

**maximum**

maximum value(s) of energy flow in and out of the component. Can be one value (ex: maximum electricity exchanges through grid) or a sequence of the same duration of the simulation duration (ex: hourly PV production). In this case, a string describing where the data is stored must be given under this form : “path/to/data.csv//column_name_of_maximum”. Value(s) must be given in the unit of the energy type attached to the object.

**Type**
float | str

**maximum_in**

maximum of the energy flow in the sense Hub -> Component. Same format as maximum, default value is maximum, crushes maximum if both maximum_in and maximum are given (only maximum_in in used in the problem constraints).

**Type**
float | str

**maximum_out**

maximum of the energy flow in the sense Component -> Hub. Same format as maximum, default value is maximum, crushes maximum if both maximum_out and maximum are given (only maximum_out in used in the problem constraints).

**Type**
float | str

**minimum**

minimum value(s) of energy flow in and out of the component. Can be one value or a sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_minimum". Value(s) must be given in the unit of the energy type attached to the object. Default is 0. ; it highly recommanded to NOT PUT NEGATIVE VALUES.

**Type**
float | str

**minimum_in**

minimum of the energy flow in the sense Hub -> Component. Same format as minimum, default value is minimum, crushes minimum if both minimum_in and minimum are given (only minimum_in in used in the problem constraints).

**Type**
float | str

**minimum_out**

minimum of the energy flow in the sense Component -> Hub. Same format as minimum, default value is minimum, crushes minimum if both minimum_out and minimum are given (only minimum_out in used in the problem constraints).

**Type**
float | str

**cost**

cost value(s) of energy flow in and out of the component per energy type unit. Can be one value or a sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_cost". Value(s) must be given in cost unit per unit of the energy type attached to the object. Default is 0. A positive value is a cost, a negative value is a gain.

**Type**
float | str

**cost_in**

minimum of the energy flow in the sense Hub -> Component. Same format as cost, default value is -cost, crushes cost if both cost_in and cost are given (only cost_in in used in the problem constraints).

**Type**
float | str

`cost_out`

minimum of the energy flow in the sense Component -> Hub. Same format as cost, default value is cost, crushes cost if both cost_out and cost are given (only cost_out in used in the problem constraints).

**Type**
float | str

`factor`

actual minimum and maximum flow values are multiplied by factor. Example of use: if the input timeseries given as maximum_out describes the production of a 1 kWc PV panel but we want to have 5 kWc of PV installed, set factor to 5 ; if the input timeseries describes a demand that we want to split as 50% dispatchable and 50% not dispatchable, create to Demand objects (one with dispatchable=True, the other with dispatchable=False) with this input timeseries and set factor=0.5 for both objects ; if the best size of component has to be automatically optimized, set factor='auto'. WARNING: 'auto' option is not compatible with dispatchable Demand objects.

**Type**
int | float | str

`factor_low_bound`

Used if factor is set to 'auto'. factor_low_bound <= factor.

**Type**
int | float

`factor_up_bound`

Used if factor is set to 'auto'. factor_up_bound => factor.

**Type**
int | float

`factor_type`

Used if factor is set to 'auto' and set the factor type. Can be 'Continuous', 'Integer' or 'Binary'. Example of use for 'Integer' : how many PV panels of 200 Wc should be installed ? Example of use for 'Binary' : should a generator be installed ? (Yes : factor=1., No : factor=0.)

**Type**
str

`installation_cost`

Cost for one unit installed (factor=1.) Total installation cost will be installation_cost * factor.

**Type**
float

`flow_vars_in`

List of length nb_of_timesteps containing the flow-in variables (from Hub to Component) of the Component at each timestep. Once the problem solved, flow_vars_in[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

**Type**
list[pulp.LpVariable]

`flow_vars_out`

List of length nb_of_timesteps containing the flow-out variables (from Component to Hub) of the Component at each timestep. Once the problem solved, flow_vars_out[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

**Type**
list[pulp.LpVariable]

**Parameters**

- **name** (*str*) – Name of the component.
- **energy** (*str*) – Main energy type of the component.
- **environment** (*str*) – Main environment of the component.
- **description** (*dict*) – Sub-section of the configuration file describing the characteristics of the links (attributes of the Component).
- **nb_of_timesteps** (*int*) – Number of timesteps of the simulation.
- **isHub** (*bool*) – Must be True if the component described is a Hub (Component subclass), else False.
- **i** (*int | None*) – To be used only if the component described is an EnvironmentsConnection (Component sub-class) to iterate within a list of specifications.
- **log** (*bool*) – If True, additional information will be printed throughout the initialization.

**get_hub**(*hubs*, *environment=None*, *energy=None*, *log=False*)

This method gets the Hub(environment, energy) to be linked to the Component object, and creates it if it doesn't exist.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **environment** (*str*) – Environment of the Hub to be selected. Default is self.environment, the main environment attached to the Component object.
- **energy** (*str*) – Energy of the Hub to be selected. Default is self.energy, the main energy type attached to the Component object.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**
Hub object corresponding to (environment, energy).

**Return type**
*Hub*(*Component*)

**link**(*hubs*, *model*, *log=False*)

This methods creates flow_vars_in and flow_vars_out lists of flow variables that will be optimized, adds them to them to the Hub equation (energy conservation equation), and add the constraints factor*minimum_in[t] <= flow_vars_in[t] <= factor*maximum_in[t] and factor*minimum_out[t] <= flow_vars_out[t] <= factor*maximum_out[t] for every timestep t between 0 and nb_of_timesteps-1. Updates the model objective function with costs attributed to the flow variables: model.objective += sum_t flow_vars_in[t]*cost_in[t] + flow_vars_out[t]*cost_out[t].

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**

Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**

pandas.DataFrame

`class components.Converter(name, description, nb_of_timesteps, log=False)`

Bases: *Component*

Sub-class of Component class representing a converter object. Allows to link different hubs within a same environment with different energy types, with efficiency factors for the conversion. Only one sense is allowed for energy conversion, from only one input hub to one or several output hubs.

**Inherited Attributes:**

name (str): Name of the component. energy (str): Main energy type of the component. environment (str): Main environment of the component. description (dict): Portion of the configuration file describing the characteristics of the links nb_of_timesteps (int): Number of timesteps of the simulation. maximum (float | str): maximum value(s) of energy flow in and out of the component. Can be one value

> (ex: maximum electricity exchanges through grid) or a sequence of the same duration of the simulation duration (ex: hourly PV production). In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_maximum". Value(s) must be given in the unit of the energy type attached to the object.

**maximum_in (float | str): maximum of the energy flow in the sense Hub -> Component. Same format as maximum,**

default value is maximum, crushes maximum if both maximum_in and maximum are given (only maximum_in in used in the problem constraints).

