

---

# **Multi-Vector Simulator (MVS)**

***Release 1.0.2dev***

**Reiner Lemoine Institut and Martha M. Hoffmann**

**Jun 07, 2023**



# GETTING STARTED

<b>1</b>	<b>Maintainers</b>	<b>3</b>
<b>2</b>	<b>Getting Started</b>	<b>5</b>
<b>3</b>	<b>Model Reference</b>	<b>15</b>
<b>4</b>	<b>API Reference</b>	<b>111</b>
<b>5</b>	<b>Indices and tables</b>	<b>197</b>
	<b>Python Module Index</b>	<b>199</b>
	<b>Index</b>	<b>201</b>

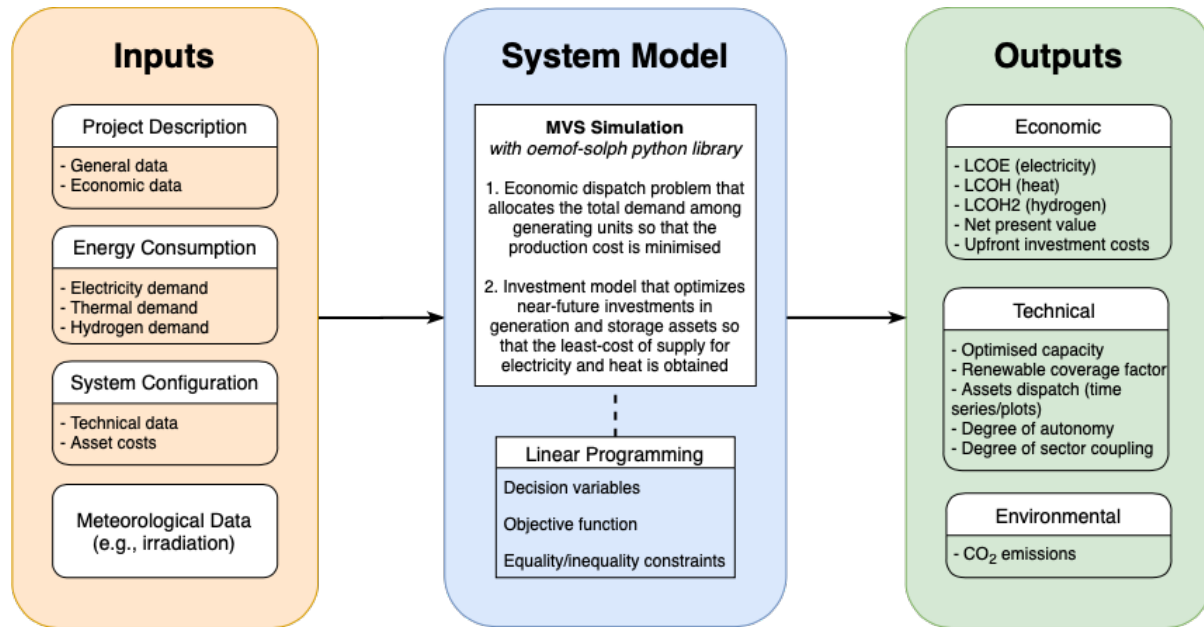


The Multi-Vector Simulator (hereafter MVS) is an `oemof`-based Python package which aims at facilitating the modelling of multi-energy carriers energy systems in island or grid connected mode.

The main goals of the MVS are

1. to minimize the production costs by determining the generating units' optimal output, which meets the total demand
2. to optimize near-future investments in generation and storage assets with the least possible cost of energy.

The MVS graphical model is divided into three connected blocks that trace the logic sequence: inputs, system model, and outputs. This is a typical representation of a simulation model:



The user is asked to provide the required data via a collection of csv files or a unique json file with particular format. The input data is split into the following categories:

- **Project description**, which entails the general information regarding the project (country, coordinates, etc.), as well as the economic data such as the discount factor, project duration, or tax
- **Energy consumption**, which is expressed as times series based on the type of energy (in this case: electrical and thermal)
- **System configuration**, in which the user specifies the technical and financial data of each asset
- **Meteorological data**, which is related to the components that generate electricity by harnessing an existing source of energy that is weather- and time-dependent (e.g., solar and wind power)

This set of input data is then translated to a linear programming problem, also known as a constrained optimization problem. The MVS is based on the `oemof-solph` python library that describes the problem by specifying an objective function to minimize the annual energy supply costs, the decision variables and the bounds and constraints.

The simulation outputs are also separated into categories:

- Economic results used for the financial evaluation, such as the levelized cost of electricity/heat or the net present value of the projected investments
- Technical results that include the optimized capacities and dispatch of each asset
- Environmental results assessing the system's environmental contribution in terms of CO2 emissions.

Additionally, different vizualizations of the results can be provided, eg. as pie charts, plots of asset dispatch and an automatic summary report.

## MAINTAINERS

The multi-vector simulator is currently maintained by staff from [Reiner Lemoine Institute](#).

The MVS is developed as a work package in the European Union's Horizon 2020 Research [E-LAND project](#)

### **Acknowledgement**

This project has received funding from the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No 824388.

### **Disclaimer**

The information and views set out in this document are those of the author(s) and do not necessarily reflect the official opinion of the European Union. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained herein





## GETTING STARTED

### 2.1 Installation

#### 2.1.1 Setup

To set up the MVS, follow the steps below:

- If python3 is not pre-installed: Install miniconda (for python 3.7: <https://docs.conda.io/en/latest/miniconda.html>)
- WINDOWS USERS: Using an Anaconda virtual environment is highly recommended for being able to fully utilize the tool. Venv environments works only for running the optimization tool (mvs\_tool). For this, updating Pandas to at least version 1.3.5 and installing the package pygraphviz as indicated in this link <https://pygraphviz.github.io/documentation/stable/install.html> is necessary. However, it is not possible to run the interactive report (mvs\_report) with venv, as it gives an error. Therefore, it is best to use conda environments.

- Open Anaconda prompt (or other software as Pycharm) to create and activate a virtual environment

```
conda create -n [your_env_name] python=3.6 activate [your env_name]
```

- Install the latest [MVS release](#):

```
pip install multi-vector-simulator
```

- Download the [cbc-solver](https://ampl.com/dl/open/cbc/) into your system from <https://ampl.com/dl/open/cbc/> and integrate it in your system, ie. unzip, place into chosen path, add path to your system variables (Windows: “System Properties” -> “Advanced” -> “Environment Variables”, requires admin-rights).

You can also follow the [steps](#) from the oemof setup instructions

- Test if that the cbc solver is properly installed by typing

```
oemof_installation_test
```

You should at least get a confirmation that the cbc solver is working

```
*****
Solver installed with oemof:

cbc: working
glpk: not working
gurobi: not working
cplex: not working

*****
oemof successfully installed.
*****
```

- Test if the MVS installation was successful by executing

```
mvs_tool
```

This should create a folder `MVS_outputs` with the example simulation's results

You can always check which version you installed with the following command

```
mvs_tool --version
```

## 2.1.2 Using the MVS

To run the MVS with custom inputs you have several options:

### Use the command line

Edit the json input file (or csv files) and run

```
mvs_tool -i path_input_folder -ext json -o path_output_folder
```

With `path_input_folder`: path to folder with input data,

`ext`: json for using a json file and csv for using csv files

and `path_output_folder`: path of the folder where simulation results should be stored.

For more information about the possible command lines options

```
mvs_tool -h
```

### Use the `main()` function

You can also execute the mvs within a script, for this you need to import

```
from multi_vector_simulator.cli import main
```

The possible arguments to this functions are:

- `overwrite` (bool): Determines whether to replace existing results in `path_output_folder` with the results of the current simulation (True) or not (False) (Command line “-f”). Default: False.
- `input_type` (str): Defines whether the input is taken from the `mvs_config.json` file (“json”) or from csv files (“csv”) located within `/csv_elements/` (Command line “-ext”). Default: json.
- `path_input_folder` (str): The path to the directory where the input CSVs/JSON files are located. Default: `inputs/` (Command line “-i”).
- `path_output_folder` (str): The path to the directory where the results of the simulation such as the plots, time series, results JSON files are saved by MVS (Command line “-o”). Default: `MVS_outputs/`.
- `display_output` (str): Sets the level of displayed logging messages. Options: “debug”, “info”, “warning”, “error”. Default: “info”.
- `lp_file_output` (bool): Specifies whether linear equation system generated is saved as lp file. Default: False.
- `pdf_report` (bool): Specify whether pdf report of the simulation's results is generated or not (Command line “-pdf”). Default: False.

- `save_png` (bool): Specify whether png figures with the simulation's results are generated or not (Command line "-png"). Default: False.

Edit the csv files (or, for devs, the json file) and run the `main()` function. The following `kwargs` are possible:

### Default settings

If you execute the `mvs_tool` command in a path where there is a folder named `inputs` (you can use the folder `input_template` for inspiration) this folder will be taken as default input folder and you can simply run

```
mvs_tool
```

A default output folder will be created, if you run the same simulation several time you would have to either overwrite the existing output file with

```
mvs_tool -f
```

Or provide another output folder's path

```
mvs_tool -o <path_to_other_output_folder>
```

### Generate pdf report or an app in your browser to visualise the results of the simulation

To use the report feature you need to install extra dependencies first

```
pip install multi-vector-simulator[report]
```

If you are using zsh terminals and receive the error message "no matches found", you might need to run

```
pip install 'multi-vector-simulator[report]'
```

Use the option `-pdf` in the command line `mvs_tool` to generate a pdf report in a simulation's output folder (by default in `MVS_outputs/report/simulation_report.pdf`):

```
mvs_tool -pdf
```

Use the option `-png` in the command line `mvs_tool` to generate png figures of the results in the simulation's output folder (by default in `MVS_outputs/`):

```
mvs_tool -png
```

To generate a report of the simulation's results, run the following command **after** a simulation generated an output folder:

```
mvs_report -i path_simulation_output_folder -o path_pdf_report
```

where `path_simulation_output_folder` should link to the folder of your simulation's output, or directly to a json file (default `MVS_outputs/json_input_processed.json`) and `path_pdf_report` is the path where the report should be saved as a pdf file.

The report should appear in your browser (at <http://127.0.0.1:8050>) as an interactive Plotly Dash app.

You can then print the report via your browser print functionality (ctrl+p), however the layout of the pdf report is only well optimized for chrome or chromium browser.

It is also possible to automatically save the report as pdf by using the option `-pdf`

```
mvs_report -i path_simulation_output_folder -pdf
```

By default, it will save the report in a `report` folder within your simulation's output folder default (`MVS_outputs/report/`). See `mvs_report -h` for more information about possible options. The `css` and images used to make the report pretty should be located under `report/assets`.

### 2.1.3 Contributing and additional information for developers

If you want to contribute to this project, please read [CONTRIBUTING.md](#). For less experienced github users, we propose a [workflow](#).

For advanced programmers: please checkout the `dev` branch that includes the latest updates and changes. You can find out about the latest changes in the [CHANGELOG.md](#) file.

## 2.2 Simulating with the MVS

The MVS can perform capacity as well as dispatch optimisations of a specific energy system. This means that both the needed additional capacity to be installed is optimised as well as the respective asset's operation. To perform an energy system simulation, a multitude of input parameters is needed. They are described in details in the [input parameters](#) section. They include economic parameters, technological parameters and project settings. Together they define all aspects of the energy system to be simulated and optimised. With these parameters, the MVS builds an energy system model which is translated to a system of linear equations. The MVS tries to find an optimal solution which minimizes the annual costs of demand supply.

In this section, we want to provide you with all information needed to design your own energy system and run your own optimisations. First we will explain the two possible ways to provide the input parameters to the MVS. Then how to draft an energy system model out of a real local energy system configuration.

### 2.2.1 Input files

All input files need to be within a folder with the following structure.

```
input_folder
├── csv_elements
│   ├── constraints.csv
│   ├── economic_data.csv
│   ├── energyBusses.csv
│   ├── energyConsumption.csv
│   ├── energyConversion.csv
│   ├── energyProduction.csv
│   ├── energyProviders.csv
│   ├── energyStorage.csv
│   ├── fixcost.csv
│   ├── project_data.csv
│   ├── simulation_settings.csv
│   └── storage_01.csv
└── time_series
    └── blank
```

└─ mvs\_config.json

The name and location of the `input_folder` is up to the user. The underlying structure and file names within this folder should not be altered (with the exception of `storage_01.csv` which is only required to match a filename provided in `energyStorage.csv`).

There are two allowed formats to provide input data to the MVS: Json or CSV (comma separated values).

The folder `time_series` is always required, it should contain the timeseries for energy demand, energy production and potentially other time-dependent parameters. To provide the inputs using the Json format, only the file `mvs_config.json` is required, whereas for the CSV format only the folder `csv_elements` is required.

The CSV format is more user-friendly to design a local energy system model and the Json format is more compact (the whole model is contained in only one file).

### Csv files: `csv_elements` folder

To use the CSV format, each of the following files have to be present in the folder `csv_elements`.

Files containing enumeration of energy system's assets (or components):

- `energyConsumption.csv` - Energy demands and paths to their time series as csv
- `energyConversion.csv` - Conversion/transformer objects, eg. transformers, generators, heat pumps
- `energyProduction.csv` - Act as energy "sources", ie. PV or wind plants, with paths to their generation time series as csv
- `energyProviders.csv` - Specifics of energy providers, eg. DSOs that are connected to the local energy system, including energy prices and feed-in tariffs
- `energyStorage.csv` - List of energy storages of the energy system
- `storage_01.csv` - Technical parameters of each energy system
- `energyBusses.csv` - Energy busses of the energy system to be simulated

Files containing enumeration of energy system's global parameters:

- `fixcost.csv` - fix project development/maintenance costs (should not be used currently)
- `simulation_settings.csv` - Simulation settings, including start date and duration
- `project_data.csv` - some generic project information
- `constraints.csv` - Constraints on the energy system
- `economic_data.csv` - Major economic parameters of the project

The detailed description of the content of those files is available in the [input parameters](#) section. Moreover, an input folder template is available [here](#).

You can conveniently create a copy of this folder in your local path with the command (after having followed [the installation steps](#)) .. code:

```
mvs_create_input_template
```

A simple example system is setup with this [input folder](#).

**Note:** Currently only one of `,`, `;` or `&` is allowed as value separation for the CSV files (each file should make a coherent use of a unique separator, otherwise leading to parsing problems).

*For developers:* the allowed separators for csv files are located in `src/constants.py` under the `CSV_SEPARATORS` variable.

---

**Note:** The name (or label) of each assets needs to be unique and used coherently across the various csv files.

---

**Note:** If the user used the CSV format to simulate a local energy system, the MVS will automatically create a Json file (`mvs_csv_config.json`) from the provided input data. The user could rename this file `mvs_config.json` and use it as input for the simulation.

---

### Json file: `mvs_config.json`

The structure of the Json file matches the one described by the `csv_elements` folder. The Json format is intended for easier exchange: via http requests for online services such as EPA for example.

Use of Json file is recommended for advanced users only.

There can only be a single Json file in your input folder and it must be named `mvs_config.json`.

An example of a Json file structure is available from the [default scenario](#) of the MVS.

### Time series: `time_series` folder

In the CSV and Json files, the value of the parameter *file\_name* are filenames. Those filenames correspond to files which must be present in the folder `time_series` in your input folder, formatted as CSV. As an example, if one asset listed in *energy production* has `generation_pv.csv` as value for the *file\_name*. The file `generation_pv.csv` containing a value of the pv generation for each timestep of the simulation should be present in the `time_series` folder.

**Note:** When a time series describes a non-dispatchable demand or an otherwise scalar value of a parameter (eg. energy price), the values of the time series can have any positive value.

---

**Note:** For non-dispatchable sources, eg. the generation of a PV plant, you need to provide a specific time series (unit: kWh/kWp, etc.). For the latter, make sure that its values are between zero and one (`[0, 1]`).

---

## 2.2.2 Defining an energy system

To define your energy system you have to fill out the CSV sheets that are provided in the folder `csv_elements`. For each asset you want to add, you have to add a new column to a file. If you do not have an asset of a specific type, simply leave the columns empty (but leave the columns with the parameter names and units).

