Energy systems modelization

EnerPyFlow practical guide

Working document - 02/10/2025

Hortense Ronzani

Summary

I. Model presentation

1. General description

EnerPyFlow is a tool to be used to modelize energy systems as linear problems and solve it. The solution consists in the optimal dispatching of energy flows between the system components.

Several energy types can be represented, such as heat and electricity, and several environments can be described. For instance, a house and a car, with connection and disconnection time slots between the two.

Consumptions can also be optimally dispatched.

It is also possible to automatically optimally design some base components characteristics, such as number of PV panels or size of a battery storage.

This tool is meant to be used to build different scenarios by combining different base components, to then compare them.

EnerPyFlow is written in Python, and the resolution is based on the linear solver library PuLP. It is fully configurable from YAML files (text files) and CSV files for input data, and an easy way to use it is proposed in a demo Jupyter Notebook.

Model inputs:

- Demand time series (e.g., electricity, heating, distances to be travelled)
- Means of production and associated potential volumes (e.g., solar production per installed kWc, connection to the grid, gas station)
- Potential marginal costs (monetary or otherwise) associated with energy flows (e.g., price or carbon intensity of grid electricity)
- Connection time series between different environments (e.g., time periods when the car is connected to the house)
- Storage & conversion available equipments and characteristics
- Optimization sense (minimize or maximize)

→ cost will be optimized

A more precise description of the input data is provided in section *3. How to edit configuration files*, and full documentation can be found in document *EnerPyFlow_docs.pdf*.

Model outputs:

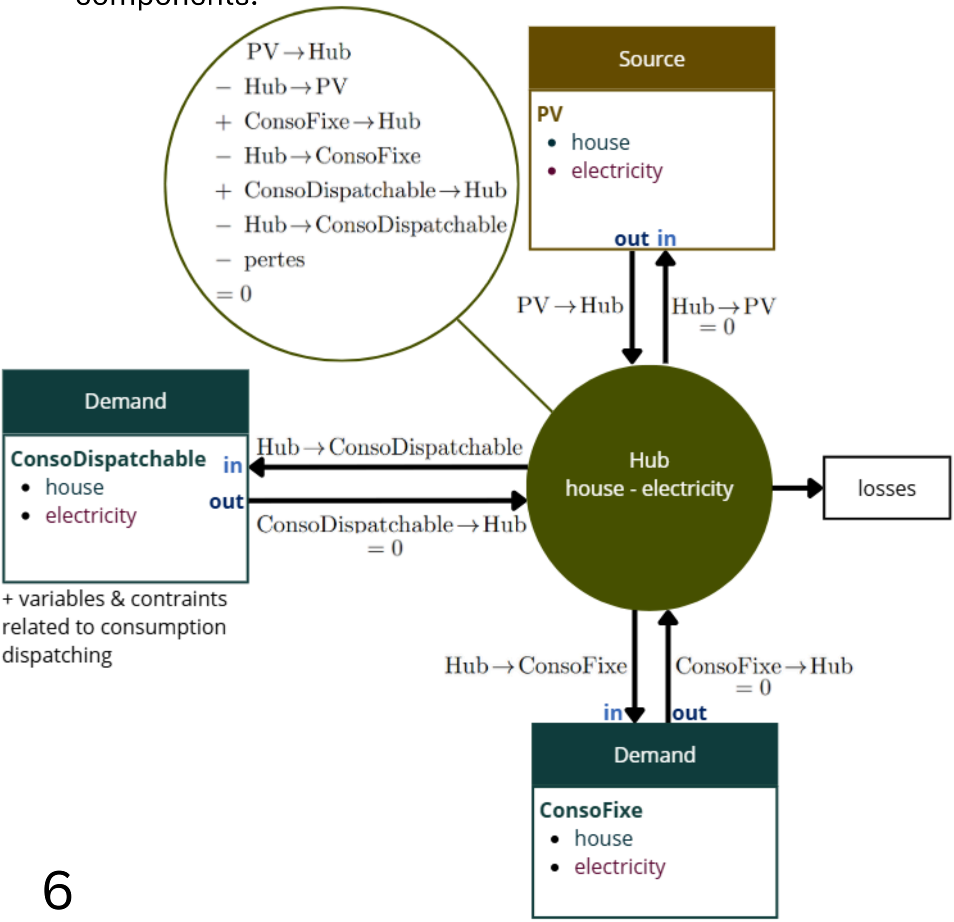
- Optimal solution for dispatching energy flows between the different base components as a .csv file and as .png plots
- Dispatching solution for consumption when it is controllable
- Optimal sizing of equipment when declared as an optimization variable (e.g., battery capacity, installed PV power)