**maximum_out (float | str): maximum of the energy flow in the sense Component -> Hub. Same format as maximum,**

default value is maximum, crushes maximum if both maximum_out and maximum are given (only maximum_out in used in the problem constraints). ALWAYS=0. FOR A CONVERTER OBJECT.

**minimum (float | str): minimum value(s) of energy flow in and out of the component. Can be one value or a**

sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_minimum". Value(s) must be given in the unit of the energy type attached to the object. Default is 0. ; it highly recommanded to NOT PUT NEGATIVE VALUES.

**minimum_in (float | str): minimum of the energy flow in the sense Hub -> Component. Same format as minimum,**

default value is minimum, crushes minimum if both minimum_in and minimum are given (only minimum_in in used in the problem constraints).

**minimum_out (float | str): minimum of the energy flow in the sense Component -> Hub. Same format as minimum,**

default value is minimum, crushes minimum if both minimum_out and minimum are given (only minimum_out in used in the problem constraints).

**cost (float | str): cost value(s) of energy flow in and out of the component per energy type unit. Can be one**

value or a sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form :

"path/to/data.csv//column_name_of_cost". Value(s) must be given in cost unit per unit of the energy type attached to the object. Default is 0. A positive value is a cost, a negative value is a gain.

**cost_in (float | str): minimum of the energy flow in the sense Hub -> Component. Same format as cost, default**
value is -cost, crushes cost if both cost_in and cost are given (only cost_in in used in the problem constraints).

**cost_out (float | str): minimum of the energy flow in the sense Component -> Hub. Same format as cost, default**
value is cost, crushes cost if both cost_out and cost are given (only cost_out in used in the problem constraints).

**factor (int | float | str): actual minimum and maximum flow values are multiplied by factor. Example of use: if the**
input timeseries given as maximum_out describes the production of a 1 kWc PV panel but we want to have 5 kWc of PV installed, set factor to 5 ; if the input timeseries describes a demand that we want to split as 50% dispatchable and 50% not dispatchable, create to Demand objects (one with dispatchable=True, the other with dispatchable=False) with this input timeseries and set factor=0.5 for both objects ; if the best size of component has to be automatically optimized, set factor='auto'. WARNING: 'auto' option is not compatible with dispatchable Demand objects.

factor_low_bound (int | float): Used if factor is set to 'auto'. factor_low_bound <= factor. factor_up_bound (int | float): Used if factor is set to 'auto'. factor_up_bound => factor. factor_type (str): Used if factor is set to 'auto' and set the factor type. Can be 'Continuous', 'Integer' or 'Binary'.

Example of use for 'Integer' : how many PV panels of 200 Wc should be installed ? Example of use for 'Binary' : should a generator be installed ? (Yes : factor=1., No : factor=0.)

**installation_cost (float): Cost for one unit installed (factor=1.) Total installation cost will be**
installation_cost * factor.

**flow_vars_in (list[pulp.LpVariable]): List of length nb_of_timesteps containing the flow-in variables (from Hub to Component)**
of the Component at each timestep. Once the problem solved, flow_vars_in[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

`input_energy`

Input energy type.

**Type**
str

`output_energies`

List of output energies.

**Type**
list[str]

`flow_vars_out (dict[str`

list[pulp.LpVariable]]): Dictionnary with output energy types as keys and lists of length nb_of_timesteps containing the flow-out variables (from Component to Hub) of the Converter at each timestep, for each type of output energy. Once the problem solved, flow_vars_out[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

**equations (dict[str**

list[pulp.LpConstraint]]): Dictionnary with output energy types as keys and lists of length nb_of_timesteps containing the conversion equations from the input energy type to each output energy type, at each timestep: flow_vars_out[output_energy][t] = conversion_ratio*flow_vars_in[t] for each timestep t between 0 and nb_of_timesteps-1.

**Parameters**

- **name** (*str*) – Name of the component.
- **description** (*dict*) – Portion of the configuration file describing the characteristics of the links
- **nb_of_timesteps** (*int*) – Number of timesteps of the simulation.
- **log** (*bool*) – If True, additional information will be printed throughout the initialization.

**build_equations**(*hubs*, *model=None*, *log=False*)

Method to build the conversion equations between output energy types and input energy type. Calls method Component.link() to build variables and constraints related to the input energy hub, creates flow_vars_out variables for the links with the output energy hubs, adds them to the output energy hubs equations, builds conversions equations between input energy flows and output energy flows and adds them to the linear problem as constraints.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**

Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**

pandas.DataFrame

**get_hub**(*hubs*, *environment=None*, *energy=None*, *log=False*)

This method gets the Hub(environment, energy) to be linked to the Component object, and creates it if it doesn't exist.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **environment** (*str*) – Environment of the Hub to be selected. Default is self.environment, the main environment attached to the Component object.
- **energy** (*str*) – Energy of the Hub to be selected. Default is self.energy, the main energy type attached to the Component object.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**

Hub object corresponding to (environment, energy).

**Return type**

*Hub*(*Component*)

**link**(*hubs*, *model*, *log=False*)

This methods creates flow_vars_in and flow_vars_out lists of flow variables that will be optimized, adds them to them to the Hub equation (energy conservation equation), and add the constraints factor*minimum_in[t] <= flow_vars_in[t] <= factor*maximum_in[t] and factor*minimum_out[t] <= flow_vars_out[t] <= factor*maximum_out[t] for every timestep t between 0 and nb_of_timesteps-1. Updates the model objective function with costs attributed to the flow variables: model.objective += sum_t flow_vars_in[t]*cost_in[t] + flow_vars_out[t]*cost_out[t].

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**

Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**

pandas.DataFrame

**class** components.**Demand**(*name*, *description*, *nb_of_timesteps*, *log=False*)

Bases: *Component*

Sub-class of Component class representing an energy demand. Can be dispatchable within some limitations or not dispatchable (default).

**Inherited Attributes:**

name (str): Name of the component. energy (str): Main energy type of the component. environment (str): Main environment of the component. description (dict): Portion of the configuration file describing the characteristics of the links nb_of_timesteps (int): Number of timesteps of the simulation. maximum (float | str): maximum value(s) of energy flow in and out of the component. Can be one value

(ex: maximum electricity exchanges through grid) or a sequence of the same duration of the simulation duration (ex: hourly PV production). In this case, a string describing where the data is stored must be given under this form : “path/to/data.csv//column_name_of_maximum”. Value(s) must be given in the unit of the energy type attached to the object.

**maximum_in (float | str): maximum of the energy flow in the sense Hub -> Component. Same format as maximum,**

default value is maximum, crushes maximum if both maximum_in and maximum are given (only maximum_in in used in the problem constraints).