The unit columns can indicate you what is the type of the parameter which is required from you (string, boolean, number) if it is not a physical unit. In case of doubts, also consider having a look in the [parameter list](#).

**Warning:** Do not delete any of the rows of the CSV's – each parameter is needed for the simulation. There will also be warnings if a parameters is missing or misspelled.

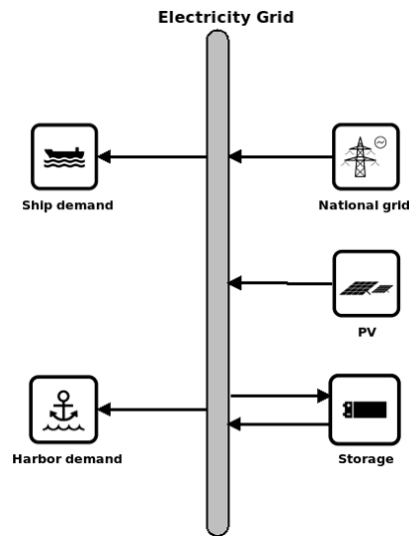
## Example of simple energy systems

Please refer to the [Examples](#) section

### Building a model from assets and energy flows

Simulating an energy system with the MVS requires a certain level of abstraction. In general, as it is based on the programming framework oemof, it will depict the energy system only as linearized model. This allows for the quick computation of the optimal system sizing and approximate dispatch, but does not replace operational management.

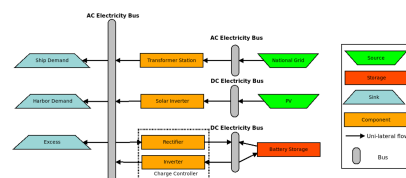
The level of abstraction and system detail needed for an MVS simulation will be explained based on an exemplary local energy system. Let's assume that we want to simulate an industrial site with some electrical demand, the grid connection, a battery as well as a PV plant. A schematic of such a system is shown below.



We can see that we have an electricity bus, to which all other components are connected, specifically demand external electricity supply and the local assets (battery and PV). However even though all those components belong to the same sector, their interconnection with the electricity bus or here the electricity grid could be detailed in the deeper manner.

As such, in reality, the battery may be on an own DC electricity bus, which is either the separate from or identical to the DC bus of the PV plant. Both DC busses would have to be interconnected with the main electricity bus (AC) through an inverter, or in case of bi-directional flow for the battery with a rectifier as well.

Just like so, the DSO could either be only providing electricity also allowing feed in, or the demand may be split up into multiple demand profiles. This granularity of information would be something that the MVS model requires to properly depict the system behaviour and resulted optimal capacities and dispatch. The information fed into the MVS via the CSV's would therefore define following components:



Ideally you sketch down the energy system you want to simulate with the above-mentioned granularity and only using sources, sinks, transformers and buses (meaning the oemof components). When interconnecting different assets make sure that you use the correct bus name in each of the CSV input files. The bus names are defined with `input_direction` and `output_direction`. If you interconnect your assets or buses incorrectly the system will still

be built but the simulation terminated. When executing a simulation, the MVS will generate a rough graphic visualisation of your energy system if you use the option `-png`.

There, all components and buses should be part of a single system (i.e. linked to each other) - otherwise you misconfigured your energy system.

**Warning:** You need to be aware that you yourself have to make sure that the units you assign to your assets and energy flows make sense. The MVS does neither perform a logical check, nor does it transform units, eg. from MWh to kWh.

## 2.3 Example of PV + Battery + Grid

Input files of simple benchmarks (PV + battery + grid) scenarios can be found [here](#)

## 2.4 Adding a timeseries for a parameter

Sometimes you may want to define a parameter not as a scalar value but as a time series. This can for example happen for efficiencies (heat pump COP during the seasons), energy prices (currently only hourly resolution), or the state of charge (for example if you want to achieve a certain stage of charge of an FCEV at a certain point of time).

You can define a scalar as a time series in the csv input files (not applicable for `energyConsumption.csv`), by replacing the scalar value with following dictionary:

```
{ 'file_name': 'your_file_name.csv', 'header': 'your_header', 'unit': 'your_unit' }
```

The feature was tested for following parameters:

- `energy_price`
- `feedin_tariff`
- `dispatch_price`
- `efficiency`

You can see an implemented example here, where the heat pump has a time-dependent efficiency:



Table 1: Example for defining a scalar parameter as a time series

	unit	heat_pump
age_installed	year	0
development_costs	currency	0
specific_costs	currency/kW	7000
efficiency	factor	{ 'file_name': 'cops_eers_test.csv', 'header': 'no_unit', 'unit': 'NA' }
inflow_direction	str	Electricity
installedCap	kW	0
label	str	Heat pump
lifetime	year	20
specific_costs_om	currency/kW/year	0
dispatch_price	currency/kWh	0
optimizeCap	bool	True
outflow_direction	str	Heat
energyVector	str	Electricity
type_oemof	str	transformer
unit	str	kW

The feature is tested with benchmark test `test_benchmark_feature_parameters_as_timeseries()`.

Example input files, where at least one parameter is defined as a time series, can be found here:

- **First example:** Defines the `energy_price` (file) of an energy provider as a time series
- **Second example:** Defines the `energy_price` (file) of an energy provider and the efficiency of a diesel generator (file) as a time series.

## 2.5 Using multiple in- or output busses

Sometimes, you may also want to have multiple input or output busses connected to a component. This is for example the case if you want to model an electrolyzer with a transformer, and want to track water consumption at the same time as you want to track electricity consumption.

You can define this, again, in the csv's. First you should provide the input, or output, busses as a list for the *energy-Conversion.csv* parameter of *inflow\_direction* or *outflow\_direction* resp.:

```
"[h2o_bus, electricity_bus]"
```

Then you need to provide the *efficiencies* and *dispatch prices* respective to each bus, for example:

```
"[0.99, 0.98]"
```

You can also provide a timeseries for one or both values. To do so, you can simply use the notation introduced in *Adding a timeseries for a parameter*:

```
"[0.99, { 'value': { 'file_name': 'your_file_name.csv', 'header': 'your_header' }, 'unit': 'your_unit' }]"
```

You can see an example here, with an electrolyzer :

Table 2: Example for defining a component with multiple inputs/outputs

	unit	electrolyser
age_installed	year	3
development_costs	currency	0
specific_costs	currency/kW	1500
efficiency	factor	“[0.01923,0.28845]”
inflow_direction	str	“[MicroGrid,Water]”
installedCap	kW	0
label	str	Electrolyser
lifetime	year	20
specific_costs_om	currency/kW/year	75
dispatch_price	currency/kWh	“[0,0.0038]”
optimizeCap	bool	True
outflow_direction	str	Local H2 grid
energyVector	str	Electricity
type_oemof	str	transformer
unit	str	kW

The features were integrated with [Pull Request #63](#) and [Pull Request #949](#).

For more information, you might also reference following issues:

- Parameters can now be a list of values, eg. efficiencies for two busses or multiple input/output vectors([Issue #52](#))
- Parameters can now be defined as a list as well as as a timeseries ([Issue #52](#), [Issue #82](#))

## 2.6 Tips & Tricks

### 2.6.1 Including sunk costs for previous investments into specific assets

Usually, the investments into existing capacities are neglected and assumed to be sunk costs of the system. The existing capacity `installedCap` as well as the age of the installed asset `age_installed` are only used to calculate when necessary re-investments take place, and how high the *replacements costs* are. But there is no option if one wants to optimize a system with pre-existing capacities of certain assets and still account for the installation costs that happened before the first time step of the simulation.

When optimizing a system with pre-existing capacities of certain assets, it can be usefull for the user to implement the installation costs of these assets in the economic evaluation. This trick triggers the replacement of those assets, thus accounting for investments costs of pre-existing assets in the scenario.

With the trick presented here, it is possible to optimize a system with a specific or a specific minimal capacity of a certain asset and still account for installation costs of the asset at the beginning of the project (in the idea of a greenfield / brownfield optimization). The presented trick works for energy production assets as well as energy conversion assets.

To apply this trick, the following manipulations must be applied to the input parameters of the asset in question:

- `optimizeCap` to False
- `installedCap` to the specific existing capacity
- `aged_installed` to the lifetime of the asset

Previous investment costs into now pre-existing asset capacities are now taken into account in the economic evaluation of a scenario.

## MODEL REFERENCE

- **How the energy system is modelled:** *Assumption behind the model | Available components for modelling | Setting constraints on model or components | Scope and limitation of the model*
- **Description of parameters:** *Input parameters | Output variables and KPIs*
- **Validation of the model:** *Validation methodology*

### 3.1 Assumptions

The MVS uses the programming framework `oemof-solph` at its core and builds an energy system model based upon its nomenclature. As such, the energy system model can be described with a linear equation system. The most important aspects of a linear equation system are described below in a generalized way, and additionally explained through the use of an example. This will enable the clear comparison to other energy system models.

#### 3.1.1 Economic Dispatch

Linear programming is a mathematical modelling and optimization technique for a system of a linear objective function subject to linear constraints. The goal of a linear programming problem is to find the optimal value for the objective function, be it a maximum or a minimum. The MVS is based on `oemof-solph`, which in turn uses `Pyomo` to create a linear problem. The economic dispatch problem in the MVS has the objective of minimizing the production cost by allocating the total demand among the generating units at each time step. The equation is the following:

$$\min Z = \sum_i a_i \cdot CAP_i + \sum_i \sum_t c_{var,i} \cdot E_i(t)$$

$$CAP_i \geq 0$$

$$E_i(t) \geq 0 \quad \forall t$$

$i$ : asset

$a_i$ : asset annuity [currency/kWp/year, currency/kW/year, currency/kWh/year]

$CAP_i$ : asset capacity [kWp, kW, kWh]

$c_{var,i}$ : variable operational or dispatch cost [currency/kWh, currency/L]

$E_i(t)$ : asset dispatch [kWh]

The annual cost function of each asset includes the capital expenditure (investment cost) and residual value, as well as the operating expenses of each asset. It is expressed as follows:

$$a_i = \left( capex_i + \sum_{k=1}^n \frac{capex_i}{(1+d)^{k \cdot t_a}} - c_{res,i} \right) \cdot CRF(T) + opex_i$$

$$CRF(T) = \frac{d \cdot (1+d)^T}{(1+d)^T - 1}$$

$capex_i$ : specific investment costs [currency/unit]

$n$ : number of replacements of an asset within project lifetime  $T$

$t_a$ : asset lifetime [years]

$CRF$ : capital recovery factor

$c_{res,i}$ : residual value of asset  $i$  at the end of project lifetime  $T$  [currency/unit]

$opex_i$ : annual operational and management costs [currency/unit/year]

$d$ : discount factor

$T$ : project lifetime [years]

The CRF is a ratio used to calculate the present value of the annuity. The discount factor can be replaced by the weighted average cost of capital (WACC), calculated by the user.

The lifetime of the asset  $t_a$  and the lifetime of the project  $T$  can be different from each other; as a result, the number of replacements  $n$  is estimated using the equation below:

$$n = \text{round} \left( \frac{T}{t_a} + 0.5 \right) - 1$$

The residual value is also known as the salvage value and it represents an estimate of the monetary value of an asset at the end of the project lifetime  $T$ . The MVS considers a linear depreciation over  $T$  and accounts for the time value of money through the use of the following equation:

$$c_{res,i} = \frac{capex_i}{(1+d)^{n \cdot t_a}} \cdot \frac{1}{T} \cdot \frac{(n+1) \cdot t_a - T}{(1+d)^T}$$

### 3.1.2 Energy Balance Equation

One main constraint that the optimization model is subject to is the energy balance equation, which specifically maintains equality between the total incoming and outgoing energy of a bus. This balancing equation is applicable to all bus types, be it electrical, thermal, hydrogen or any other energy carrier.

$$\sum E_{in,i}(t) - \sum E_{out,j}(t) = 0 \quad \forall t$$

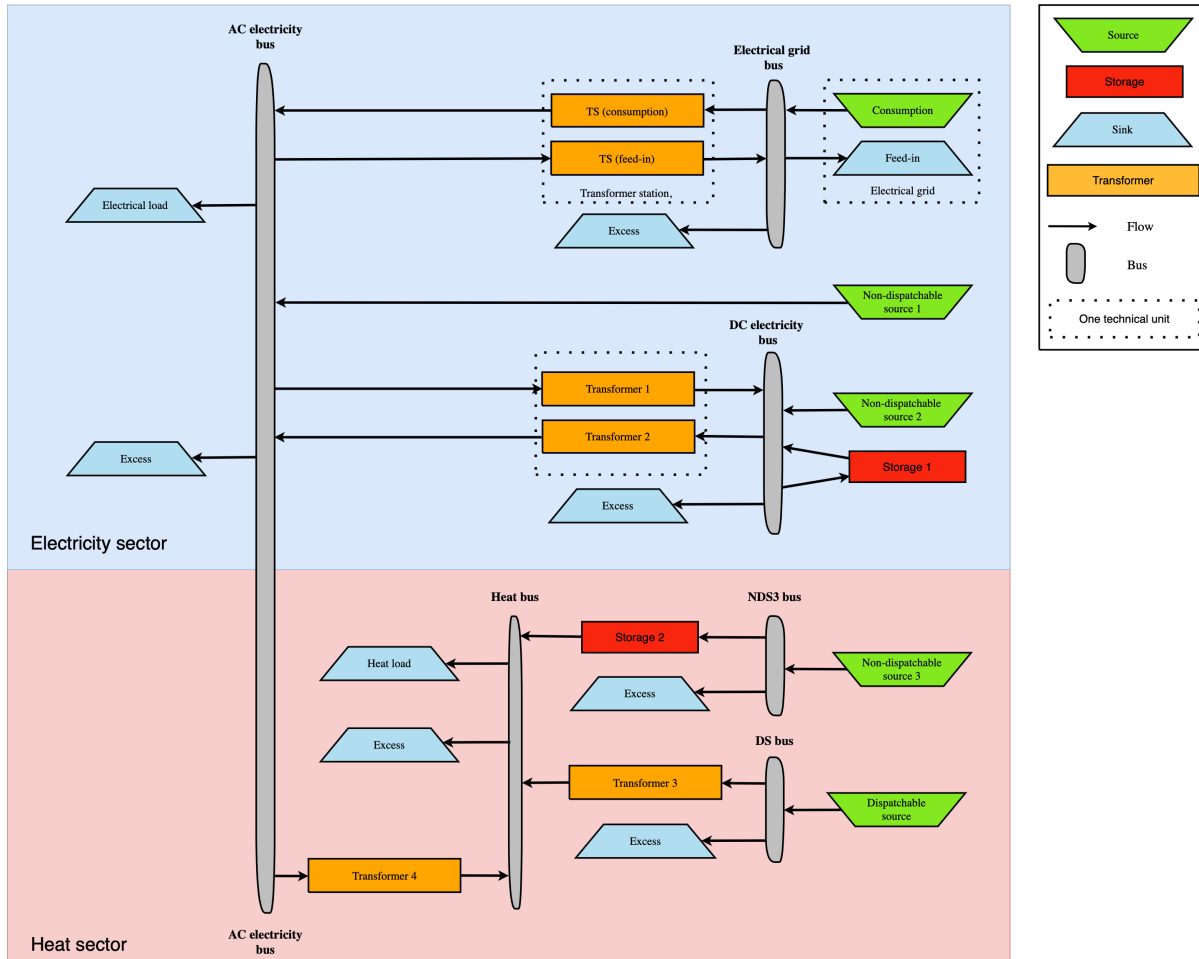
$E_{in,i}$ : energy flowing from asset  $i$  to the bus

$E_{out,j}$ : energy flowing from the bus to asset  $j$

It is very important to note that assets  $i$  and  $j$  can be the same asset (e.g. a battery with an electrical inflow/outflow). *Oemof-solph* allows both  $E_{in}$  and  $E_{out}$  to be larger than zero in the same time step  $t$  (see [Infeasible bi-directional flow in one timestep](#)).

### 3.1.3 Example: Sector Coupled Energy System

In order to understand the component models, a generic sector coupled energy system example is shown in the figure below. It brings together the electricity and heat sector through a transformer (Transformer 4) which connects the two sector buses.