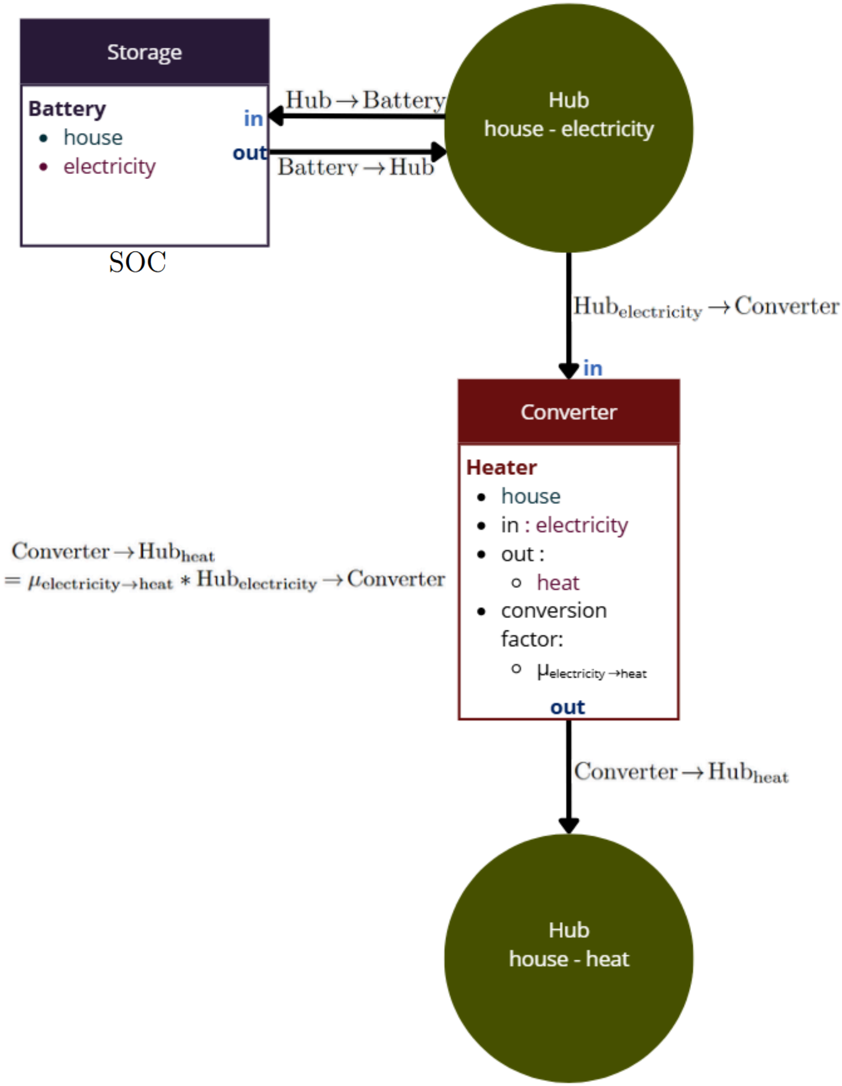
2. Architecture: base components

Base components can be of 4 types: **Source**, **Demand**, **Storage** and **Converter**.

Each base component has to be linked to one energy type (input energy type for Converter objects) and one environment.

Hubs objects will automatically be created for each (energy type, environment) couple in order to inter-connect base components.