**maximum_out (float | str): maximum of the energy flow in the sense Component -> Hub. Same format as maximum,**

default value is maximum, crushes maximum if both maximum_out and maximum are given (only maximum_out in used in the problem constraints). ALWAYS=0. FOR A DEMAND OBJECT.

**minimum (float | str): minimum value(s) of energy flow in and out of the component. Can be one value or a**

sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form : “path/to/data.csv//column_name_of_minimum”. Value(s) must be given in the unit of the energy type attached to the object. ALWAYS=0. FOR A DEMAND OBJECT.

**minimum_in (float | str): minimum of the energy flow in the sense Hub -> Component. Same format as minimum,**
default value is minimum, crushes minimum if both minimum_in and minimum are given (only minimum_in in used in the problem constraints). If demand is not dispatchable, will be force to minimum_in=value.

**minimum_out (float | str): minimum of the energy flow in the sense Component -> Hub. Same format as minimum,**
default value is minimum, crushes minimum if both minimum_out and minimum are given (only minimum_out in used in the problem constraints). If demand is not dispatchable, will be force to minimum_in=value.

**cost (float | str): cost value(s) of energy flow in and out of the component per energy type unit. Can be one**
value or a sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_cost". Value(s) must be given in cost unit per unit of the energy type attached to the object. Default is 0. A positive value is a cost, a negative value is a gain.

**cost_in (float | str): minimum of the energy flow in the sense Hub -> Component. Same format as cost, default**
value is -cost, crushes cost if both cost_in and cost are given (only cost_in in used in the problem constraints).

**cost_out (float | str): minimum of the energy flow in the sense Component -> Hub. Same format as cost, default**
value is cost, crushes cost if both cost_out and cost are given (only cost_out in used in the problem constraints).

**factor (int | float | str): actual minimum and maximum flow values are multiplied by factor. Example of use: if the**
input timeseries given as maximum_out describes the production of a 1 kWc PV panel but we want to have 5 kWc of PV installed, set factor to 5 ; if the input timeseries describes a demand that we want to split as 50% dispatchable and 50% not dispatchable, create to Demand objects (one with dispatchable=True, the other with dispatchable=False) with this input timeseries and set factor=0.5 for both objects. 'AUTO' OPTION IS NOT COMPATIBLE WITH DISPATCHABLE DEMAND OBJECTS.

factor_low_bound (int | float): Used if factor is set to 'auto'. factor_low_bound <= factor. factor_up_bound (int | float): Used if factor is set to 'auto'. factor_up_bound => factor. factor_type (str): Used if factor is set to 'auto' and set the factor type. Can be 'Continuous', 'Integer' or 'Binary'.

Example of use for 'Integer' : how many PV panels of 200 Wc should be installed ? Example of use for 'Binary' : should a generator be installed ? (Yes : factor=1., No : factor=0.)

**installation_cost (float): Cost for one unit installed (factor=1.) Total installation cost will be**
installation_cost * factor.

**flow_vars_in (list[pulp.LpVariable]): List of length nb_of_timesteps containing the flow-in variables (from Hub to Component)**
of the Component at each timestep. Once the problem solved, flow_vars_in[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

**flow_vars_out (list[pulp.LpVariable]): List of length nb_of_timesteps containing the flow-out variables (from Hub to Component)**

of the Component at each timestep. Once the problem solved, flow_vars_out[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

`dispatchable`

True if demand is dispatchable, else False.

**Type**
bool

`dispatch_window`

Used only if demand is dispatchable. Indicates the extend to when each chunk of initial demand can be displaced. Example: if dispatch_window=24, each chunk of initial demand can be dispatched within the time window [-12, +12].

**Type**
int

`value`

List of length nb_of_timesteps with initial demand values, in energy type unit.

**Type**
List[float]

`y_vars`

Only if demand is dispatchable. Table of variables indicating when each chunk of initial demand is displaced.

**Type**
pandas.DataFrame

**Parameters**

- **name** (*str*) – Name of the component.
- **description** (*dict*) – Portion of the configuration file describing the characteristics of the component links.
- **nb_of_timesteps** (*int*) – Number of timesteps of the simulation.
- **log** (*bool*) – If True, additional information will be printed throughout the initialization.

`build_equations`(*hubs*, *model*, *log=False*)

Method to build demand-specific variables and constraints. Calls method Component.link() to build variables and constraints related to the energy flows, and if demand is dispatchable, creates dispatching variables and equations variables and adds them to the linear problem as constraints.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**
Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**
pandas.DataFrame

**get_hub**(*hubs*, *environment=None*, *energy=None*, *log=False*)

This method gets the Hub(environment, energy) to be linked to the Component object, and creates it if it doesn't exist.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **environment** (*str*) – Environment of the Hub to be selected. Default is self.environment, the main environment attached to the Component object.
- **energy** (*str*) – Energy of the Hub to be selected. Default is self.energy, the main energy type attached to the Component object.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**

Hub object corresponding to (environment, energy).

**Return type**

*Hub*(*Component*)

**link**(*hubs*, *model*, *log=False*)

This methods creates flow_vars_in and flow_vars_out lists of flow variables that will be optimized, adds them to them to the Hub equation (energy conservation equation), and add the constraints factor*minimum_in[t] <= flow_vars_in[t] <= factor*maximum_in[t] and factor*minimum_out[t] <= flow_vars_out[t] <= factor*maximum_out[t] for every timestep t between 0 and nb_of_timesteps-1. Updates the model objective function with costs attributed to the flow variables: model.objective += sum_t flow_vars_in[t]*cost_in[t] + flow_vars_out[t]*cost_out[t].

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**

Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**

pandas.DataFrame

class components.**EnvironmentsConnection**(*environment1*, *environment2*, *descriptions*, *i*, *nb_of_timesteps*, *log=False*)

Bases: *Component*

Sub-class of Component class used to represent the interface between two different environments.

**Inherited Attributes:**

name (str): Name of the component. energy (str): Not used, forced to energy=''. environment (str): Main environment of the component. For an EnvironmentsConnection object, corresponds

to environment1.

description (dict): Portion of the configuration file describing the characteristics of the links nb_of_timesteps (int): Number of timesteps of the simulation. maximum (float | str): maximum value(s) of energy flow in and out of the component. Can be one value

(ex: maximum electricity exchanges through grid) or a sequence of the same duration of the simulation duration (ex: hourly PV production). In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_maximum". Value(s) must be given in the unit of the energy type attached to the object.

**maximum_in (float | str): maximum of the energy flow in the sense Hub -> Component. Same format as maximum,**
default value is maximum, crushes maximum if both maximum_in and maximum are given (only maximum_in in used in the problem constraints).

**maximum_out (float | str): maximum of the energy flow in the sense Component -> Hub. Same format as maximum,**
default value is maximum, crushes maximum if both maximum_out and maximum are given (only maximum_out in used in the problem constraints).