For the sake of simplicity, the following table gives an example for each asset type with an abbreviation to be used in the energy balance and component equations.

Table 1: Asset Types and Examples

Asset Type	Asset Example	Abbreviation	Unit
Non-dispatchable source 1	Wind turbine	wind	kW
Non-dispatchable source 2	Photovoltaic panels	pv	kWp
Storage 1	Battery energy storage	bat	kWh
Transformer 1	Rectifier	rec	kW
Transformer 2	Solar inverter	inv	kW
Non-dispatchable source 3	Solar thermal collector	stc	kWth
Storage 2	Thermal energy storage	tes	kWth
Dispatchable source	Heat source (e.g., biogas)	heat	L
Transformer 3	Turbine	turb	kWth
Transformer 4	Heat pump	hp	kWth

All grid and dispatchable source asset types are assumed to be available 100% of the time with no consumption limits. For each bus in the system, the MVS automatically includes a sink component for excess energy related to the bus, which is denoted  $E_{ex}$  in the equations. This excess sink accounts for the extra energy in the system that has to be dumped.

### Electricity Grid Equation

The electricity grid is modeled through a feed-in and a consumption node. Transformers limit the peak flow into or from the local electricity line, and electricity sold to the grid experiences losses in the transformer ( $ts, f$ ).

$$E_{grid,c}(t) - E_{grid,f}(t) + E_{ts,f}(t) \cdot \eta_{ts,f} - E_{ts,c}(t) = 0 \quad \forall t$$

$E_{grid,c}$ : energy consumed from the electricity grid

$E_{grid,f}$ : energy fed into the electricity grid

$E_{ts,c}$ : transformer station feed-in

$\eta_{ts,f}$ : transformer station efficiency

$E_{ts,f}$ : transformer station consumption

### Non-Dispatchable Source Equations

Non-dispatchable sources in the sector coupled energy system example are wind, PV and solar thermal power. Their generation is determined by the provided timeseries of instantaneous generation, providing  $\alpha, \beta, \gamma$  in relation to wind, PV and solar thermal power respectively.

$$E_{wind}(t) = CAP_{wind} \cdot \alpha_{wind}(t) \quad \forall t$$

$$E_{pv}(t) = CAP_{pv} \cdot \beta_{pv}(t) \quad \forall t$$

$$E_{stc}(t) = CAP_{stc} \cdot \gamma_{stc}(t) \quad \forall t$$

$E_{wind}$ : energy generated from the wind turbine  
 $CAP_{wind}$ : wind turbine capacity [kW]  
 $\alpha_{wind}$ : instantaneous wind turbine performance metric [kWh/kW]  
 $E_{pv}$ : energy generated from the PV panels  
 $CAP_{pv}$ : PV panel capacity [kWp]  
 $\beta_{pv}$ : instantaneous PV specific yield [kWh/kWp]  
 $E_{stc}$ : energy generated from the solar thermal collector  
 $CAP_{stc}$ : Solar thermal collector capacity [kWth]  
 $\gamma_{stc}$ : instantaneous collector's production [kWh/kWth]

## Storage Model

There are two storages in the defined example system: An electrical energy storage (Storage 1, *bat*) and a thermal energy storage (Storage 2, *tes*). Below, the equations for Storage 1 are provided, but Storage 2 follows analogous equations for charge, discharge and bounds.

$$E_{bat}(t) = E_{bat}(t-1) + E_{bat,in}(t) \cdot \eta_{bat,in} - \frac{E_{bat,out}}{\eta_{bat,out}} - E_{bat}(t-1) \cdot \epsilon \quad \forall t$$

$$CAP_{bat} \cdot SOC_{min} \leq E_{bat}(t) \leq CAP_{bat} \cdot SOC_{max} \quad \forall t$$

$$0 \leq E_{bat}(t) - E_{bat}(t-1) \leq CAP_{bat} \cdot C_{rate,in} \quad \forall t$$

$$0 \leq E_{bat}(t-1) - E_{bat}(t) \leq CAP_{bat} \cdot C_{rate,out} \quad \forall t$$

$E_{bat}$ : energy stored in the battery at time t  
 $E_{bat,in}$ : battery charging energy  
 $\eta_{bat,in}$ : battery charging efficiency  
 $E_{bat,out}$ : battery discharging energy  
 $\eta_{bat,out}$ : battery discharging efficiency  
 $\epsilon$ : decay per time step  
 $CAP_{bat}$ : battery capacity [kWh]  
 $SOC_{min}$ : minimum state of charge  
 $SOC_{max}$ : maximum state of charge  
 $C_{rate,in}$ : battery charging rate  
 $C_{rate,out}$ : battery discharging rate

## DC Electricity Bus Equation

The following equation illustrates an example of a DC bus with a battery, PV and a bi-directional inverter.

$$E_{pv}(t) + E_{bat,out}(t) \cdot \eta_{bat,out} + E_{rec}(t) \cdot \eta_{rec} - E_{inv}(t) - E_{bat,in} - E_{ex}(t) = 0 \quad \forall t$$

$E_{rec}$ : rectifier energy  
 $\eta_{rec}$ : rectifier efficiency  
 $E_{inv}$ : inverter energy

### AC Electricity Bus Equation

This equation describes the local electricity grid and all connected assets:

$$E_{ts,c}(t) \cdot \eta_{ts,c} + E_{wind}(t) + E_{inv}(t) \cdot \eta_{inv} - E_{ts,c}(t) - E_{rec}(t) - E_{hp}(t) - E_{el}(t) - E_{ex}(t) = 0 \quad \forall t$$

$\eta_{ts,c}$ : transformer station efficiency

$\eta_{inv}$ : inverter efficiency

$E_{hp}$ : heat pump electrical consumption

$E_{el}$ : electrical load

### Heat Bus Equation

This equation describes the heat bus and all connected assets:

$$E_{tes}(t) \cdot \eta_{tes} + E_{turb}(t) \cdot \eta_{turb} + E_{hp}(t) \cdot COP - E_{th}(t) - E_{ex}(t) = 0$$

$\eta_{tes}$ : thermal storage efficiency

$\eta_{turb}$ : turbine efficiency

$COP$ : heat pump coefficient of performance

$E_{th}$ : heat load

### NDS3 Bus Equation

The NDS3 Bus is an example of a bus which does not serve both as the input and output of a storage system. Instead, the thermal storage is charged from the NDS3 bus, but discharges into the heat bus.

$$E_{stc}(t) - E_{tes}(t) - E_{ex}(t) = 0$$

$E_{tes}$ : thermal energy storage

### DS Bus Equation

The DS Bus shows an example of a fuel source providing an energy carrier (biogas) to a transformer (turbine).

$$E_{heat}(t) - E_{turb}(t) - E_{ex}(t) = 0$$

$E_{heat}$ : thermal energy (biogas) production

$E_{turb}$ : turbine (biogas turbine) energy

### 3.1.4 Cost calculations

The optimization of the MVS is mainly based on costs. There is, however, the possibility of introducing additional constraints which will impact the optimization results e.g. implementing a maximum installable capacity limit (comp. *maximumCap*) or adding constraints for certain key performance indicators (see *Constraints*). In order to optimize the energy systems properly, the economic data provided with the input data has to be pre-processed (also see *Economic Dispatch*) and then also post-processed when evaluating the results. The following assumptions are therefore important:



- **Project lifetime:** The simulation has a defined project lifetime, for which continuous operation is assumed - which means that the first year of operation is considered to be the same as the last year of operation. Existing and optimized assets have to be replaced (if their lifetime precedes the system lifetime) to make this possible.
- **Simulation duration:** It is advisable to simulate the whole year to find the most suitable combination of energy assets for your system. Sometimes however you might want to look at specific seasons to see their effect - this is possible in the MVS by choosing a specific start date and simulation duration.
- **Asset costs:** Each asset can have development costs, specific investment costs, specific operation and management costs as well as dispatch costs.
  - *Replacement costs* are calculated based on the lifetime of the assets, and residual values are paid at the end of the project.
  - *Development costs* are costs that will occur regardless of the installed capacity of an asset - even if it is not installed at all. It stands for system planning and licensing costs. If you have optimized your energy system and see that an asset might not be favourable (zero optimized capacities), you might want to run the simulation again and remove the asset, or remove the development costs of the asset.
  - *Specific investment costs* and *specific operation and maintenance costs* are used to calculate the annual expenditures that an asset has per year, in the process also adding the replacement costs.
  - *Dispatch price* can often be set to zero, but are supposed to cover instances where utilization of an asset requires increased operation and maintenance or leads to wear.
- **Pre-existing capacities:** It is possible to add assets that already exist in your energy system with their capacity and age.
  - *Replacements* - To ensure that the energy system operates continuously, the existing assets are replaced with the same capacities when they reached their end of life within the project lifetime.
  - *Replacement costs* are calculated based on the lifetime of the asset in general and the age of the pre-existing capacities
- **Fix project costs:** It is possible to define fix costs of the project - this is important if you want to compare different project locations with each other. You can define...
  - *Development costs*, which could for example stand for the cost of licenses of the whole energy system
  - *(Specific) investment costs*, which could be an investment into land or buildings at the project site. When you define a lifetime for the investment, the MVS will also consider replacements and reimbursements.
  - *(Specific) operation and management costs*, which can cover eg. the salaries of at the project site

### 3.1.5 Weighting of energy carriers

To be able to calculate sector-wide key performance indicators, it is necessary to assign weights to the energy carriers based on their usable potential. In the conference paper handed in to the CIRED workshop, we have proposed a methodology comparable to Gasoline Gallon Equivalents.

After thorough consideration, it has been decided to base the equivalence in tonnes of oil equivalent (TOE). Electricity has been chosen as a baseline energy carrier, as our pilot sites mainly revolve around it and also because we believe that this energy carrier will play a larger role in the future. For converting the results into a more conventional unit, we choose crude oil as a secondary baseline energy carrier. This also enables comparisons with crude oil price developments in the market. For most KPIs, the baseline energy carrier used is of no relevance as the result is not dependent on it. This is the case for KPIs such as the share of renewables at the project location or its self-sufficiency. The choice of the baseline energy carrier is relevant only for the levelized cost of energy (LCOE), as it will either provide a system-wide supply cost in Euro per kWh electrical or per kg crude oil.

First, the conversion factors to kg crude oil equivalent [1] were determined (see *Conversion factors: kg crude oil equivalent (kgoe) per unit of a fuel* below). These are equivalent to the energy carrier weighting factors with baseline energy carrier crude oil.

Following conversion factors and energy carriers are defined:

Table 2: Conversion factors: kg crude oil equivalent (kgoe) per unit of a fuel

Energy carrier	Unit	Value
H2 [3]	kgoe/kgH2	2.87804
LNG	kgoe/kg	1.0913364
Crude oil	kgoe/kg	1
Gas oil/diesel	kgoe/litre	0.81513008
Kerosene	kgoe/litre	0.0859814
Gasoline	kgoe/litre	0.75111238
LPG	kgoe/litre	0.55654228
Ethane	kgoe/litre	0.44278427
Electricity	kgoe/kWh(el)	0.0859814
Biodiesel	kgoe/litre	0.00540881
Ethanol	kgoe/litre	0.0036478
Natural gas	kgoe/litre	0.00080244
Heat	kgoe/kWh(therm)	0.086
Heat	kgoe/kcal	0.0001
Heat	kgoe/BTU	0.000025

The values of ethanol and biodiesel seem comparably low in [1] and [2] and do not seem to be representative of the net heating value (or lower heating value) that was expected to be used here.

From this, the energy weighting factors are calculated using the electricity content for crude oil as baseline (see *Electricity equivalent conversion per unit of a fuel* below).

Table 3: Electricity equivalent conversion per unit of a fuel

Energy carrier	Unit	Value
LNG	kWh(elec)/kg	12.6927
Crude oil	kWh(elec)/kg	11.6304
Diesel	kWh(elec)/litre	9.4803
Kerosene	kWh(elec)/litre	8.9080
Gasoline	kWh(elec)/litre	8.7358
LPG	kWh(elec)/litre	6.4728
Ethane	kWh(elec)/litre	5.1498
H2	kWh(elec)/kgH2	33.4728
Electricity	kWh(elec)/kWh(el)	1
Biodiesel	kWh(elec)/litre	0.0629
Ethanol	kWh(elec)/litre	0.0424
Natural gas	kWh(elec)/litre	0.009
Heat	kWh(elec)/kWh(therm)	1.0002
Heat	kWh(elec)/kcal	0.0011
Heat	kWh(elec)/BTU	0.0003

With this, the equivalent potential of an energy carrier  $E_{\text{elec},i}$ , compared to electricity, can be calculated with its conversion factor  $w_i$  as:

$$E_{\text{elec},i} = E_i \cdot w_i$$

As it can be noticed, the conversion factor between heat (kWh(therm)) and electricity (kWh(el)) is almost 1. The deviation stems from the data available in source [1] and [2]. The equivalency of heat and electricity can be a source of discussion, as from an exergy point of view these energy carriers can not be considered equivalent. When combined, say with a heat pump, the equivalency can also result in ripple effects in combination with the minimal renewable factor or the minimal degree of autonomy, which need to be evaluated during the pilot simulations.

For the most part, the energy carrier weighting factors are similar to the lower heating value of the fuel in question. A stark deviation is noticable for ethanol and biodiesel. This deviation should be investigated further. In the future, it should be discussed whether it would be better to directly use the lower heating values of a fuel as its energy carrier weighting factor, as this would be more intuitive.

---

**Note:** The `energy_vector` of each of the assets and busses must be identical in spelling to one of the energy carriers defined in the above table. Spaces should be translated to underscores (ie. Crude oil as an energy carrier is defined as `Crude_oil` in the input files). Other energy carriers can not be parsed and will raise a warning. Please note that *Heat* currently has to be measured in kWh(thermal).

---

#### Code

Currently, the energy carrier conversion factors are defined in `constants.py` with `DEFAULT_WEIGHTS_ENERGY_CARRIERS`. New energy carriers should be added to its list when needed. Unknown carriers raise an `UnknownEnergyVectorError` error.

#### Comment

Please note that the energy carrier weighting factor is not applied dependent on the LABEL of the energy asset, but based on its energy vector. Let us consider an example:

In our system, we have a dispatchable *diesel fuel source*, with dispatch carrying the unit *l Diesel*. The energy vector needs to be defined as *Diesel* for the energy carrier weighting to be applied, ie. the energy vector of *diesel fuel source* needs to be *Diesel*. This will also have implications for the KPI: For example, the *degree of sector coupling* will reach its maximum, when the system only has heat demand and all of it is provided by processing diesel fuel. If you want to portrait diesel as something inherent to heat supply, you will need to make the diesel source a heat source, and set its *dispatch costs* to currency/kWh, ie. divide the diesel costs by the heating value of the fuel.

#### Comment

In the MVS, there is no distinction between energy carriers and energy vector. For *Electricity* of the *Electricity* vector this may be self-explanatory. However, the energy carriers of the *Heat* vector can have different technical characteristics: A fluid on different temperature levels. As the MVS measures the energy content of a flow in kWh(thermal) however, this distinction is only relevant for the end user to be aware of, as two assets that have different energy carriers as an output should not be connected to one and the same bus if a detailed analysis is expected. An example of this would be, that a system where the output of the diesel boiler as well as the output of a solar thermal panel are connected to the same bus, eventhough they can not both supply the same kind of heat demands (radiator vs. floor heating). This, however, is something that the end-user has to be aware of themselves, eg. by defining self-explanatory labels.

### 3.1.6 Emission factors

In order to optimise the energy system with minimum emissions, it is important to calculate emission per unit of fuel consumption.

In table *Emission factors: Kg of CO2 equivalent per unit of fuel consumption* the emission factors for energy carriers are defined. These values are based on direct emissions during stationary consumption of the mentioned fuels.