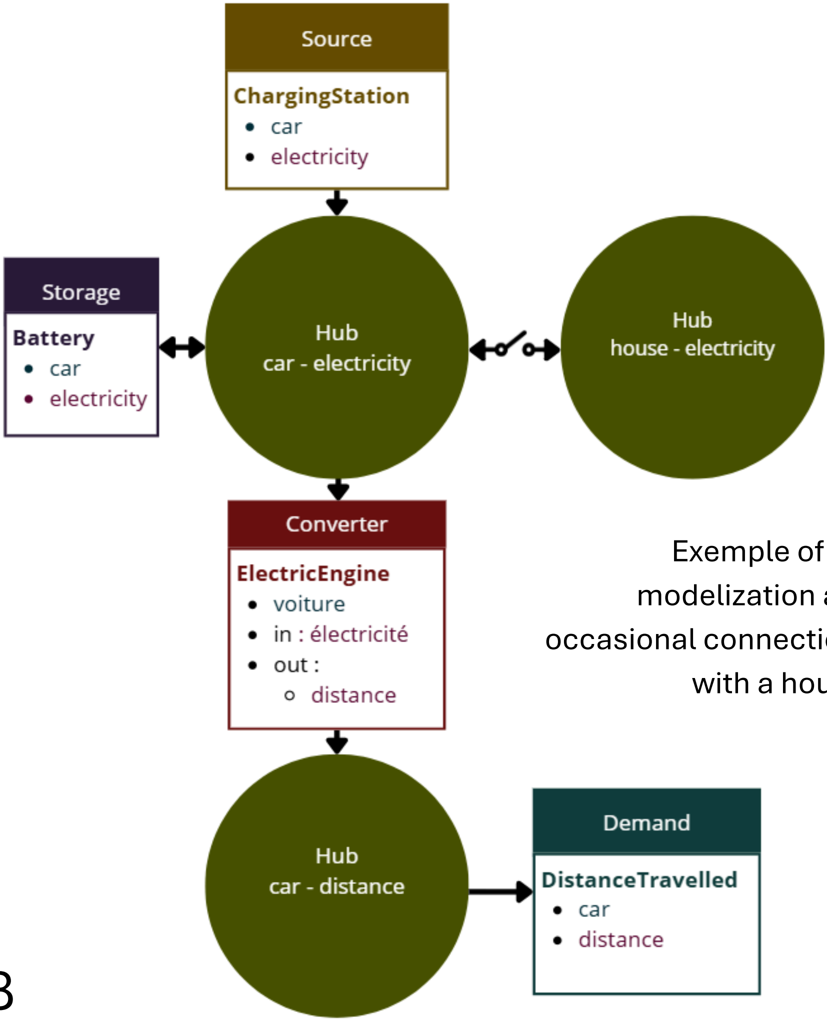




Two hubs of different energy types can only be linked through a Converter object (e.g., electric or thermal engine, electric heater).

3. Architecture: environments connections

In one environment, objects of a same energy type are always linked. Integration of an electric car in an energy system doesn't match this situation: it is here considered as a distinct environment, with its own Hubs, that can be linked or not to the house depending on whether the car is home or not.



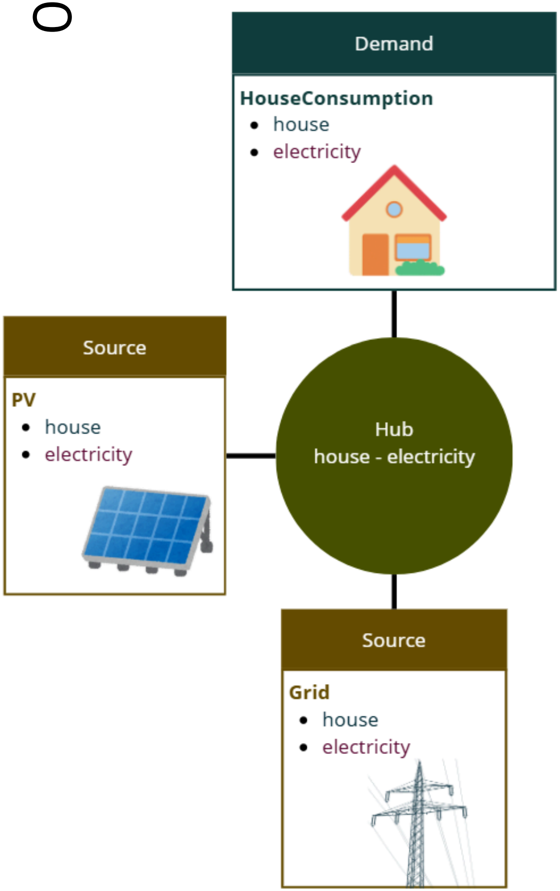
Exemple of car modelization and occasional connections with a house.