**minimum (float | str): minimum value(s) of energy flow in and out of the component. Can be one value or a**
sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_minimum". Value(s) must be given in the unit of the energy type attached to the object.

**minimum_in (float | str): minimum of the energy flow in the sense Hub -> Component. Same format as minimum,**
default value is minimum, crushes minimum if both minimum_in and minimum are given (only minimum_in in used in the problem constraints). If demand is not dispatchable, will be force to minimum_in=value.

**minimum_out (float | str): minimum of the energy flow in the sense Component -> Hub. Same format as minimum,**
default value is minimum, crushes minimum if both minimum_out and minimum are given (only minimum_out in used in the problem constraints). If demand is not dispatchable, will be force to minimum_in=value.

**cost (float | str): cost value(s) of energy flow in and out of the component per energy type unit. Can be one**
value or a sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_cost". Value(s) must be given in cost unit per unit of the energy type attached to the object. Default is 0. A positive value is a cost, a negative value is a gain.

**cost_in (float | str): minimum of the energy flow in the sense Hub -> Component. Same format as cost, default**
value is -cost, crushes cost if both cost_in and cost are given (only cost_in in used in the problem constraints).

**cost_out (float | str): minimum of the energy flow in the sense Component -> Hub. Same format as cost, default**
value is cost, crushes cost if both cost_out and cost are given (only cost_out in used in the problem constraints).

**factor (int | float | str): actual minimum and maximum flow values are multiplied by factor. Example of use: if the**
input timeseries given as maximum_out describes the production of a 1 kWc PV panel but we want to have 5 kWc of PV installed, set factor to 5 ; if the input timeseries describes a demand that we want to split as 50% dispatchable and 50% not dispatchable, create to Demand objects (one with dispatchable=True, the other with dispatchable=False) with this input timeseries and set factor=0.5 for both objects.

factor_low_bound (int | float): Used if factor is set to 'auto'. factor_low_bound <= factor. factor_up_bound (int | float): Used if factor is set to 'auto'. factor_up_bound => factor. factor_type (str): Used if factor is set to 'auto' and set the factor type. Can be 'Continuous', 'Integer' or 'Binary'.

Example of use for 'Integer' : how many PV panels of 200 Wc should be installed ? Example of use for 'Binary' : should a generator be installed ? (Yes : factor=1., No : factor=0.)

**installation_cost (float): Cost for one unit installed (factor=1.) Total installation cost will be**
installation_cost * factor.

**flow_vars_in (list[pulp.LpVariable]): List of length nb_of_timesteps containing the flow-in variables (from Hub to Component)**
of the Component at each timestep. Once the problem solved, flow_vars_in[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

**flow_vars_out (list[pulp.LpVariable]): List of length nb_of_timesteps containing the flow-out variables (from Hub to Component)**
of the Component at each timestep. Once the problem solved, flow_vars_out[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

**environment1**

Name of 1st environment to connect.

**Type**
str

**environment2**

Name of 2nd environment to connect.

**Type**
str

**vars_out (dict[str**

list[pulp.LpVariable]]): Dictionnary with energy types common to both environment1 and environment2 as keys, containing lists of length nb_of_timesteps containing the flow variables from Hub(environment1) to Hub(environment2) for each common energy type. Once the problem solved, vars_out[energy][t].value() returns the optimized value of these flow variables from for each timestep t between 0 and nb_of_timesteps-1.

**vars_in (dict[str**

list[pulp.LpVariable]]): Dictionnary with energy types common to both environment1 and environment2 as keys, containing lists of length nb_of_timesteps containing the flow variables from Hub(environment2) to Hub(environment1) for each common energy type. Once the problem solved, vars_in[energy][t].value() returns the optimized value of these flow variables from for each timestep t between 0 and nb_of_timesteps-1.

**Parameters**

- **environment1** (*str*) – Name of 1st environment to connect.
- **environment2** (*str*) – Name of 2nd environment to connect.
- **descriptions** (*dict*) – Sub-section of the configuration file describing the characteristics of the links.
- **i** (*int*) – Used to iterate within a list of specifications.
- **nb_of_timesteps** (*int*) – Number of timesteps of the simulation.

- **log** (*bool*) – If True, additional information will be printed throughout the initialization.

**connect_as_input**(*hubs*, *model*, *log=False*)

Creates vars_out and vars_in to store the flow variables between hubs of environment1 and hubs of environment2, add them to hubs of environment1 and hubs of environment2 equations, adds minimum and maximum constraints (describing the connection conditions between the two environments) to the linear problem, and updates the model objective function with costs attributed to the flow variables.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**

Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**

pandas.DataFrame

**get_hub**(*hubs*, *environment=None*, *energy=None*, *log=False*)

This method gets the Hub(environment, energy) to be linked to the Component object, and creates it if it doesn't exist.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **environment** (*str*) – Environment of the Hub to be selected. Default is self.environment, the main environment attached to the Component object.
- **energy** (*str*) – Energy of the Hub to be selected. Default is self.energy, the main energy type attached to the Component object.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**

Hub object corresponding to (environment, energy).

**Return type**

*Hub*(*Component*)

**link**(*hubs*, *model*, *log=False*)

This methods creates flow_vars_in and flow_vars_out lists of flow variables that will be optimized, adds them to them to the Hub equation (energy conservation equation), and add the constraints factor*minimum_in[t] <= flow_vars_in[t] <= factor*maximum_in[t] and factor*minimum_out[t] <= flow_vars_out[t] <= factor*maximum_out[t] for every timestep t between 0 and nb_of_timesteps-1. Updates the model objective function with costs attributed to the flow variables: model.objective += sum_t flow_vars_in[t]*cost_in[t] + flow_vars_out[t]*cost_out[t].

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**
Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**
pandas.DataFrame

**class** `components.Hub`(*environment*, *energy*, *nb_of_timesteps*, *log=False*)

Bases: *Component*

Sub-class of Component class representing the link between other Components belonging to the same environment with the same energy type. Examples: Hub(house, electricity), Hub(house, heat) or Hub(car, electricity). Is used to implement the energy conservation equation within each environment and for each energy type.

**Inherited attributes:**
name (str): Name of the hub. environment (str): Main environment of the hub. energy (str): Main energy type of the hub. nb_of_timesteps (int): Number of timesteps of the simulation.

`unused_energy`

List of variables representing the energy losses of the hub at each timestep.

**Type**
list[pulp.LpVariable]

`equation`

List of equations representing the conservation of energy (everything entering the hub = everything going out of the hub) at each timestep.

**Type**
list[pulpLpConstraint]

`component_names`

List of names of the components linked to the hub.