Table 4: Emission factors: Kg of CO2 equivalent per unit of fuel consumption

Energy carrier	Unit	Value	Source
Diesel	kgCO2eq/litre	2.7	[4] Page No. 26
Gasoline	kgCO2eq/litre	2.3	[4] Page No. 26
Kerosene	kgCO2eq/litre	2.5	[4] Page No. 26
Natural gas	kgCO2eq/m3	1.9	[4] Page No. 26
LPG	kgCO2eq/litre	1.6	[4] Page No. 26
Biodiesel	kgCO2eq/litre	0.000125	[5] Page No. 6
Bioethanol	kgCO2eq/litre	0.0000807	[5] Page No. 6
Biogas	kgCO2eq/m3	0.12	[6] Page No. 1

In table *CO2 Emission factors: grams of CO2 equivalent per kWh of electricity consumption* the CO2 emissions for Germany and the four pilot sites (Norway, Spain, Romania, India) are defined:

Table 5: CO2 Emission factors: grams of CO2 equivalent per kWh of electricity consumption

Country	Unit	Value	Source
Germany	gCO2eq/kWh	338	[7] Fig. 2
Norway	gCO2eq/kWh	19	[7] Fig. 2
Spain	gCO2eq/kWh	207	[7] Fig. 2
Romania	gCO2eq/kWh	293	[7] Fig. 2
India	gCO2eq/kWh	708	[8] Page No. 7

The values mentioned in the table above account for emissions during the complete life cycle. This includes emissions during energy production, energy conversion, energy storage and energy transmission.

### 3.1.7 Input verification

The inputs for a simulation with the MVS are subjected to a couple of verification tests to make sure that the inputs result in valid oemof simulations. This should ensure:

- Uniqueness of labels (C1.check\_for\_label\_duplicates): This function checks if any LABEL provided for the energy system model in dict\_values is a duplicate. This is not allowed, as oemof can not build a model with identical labels.
- No levelized costs of generation lower than feed-in tariff of same energy vector in case of investment optimization (optimizeCap is True) (C1.check\_feedin\_tariff\_vs\_levelized\_cost\_of\_generation\_of\_providers): Raises error if feed-in tariff > levelized costs of generation if maximumCap is None for energy asset in ENERGY\_PRODUCTION. This is not allowed, as oemof otherwise may be subjected to an unbound problem, ie. a business case in which an asset should be installed with infinite capacities to maximize revenue. If maximumCap is not None a logging.warning is shown as the maximum capacity of the asset will be installed.
- No feed-in tariff higher than energy price from an energy provider (C1.check\_feedin\_tariff\_vs\_energy\_price): Raises error if feed-in tariff > energy price of any asset in energyProvider.csv. This is not allowed, as oemof otherwise is subjected to an unbound and unrealistic problem, eg. one where the owner should consume electricity to feed it directly back into the grid for its revenue.
- Assets have well-defined energy vectors and belong to an existing bus (C1.check\_if\_energy\_vector\_of\_all\_assets\_is\_valid): Validates for all assets, whether energyVector is defined within DEFAULT\_WEIGHTS\_ENERGY\_CARRIERS and within the energyBusses.

- Energy carriers used in the simulation have defined factors for the electricity equivalency weighting (`C1.check_if_energy_vector_is_defined_in_DEFAULT_WEIGHTS_ENERGY_CARRIERS`): Raises an error message if an energy vector is unknown. It then needs to be added to the `DEFAULT_WEIGHTS_ENERGY_CARRIERS` in `constants.py`
- An energy bus is always connected to one inflow and one outflow (`C1.check_for_sufficient_assets_on_busses`): Validating model regarding busses - each bus has to have more than two assets connected to it, excluding energy excess sinks
- Time series of energyProduction assets that are to be optimized have specific generation profiles (`C1.check_non_dispatchable_source_time_series`, `C1.check_time_series_values_between_0_and_1`): Raises error if time series of non-dispatchable sources are not between [0, 1].
- Provided timeseries are checked for NaN values, which are replaced by zeroes (`C0.replace_nans_in_timeseries_with_0`).
- Asset capacities connected to each bus are sized sufficiently to fulfill the maximum demand (`C1.check_energy_system_can_fulfill_max_demand`): Logs a logging.warning message if the aggregated installed capacity and maximum capacity (if applicable) of all conversion, generation and storage assets connected to one bus is smaller than the maximum demand. The check is applied to each bus of the energy system. Check passes when the potential peak supply is larger then or equal to the peak demand on the bus, or if the maximum capacity of an asset is set to None when optimizing.

## 3.2 Component models

The component models of the MVS result from the used python-library `oemof-solph` for energy modeling.

It requires component models to be simplified and linearized. This is the reason why the MVS can provide a pre-feasibility study of a specific system setup, but not the final sizing and system design. The types of assets are presented below.

### 3.2.1 Energy consumption

Demands within the MVS are added as energy consumption assets in *energyConsumption.csv*. Most importantly, they are defined by a timeseries, representing the demand profile, and their energy vector. A number of demand profiles can be defined for one energy system, both of the same and different energy vectors. The main optimization goal for the MVS is to supply the defined demand without fail for all of the timesteps in the simulation with the least cost of supply (comp. *Economic Dispatch*).

### 3.2.2 Energy production

#### Non-dispatchable sources of generation

*Examples:*

- PV plants
- Wind plants
- Run-of-the-river hydro power plants
- Solar thermal collectors

Variable renewable energy (VRE) sources, like wind and PV, are non-dispatchable due to their fluctuations in supply. They are added as sources in *energyProduction.csv*.

The fluctuating nature of non-dispatchable sources is represented by generation time series that show the respective production for each time step of the simulated period. In energy system modelling it is common to use hourly time series. The name of the file containing the time series is added to *energyProduction.csv* with the parameter *file\_name*. For further requirements concerning the time series see section *Time series: time\_series folder*.

If you cannot provide time series for your VRE assets you can consider to calculate them by using models for generating feed-in time series from weather data. The following is a list of examples, which is not exhaustive:

- PV: *pplib* , *Renewables Ninja* (download capacity factors), *altilite*
- Wind: *windpowerlib*, *Renewables Ninja* (download capacity factors), *altilite*
- Hydro power (run-of-the-river): *hydropowerlib*
- Solar thermal: *flat plate collectors* of *oemof.thermal*

### Dispatchable sources of generation

*Examples:*

- Fuel sources
- Deep-ground geothermal plant (ground assumed to allow unlimited extraction of heat, not depending on season)

Fuel sources are added as dispatchable sources, which can have development, investment, operational and dispatch costs. They are added to *energyProduction.csv*, while setting *file\_name* to *None*.

Fuel sources are for example needed as source for a diesel generator (diesel), biogas plant (gas) or a condensing power plant (gas, coal, ...), see *Energy conversion*.

Energy providers, even though also dispatchable sources of generation, should be added via *energyProviders.csv*, as there are some additional features available then, see *Energy providers*.

Both energy providers and the additional fuel sources are limited to the options of energy carriers provided in the table of *Electricity equivalent conversion per unit of a fuel*, as the default weighting factors to translate the energy carrier into electricity equivalent need to be defined.

### 3.2.3 Energy conversion

*Examples:*

- Electric transformers (rectifiers, inverters, transformer stations, charge controllers)
- HVAC and Heat pumps (as heater and/or chiller)
- Combined heat and power (CHP) and other condensing power plants
- Diesel generators
- Electrolyzers
- Biogas power plants

Conversion assets are added as transformers and are defined in *energyConversion.csv*.

The parameters *dispatch\_price*, *efficiency* and *installedCap* of transformers are assigned to their output flows. This means that these parameters need to be provided for the output of the asset and that the costs of the input, (e.g. cost of fuel) are not included in its *dispatch\_price* but in the *dispatch\_price* of the fuel source, see *Dispatchable sources of generation*.

Conversion assets can be defined with multiple inputs or multiple outputs, but one asset currently cannot have both, multiple inputs and multiple outputs. Note that multiple inputs/output is possible but this feature is not currently tested.

### Electric transformers

Electric rectifiers and inverters that are transforming electricity in one direction only, are simply added as transformers. Bidirectional converters and transformer stations are defined by two transformers that are optimized independently from each other, if optimized. The same accounts for charge controllers for a *Battery energy storage system (BESS)* that are defined by two transformers, one for charging and one for discharging. The parameters *dispatch\_price*, *efficiency* and *installedCap* need to be given for the electrical output power of the electric transformers.

---

**Note:** When using two conversion objects to emulate a bidirectional conversion asset, their capacity should be inter-dependent. This is currently not the case, see *Infeasible bi-directional flow in one timestep*.

---

### Heating, Ventilation, and Air Conditioning (HVAC)

Like other conversion assets, devices for heating, ventilation and air conditioning (HVAC) are added as transformers. As the parameters *dispatch\_price*, *efficiency* and *installedCap* are assigned to the output flows they need to be given for the nominal heat output of the HVAC.

Different types of HVAC can be modelled. Except for an air source device with ambient temperature as heat reservoir, the device could be modelled with two inputs (electricity and heat) in case the user is interested in the heat reservoir. This has not been tested yet. Also note that currently efficiencies are assigned to the output flows the see [issue #799](#). Theoretically, a HVAC device can be modelled with multiple outputs (heat, cooling, ...); this has not been tested yet.

The efficiency of HVAC systems is defined by the coefficient of performance (COP), which is strongly dependent on the temperature. In order to take account of this, the efficiency can be defined as time series, see section *Adding a timeseries for a parameter*. If you do not provide your own COP time series you can calculate them with `oemof.thermal`, see [documentation on compression heat pumps and chillers](#) and [documentation on absorption chillers](#).

### Electrolyzers

Electrolyzers are added as transformers with a constant or time dependent but in any case pre-defined efficiency. The parameters *dispatch\_price*, *efficiency* and *installedCap* need to be given for the output of the electrolyzers (hydrogen).

Currently, electrolyzers are modelled with only one input flow (electricity), not taking into account the costs of water; see [issue #799](#). The minimal operation level and consumption in standby mode are not taken into account, yet, see [issue #50](#).

### Condensing power plants and Combined heat and power (CHP)

Condensing power plants are added as transformers with one input (fuel) and one output (electricity), while CHP plants are defined with two outputs (electricity and heat). The parameters *dispatch\_price*, *efficiency* and *installedCap* need to be given for the electrical output power (and nominal heat output) of the power plant, while fuel costs need to be included in the *dispatch\_price* of the fuel source.

The ratio between the heat and electricity output of a CHP is currently simulated as fix values. This might be changed in the future by using the `ExtractionTurbineCHP` or the `GenericCHP` component of `oemof`, see [issue #803](#)

Note that multiple inputs/output have not been tested yet.



### Other fuel powered plants

Fuel powered conversion assets, such as diesel generators and biogas power plants, are added as transformers. The parameters *dispatch\_price*, *efficiency* and *installedCap* need to be given for the electrical output power of the diesel generator or biogas power plant. As described above, the costs for diesel and gas need to be included in the *dispatch\_price* of the fuel source.

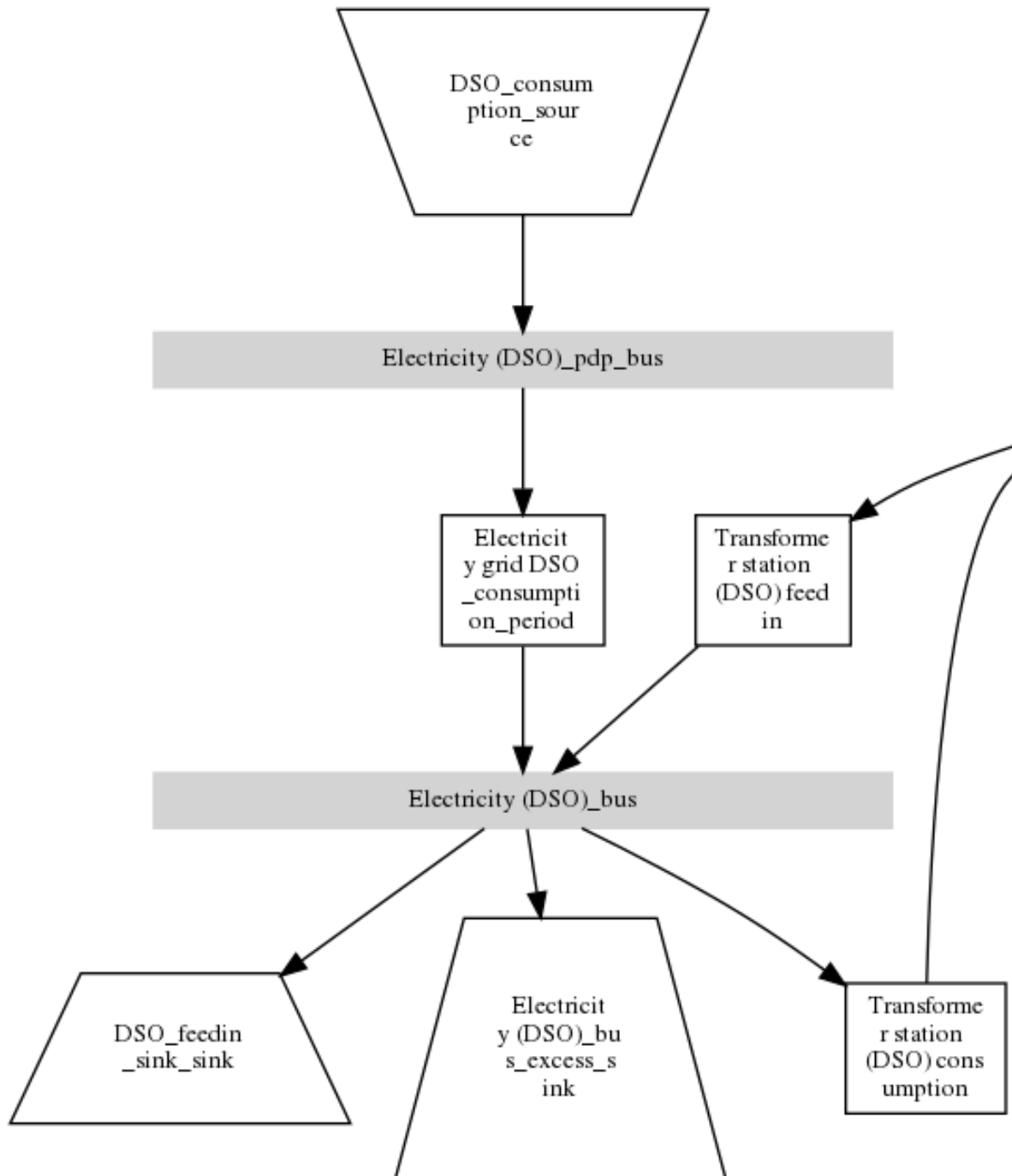
### 3.2.4 Energy providers

The energy providers are the most complex assets in the MVS model. They are composed of a number of sub-assets

- Energy consumption source, providing the energy required from the system with a certain price
- Energy peak demand pricing “transformers”, which represent the costs induced due to peak demand
- Bus connecting energy consumption source and energy peak demand pricing transformers
- Energy feed-in sink, able to take in generation that is provided to the energy provider for revenue
- Optionally: Transformer Station connecting the energy provider bus to the energy bus of the LES

With all these components, the energy provider can be visualized as follows:





## Variable energy consumption prices (time-series)

Energy consumption prices can be added as values that vary over time. See section *Time series: time\_series folder* or more information.

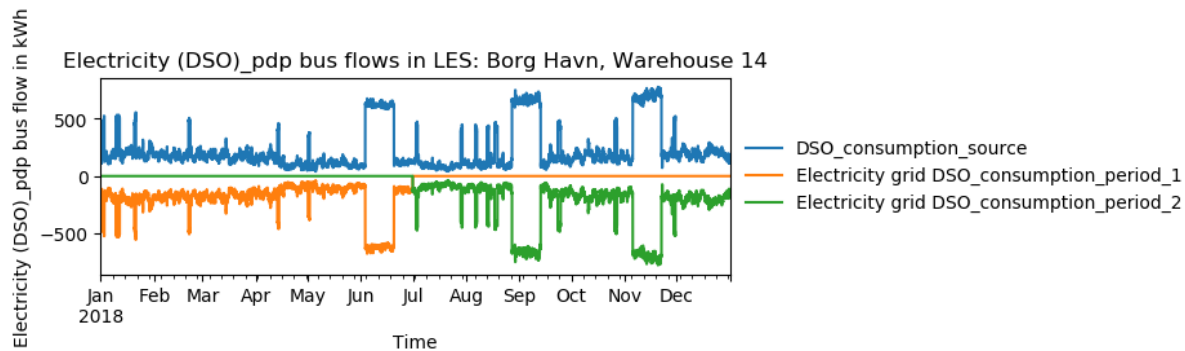
## Peak demand pricing

A peak demand pricing scheme is based on an electricity tariff, that requires the consumer not only to pay for the aggregated energy consumption in a time period (eg. kWh electricity), but also for the maximum peak demand (load, eg. kW power) towards the grid of the energy provider within a specific pricing period.