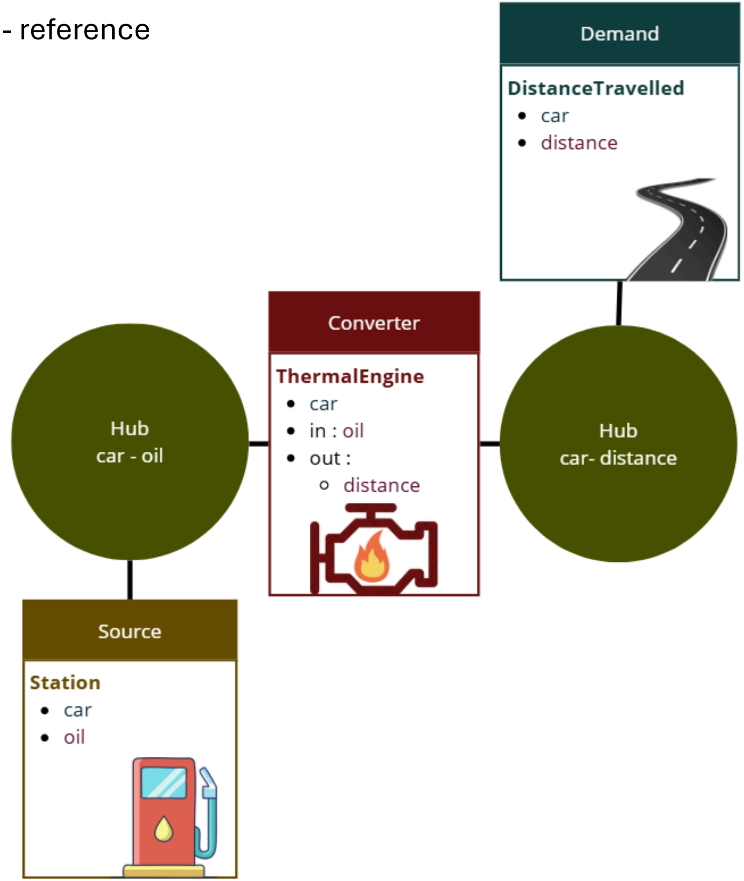
4. Demo

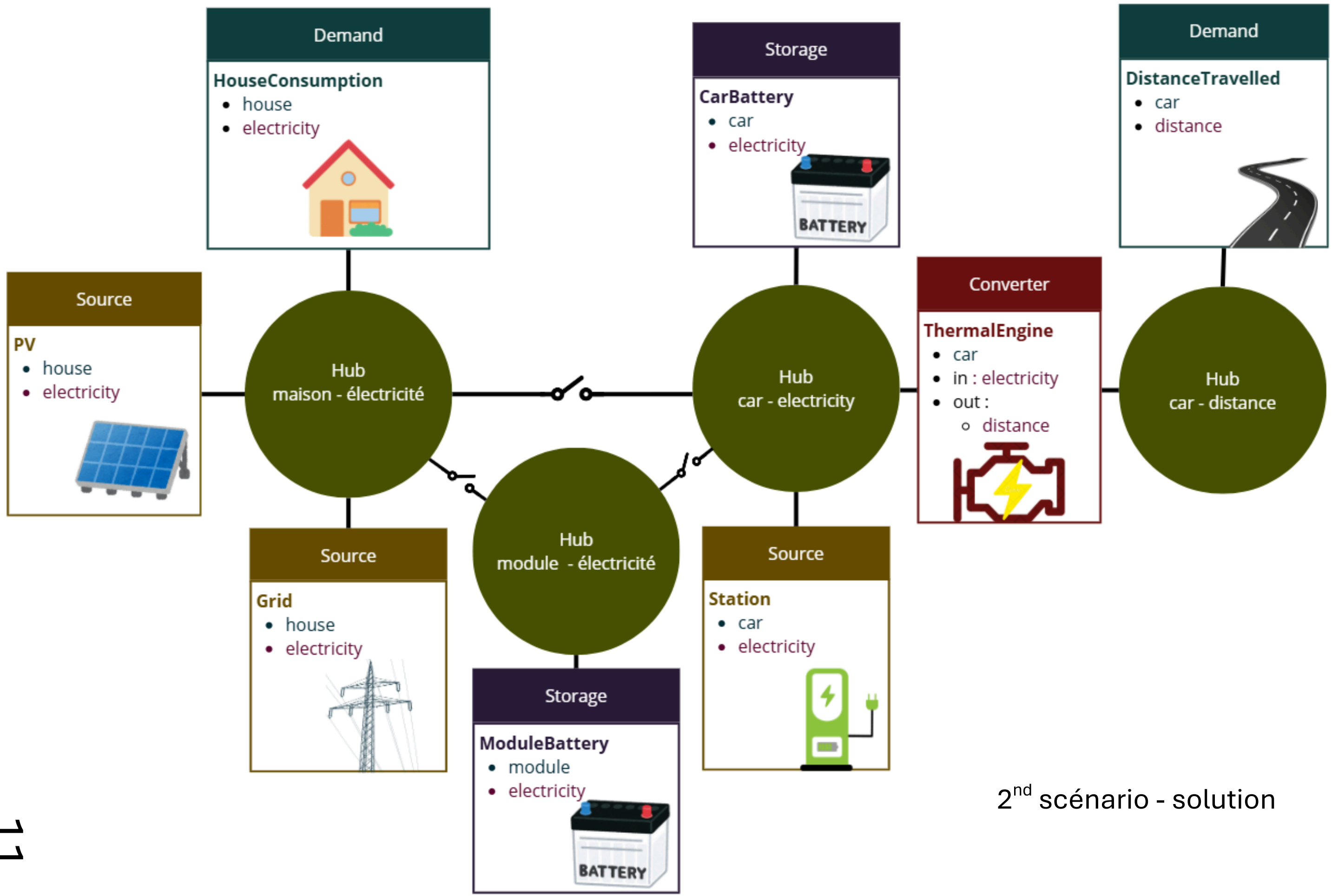
The following demo case is composed of a house, with PV on the roof and a grid connection and a car. In a 1st reference scenario, car is thermal and in a 2nd scenario, it is electric and there is an extra battery module. This module is embarked in the car for long travels and is left plugged to the house as a stationnary storage rest of the time.