**Type**
list[str]

**Parameters**

- **environment** (*str*) – Main environment of the hub.
- **energy** (*str*) – Main energy type of the hub.
- **nb_of_timesteps** (*int*) – Number of timesteps of the simulation.
- **log** (*bool*) – If True, additional information will be printed throughout the initialization.

`add_link`(*variables*, *component_name*, *sign='+'*, *log=False*)

Link a new component to the hub by adding its flow variables to the hub equation.

**Parameters**

- **variables** (*list[pulp.LpVariable]*) – Liste of length nb_of_timesteps with flow variables to be added to the hub equation.
- **component_name** (*str*) – Name of the component to be linked.
- **sign** (*str*) – Can be '+' or '-'. Describes whether it is flow_in or flow_out variables. If sign='+' then +variables[t] is added to the left hand side of the hub equation, else if sign='-' then -variables[t] is added to the left hand side of the hub equation.

- **log** (*bool*) – If True, additional information will be printed throughout the initialization.

**get_hub**(*hubs*, *environment=None*, *energy=None*, *log=False*)

This method gets the Hub(environment, energy) to be linked to the Component object, and creates it if it doesn't exist.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **environment** (*str*) – Environment of the Hub to be selected. Default is self.environment, the main environment attached to the Component object.
- **energy** (*str*) – Energy of the Hub to be selected. Default is self.energy, the main energy type attached to the Component object.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**

Hub object corresponding to (environment, energy).

**Return type**

*Hub*(*Component*)

**link**(*hubs*, *model*, *log=False*)

This methods creates flow_vars_in and flow_vars_out lists of flow variables that will be optimized, adds them to them to the Hub equation (energy conservation equation), and add the constraints factor*minimum_in[t] <= flow_vars_in[t] <= factor*maximum_in[t] and factor*minimum_out[t] <= flow_vars_out[t] <= factor*maximum_out[t] for every timestep t between 0 and nb_of_timesteps-1. Updates the model objective function with costs attributed to the flow variables: model.objective += sum_t flow_vars_in[t]*cost_in[t] + flow_vars_out[t]*cost_out[t].

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**

Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**

pandas.DataFrame

**class** components.**Source**(*name*, *description*, *nb_of_timesteps*, *log=False*)

Bases: `Component`

Sub-class of Component class representing an energy source. Note that if specified,it is possible to send energy toward the source (reinjecting electricity to the grid for instance).

**Inherited Attributes:**

**name (str)**

Name of the component. energy (str): Main energy type of the component. environment (str): Main environment of the component.

**description (dict)**

Portion of the configuration file describing the characteristics of the links

**nb_of_timesteps (int)**
Number of timesteps of the simulation.

**maximum (float | str)**
Maximum value(s) of energy flow in and out of the component. Can be one value (ex: maximum electricity exchanges through grid) or a sequence of the same duration of the simulation duration (ex: hourly PV production). In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_maximum". Value(s) must be given in the unit of the energy type attached to the object.

**maximum_in (float | str)**
Maximum of the energy flow in the sense Hub -> Component. Same format as maximum, default value is maximum, crushes maximum if both maximum_in and maximum are given (only maximum_in in used in the problem constraints).

**maximum_out (float | str)**
Maximum of the energy flow in the sense Component -> Hub. Same format as maximum, default value is maximum, crushes maximum if both maximum_out and maximum are given (only maximum_out in used in the problem constraints).

**minimum (float | str)**
Minimum value(s) of energy flow in and out of the component. Can be one value or a sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_minimum". Value(s) must be given in the unit of the energy type attached to the object.

**minimum_in (float | str): minimum of the energy flow in the sense Hub -> Component. Same format as minimum,**
default value is minimum, crushes minimum if both minimum_in and minimum are given (only minimum_in in used in the problem constraints). If demand is not dispatchable, will be force to minimum_in=value.

**minimum_out (float | str): minimum of the energy flow in the sense Component -> Hub. Same format as minimum,**
default value is minimum, crushes minimum if both minimum_out and minimum are given (only minimum_out in used in the problem constraints). If demand is not dispatchable, will be force to minimum_in=value.

**cost (float | str): cost value(s) of energy flow in and out of the component per energy type unit. Can be one**
value or a sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_cost". Value(s) must be given in cost unit per unit of the energy type attached to the object. Default is 0. A positive value is a cost, a negative value is a gain.

**cost_in (float | str): minimum of the energy flow in the sense Hub -> Component. Same format as cost, default**
value is -cost, crushes cost if both cost_in and cost are given (only cost_in in used in the problem constraints).

**cost_out (float | str): minimum of the energy flow in the sense Component -> Hub. Same format as cost, default**
value is cost, crushes cost if both cost_out and cost are given (only cost_out in used in the problem constraints).

**factor (int | float | str): actual minimum and maximum flow values are multiplied by factor. Example of use: if the**
input timeseries given as maximum_out describes the production of a 1 kWc PV panel but

we want to have 5 kWc of PV installed, set factor to 5 ; if the input timeseries describes a demand that we want to split as 50% dispatchable and 50% not dispatchable, create to Demand objects (one with dispatchable=True, the other with dispatchable=False) with this input timeseries and set factor=0.5 for both objects.

factor_low_bound (int | float): Used if factor is set to 'auto'. factor_low_bound <= factor. factor_up_bound (int | float): Used if factor is set to 'auto'. factor_up_bound => factor. factor_type (str): Used if factor is set to 'auto' and set the factor type. Can be 'Continuous', 'Integer' or 'Binary'.

Example of use for 'Integer' : how many PV panels of 200 Wc should be installed ? Example of use for 'Binary' : should a generator be installed ? (Yes : factor=1., No : factor=0.)

**installation_cost (float): Cost for one unit installed (factor=1.) Total installation cost will be**
installation_cost * factor.

**flow_vars_in (list[pulp.LpVariable]): List of length nb_of_timesteps containing the flow-in variables (from Hub to Component)**
of the Component at each timestep. Once the problem solved, flow_vars_in[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

**flow_vars_out (list[pulp.LpVariable]): List of length nb_of_timesteps containing the flow-out variables (from Hub to Component)**
of the Component at each timestep. Once the problem solved, flow_vars_out[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

**Parameters**

- **name** (*str*) – Name of the component.
- **description** (*dict*) – Portion of the configuration file describing the characteristics of the component links.
- **nb_of_timesteps** (*int*) – Number of timesteps of the simulation.
- **log** (*bool*) – If True, additional information will be printed throughout the initialization.

**build_equations**(*hubs*, *model*, *log=False*)

Calls method Component.link() to build variables and constraints related to the energy flows.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**
Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**
pandas.DataFrame

**get_hub**(*hubs*, *environment=None*, *energy=None*, *log=False*)

This method gets the Hub(environment, energy) to be linked to the Component object, and creates it if it doesn't exist.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **environment** (*str*) – Environment of the Hub to be selected. Default is self.environment, the main environment attached to the Component object.
- **energy** (*str*) – Energy of the Hub to be selected. Default is self.energy, the main energy type attached to the Component object.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**
Hub object corresponding to (environment, energy).