In the MVS, this information is gathered in `energyProviders` assets with:

- `multi_vector_simulator.utils.constants_json_strings.PEAK_DEMAND_PRICING_PERIOD` as the period used in peak demand pricing. Possible values are 1 (yearly), 2 (half-yearly), 3 (each trimester), 4 (quarterly), 6 (every 2 months) and 12 (each month). If you have a `simulation_duration < 365` days, the periods will still be set up assuming a year! This means, that if you are simulating 14 days, you will never be able to have more than one peak demand pricing period in place.
- `multi_vector_simulator.utils.constants_json_strings.PEAK_DEMAND_PRICING` as the costs per peak load unit, eg. kW

To represent the peak demand pricing, the MVS adds a “transformer” that is optimized with specific operation and maintenance costs per year equal to the `PEAK_DEMAND_PRICING` for each of the pricing periods. For two peak demand pricing periods, the resulting dispatch could look as following:



## 3.2.5 Energy storage

Energy storages such as battery storages, thermal storages or H2 storages are modelled with the `GenericStorage` component of `oemof.solph`. They are designed for one input and one output and are defined within the files `energyStorage.csv` and `storage_*.csv`.

The state of charge of a storage at the first and last time step of an optimization are equal. Charge and discharge of the whole capacity of the energy storage are possible within one time step in case the capacity of the storage is not optimized. In case of capacity optimization charge and discharge is limited by the *c-rate*.

## Battery energy storage system (BESS)

BESS are modelled as `GenericStorage` like described above. The BESS can either be connected directly to the electricity bus of the LES or via a charge controller that manages the BESS. When choosing the second option, the capacity of the charge controller can be optimized individually, which takes its specific costs and lifetime into consideration. If you do not want to optimize the charge controller's capacity you can take its costs and efficiency into account when defining the storage's input and output power, see *storage\_\*.csv*. A charge controller is defined by two transformers, see section *Energy conversion* above.

Note that capacity reduction over the lifetime of a BESS that may occur due to different effects during aging cannot be taken into consideration in MVS. A possible workaround for this could be to manipulate the lifetime.

## Hydrogen storage (H2)

Hydrogen storages are modelled as all storage types in MVS with as `GenericStorage` like described above.

The most common hydrogen storages store H2 as liquid under temperatures lower than -253 °C or under high pressures. The energy needed to provide these requirements cannot be modelled via the storage component as another energy sector such as cooling or electricity is needed. It could therefore, be modelled as an additional demand of the energy system, see *issue #811*

## Thermal energy storage

Thermal energy storages of the type sensible heat storage (SHS) are modelled as `GenericStorage` like described above. The implementation of a specific type of SHS, the stratified thermal energy storage, is described in section *Stratified thermal energy storage*. The modelling of latent-heat (or Phase-change) and chemical storages have not been tested with MVS, but might be achieved by precalculations.

## Stratified thermal energy storage

Stratified thermal energy storage is defined by the two optional parameters *fixed\_thermal\_losses\_relative* and *fixed\_thermal\_losses\_absolute*. If they are not included in *storage\_\*.csv* or are equal to zero, then a normal generic storage is simulated instead. These two parameters are used to take into account temperature dependent losses of a thermal storage. To model a thermal energy storage without stratification, the two parameters are not set. The default values of *fixed\_thermal\_losses\_relative* and *fixed\_thermal\_losses\_absolute* are zero. Except for these two additional parameters the stratified thermal storage is implemented in the same way as other storage components.

Precalculations of the *installedCap*, *efficiency*, *fixed\_thermal\_losses\_relative* and *fixed\_thermal\_losses\_absolute* can be done orientating on the stratified thermal storage component of *oemof.thermal*. The parameters U-value, volume and surface of the storage, which are required to calculate *installedCap*, can be precalculated as well.

The efficiency  $\eta$  of the storage is calculated as follows:

$$\eta = 1 - \text{loss\_rate}$$

This example shows how to do precalculations using stratified thermal storage specific input data:

```
from oemof.thermal.stratified_thermal_storage import (
    calculate_storage_u_value,
    calculate_storage_dimensions,
    calculate_capacities,
    calculate_losses,
)
```

(continues on next page)

(continued from previous page)

```
# Precalculation
u_value = calculate_storage_u_value(
    input_data['s_iso'],
    input_data['lamb_iso'],
    input_data['alpha_inside'],
    input_data['alpha_outside'])

volume, surface = calculate_storage_dimensions(
    input_data['height'],
    input_data['diameter']
)

nominal_storage_capacity = calculate_capacities(
    volume,
    input_data['temp_h'],
    input_data['temp_c'])

loss_rate, fixed_losses_relative, fixed_losses_absolute = calculate_losses(
    u_value,
    input_data['diameter'],
    input_data['temp_h'],
    input_data['temp_c'],
    input_data['temp_env'])
```

Please see the *oemof.thermal* [examples](#) and the [documentation](#) for further information.

For an investment optimization the height of the storage should be left open in the precalculations and *installedCap* should be set to 0 or NaN.

An implementation of the stratified thermal storage component has been done in [pvcompare](#). You can find the precalculations of the stratified thermal energy storage made in *pvcompare* [here](#).

### 3.2.6 Energy excess

---

**Note:** Energy excess components are implemented **automatically** by MVS! You do not need to define them yourself.

---

An energy excess sink is placed on each of the LES energy busses, and therefore energy excess is allowed to take place on each bus of the LES. This means that there are assumed to be sufficient vents (heat) or resistors (electricity) to dump excess (waste) generation. Excess generation can only take place when a non-dispatchable source is present or if an asset is allowed to supply energy without any fuel or dispatch costs.

In case of excessive excess energy, a warning is issued that it seems to be cheaper to have high excess generation than investing into more capacities. High excess energy can for example result into an optimized inverter capacity that is smaller than the peak generation of installed PV. The model becomes unrealistic when the excess is very high.

## 3.3 Constraints

Constraints are controlled via the file *constraints.csv*.

### 3.3.1 Minimal renewable factor constraint

The minimal renewable factor constraint requires the capacity and dispatch optimization of the MVS to reach at least the minimal renewable factor defined within the constraint. The renewable share of the optimized energy system may also be higher than the minimal renewable factor.

The minimal renewable factor is applied to the minimal renewable factor of the whole, sector-coupled energy system, but not to specific sectors. As such, energy carrier weighting plays a role and may lead to unexpected results. The constraint reads as follows:

$$\text{minimalrenewablefactor} \leq \frac{\sum \text{renewablegeneration} \cdot \text{weightingfactor}}{\sum \text{renewablegeneration} \cdot \text{weightingfactor} + \sum \text{non-renewablegeneration} \cdot \text{weightingfactor}}$$

Please be aware that the minimal renewable factor constraint defines bounds for the *Renewable factor (renewable\_factor)* of the system, ie. taking into account both local generation as well as renewable supply from the energy providers. The constraint explicitly does not aim to reach a certain *Renewable share of local generation (renewable\_share\_of\_local\_generation)* on-site.

#### Deactivating the constraint

The minimal renewable factor constraint is deactivated by inserting the following row in *constraints.csv* as follows:

```
`minimal_renewable_factor,factor,0`
```

#### Activating the constraint

The constraint is enabled when the value of the minimal renewable factor factor is above 0 in *constraints.csv*:

```
`minimal_renewable_factor,factor,0.3`
```

Depending on the energy system, especially when working with assets which are not to be capacity-optimized, it is possible that the minimal renewable factor criterion cannot be met. The simulation terminates in that case. If you are not sure if your energy system can meet the constraint, set all `optimize_Cap` parameters of your optimizable assets to `True`, and then investigate further.

Also, if you are aiming at very high minimal renewable factors, the simulation time can increase drastically. If you do not get a result after an excessive simulation time (e.g. 10 times the simulation without constraints), you should consider terminating the simulation and trying with a lower minimum renewable share.

The minimum renewable share is introduced to the energy system by `D2.constraint_minimal_renewable_share()` and a validation test is performed with `E4.minimal_constraint_test()`.

### 3.3.2 Minimal degree of autonomy constraint

The minimal degree of autonomy constraint requires the capacity and dispatch optimization of the MVS to reach at least the minimal degree of autonomy defined within the constraint. The degree of autonomy of the optimized energy system may also be higher than the minimal degree of autonomy. For more details, refer to the definition of *degree of autonomy*

The minimal degree of autonomy is applied to the whole, sector-coupled energy system, but not to specific sectors. As such, energy carrier weighting plays a role and may lead to unexpected results.

The constraint reads as follows:

$$\text{minimal degree of autonomy} \leq DA = \frac{\sum E_{demand,i} \cdot w_i - \sum E_{consumption,provider,j} \cdot w_j}{\sum E_{demand,i} \cdot w_i}$$

#### Deactivating the constraint

The minimal degree of autonomy constraint is deactivated by inserting the following row in *constraints.csv* as follows:

```
`minimal_degree_of_autonomy,factor,0`
```

#### Activating the constraint

The constraint is enabled when the value of the minimal degree of autonomy is above 0 in *constraints.csv*:

```
`minimal_degree_of_autonomy,factor,0.3`
```

Depending on the energy system, especially when working with assets which are not subject to the optimization of their capacities, it is possible that the minimal degree of autonomy criterion cannot be met. The simulation terminates in that case. If you are not sure if your energy system can meet the constraint, set all *optimizeCap* parameters of your optimizable assets to True, and then investigate further.

The minimum degree of autonomy is introduced to the energy system by *D2.constraint\_minimal\_degree\_of\_autonomy()* and a validation test is performed with *E4.minimal\_constraint\_test()*.

### 3.3.3 Maximum emission constraint

The maximum emission constraint limits the maximum amount of total emissions per year of the energy system. It allows the capacity and dispatch optimization of the MVS to result into a maximum amount of emissions defined by the maximum emission constraint. The yearly emissions of the optimized energy system may also be lower than the maximum emission constraint.

---

**Note:** The maximum emissions constraint currently does not take into consideration life cycle emissions, also see *Total GHG emissions (total\_emissions)* section for an explanation.

---

#### Activating the constraint

The maximum emissions constraint is enabled by inserting the following row in *constraints.csv* as follows:

```
`maximum_emissions,kgCO2eq/a,8000000`
```

#### Deactivating the constraint

The constraint is deactivated by setting the value in *constraints.csv* to None:

```
`maximum_emissions,kgCO2eq/a,None`
```

The unit of the constraint is *kgCO2eq/a*. To pick a realistic value for this constraint you can e.g.:

- Firstly, optimize your system without the constraint to get an idea about the scale of the emissions and then, secondly, set the constraint and lower the emissions step by step until you reach an unbound problem (which then represents the non-achievable minimum of emissions for your energy system)
- Check the emissions targets of your region/country and disaggregate the number

The maximum emissions constraint is introduced to the energy system by *D2.constraint\_maximum\_emissions()* and a validation test is performed with *E4.maximum\_emissions\_test()*.

### 3.3.4 Net zero energy (NZE) constraint

The net zero energy (NZE) constraint requires the capacity and dispatch optimization of the MVS to result into a net zero system, but can also result in a plus energy system. The degree of NZE of the optimized energy system may be higher than 1, in case of a plus energy system. Please find the definition of net zero energy (NZE) and the KPI here: [Degree of Net Zero Energy \(degree\\_of\\_nze\)](#).

Some definitions of NZE systems in literature allow the energy system's demand solely be provided by locally generated renewable energy. In MVS this is not the case - all locally generated energy is taken into consideration. To enlarge the share of renewables in the energy system you can use the [Minimal renewable factor constraint](#).

The NZE constraint is applied to the whole, sector-coupled energy system, but not to specific sectors. As such, energy carrier weighting plays a role and may lead to unexpected results. The constraint reads as follows:

$$\sum_i E_{\text{feedin},\text{provider}}(i) \cdot w_i - E_{\text{consumption},\text{provider}}(i) \cdot w_i \geq 0$$

#### Deactivating the constraint

The NZE constraint is deactivated by inserting the following row in [constraints.csv](#) as follows:

```
`net_zero_energy,bool,False`
```

#### Activating the constraint

The constraint is enabled when the value of the NZE constraint is set to **True** in [constraints.csv](#):

```
`net_zero_energy,bool,True`
```

Depending on the energy system, especially when working with assets which are not subject to the optimization of their capacities, it is possible that the NZE criterion cannot be met. The simulation terminates in that case. If you are not sure whether your energy system can meet the constraint, set all [optimizeCap](#) parameters of your optimizable assets to **True**, and then investigate further.

The net zero energy constraint is introduced to the energy system by [D2.constraint\\_net\\_zero\\_energy\(\)](#) and a validation test is performed with [E4.net\\_zero\\_energy\\_test\(\)](#).

## 3.4 Limitations

When running simulations with the MVS, there are certain peculiarities to be aware of. The peculiarities can be considered as limitations, some of which are merely model assumptions and others are drawbacks of the model. A number of those are inherited due to the nature of the MVS and its underlying modules. The following table ([Limitations](#)) lists the MVS limitations based on their type.

Table 6: Limitations

Inherited	Can be addressed
<i>Infeasible bi-directional flow in one timestep</i>	<i>Need to model one technical unit with two transformer assets</i>
<i>Simplified linear component models</i>	<i>Random excess energy distribution</i>
<i>No degradation of efficiencies over a component lifetime</i>	<i>Renewable energy share definition relative to energy carriers</i>
<i>Perfect foresight</i>	<i>Energy carrier weighting</i>
	<i>Events of energy shortage or grid interruption cannot be modelled</i>

### 3.4.1 Infeasible bi-directional flow in one timestep

#### Limitation

It is not possible to model two flows in opposite directions during the same time step.

#### Reason

The MVS is based on the python library `oemof-solph`. Its generic components are used to set up the energy system. As a ground rule, the components of `oemof-solph` are unidirectional. This means that for an asset that is bidirectional two transformer objects have to be used. Examples for this are:

- Physical bi-directional assets, eg. inverters
- Logical bi-directional assets, eg. consumption from the grid and feed-in to the grid

To achieve the real-life constraint one flow has to be zero when the other is larger zero, one would have to implement following relation:

$$E_{in} \cdot E_{out} = 0$$

However, this relation creates a non-linear problem and can not be implemented in `oemof-solph`.

#### Implications

This limitation means that the MVS might result in infeasible dispatch of assets. For instance, a bus might be supplied by a rectifier and itself supplying an inverter at the same time step  $t$ , which cannot logically happen if these assets are part of one physical bi-directional inverter. Another case that could occur is feeding the grid and consuming from it at the same time  $t$ .

Under certain conditions, including excess generation as well as dispatch costs of zero, the infeasible dispatch can also be observed for batteries and result in a parallel charge and discharge of the battery. If this occurs, a solution may be to set a marginal dispatch cost of battery charge.

### 3.4.2 Simplified linear component models

#### Limitation

The MVS simplifies the component model of some assets.

- Generators have an efficiency that is not load-dependent
- Storage have a charging efficiency that is not SOC-dependent
- Turbines are implemented without ramp rates

#### Reason

The MVS is based `oemof-solph` python library and uses its generic components to set up an energy system. Transformers and storages cannot have variable efficiencies, because otherwise the system of equation to solve would not be linear.

#### Implications

Simplifying the implementation of some component specifications can be beneficial for the ease of the model, however, it contributes to the lack of realism and might result in less accurate values. The MVS trades off the decreased level of detail for a quick evaluation of its scenarios, which are often only used for a pre-feasibility analysis.