Both diagrams are represented on the following pages.

1st scénario - reference







2nd scénario - solution

II. Starting guide

1. Getting started with Python Jupyter Notebooks

1.Download & install Python at
https://www.python.org/downloads/ (most recent version)
Click ‘add path to global environment’ for the installation.

2. Open the terminal (search ‘cmd’ in Windows search bar).

3. Run the command line

    pip install jupyter-notebook

4. Downloads files from git hub repository:
https://github.com/hortense-ronzani/EnerPyFlow/tree/main.
Unzip them and save them in a repository MyRepository.

5. Go to MyRepository through the cmd by navigating with command

    ls path/to/MyRepository

6. Run the command:

    jupyter-notebook workflow_light.ipynb

to open the demo file workflow.ipynb with Jupyter.

If you’ve never done anything like this before or if you’re in trouble, call me or call someone who’s used to navigate through files via cmd. It can be tricky on Stellantis’ computer (hint: user ≠ users).

2. Files description

Model repository is available at: https://github.com/hortense-ronzani/home-energy-system

It contains the following file:

- **Energy Systems Modelization - EnerPyFlow practical guide.pdf** → quick presentation and starting guide i.e. present document.
- **EnerPyFlow_docs.pdf** → Complete description of the code, each class, function and method.
- **elements_list.yaml** → 1st configuration file to be edited to enter the system components characteristics. Can be opened by any text editor.
- **general_config_file.yaml** → 2nd configuration file to be edited with general information about the simulation and environments connections. Can be open by any text editor.
- **model.py** → Python module allowing to build a model. Should not be modified.
- **components.py** → Python module allowing to build components of the model. Should not be modified.
- **utils.py** → Python module with useful general functions for model.py and components.py. Should not be modified.
- **data_sample.csv** → One week of hourly data for a demo with house, car and module modelization.
- **workflow_demo.ipynb** → Jupyter Notebook ready-to-run to demonstrate the main functionnalities of the model. Requires Python and Jupyter notebook extension to be installed.

3. How to edit configuration files

Configuration files are elements_list.yaml and general_config.yaml.

elements_list.yaml is divided in 4 sections: Source, Demand, Storage, Converter. Each one of these sections can be used to declare corresponding base components, with these parameters:

Source

name	default values	
• activate*		True | False
• energy*		str
• environment*		str
• maximum	*1000000.*	float | str (path)
• maximum_in	maximum	float | str (path)
• maximum_out	maximum	float | str (path)
• minimum	*0.*	float | str (path)
• minimum_in	minimum	float | str (path)
• minimum_out	minimum	float | str (path)
• cost	*0.*	float | str (path)
• cost_in	-cost	float | str (path)
• cost_out	cost	float | str (path)
• factor	*1.*	float | int | 'auto'
• factor_low_bound	*1.*	float | int
• factor_up_bound	*1.*	float | int
• factor_type	*'Continuous'*	'Continuous' | 'Integer' | 'Binary'
• installation_cost	*0.*	float

* mandatory

Demand

name	default values	
• activate*		True | False
• energy*		str
• environment*		str
• value_type*		'constant' | 'hourly'
• value*		float | str (path)
• dispatchable	*False*	True | False
• dispatch_window*		int
• maximum	*1000000.*	float | str (path)
• maximum_in	value* |maximum[1]	float | str (path)
• maximum_out	**0.**[2]	float | str (path)
• minimum	*0.*	float | str (path)
• minimum_in	value* | minimum[1]	float | str (path)
• minimum_out	**0.**[2]	float | str (path)
• cost	*0.*	float | str (path)
• cost_in	-cost	float | str (path)
• cost_out	cost	float | str (path)
• factor	*1.*	float | int
• factor_low_bound	*1.*	float | int
• factor_up_bound	*1.*	float | int
• factor_type	*'Continuous'*	'Continuous' | 'Integer' | 'Binary'
• installation_cost	*0.*	float

* mandatory if dispatchable
[1] if not dispatchable | if dispatchable
[2] will be forced to 0. anyway

* mandatory

Storage

name	default values	
• activate*		True | False
• energy*		str
• environment*		str
• capacity*		float
• initial_SOC*		float
• final_SOC*		float
• efficiency	*1.*	float
• calendar_losses	*1.*	float
• volume_factor	*1.*	float | int | 'auto'
• volume_factor_low_bound	*1.*	float | int
• volume_factor_up_bound	*1.*	float | int
• volume_factor_type	*'Continuous'*	'Continuous' | 'Integer' | 'Binary'
• volume_installation_cost	*0.*	float
• maximum	*1000000.*	float | str (path)
• maximum_in	maximum	float | str (path)
• maximum_out	maximum	float | str (path)
• minimum	*1000000.*	float | str (path)
• minimum_in	minimum	float | str (path)
• minimum_out	minimum	float | str (path)
• cost	*0.*	float | str (path)
• cost_in	-cost	float | str (path)
• cost_out	cost	float | str (path)
• factor	*1.*	float | int | 'auto'
• factor_low_bound	*1.*	float | int
• factor_up_bound	*1.*	float | int
• factor_type	*'Continuous'*	'Continuous' | 'Integer' | 'Binary'
• installation_cost	*0.*	float