**Return type**
*Hub*(*Component*)

**link**(*hubs*, *model*, *log=False*)

This methods creates flow_vars_in and flow_vars_out lists of flow variables that will be optimized, adds them to them to the Hub equation (energy conservation equation), and add the constraints factor*minimum_in[t] <= flow_vars_in[t] <= factor*maximum_in[t] and factor*minimum_out[t] <= flow_vars_out[t] <= factor*maximum_out[t] for every timestep t between 0 and nb_of_timesteps-1. Updates the model objective function with costs attributed to the flow variables: model.objective += sum_t flow_vars_in[t]*cost_in[t] + flow_vars_out[t]*cost_out[t].

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**
Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**
pandas.DataFrame

**class** components.**Storage**(*name*, *description*, *nb_of_timesteps*, *log=False*)

Bases: `Component`

Sub-class of Component class representing a storage device.

**Inherited Attributes:**

name (str): Name of the component. energy (str): Main energy type of the component. environment (str): Main environment of the component. description (dict): Portion of the configuration file describing the characteristics of the links nb_of_timesteps (int): Number of timesteps of the simulation. maximum (float | str): maximum value(s) of energy flow in and out of the component. Can be one value

> (ex: maximum electricity exchanges through grid) or a sequence of the same duration of the simulation duration (ex: hourly PV production). In this case, a string describing where the data is stored must be given under this form : “path/to/data.csv//column_name_of_maximum”. Value(s) must be given in the unit of the energy type attached to the object.

**maximum_in (float | str): maximum of the energy flow in the sense Hub -> Component. Same format as maximum,**
default value is maximum, crushes maximum if both maximum_in and maximum are given (only maximum_in in used in the problem constraints).

**maximum_out (float | str): maximum of the energy flow in the sense Component -> Hub. Same format as maximum,**
default value is maximum, crushes maximum if both maximum_out and maximum are given (only maximum_out in used in the problem constraints).

**minimum (float | str): minimum value(s) of energy flow in and out of the component. Can be one value or a**
sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_minimum". Value(s) must be given in the unit of the energy type attached to the object. Default is 0. ; it highly recommanded to NOT PUT NEGATIVE VALUES.

**minimum_in (float | str): minimum of the energy flow in the sense Hub -> Component. Same format as minimum,**
default value is minimum, crushes minimum if both minimum_in and minimum are given (only minimum_in in used in the problem constraints).

**minimum_out (float | str): minimum of the energy flow in the sense Component -> Hub. Same format as minimum,**
default value is minimum, crushes minimum if both minimum_out and minimum are given (only minimum_out in used in the problem constraints).

**cost (float | str): cost value(s) of energy flow in and out of the component per energy type unit. Can be one**
value or a sequence of the same duration of the simulation duration. In this case, a string describing where the data is stored must be given under this form : "path/to/data.csv//column_name_of_cost". Value(s) must be given in cost unit per unit of the energy type attached to the object. Default is 0. A positive value is a cost, a negative value is a gain.

**cost_in (float | str): minimum of the energy flow in the sense Hub -> Component. Same format as cost, default**
value is -cost, crushes cost if both cost_in and cost are given (only cost_in in used in the problem constraints).

**cost_out (float | str): minimum of the energy flow in the sense Component -> Hub. Same format as cost, default**
value is cost, crushes cost if both cost_out and cost are given (only cost_out in used in the problem constraints).

**factor (int | float | str): actual minimum and maximum flow values are multiplied by factor. Example of use: if the**
input timeseries given as maximum_out describes the production of a 1 kWc PV panel but we want to have 5 kWc of PV installed, set factor to 5 ; if the input timeseries describes a demand that we want to split as 50% dispatchable and 50% not dispatchable, create to Demand objects (one with dispatchable=True, the other with dispatchable=False) with this input timeseries and set factor=0.5 for both objects ; if the best size of component has to be automatically optimized, set factor='auto'. WARNING: 'auto' option is not compatible with dispatchable Demand objects.

factor_low_bound (int | float): Used if factor is set to 'auto'. factor_low_bound <= factor. factor_up_bound (int | float): Used if factor is set to 'auto'. factor_up_bound => factor. factor_type (str): Used if factor is set to 'auto' and set the factor type. Can be 'Continuous', 'Integer' or 'Binary'.

Example of use for 'Integer' : how many PV panels of 200 Wc should be installed ? Example of use for 'Binary' : should a generator be installed ? (Yes : factor=1., No : factor=0.)

**installation_cost (float): Cost for one unit installed (factor=1.) Total installation cost will be**
installation_cost * factor.

**flow_vars_in (list[pulp.LpVariable]): List of length nb_of_timesteps containing the flow-in variables (from Hub to Component)**
of the Component at each timestep. Once the problem solved, flow_vars_in[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

**flow_vars_out (list[pulp.LpVariable]): List of length nb_of_timesteps containing the flow-out variables (from Hub to Component)**
of the Component at each timestep. Once the problem solved, flow_vars_out[t].value() returns the optimized value of the variable for each timestep t between 0 and nb_of_timesteps-1.

`SOC`

List of length nb_of_timesteps of variables describing the energy contained in the storage at every timestep.

**Type**
list[pulp.LpVariable]

`capacity`

Storage capacity, in the same unit as its energy type.

**Type**
float

`initial_SOC`

Initial storage state-of-charge rate (at t=0), must be comprised between 0. and 1.

**Type**
float

`final_SOC`

Final storage state-of-charge rate (at t=nb_of_timesteps - 1), must be comprised between 0. and 1.

**Type**
float

`efficiency`

Storage efficiency = energy out / energy in. Must be comprised between 0. and 1.

**Type**
float

`calendar_loss`

Loss of energy stored at each timestep = energy stored at t / energy stored at t-1 without external flows. Must be comprised between 0. and 1.

**Type**
float

`volume_factor`

Actual storage capacity is multiplied by volume_factor: 0 <= SOC[t] <= volume_factor * capacity for each timestep t. If volume_factor is set to 'auto', best volume_factor will be automatically determined.

**Type**
float | int | str

**volume_factor_type**

Used if volume_factor is set to 'auto' and set the factor type. Can be 'Continuous', 'Integer' or 'Binary'.

**Type**
str

**volume_factor_low_bound**

Used if volume_factor is set to 'auto' and set the factor type. volume_factor_low_bound <= volume_factor.

**Type**
float

**volume_factor_up_bound**

Used if volume_factor is set to 'auto' and set the factor type. volume_factor_up_bound => volume_factor.