### 3.4.3 No degradation of efficiencies over a component lifetime

#### Limitation

The MVS does not degrade the efficiencies of assets over the lifetime of the project, eg. in the case of production assets like PV panels.

#### Reason

The simulation of the MVS is only based on a single reference year, and it is not possible to take into account multi-year degradation of asset efficiency.

#### Implications

This results in an overestimation of the energy generated by the asset, which implies that the calculation of some other results might also be overestimated (e.g. overestimation of feed-in energy). The user can circumvent this by applying a degradation factor manually to the generation time series used as an input for the MVS.

### 3.4.4 Perfect foresight

#### Limitation

The optimal solution of the energy system is based on perfect foresight.

#### Reason

As the MVS and thus `oemof-solph`, which is handling the energy system model, know the generation and demand profiles for the whole simulation time and solve the optimization problem based on a linear equation system, the solver knows their dispatch for certain, whereas in reality the generation and demand could only be forecasted.

#### Implications

The perfect foresight can lead to suspicious dispatch of assets, for example charging of a battery right before a (in real-life) random blackout occurs. The systems optimized with the MVS therefore, represent their optimal potential, which in reality could not be reached. The MVS has thus a tendency to underestimate the needed battery capacity or the minimal state of charge for backup purposes, and also designs the PV system and backup power according to perfect forecasts. In reality, operational margins would need to be considered.

### 3.4.5 Optimization precision

#### Limitation

Marginal capacities and flows below a threshold of  $10^{-6}$  are rounded to zero.

#### Reason

The MVS makes use of the open energy modelling framework (`oemof`) by using `oemof-solph`. For the MVS, we use the `cbc-solver` with a `ratioGap=0.03`. This influences the precision of the optimized decision variables, ie. the optimized capacities as well as the dispatch of the assets. In some cases the dispatch and capacities vary around 0 with fluctuations of the order of floating point precision (well below  $<10e-6$ ), thus resulting sometimes in marginal fluctuations dispatch or capacities around 0. When calculating KPI from these decision variables, the results can be nonsensical, for example leading to SoC curves with negative values or values far above the viable value 1. As the reason for these inconsistencies is known, the MVS enforces the capacities and dispatch of to be above  $10e-6$ , ie. all capacities or flows smaller than that are automatically set to zero. This is applied to absolute values, so that irregular (and incorrect) values for decision variables can still be detected.

#### Implications

If your energy system has demand or resource profiles that include marginal values below the threshold of  $10^{-6}$ , the MVS will not result in appropriate results. For example, that means that if you have an energy system with usually is measured in *MW* but one demand is in the *W* range, the dispatch of assets serving this minor demand is not displayed correctly. Please consider using *kW* or even *W* as a base unit then.

### 3.4.6 Extension of KPIs necessary

#### Limitation

Some important KPIs usually required by developers are currently not implemented within the MVS:

- Internal rate of return (IRR)
- Payback period
- Return on equity (ROE),

#### Reason

The MVS tool is a work in progress and this can still be addressed in the future.

#### Implications

The absence of such indicators might affect decision-making.

### 3.4.7 Random excess energy distribution

#### Limitation

There is random excess distribution between the feed-in sink and the excess sink when no feed-in-tariff is assumed in the system.

#### Reason

Since there is no feed-in-tariff to benefit from, the MVS randomly distributes the excess energy between the feed-in and excess sinks. As such, the distribution of excess energy changes when running several simulations for the same input files.

#### Implications

On the first glance, the distribution of excess energy onto both feed-in sink and excess sink may seem off to the end-user. Other than these inconveniences, there are no real implications that affect the capacity and dispatch optimization. When a degree of self-supply and self-consumption is defined, the limitation might tarnish these results.

### 3.4.8 Renewable energy share definition relative to energy carriers

#### Limitation

The current renewable energy share depends on the share of renewable energy production assets directly feeding the load. The equation to calculate the share also includes the energy carrier rating as described here below:

$$RES = \frac{\sum_i E_{RE,generation}(i) \cdot w_i}{\sum_i E_{RE,generation}(i) \cdot w_i + \sum_k E_{nonRE,generation}(k) \cdot w_k}$$

with *i*: renewable energy asset

*k*: non-renewable energy asset

#### Reason

The MVS tool is a work in progress and this can still be addressed in the future.

#### **Implications**

This might result in different values when comparing them to other models. Another way to calculate it is by considering the share of energy consumption supplied from renewable sources.

### **3.4.9 Energy carrier weighting**

#### **Limitation**

The MVS assumes a usable energy content rating for every energy carrier. The current version assumes that 1 kWh thermal is equivalent to 1 kWh electricity.

#### **Reason**

This is an approach that the MVS currently uses.

#### **Implications**

By weighing the energy carriers according to their energy content (Gasoline Gallon Equivalent (GGE)), the MVS might result in values that can't be directly assessed. Those ratings affect the calculation of the levelized cost of the energy carriers, but also the minimum renewable energy share constraint.

### **3.4.10 Events of energy shortage or grid interruption cannot be modelled**

#### **Limitation**

The MVS assumes no shortage or grid interruption in the system.

#### **Reason**

The aim of the MVS does not cover this scenario.

#### **Implications**

Electricity shortages due to power cuts might happen in real life and the MVS currently omits this scenario. If a system is self-sufficient but relies on grid-connected PV systems, the latter stop feeding the load if any power cuts occur and the battery storage systems might not be enough to serve the load thus resulting energy shortage.

### **3.4.11 Need to model one technical unit with two transformer assets**

#### **Limitation**

Two transformer objects representing one technical unit in real life are currently unlinked in terms of capacity and attributed costs.

#### **Reason**

The MVS uses oemof-solph's generic components which are unidirectional so for a bidirectional asset, two transformer objects have to be used.

#### **Implications**

Since only one input is allowed, such technical units are modelled as two separate transformers that are currently unlinked in the MVS (e.g., hybrid inverter, heat pump, distribution transformer, etc.). This raises a difficulty to define costs in the input data. It also results in two optimized capacities for one logical unit.

This limitation can be addressed with a constraint which links both capacities of one logical unit, and therefore solves both the problem to attribute costs and the previously differing capacities.

## 3.5 Input parameters

### 3.5.1 Parameters in each category/CSV file

Important note: Each asset and bus needs to have a unique label. In the csv input files, these are defined by the column headers. The input parameters are gathered under the following categories. These categories reflect the structure of the csv input files or the firsts keys of the json input file.

#### **constraints.csv**

The file *constraints.csv* includes the following parameter(s):

- *maximum\_emissions*
- *minimal\_degree\_of\_autonomy*
- *minimal\_renewable\_factor*
- *net\_zero\_energy*

#### **economic\_data.csv**

The file *economic\_data.csv* includes all economic data that the simulation will use. This includes the following parameters:

- *currency*
- *discount\_factor*
- *project\_duration*
- *tax*

#### **energyBusses.csv**

The file *energyBusses.csv* defines all busses required to build the energy system. It includes following parameters:

- *energyVector*

#### **energyConsumption.csv**

The file *energyConsumption.csv* defines all energy demands that should be included in the energy system. It includes the following parameters:

- *energyVector*
- *file\_name*
- *inflow\_direction*
- *outflow\_direction*
- *type\_oemof*
- *unit*

### energyConversion.csv

The file *energyConversion.csv* defines the assets that convert one energy carrier into another one, eg. inverters or generators. Following parameters define them:

- *age\_installed*
- *development\_costs*
- *dispatch\_price*
- *efficiency*
- *energyVector*
- *inflow\_direction*
- *installedCap*
- *lifetime*
- *optimizeCap*
- *outflow\_direction*
- *specific\_costs*
- *specific\_costs\_om*
- *type\_oemof*
- *unit*
- *beta*

### energyProduction.csv

The file *energyProduction.csv* defines the assets that serve as energy sources, eg. fuel sources or PV plants. They include the following parameters:

- *age\_installed*
- *development\_costs*
- *dispatch\_price*
- *emission\_factor*
- *energyVector*
- *file\_name*
- *installedCap*
- *lifetime*
- *maximumCap*
- *optimizeCap*
- *renewableAsset*
- *specific\_costs*
- *specific\_costs\_om*
- *type\_oemof*

- *unit*

### **energyProviders.csv**

The file *energyProviders.csv* defines the energy providers of the energy system. They include the following parameters:

- *emission\_factor*
- *energy\_price*
- *energyVector*
- *feedin\_tariff*
- *inflow\_direction*
- *optimizeCap*
- *outflow\_direction*
- *peak\_demand\_pricing*
- *peak\_demand\_pricing\_period*
- *renewable\_share*
- *type\_oemof*
- *unit*

### **energyStorage.csv**

The file *energyStorage.csv* defines the storage assets included in the energy system. It does not hold all needed parameters, but requires *storage\_xx.csv* to be defined as well. The file *energyStorage.csv* includes the following parameters:

- *energyVector*
- *file\_name*
- *inflow\_direction*
- *optimizeCap*
- *outflow\_direction*
- *storage\_filename*
- *type\_oemof*

### **fixcost.csv**

The parameters must be filled for all three columns/components namely: *distribution\_grid*, *engineering* and *operation*. The file *fixcost.csv* includes the following parameters:

- *development\_costs*
- *dispatch\_price*
- *label*
- *lifetime*
- *specific\_costs*

- *specific\_costs\_om*

### **project\_data.csv**

The file *project\_data.csv* includes the following parameters:

- *country*
- *latitude*
- *longitude*
- *project\_id*
- *project\_name*
- *scenario\_description*
- *scenario\_id*
- *scenario\_name*

### **simulation\_settings.csv**

The file *simulation\_settings.csv* includes the following parameters:

- *evaluated\_period*
- *output\_lp\_file*
- *start\_date*
- *timestep*

### **storage\_\*.csv**

The \* in the storage filename is the number identifying the storage. It depends on the number of storage components (such as batteries, etc.) present in the system. For e.g., there should be two storage files named *storage\_01.csv* and *storage\_02.csv* if the system contains two storage components. The file *storage\_xx.csv* contains the following parameters:

- *c-rate*
- *development\_costs*
- *dispatch\_price*
- *efficiency*
- *fixed\_thermal\_losses\_absolute*
- *fixed\_thermal\_losses\_relative*
- *installedCap*
- *lifetime*
- *soc\_initial*
- *soc\_max*
- *soc\_min*
- *specific\_costs*

- *specific\_costs\_om*
- *unit*

### 3.5.2 Table of parameters

The input parameters are gathered in the table below. Each parameter is provided with unit, type and example values. For more information about one parameter, please click on it.

Table 7: Parameters summary

Parameter	Type	Unit	Default
<i>age_installed</i>	numeric	Year	0
<i>c-rate</i>	numeric	Factor	1
<i>country</i>	str	None	None
<i>currency</i>	str	None	EUR
<i>development_costs</i>	numeric	currency	0
<i>discount_factor</i>	numeric	Factor	0
<i>dispatch_price</i>	numeric	currency/kWh	0
<i>efficiency</i>	numeric	Factor	1
<i>emission_factor</i>	numeric	kgCO2eq/asset unit	0
<i>energy_price</i>	numeric	currency/energy carrier unit	0
<i>energyVector</i>	str	None	Electricity
<i>evaluated_period</i>	numeric	Day	365
<i>feedin_tariff</i>	numeric	currency/kWh	0
<i>file_name</i>	str	None	None
<i>fixed_thermal_losses_absolute</i>	numeric	factor	0
<i>fixed_thermal_losses_relative</i>	numeric	factor	0
<i>inflow_direction</i>	str	None	None
<i>installedCap</i>	numeric	kWp	0
<i>label</i>	str	None	None
<i>latitude</i>	numeric	None	None
<i>lifetime</i>	numeric	Year	20
<i>longitude</i>	numeric	None	None
<i>maximum_emissions</i>	numeric	kgCO2eq/a	None
<i>maximumCap</i>	numeric	kWp	None
<i>minimal_degree_of_autonomy</i>	numeric	factor	0
<i>minimal_renewable_factor</i>	numeric	factor	0
<i>net_zero_energy</i>	boolean	None	False
<i>optimizeCap</i>	boolean	None	False
<i>outflow_direction</i>	str	None	None
<i>output_lp_file</i>	boolean	None	False
<i>peak_demand_pricing</i>	numeric	currency/kW	0
<i>peak_demand_pricing_period</i>	numeric	times per year (1,2,3,4,6,12)	1
<i>project_duration</i>	numeric	Years	20
<i>project_id</i>	str	None	None
<i>project_name</i>	str	None	None
<i>renewable_share</i>	numeric	Factor	0
<i>renewableAsset</i>	boolean	None	False
<i>scenario_description</i>	str	None	None
<i>scenario_id</i>	str	None	None
<i>scenario_name</i>	str	None	None

continues on next page



Table 7 – continued from previous page

Parameter	Type	Unit	Default
<i>soc_initial</i>	numeric	None or factor	None
<i>soc_max</i>	numeric	Factor	1
<i>soc_min</i>	numeric	Factor	0
<i>specific_costs</i>	numeric	currency/unit	0
<i>specific_costs_om</i>	numeric	currency/unit/year	0
<i>start_date</i>	str	None	None
<i>storage_filename</i>	str	None	None
<i>tax</i>	numeric	Factor	0
<i>timestep</i>	numeric	Minutes	60
<i>type_oemof</i>	str	None	None
<i>unit</i>	str		None
<i>beta</i>	numeric	factor	0

### 3.5.3 List of parameters

Below is the list of all the parameters of MVS, sorted in alphabetical order. Each of the parameters has the following properties

**Definition**

parameter's definition, could also contain potential use cases of the parameter

**Type**

str (text), numeric (integer or double precision number), boolean (True or False)

**Unit**

physical unit

**Example**

an example of parameter's value

**Restrictions**

specific restrictions on the parameter's value (e.g., "positive integer number", "must be an even number", "must be one of ['val1', 'val2']")

**Default**

default parameter's value

#### age\_installed

**Definition**

The number of years the asset has already been in operation.

**Type**

numeric

**Unit**

Year

**Example**

10

**Restrictions**

Natural number

**Default**

0

This parameter is used within the following categories: *energyConversion.csv*, *energyProduction.csv*

**c-rate**

**Definition**

C-rate is the rate at which the storage can charge or discharge relative to the nominal capacity of the storage. A c-rate of 1 implies that the battery can discharge or charge completely in a single timestep.

**Type**

numeric

**Unit**

Factor

**Example**

*storage capacity*: NaN, *input power*: 1, *output power*: 1

**Restrictions**

Real number between 0 and 1. Only the columns “input power” and “output power” require a value, in column “storage capacity” c\_rate should be set to NaN.

**Default**

1

This parameter is used within the following categories: *storage\_\*.csv*

**country**

**Definition**

Name of the country where the project is being deployed

**Type**

str

**Unit**

None

**Example**

Norway

**Restrictions**

None

**Default**

None

This parameter is used within the following categories: *project\_data.csv*

**currency****Definition**

The currency of the country where the project is implemented.

**Type**

str

**Unit**

None

**Example**

EUR

**Restrictions**

None

**Default**

EUR

This parameter is used within the following categories: *economic\_data.csv*

**development\_costs****Definition**

A fixed cost to implement the asset, eg. planning costs which do not depend on the (optimized) asset capacity.

**Type**

numeric

**Unit**

currency

**Example**

10000

**Restrictions**

Positive real number

**Default**

0

This parameter is used within the following categories: *energyConversion.csv*, *storage\_\*.csv*, *energyProduction.csv*, *fixcost.csv*

**discount\_factor****Definition**

Discount factor is the factor which accounts for the depreciation in the value of money in the future, compared to the current value of the same money. The common method is to calculate the weighted average cost of capital (WACC) and use it as the discount rate.

**Type**

numeric

**Unit**

Factor

**Example**

0.06

**Restrictions**

Between 0 and 1

**Default**

0

This parameter is used within the following categories: *economic\_data.csv*

## dispatch\_price

**Definition**

Variable cost associated with a flow through/from the asset (eg. Euro/kWh).