* mandatory

Converter

name	default values	
• activate*		True | False
• input_energy*		str
• environment*		str
• output_energies*		dict(str: float)
• maximum	*1000000.*	float | str (path)
• maximum_in	maximum	float | str (path)
• maximum_out	maximum	float | str (path)
• minimum	*1000000.*	float | str (path)
• minimum_in	minimum	float | str (path)
• minimum_out	minimum	float | str (path)
• cost	*0.*	float | str (path)
• cost_in	-cost	float | str (path)
• cost_out	cost	float | str (path)
• factor	*1.*	float | int | 'auto'
• factor_low_bound	*1.*	float | int
• factor_up_bound	*1.*	float | int
• factor_type	*'Continuous'*	'Continuous' | 'Integer' | 'Binary'
• installation_cost	*0.*	float

* mandatory

As for general_config_file.yaml, it should give the following information:

- Number of run *run_num* (int).
- Name of run run_name (str).
- Time axis as path to a csv file column *time* (str), under this form: 'path/to/file.csv//column_name'.
- Sense of optimization optimization_sense (str): 'minimize' or 'maximize'.

Additionally, connections between different environments can be declared and parameterized in the same file, using this template:

EnvironmentsConnection

	default values	
environment_1*		
• envs* (environment_2)		list[str]
• maximum	*1000000.*	list[float | str (path)]
• maximum_in	maximum	list[float | str (path)]
• maximum_out	maximum	list[float | str (path)]
• minimum	*1000000.*	list[float | str (path)]
• minimum_in	minimum	list[float | str (path)]
• minimum_out	minimum	list[float | str (path)]
• cost	*0.*	list[float | str (path)]
• cost_in	-cost	list[float | str (path)]
• cost_out	cost	list[float | str (path)]
• factor	*1.*	list[float | int | 'auto']
• factor_low_bound	*1.*	list[float | int]
• factor_up_bound	*1.*	list[float | int]
• factor_type	*'Continuous'*	list['Continuous' | 'Integer' | 'Binary']
• installation_cost	*0.*	list[float]

* mandatory

Periods of connection and disconnection can be passed through the *maximum* arguments by setting it to 0. when environments are disconnected.

4. Workflow description: main functions

Create your model with the info entered in the two configuration files:

```
myModel = Model(config_file='general_config_file.yaml',
                elements_list='elements_list.yaml')
```

Identify all possible hubs:

```
myModel.initialize_hubs()
```

Create base components and link them to the Hubs:

```
myModel.build_environment_level_variables_and_constraints()
```

Create environments connections and link them to the Hubs:

```
myModel.connect_environments()
```

Hubs equations are now completed. Add them to the linear problem constraints:

```
myModel.add_hubs_equations_to_model()
```

Solve and save results:

```
myModel.solve()
```

Get the value of the objective function:

```
myModel.objective_value
```

Plot all flow variables values grouped by Hubs:

```
myModel.plot_hubs()
```

Plot SOC values for all Storage components:

```
myModel.plot_SOC()
```

Get factor and/or volume factor values for a list of components, e.g.:

```
components = ['car_battery', 'module_battery', 'PV']
factors = ['volume_factor', 'volume_factor', 'factor']
myModel.get_design(components=components,
                   factors=factors)
```