**Type**
float

**volume_installation_cost**

Cost for one unit installed (volume_factor=1.) Total installation due to storage volume will be volume_installation_cost * volume_factor.

**Type**
float

**equation**

List of length nb_of_timesteps containing the storage energy conservation equations for every timestep t>0 : SOC[t] = calendar_loss*SOC[t-1] + (efficiency)^(1/2)*flow_vars_in[t-1] - efficiency^(-1/2)*flow_vars_out[t-1].

**Type**
list[pulp.LpConstraints]

**Parameters**

- **name** (*str*) – Name of the component.
- **description** (*dict*) – Portion of the configuration file describing the characteristics of the component links.
- **nb_of_timesteps** (*int*) – Number of timesteps of the simulation.
- **log** (*bool*) – If True, additional information will be printed throughout the initialization.

**build_equations**(*hubs*, *model*, *log=False*)

Method to build storage-specific variables and constraints. Calls method Component.link() to build variables and constraints related to the energy flows, creates SOC variables for the energy stored at each timestep, builds equations to modelize the storage capacity and energy flows and adds them to the linear problem as constraints.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**
Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**
pandas.DataFrame

**get_hub**(*hubs*, *environment=None*, *energy=None*, *log=False*)

This method gets the Hub(environment, energy) to be linked to the Component object, and creates it if it doesn't exist.

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **environment** (*str*) – Environment of the Hub to be selected. Default is self.environment, the main environment attached to the Component object.
- **energy** (*str*) – Energy of the Hub to be selected. Default is self.energy, the main energy type attached to the Component object.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**
Hub object corresponding to (environment, energy).

**Return type**
*Hub*(*Component*)

**link**(*hubs*, *model*, *log=False*)

This methods creates flow_vars_in and flow_vars_out lists of flow variables that will be optimized, adds them to them to the Hub equation (energy conservation equation), and add the constraints factor*minimum_in[t] <= flow_vars_in[t] <= factor*maximum_in[t] and factor*minimum_out[t] <= flow_vars_out[t] <= factor*maximum_out[t] for every timestep t between 0 and nb_of_timesteps-1. Updates the model objective function with costs attributed to the flow variables: model.objective += sum_t flow_vars_in[t]*cost_in[t] + flow_vars_out[t]*cost_out[t].

**Parameters**

- **hubs** (*pandas.DataFrame*) – Table of all hubs.
- **model** (*pulp.LpProblem*) – Pulp linear problem to be optimized.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**Returns**
Updated table of all hubs. pulp.LpProblem: Updated pulp linear problem to be optimized.

**Return type**
pandas.DataFrame

components.**get_default_values_Component_phase2**(*o*)

Gets default values of the attributes of a Component that depends on other attributes values.

**Parameters**
**o** (Component) – Component object to read the already created attributes from.

**Returns**
Dictionnary with new attributes names as keys containing their default values.

**Return type**
dict()

CHAPTER

# TWO

# MODEL MODULE

**class** model.**Model**(*config_file='general_config_file.yaml'*, *elements_list='elements_list.yaml'*, *data=''*)

Bases: object

Class for the modelization of an energy system from configuration files A Model objects allows to build linear variables and equations, solve them as a linear problem, access all information relative to the simulation, and display the results once solved (optimized values of all variables, value of the objective function).

**config_file**

General configuration file.

**Type**
dict

**elements_list**

Description of all base components of the system.

**Type**
dict

**run_num**

Number of the run, for identification and saving purpose.

**Type**
int

**run_name**

Name of the run, for identification and saving purpose.

**Type**
str

**energies**

List of all possible energy types.

**Type**
list[str]

**environments**

List of all possible environments.

**Type**
list[str]

`data`
: [Optional] Dataframe with input data.

**Type**
pandas.Dataframe | NoneType

`nb_of_timesteps`
: Number of timesteps of the simulation.

**Type**
int

`time`
: List of length nb_of_timesteps with timestamps.

**Type**
List[str]

`model`
: Pulp linear formulation of the optimization problem.

**Type**
pulp.LpProblem

`components (dict[str`
: dict[str: Components]]): Dictionnary of all base components added to the system, with base components type (Source, Demand, Storage, Converter) and components names as keys.

`directory`
: Directory path to save configuration files, results and plots.

**Type**
str

`hubs`
: Data table with index=environments and columns=energies representing of all possible hubs, filled with Hub(environment, energy) objects, and None when it doesn't exist.

**Type**
pandas.Dataframe

`environmentsConnections (dict[str`
: EnvironmentsConnection]) : Dictionnary of all environments connections, represented by EnvironmentsConnection objects.

`simu_time`
: Timestamp of the instant where the optimization is completed.

**Type**
datetime

`objective_value`
: Value of the objective function once the optimization is completed.

**Type**
float

`dispatch`
: Dataframe of length nb_of_timesteps with flow variables and SOC values (optimal solution of the problem).

**Type**
pandas.Dataframe

**Parameters**

- **config_file** (*str*) – Path to general yaml configuration file.
- **elements_list** (*str*) – Path to elements list yaml configuration file.
- **data** (*str*) – [Optional] Path to input data csv file. Could be usefull to display information relative to input data in results plot.

**add_hubs_equations_to_model**(*log=False*)

Adds hubs equations to the model, once all base components are added to the hubs equations. Updates model.

TO BE USED AFTER build_environment_level_variables_and_constraints() AND connect_environments().

**Parameters**
**log** (*bool*) – If True, additional information will be printed throughout the process.

**build_environment_level_variables_and_constraints**(*log=False*)

Builds all base components (Source, Demand, Storage, Converter), including their variables and internal constraints (example: dispatching equations for dispatchable Demand objects, storage equations for Storage objects, conversion equations for Conversion objects). Updates hubs table, components dictionnary, and model. TO BE USED AFTER initialize_hubs().

**Parameters**
**log** (*bool*) – If True, additional information will be printed throughout the process.

**connect_environments**(*log=False*)

Builds flow variables between different environments according to the connections descriptions given in the general configuration file. Store EnvironmentsConnection object in environmentsConnections dictionnary. Updates hubs table and model.

**Parameters**
**log** (*bool*) – If True, additional information will be printed throughout the process.

**get_design**(*components*, *factors*, *units=None*)

Shortcut to get the values of components attributes.

**Parameters**

- **components** (*list[str]*) – List of components names.
- **factors** (*list[str]*) – List of associated factors to be displayed, must be the same length as components (with typically "factor" or "volume_factor" as values).
- **units** (*list[str] | None*) – List of associated units to be displayed, must be the same length as components if not None.

**Returns**
Table filled with factors values

**Return type**
pandas.Dataframe

**hub_vars**(*environment*, *energy*)

Gets all flow variable names linked to the hub(environment, energy.)