**Type**

numeric

**Unit**

currency/kWh

**Example**

0.64 or “[0.3, 0.26]” for multiple input or output busses

**Restrictions**

In “storage\_xx.csv” only the columns “input power” and “output power” require a value, in column “storage capacity” dispatch\_price should be set to NaN. In conversion assets, there should be different dispatch prices provided for each input or output busses. For two output busses (for example a heat pump), then write “[0.1, 0.4]”

**Default**

0

This parameter is used within the following categories: *energyConversion.csv*, *energyProduction.csv*, *storage\_\*.csv*, *fixcost.csv*

## efficiency

**Definition**

Ratio of energy output/energy input. The battery efficiency is the ratio of the energy taken out from the battery, to the energy put into the battery. It means that it is not possible to retrieve as much energy as provided to the battery due to the discharge losses. The efficiency of the “input power” and “output power” columns should be set equal to the charge and discharge efficiencies respectively, while the “storage capacity” efficiency should be equal to the storage’s efficiency/ability to hold charge over time ( $= 1 - \text{self-discharge/decay}$ ), which is usually in the range of 0.95 to 1 for electrical storages.

**Type**

numeric

**Unit**

Factor

**Example**

0.95 or “[0.91, 0.98]” for multiple input or output busses

**Restrictions**

Between 0 and 1

**Default**

1

This parameter is used within the following categories: *energyConversion.csv*, *storage\_\*.csv*

**emission\_factor****Definition**

Emissions per unit dispatch of an asset.

**Type**

numeric

**Unit**

kgCO<sub>2</sub>eq/asset unit

**Example**

14.4

**Restrictions**

Positive real number

**Default**

0

This parameter is used within the following categories: *energyProviders.csv*, *energyProduction.csv*

**energy\_price****Definition**

Price of energy carrier sourced from the utility grid.

**Type**

numeric

**Unit**

currency/energy carrier unit

**Example**

0.1

**Restrictions**

None

**Default**

0

This parameter is used within the following categories: *energyProviders.csv*

## energyVector

**Definition**

Energy vector/commodity. Convention: For an energy conversion asset define energyVector of the output. For a sink define based on inflow. For a source define based on output flow. For a storage, define based on stored energy carrier.

**Type**

str

**Unit**

None

**Example**

Electricity

**Restrictions**

One of “Electricity”, “Gas”, “Bio-Gas”, “Diesel”, “Heat”, “H2”

**Default**

Electricity

This parameter is used within the following categories: *energyBusses.csv*, *energyConsumption.csv*, *energyProduction.csv*, *energyStorage.csv*, *energyProviders.csv*, *energyConversion.csv*

## evaluated\_period

**Definition**

The number of days simulated with the energy system model.

**Type**

numeric

**Unit**

Day

**Example**

365

**Restrictions**

Natural number

**Default**

365

This parameter is used within the following categories: *simulation\_settings.csv*

## feedin\_tariff

**Definition**

Price received for feeding electricity into the grid.

**Type**

numeric

**Unit**

currency/kWh

**Example**

0.7

**Restrictions**

Real number between 0 and 1

**Default**

0

This parameter is used within the following categories: *energyProviders.csv*

**file\_name****Definition**

Name of a csv file containing the input generation or demand timeseries.

**Type**

str

**Unit**

None

**Example**

demand\_harbor.csv

**Restrictions**

This file must be placed in a folder named “time\_series” inside your input folder.

**Default**

None

This parameter is used within the following categories: *energyConsumption.csv*, *energyProduction.csv*, *energyStorage.csv*

**fixed\_thermal\_losses\_absolute****Definition**

Thermal losses of storage independent of state of charge and independent of nominal storage capacity between two consecutive timesteps.

**Type**

numeric

**Unit**

factor

**Example**

0.0003

**Restrictions**

Between 0 and 1

**Default**

0

This parameter is used within the following categories: *storage\_\*.csv*

### fixed\_thermal\_losses\_relative

**Definition**

Thermal losses of storage independent of state of charge between two consecutive timesteps relative to nominal storage capacity.

**Type**

numeric

**Unit**

factor

**Example**

0.0016

**Restrictions**

Between 0 and 1

**Default**

0

This parameter is used within the following categories: *storage\_\*.csv*

### inflow\_direction

**Definition**

The label of the bus/component from which the energyVector is arriving into the asset.

**Type**

str

**Unit**

None

**Example**

Electricity or “[Electricity, Heat]” for multiple input busses

**Restrictions**

None

**Default**

None

This parameter is used within the following categories: *energyConsumption.csv*, *energyConversion.csv*, *energyProviders.csv*, *energyStorage.csv*

### installedCap

**Definition**

The already existing installed capacity in-place. If the project lasts longer than its remaining lifetime, its replacement costs will be taken into account.

**Type**

numeric

**Unit**

kWp

**Example**

50



**Restrictions**

Each component in the “energy production” category must have a value.

**Default**

0

This parameter is used within the following categories: *energyConversion.csv*, *energyProduction.csv*, *storage\_\*.csv*

**label****Definition**

Name of the asset for display purposes

**Type**

str

**Unit**

None

**Example**

pv\_plant\_01

**Restrictions**

Input the names in a computer friendly format, preferably with underscores instead of spaces, and avoiding special characters

**Default**

None

This parameter is used within the following categories: *fixcost.csv*

**latitude****Definition**

Latitude coordinate of the project’s geographical location.

**Type**

numeric

**Unit**

None

**Example**

45.641603

**Restrictions**

Should follow geographical convention

**Default**

None

This parameter is used within the following categories: *project\_data.csv*

## lifetime

**Definition**

Number of operational years of the asset until it has to be replaced.

**Type**

numeric

**Unit**

Year

**Example**

30

**Restrictions**

Natural number

**Default**

20

This parameter is used within the following categories: *energyConversion.csv*, *energyProduction.csv*, *storage\_\*.csv*, *fixcost.csv*

## longitude

**Definition**

Longitude coordinate of the project's geographical location.

**Type**

numeric

**Unit**

None

**Example**

10.95787

**Restrictions**

Should follow geographical convention

**Default**

None

This parameter is used within the following categories: *project\_data.csv*

## maximum\_emissions

**Definition**

The maximum amount of total emissions in the optimized energy system.

**Type**

numeric

**Unit**

kgCO<sub>2</sub>eq/a

**Example**

100000

**Restrictions**

Acceptable values are either a positive real number or None

**Default**

None

This parameter is used within the following categories: *constraints.csv*

**maximumCap****Definition**

The maximum total capacity of an asset that can be installed at the project site. This includes the installed and the also the maximum additional capacity possible. An example would be that a roof can only carry 50 kWp PV (maximumCap), whereas the installed capacity is already 10 kWp. The optimization would only be allowed to add 40 kWp PV at maximum.

**Type**

numeric

**Unit**

kWp

**Example**

1050

**Restrictions**

Acceptable values are either a positive real number or None

**Default**

None

This parameter is used within the following categories: *energyProduction.csv*

**minimal\_degree\_of\_autonomy****Definition**

The minimal degree of autonomy that needs to be met by the optimization.

**Type**

numeric

**Unit**

factor

**Example**

0.3

**Restrictions**

Between 0 and 1

**Default**

0

This parameter is used within the following categories: *constraints.csv*

### minimal\_renewable\_factor

**Definition**

The minimum share of energy supplied by renewable generation in the optimized energy system.  
Insert the value 0 to deactivate this constraint.

**Type**

numeric

**Unit**

factor

**Example**

0.7

**Restrictions**

Between 0 and 1

**Default**

0

This parameter is used within the following categories: [constraints.csv](#)

### net\_zero\_energy

**Definition**

Specifies whether optimization needs to result into a net zero energy system (True) or not (False).

**Type**

boolean

**Unit**

None

**Example**

True

**Restrictions**

Acceptable values are either True or False.

**Default**

False

This parameter is used within the following categories: [constraints.csv](#)

### optimizeCap

**Definition**

Allow the user to perform capacity optimization for an asset.

**Type**

boolean

**Unit**

None

**Example**

True

**Restrictions**

Permissible values are either True or False

**Default**

False

This parameter is used within the following categories: *energyConversion.csv*, *energyProduction.csv*, *energyProviders.csv*, *energyStorage.csv*

**outflow\_direction****Definition**

The label of bus/component towards which the energyVector is leaving from the asset.

**Type**

str

**Unit**

None

**Example**

Electricity or “[Electricity, Heat]” for multiple output busses

**Restrictions**

None

**Default**

None

This parameter is used within the following categories: *energyConsumption.csv*, *energyConversion.csv*, *energyProviders.csv*, *energyStorage.csv*

**output\_lp\_file****Definition**

Enable the generation of a file with the linear equation system describing the simulation, ie., with the objective function and all the constraints. This lp file enables the user look at the underlying equations of the optimization.

**Type**

boolean

**Unit**

None

**Example**

False

**Restrictions**

Acceptable values are either True or False

**Default**

False

This parameter is used within the following categories: *simulation\_settings.csv*

### peak\_demand\_pricing

**Definition**

Price to be paid additionally for energy-consumption based on the peak demand of a given period.

**Type**

numeric

**Unit**

currency/kW

**Example**

60

**Restrictions**

None

**Default**

0

This parameter is used within the following categories: *energyProviders.csv*

See also: *peak\_demand\_pricing\_period*

### peak\_demand\_pricing\_period

**Definition**

Number of reference periods in one year for the peak demand pricing.

**Type**

numeric

**Unit**

times per year (1,2,3,4,6,12)

**Example**

2

**Restrictions**

Only one of the following are acceptable values: 1 (yearly), 2, 3 ,4, 6, 12 (monthly)

**Default**

1

This parameter is used within the following categories: *energyProviders.csv*

See also: *peak\_demand\_pricing*

### project\_duration

**Definition**

The number of years the project is intended to be operational. The project duration also sets the installation time of the assets used in the simulation. After the project ends these assets are 'sold' and the refund is charged against the initial investment costs.

**Type**

numeric

**Unit**

Years

**Example**

30

**Restrictions**

Natural number

**Default**

20

This parameter is used within the following categories: *economic\_data.csv*

**project\_id****Definition**

Users can assign a project ID as per their preference.

**Type**

str

**Unit**

None

**Example**

1

**Restrictions**

Cannot be the same as an already existing project

**Default**

None

This parameter is used within the following categories: *project\_data.csv*

**project\_name****Definition**

Users can assign a project name as per their preference.

**Type**

str

**Unit**

None

**Example**

Borg Havn

**Restrictions**

None

**Default**

None

This parameter is used within the following categories: *project\_data.csv*

## renewable\_share

**Definition**

The share of renewables in the generation mix of the energy supplied by the DSO (utility).

**Type**

numeric

**Unit**

Factor

**Example**

0.1

**Restrictions**

Real number between 0 and 1

**Default**

0

This parameter is used within the following categories: *energyProviders.csv*

## renewableAsset

**Definition**

Allow the user to tag as asset as renewable.

**Type**

boolean

**Unit**

None

**Example**

True

**Restrictions**

Acceptable values are either True or False

**Default**

False

This parameter is used within the following categories: *energyProduction.csv*

## scenario\_description

**Definition**

Brief description of the scenario being simulated.

**Type**

str

**Unit**

None

**Example**

This scenario simulates a sector-coupled energy system

**Restrictions**

None



**Default**

None

This parameter is used within the following categories: *project\_data.csv*

**scenario\_id****Definition**

Users can assign a scenario id as per their preference.

**Type**

str

**Unit**

None

**Example**

1

**Restrictions**

Cannot be the same as an already existing scenario within the project

**Default**

None

This parameter is used within the following categories: *project\_data.csv*

**scenario\_name****Definition**

Users can assign a scenario name as per their preference.

**Type**

str

**Unit**

None

**Example**

Warehouse 14

**Restrictions**

None

**Default**

None

This parameter is used within the following categories: *project\_data.csv*

**soc\_initial****Definition**

The level of charge (as a factor of the actual capacity) in the storage in the initial (0) time-step.

**Type**

numeric

**Unit**

None or factor

**Example**

storage capacity: None, input power: NaN, output power: NaN

**Restrictions**

Acceptable values are either None or the factor. Only the column `storage capacity` requires a value, in column `input power` and `output power` `soc_initial` should be set to NaN. The `soc_initial` has to be within the [0,1] interval.

**Default**

None

This parameter is used within the following categories: *storage\_\*.csv*

**soc\_max****Definition**

The maximum permissible level of charge in the battery (generally, it is when the battery is filled to its nominal capacity), represented by the value 1.0. Users can also specify a certain value as a factor of the actual capacity.

**Type**

numeric

**Unit**

Factor

**Example**

storage capacity: 1, input power: NaN, output power: NaN

**Restrictions**

Only the column `storage capacity` requires a value, in column `input power` and `output power` `soc_max` should be set to NaN. The `soc_max` has to be in the [0,1] interval.

**Default**

1

This parameter is used within the following categories: *storage\_\*.csv*

**soc\_min****Definition**

The minimum permissible level of charge in the battery as a factor of the nominal capacity of the battery.

**Type**

numeric

**Unit**

Factor

**Example**

storage capacity:0.2, input power: NaN, output power: NaN

**Restrictions**

Only the column `storage capacity` requires a value, in column `input power` and `output power` `soc_min` should be set to NaN. The `soc_min` has to be in the [0,1] interval.

**Default**

0

This parameter is used within the following categories: *storage\_\*.csv*

**specific\_costs****Definition**

Actual CAPEX of an asset, i.e., specific investment costs

**Type**

numeric

**Unit**

currency/unit

**Example**

4000

**Restrictions**

None

**Default**

0

This parameter is used within the following categories: *energyConversion.csv*, *energyProduction.csv*, *storage\_\*.csv*, *fixcost.csv*

**specific\_costs\_om****Definition**

Actual OPEX of an asset, i.e., specific operational and maintenance costs.

**Type**

numeric

**Unit**

currency/unit/year

**Example**

120

**Restrictions**

None

**Default**

0

This parameter is used within the following categories: *energyConversion.csv*, *energyProduction.csv*, *storage\_\*.csv*, *fixcost.csv*

**start\_date**

**Definition**

The date and time on which the simulation starts at the first step.

**Type**

str

**Unit**

None

**Example**

2018-01-01 00:00:00

**Restrictions**

Acceptable format is YYYY-MM-DD HH:MM:SS

**Default**

None

This parameter is used within the following categories: *simulation\_settings.csv*

**storage\_filename**

**Definition**

Name of a csv file containing the properties of a storage component

**Type**

str

**Unit**

None

**Example**

storage\_01.csv

**Restrictions**

Follows the convention of 'storage\_xx.csv' where 'xx' is a number. This file must be placed in a folder named "csv\_elements" inside your input folder.

**Default**

None

This parameter is used within the following categories: *energyStorage.csv*

**tax****Definition**

Tax factor.

**Type**

numeric

**Unit**

Factor

**Example**

0

**Restrictions**

Between 0 and 1

**Default**

0

This parameter is used within the following categories: *economic\_data.csv*

**timestep****Definition**

Length of the time-steps.

**Type**

numeric

**Unit**

Minutes

**Example**

60

**Restrictions**

Can only be 60 at the moment

**Default**

60

This parameter is used within the following categories: *simulation\_settings.csv*

**type\_oemof****Definition**

Input the type of OEMOF component. For example, a PV plant would be a source, a solar inverter would be a transformer, etc. The *type\_oemof* will later on be determined through the EPA.

**Type**

str

**Unit**

None

**Example**

sink

**Restrictions**

*sink* or *source* or one of the other component classes of OEMOF.

**Default**

None

This parameter is used within the following categories: *energyConsumption.csv*, *energyConversion.csv*, *energyProduction.csv*, *energyProviders.csv*, *energyStorage.csv*

**unit**

**Definition**

Unit associated with the capacity of the component.

**Type**

str

**Unit**

nan

**Example**

Storage could have units like kW or kWh, transformer station could have kVA, and so on.