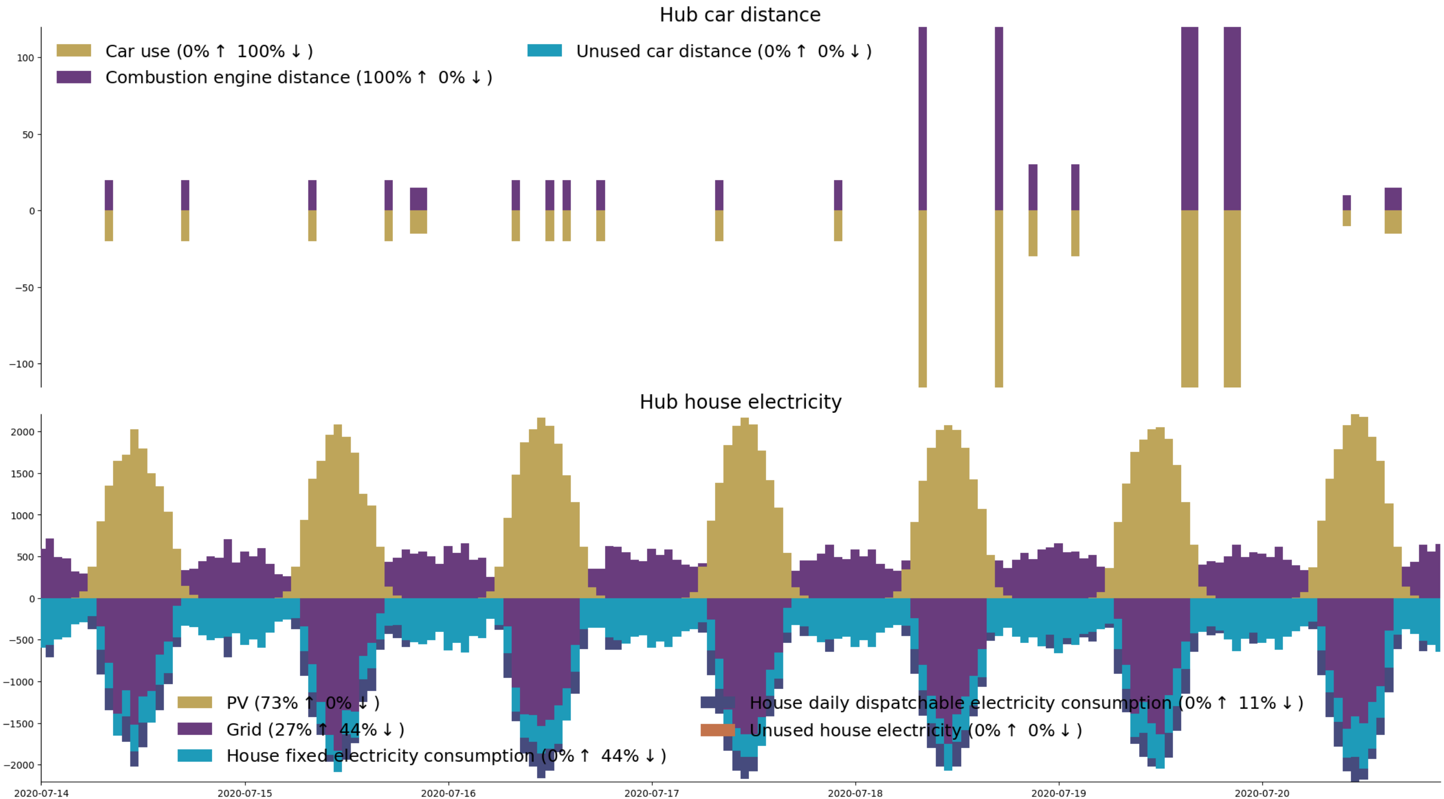
5. Demo: results

Energy flow values after optimization are saved in a file named run_num_run_name. Other results like sizing of equipments and value of the objective function can be obtained through commands in the workflow_light.ipynb file.

Some examples of plots of energy flows are given on next pages for the two upper-mentionned scenarios.