**Parameters**

- **environment** (*str*) – Environment name.
- **energy** (*str*) – Energy type name.

**Returns**
list[str]

**initialize_hubs**()

Creates hubs attribute.

**plot_SOC**(*variables='all'*, *unit=''*, *save=False*, *log=False*)

Plots SOC values depending on the time.

**Parameters**

- **variables** (*str | list[str]*) – If 'all', all SOC variables will be displayed. Else, list of names of variables to be displayed.
- **unit** (*str*) – Unit of variables, for legend.
- **save** (*bool*) – If True, plot will be saved as a png file in the run directory.
- **log** (*bool*) – If True, additional information will be printed throughout the process.

**plot_hubs**(*save=False*, *log=False*, *co=False*, *env1=''*, *env2=''*, *price=None*, *unit=''*, *start=0*, *end=0*)

Plots energy flows of all hubs that exist, depending on the time.

**Parameters**

- **save** (*bool*) – If True, plot will be saved as a png file in the run directory.
- **log** (*bool*) – If True, additional information will be printed throughout the process.
- **co** (*bool*) – If True, periods of connection between two environments will be shown on all subplots.
- **env1** (*str*) – If co==True, name of the 1st environment of the connection to be displayed.
- **env2** (*str*) – If co==True, name of the 2nd environment of the connection to be displayed.
- **price** (*tuple(str | float, str)| None*) – If not None, additional line to be plotted on a second axis on all subplots. Must be under the form (path_to_column | value). Ex: ('data_sample.csv//Electricity_price (euros/MWh)', 'Electricity price (€/MWh)').
- **unit** (*str*) – Unit of energy flows, for legend.
- **start** (*int*) – Index of the data where to start to plot. Ex: start=2 -> start at the second value.
- **end** (*int*) – len(dispatch)-end is the index of the data where to stop to plot Ex: stop=1 -> stop at the penultimate value.

**solve**(*log=False*)

Solves the pulp problem, and save flow variables and SOC values in dispatch.

**Parameters**
**log** (*bool*) – If True, additional information will be printed throughout the process.

CHAPTER

THREE

# UTILS MODULE

utils.**bound**(*value*, *t*)

If value described the value of a function f depending on time t, returns f(t). If value is a constant, returns value. If value is a list, returns element number t of the list.

**Parameters**

- **value** (*any, list[any]*) – Value or list of values.
- **t** (*int*) – Timestep or index.

**Returns**
f(t)

**Return type**
any

utils.**capitalise**(*string*)

Capitalises only the first letter of a string, leaving the other characters unchanged.

**Parameters**
**string** (*str*) – A string to be transformed.

**Returns**
String transformed.

**Return type**
str

utils.**get_chronicle**(*value*, *log=False*, *i=None*)

From a value in a configuration file, returns the value if it's raw data, and gets raw data indicated by the value if value is a path to a column of a csv file.

**Parameters**

- **value** (*str | int | float*) – Data or path to location to get data from.
- **log** (*bool*) – If True, additionnal information will be printed throughout the process.
- **i** (*int*) – Index used to iterate within a list of specifications.

**Returns**
Value(s).

**Return type**
list[float] | str | int | float

`utils.`**`get_components`**(*file_of_components_list*, *components_class*)

Gets information regarding components belonging to a type of base components.

**Parameters**

- **file_of_components_list** (*str*) – Path to a yaml components list configuration file.
- **components_class** (*str*) – Name of the class of base components to be found (Source, Storage, Demand, Converter).

**Returns**

**any]: Dictionnary with component names related to type components_class as keys, with their configuration**
information.

**Return type**
dict[str

`utils.`**`get_from_elements_list`**(*keyword*, *elements_list*)

Looks for lines in a file that contains a keyword, a returns the value of the elements describing it. Ex: keyword = 'energy' ; line = 'energy: electricity' ; then this function returns ['electricity'].

**Parameters**

- **keyword** (*str*) – The keyword to look for.
- **elements_list** (*str*) – Path to the file.

**Returns**
Lis of elements found associated to the keyword.

**Return type**
list[str]

`utils.`**`get_label`**(*string*)

Transforms a variable name into a label to be displayed by replacing '_' character by spaces.

**Parameters**
**string** (*str*) – A string to be transformed.

**Returns**
String transformed.

**Return type**
str

`utils.`**`get_timeline`**(*time*)

Builds a timeline in the list of datetime format from a list of strings.

**Parameters**
**time** (*list[str]*) – List of strings of the form '%Y%m%d:%H' (ex: 20252909:12 = September 29th, 2025 12h).

**Returns**
List of datetime objects.

**Return type**
list[datetime]

`utils.`**`plot_one_hub`**(*dispatch*, *top_var*, *variables*, *ax*, *title*, *price=None*, *co=False*, *env1_env2=None*, *env1=''*, *env2=''*, *unit=None*, *start=0*, *end=0*)

Plot the flow variables linked to a hub depending on the time as a cumulated step plot.

**Parameters**

- **dispatch** (*pandas.Dataframe*) – Dataframe containing data to be displayed.
- **(str** (*top_var*) – None): Name of a variable stored in dispatch to be displayed as a line plot with the same unit as the main variables. Must match a column name of dispatch.
- **variables** (*list[str]*) – List of names of variables to be displayed, must match some column names of dispatch.
- **ax** (*matplotlib.axes._axes.Axes*) – ax object to add the plots to.
- **title** (*str*) – Title of the subplot.
- **price** (*tuple(str | float, str)| None*) – If not None, additional line to be plotted on a second axis on all subplots. Must be under the form (path_to_column | value). Ex: ('data_sample.csv//Electricity_price (euros/MWh)', 'Electricity price (€/MWh)').
- **co** (*bool*) – If True, periods of connection between two environments will be shown on all subplots.
- **env1_env2** (*list[float]*) – Timeseries of the periods of connection between environment1 and environment2.
- **env1** (*str*) – If co==True, name of the 1st environment of the connection to be displayed.
- **env2** (*str*) – If co==True, name of the 2nd environment of the connection to be displayed.
- **unit** (*str*) – Unit of the data.
- **start** (*int*) – Index of the data where to start to plot. Ex: start=2 -> start at the second value.
- **end** (*int*) – len(dispatch)-end is the index of the data where to stop to plot Ex: stop=1 -> stop at the penultimate value.

**Returns**

ax object with the plots added.

**Return type**

ax (matplotlib.axes._axes.Axes)

utils.**value**(*flow_vars_t*)

Gets the numerical value of the element.

**Parameters**

**flow_vars_t** (*float | int | pulp.LpVariable*) – Element to get the value from.

**Returns**

Value of the element.

**Return type**

float | int

# PYTHON MODULE INDEX

# INDEX