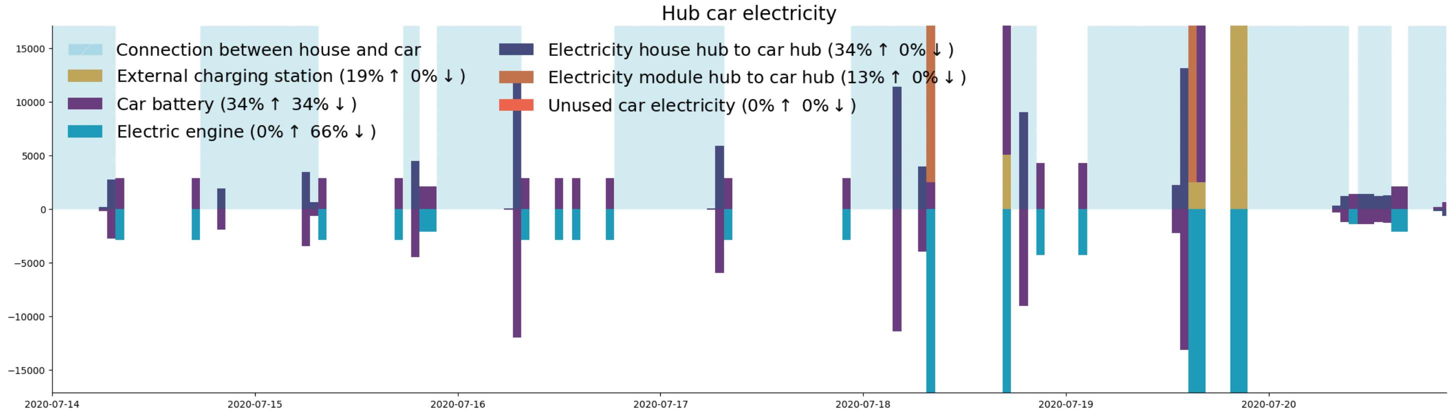
1st scénario - reference

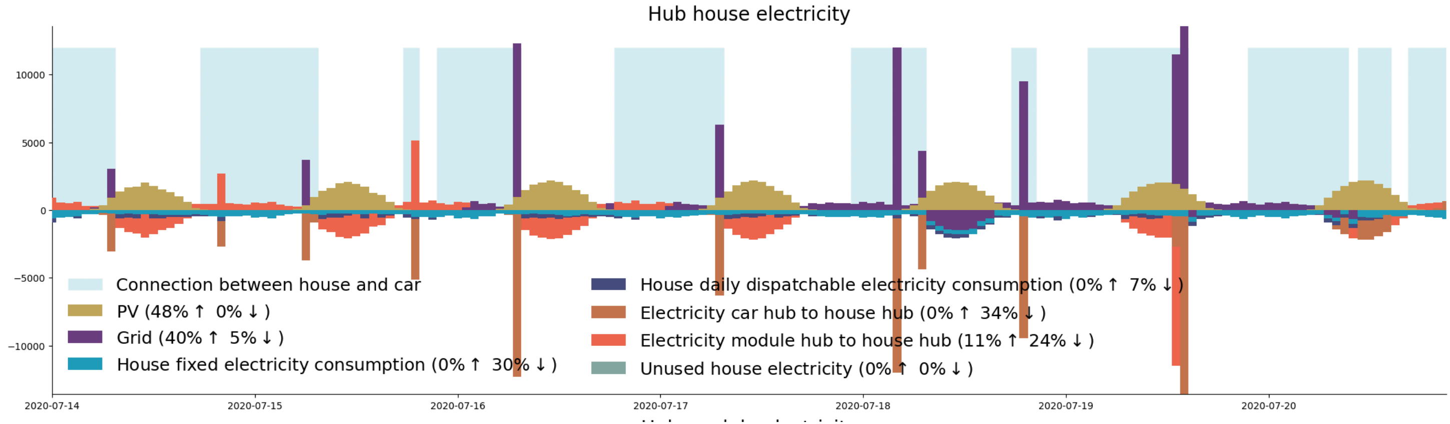
Cost of energy: 112,43€

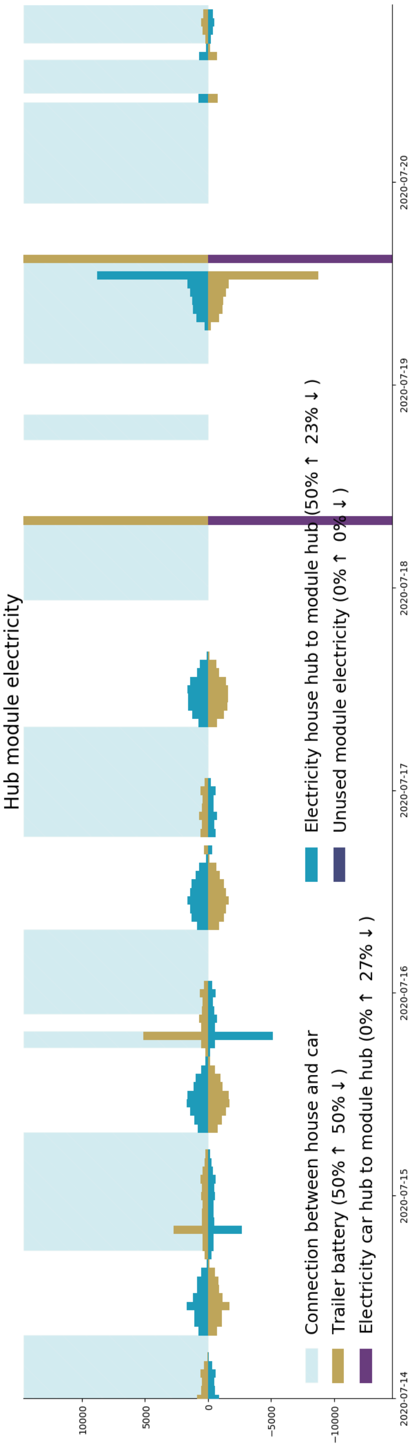


2nd scénario - solution

Cost of energy: 27,48€







Hub module electricity
Connection between house and car
Trailer battery (50% ↑ 50% ↓)
Electricity car hub to module hub (0% ↑ 27% ↓)
Electricity house hub to module hub (50% ↑ 23% ↓)
Unused module electricity (0% ↑ 0% ↓)
10000
5000
0
−5000
−10000
2020-07-14
2020-07-15
2020-07-16
2020-07-17
2020-07-18
2020-07-19
2020-07-20