**Restrictions**

Appropriate scientific unit

**Default**

None

This parameter is used within the following categories: *energyConsumption.csv*, *energyConversion.csv*, *energyProduction.csv*, *energyProviders.csv*, *storage\_\*.csv*

**beta**

**Definition**

Power loss index for CHPs

**Type**

numeric

**Unit**

factor

**Example**

0.6

**Restrictions**

Between 0 and 1

**Default**

0

This parameter is used within the following categories: *energyConversion.csv*

## 3.6 Outputs of a simulation

After optimization of an energy system, the MVS evaluates the simulation output. It evaluates the flows, costs and performance of the system. As a result, it calculates a number of *key performance indicators (KPI)*, namely *economic*, *technical* and *environmental* KPI. Depending on the simulation settings, it can also generate different *output files and figures* of the results, including an *automatic report* in pdf or html format.

### 3.6.1 Overview of Key Performance Indicators

Technical KPI are calculated to assess the performance of a simulated energy system, ie. represent the technical system configuration and operation. They are calculated based on the asset capacities and asset dispatch. They should allow the comparison of different energy system topologies and different project sites with each other. These are the calculated technical KPI:

- *Aggregated flow*
- *Average flow*
- *Degree of Autonomy*
- *Degree of Net Zero Energy*
- *Dispatch of an asset*
- *Onsite energy fraction*
- *Onsite energy matching*
- *Optimal additional capacity*
- *Peak flow*
- *Renewable factor*
- *Renewable share of local generation*
- *Energy import*
- *Energy demand*
- *Energy excess*
- *Energy export*
- *Total local generation*
- *Total non-renewable local generation*
- *Total renewable local generation*
- *Total non-renewable energy use*
- *Total renewable energy use*

Economic KPI are calculated to assess the costs of a simulated energy system. They include the costs per asset as well as the system's overall costs. Relative values like the levelized costs of supply allow a comparison to other investment options. These are the calculated economic KPI:

- *Annual operation, maintenance and dispatch expenses*
- *Annuity*
- *Costs attributed to a specific sector*
- *Operation and maintenance costs*

- *Dispatch costs*
- *Investment costs*
- *Operation, maintenance and dispatch costs*
- *Net Present Costs (NPC)*
- *Upfront investment costs*
- *Levelized cost of throughput*
- *Levelized costs of electricity equivalent*
- *Replacement costs*

Environmental KPI are calculated to assess the impact of a simulated energy system on the environment. These are the calculated environmental KPI:

- *Specific GHG per electricity equivalent*
- *Total GHG emissions*

Additionally to the KPI, the MVS can also generate a number of output files, which can be shared with other partners and used to visualize the system's behaviour and performance. These are the calculated files KPI:

- *Simulation report*
- *Bar chart of optimal capacities*
- *Excel file with all KPI*
- *Excel file with dispatch timeseries*
- *Simulation data after pre-processing (JSON)*
- *Simulation data and results (JSON)*
- *MVS logfile*
- *Energy system model visualization*
- *Pie charts of cost parameters*
- *Dispatch of all assets on a bus*
- *Input timeseries*

In the sections *economic*, *technical* and *environmental* KPI, these indicators are further defined and in *Files* the possible exportable figures and files are presented. This takes place with the following structure:

**Definition**

Definition of the defined KPI, can be used as tooltips.

**Type**

One of Numeric, Figure, Excel File, JSON, Time series, Logfile or html/pdf

**Unit**

Unit of the KPI, multiple units possible if KPI can be applied to individual sectors (see also: *Suffixes of KPI*).

**Valid Interval**

Expected valid range of the KPI. Exceptions are possible under certain conditions.

**Related indicators**

List of indicators that are related to the described KPI, either because they are part of its calculation or can be compared to it.



Besides these parameters attributes, the underlying equation of a specific KPI may be presented and explained, or further hints might be provided for the parameter evaluation or for special cases.

### 3.6.2 Suffixes of KPI

The KPI of the MVS can be calculated per asset, for each sector or for the overall system.

KPI calculated per asset are not included in the scalar results of the automatic report or in the stored Excel file, but are displayed separately. They do not need suffixes, as they are always displayed in tables next to the respective asset.

KPI calculated for each vector are specifically these KPI that aggregate the dispatch and costs of multiple assets. For cost-related KPI, such aggregating KPI have the energy vector they are describing as a suffix. An example would be the `attributed_costs` of each energy vector - the attributed costs of the electricity and H2 sector would be called `attributed_costs_electricity` and `attributed_costs_H2` respectively. For technical KPI, this suffix also applies, but additionally, due to the *energy carrier weighting*, they also feature the suffix `electricity` equivalent when the weighting has been applied. The energy demand of the system is an example: the demand per sector would be `total_demand_electricity` and `total_demand_H2`. To be able to aggregate these cost into an overall KPI for the system, the electricity equivalents of both values are calculated. They then are named `total_demand_electricity_electricity_equivalent` and `total_demand_H2_electricity_equivalent`.

KPI that describe the costs of the overall energy system do not have suffixes. Technical KPI often have the suffix `electricity_equivalent` to underline the energy carrier that the parameter is relative to.

### 3.6.3 Economic KPI

All the KPI related to costs described below are provided in net present value.

#### Net Present Costs (NPC) (`costs_total`)

##### Definition

Net present costs of the system for the whole project duration, includes all operation, maintenance and dispatch costs as well as the investment costs (including replacements). Applied to a single asset, the costs can also be called present costs of the asset.

##### Type

Numeric

##### Unit

currency

##### Valid Interval

$\geq 0$

##### Related indicators

*Operation and maintenance costs (`costs_cost_om`) | Dispatch costs (`costs_dispatch`) | Investment costs (`costs_investment_over_lifetime`) | Operation, maintenance and dispatch costs (`costs_om_total`) | Up-front investment costs (`costs_upfront_in_year_zero`)*

The Net present costs (NPC) is the present value of all the costs associated with installation, operation, maintenance and replacement of energy assets within the optimized multi-vector energy system over the whole project lifetime, deducting the present value of the residual value of asset at project end and as well as all the revenues that it earns over the project lifetime. The capital recovery factor (CRF) is used to calculate the present value of the cash flows.

$$NPC = \sum_i (c_{specific} + c_{replacement} + c_{residual}) \cdot CAP_i + \sum_i \sum_t E_i(t) \cdot p_{dispatch}$$

**Operation and maintenance costs (costs\_cost\_om)****Definition**

Costs for fix annual operation and maintenance costs over the whole project lifetime, which do not depend on the assets dispatch but solely on installed capacity. An example would be the maintenance costs for cleaning the installed PV capacity.

**Type**

Numeric

**Unit**

currency

**Valid Interval**

$\geq 0$

**Related indicators**

*Dispatch costs (costs\_dispatch) | Investment costs (costs\_investment\_over\_lifetime) | Operation, maintenance and dispatch costs (costs\_om\_total) | Net Present Costs (NPC) (costs\_total) | Upfront investment costs (costs\_upfront\_in\_year\_zero)*

**Operation, maintenance and dispatch costs (costs\_om\_total)****Definition**

Costs for annual operation and maintenance costs as well as dispatch of all assets of the energy system, for the whole project duration.

**Type**

Numeric

**Unit**

currency

**Valid Interval**

$\geq 0$

**Related indicators**

*Operation and maintenance costs (costs\_cost\_om) | Dispatch costs (costs\_dispatch) | Investment costs (costs\_investment\_over\_lifetime) | Net Present Costs (NPC) (costs\_total) | Upfront investment costs (costs\_upfront\_in\_year\_zero)*

**Dispatch costs (costs\_dispatch)****Definition**

Dispatch costs over the whole project lifetime including all expenditures that depend on the dispatch of assets (e.g. fuel costs, electricity consumption from the external grid, costs for operation and maintainance that depend on the throughput of an asset)

**Type**

Numeric

**Unit**

currency

**Valid Interval**

$\geq 0$

**Related indicators**

*Operation and maintenance costs (costs\_cost\_om) | Investment costs (costs\_investment\_over\_lifetime) | Operation, maintenance and dispatch costs (costs\_om\_total) | Net Present Costs (NPC) (costs\_total) | Upfront investment costs (costs\_upfront\_in\_year\_zero)*

**Investment costs (costs\_investment\_over\_lifetime)****Definition**

Investment costs over the whole project lifetime, including all replacement costs.

**Type**

Numeric

**Unit**

currency

**Valid Interval**

$\geq 0$

**Related indicators**

*Operation and maintenance costs (costs\_cost\_om) | Dispatch costs (costs\_dispatch) | Operation, maintenance and dispatch costs (costs\_om\_total) | Net Present Costs (NPC) (costs\_total) | Upfront investment costs (costs\_upfront\_in\_year\_zero) | Replacement costs (replacement\_costs\_during\_project\_lifetime)*

**Upfront investment costs (costs\_upfront\_in\_year\_zero)****Definition**

The costs which will have to be paid upfront when project begins, ie. In year 0. These are the investment and fix project costs into the chosen configuration.

**Type**

Numeric

**Unit**

currency

**Valid Interval**

$\geq 0$

**Related indicators**

*Operation and maintenance costs (costs\_cost\_om) | Dispatch costs (costs\_dispatch) | Investment costs (costs\_investment\_over\_lifetime) | Operation, maintenance and dispatch costs (costs\_om\_total) | Net Present Costs (NPC) (costs\_total)*

**Replacement costs (replacement\_costs\_during\_project\_lifetime)****Definition**

Costs for replacement of assets which occur over the project lifetime.

**Type**

Numeric

**Unit**

currency

**Valid Interval**

$\geq 0$

**Related indicators**

*Investment costs (costs\_investment\_over\_lifetime)*

**Costs attributed to a specific sector (attributed\_costs)****Definition**

Costs attributed to supplying the demand of a specific sector, based on the *net present costs (NPC)* of the energy system and the share of the sector demand compared to the overall system demand.

**Type**

Numeric

**Unit**

currency

**Valid Interval**

$\geq 0$

**Related indicators**

*Net Present Costs (NPC) (costs\_total)*

A multi-vector energy system connects energy vectors into a joined energy system and the system is then designed to have an optimal, joined operation. With other systems, the costs associated to each individual energy vector would be used to calculate the costs to supply the individual sector. With the multi-vector system, this could lead to distorted costs - for example if there is a lot of PV (electricity sector), which in the end is only supplying an electrolyzer (H2 sector). The investment and operational costs of the electricity sector assets would thus turn out to be very high, which could be considered unfair as the electricity from PV is solely used to provide the H2 demand. Therefore, we define the *attributed costs of each energy vector*, to determine how much of the overall system costs should be attributed to one sector, depending on the energy demand it has compared to the other sectors. To be able to compare the demands of different energy carriers, *energy carrier weighting* is applied.

**Annuity (annuity\_total)****Definition**

Annuity of the assets costs over the project lifetime or the energy system's *net present costs (NPC)*.

**Type**

Numeric

**Unit**

currency/a

**Valid Interval**

$\geq 0$

**Related indicators**

*Annual operation, maintenance and dispatch expenses (annuity\_om) | Net Present Costs (NPC) (costs\_total)*

## Annual operation, maintenance and dispatch expenses (annuity\_om)

### Definition

Annuity of the operation, maintenance and dispatch costs of the asset or energy system, i.e. ballpark number of the annual expenses for asset or system operation.

### Type

Numeric

### Unit

currency/a

### Valid Interval

$\geq 0$

### Related indicators

*Annuity (annuity\_total) | Operation, maintenance and dispatch costs (costs\_om\_total)*

## Levelized costs of electricity equivalent (levelized\_costs\_of\_electricity\_equivalent)

### Definition

Levelized cost of energy of the sector-coupled energy system, calculated from the systems annuity and the total system demand in electricity equivalent.

### Type

Numeric

### Unit

currency/kWheq

### Valid Interval

$\geq 0$

### Related indicators

*Net Present Costs (NPC) (costs\_total) | Energy demand (total\_demand)*

Specific electricity supply costs, eg. levelized costs of electricity are commonly used to compare the supply costs of different investment decisions or also energy provider prices to local generation costs. However, the a multi-vector energy system connects energy vectors into a joined energy system and the optimization objective of the MVS then is to minimize the overall energy costs, without distinguishing between the different sectors. This sector-coupled energy system is then designed to have an optimal, joined operation. With other systems, the costs associated to each individual energy vector would be used to calculate the levelized costs of energy (LCOEnergy). With the multi-vector system, this could lead to distorted costs - for example if there is a lot of PV (electricity sector), which in the end is only supplying an electrolyzer (H2 sector). The LCOE of electricity would thus turn out to be very high, which could be considered unfair as the electricity from PV is solely used to provide the H2 demand. Therefore, we define the *attributed costs of each energy vector*, to determine how much of the overall system costs should be attributed to one sector, depending on the energy demand it has compared to the other sectors. To be able to compare the demands of different energy carriers, *energy carrier weighting* is applied.

Therefore the levelized costs of energy (LCOEnergy) for energy carrier  $i$  are defined based on the annuity of the attributed costs, the CRF and the demand of one energy sector  $E_{dem,i}$ :

$$LCOEnergy_i = \frac{\text{Attributed costs} \cdot CRF}{\sum_t E_{dem,i}(t)}$$

The LCOEnergy are calculated for each sector (resulting in the levelized costs of electricity, heat, H2...), but also for the overall energy system. For the overall energy system, the levelized costs of electricity equivalent are calculated, as this system may supply different energy vectors.

**Levelized cost of throughput (levelized\_cost\_of\_energy\_of\_asset)****Definition**

Cost per kWh throughput through an asset, based on the assets costs during the project lifetime as well as the total throughput through the asset in the project lifetime. For generation assets, equivalent to the levelized cost of generation.

**Type**

Numeric

**Unit**

currency/kWh

**Valid Interval**

$\geq 0$

**Related indicators**

*Annuity (annuity\_total) | Aggregated flow (annual\_total\_flow)*

This KPI measures the cost of generating 1 kWh for each asset in the system. It can be used to assess and compare the available alternative methods of energy production. The levelized cost of energy of an asset ( $LCOE\ ASSET_i$ ) is usually obtained by looking at the lifetime costs of building and operating the asset per unit of total energy throughput of an asset over the assumed lifetime [currency/kWh].

Since not all assets are production assets, the MVS distinguishes between the type of assets. For assets in `energyConversion` and `energyProduction` the MVS calculates the  $LCOE\ ASSET_i$  by dividing the total annuity  $a_i$  of the asset  $i$  by the total flow  $\sum_t E_i(t)$ .

$$LCOE\ ASSET_i = \frac{a_i}{\sum_t E_i(t)}$$

For assets in `energyStorage`, the MVS sums the annuity for `storage capacity`  $a_{i,sc}$ , `input power`  $a_{i,ip}$  and `output power`  $a_{i,op}$  and divides it by the output power total flow  $\sum_t E_{i,op}(t)$ .

$$LCOE\ ASSET_i = \frac{a_{i,sc} + a_{i,ip} + a_{i,op}}{\sum_t E_{i,op}(t)}$$

If the total flow is 0 in any of the previous cases, then the  $LCOE\ ASSET$  is set to `None`.

$$LCOE\ ASSET_i = None$$

For assets in `energyConsumption`, the MVS outputs 0 for the  $LCOE\ ASSET_i$ .

$$LCOE\ ASSET_i = 0$$

**3.6.4 Technical KPI****Optimal additional capacity (optimizedAddCap)****Definition**

Capacity added to installed capacity for optimal economic system performance.

**Type**

Numeric

**Unit**

kW, kWh, kWp, ...

**Valid Interval**

$\geq 0$

**Related indicators***Peak flow (peak\_flow)***Dispatch of an asset (flow)****Definition**

Optimized dispatch of an asset in the optimized energy system, ie. its generation or throughput.

**Type**

Time series (with time stamps and values)

**Unit**

kW,kgH2,...

**Valid Interval**

NA

**Related indicators**

*Dispatch of all assets on a bus | Peak flow (peak\_flow) | Aggregated flow (annual\_total\_flow) | Average flow (average\_flow)*

**Peak flow (peak\_flow)****Definition**

Peak of the dispatch of an asset.

**Type**

Numeric

**Unit**

kW

**Valid Interval**

$\geq 0$

**Related indicators**

*Average flow (average\_flow) | Aggregated flow (annual\_total\_flow)*

**Average flow (average\_flow)****Definition**

Average value of the assets dispatch. The ratio of average dispatch to peak dispatch indicates how much the asset is used in comparison to its actual installed capacity.

**Type**

Numeric

**Unit**

kWh

**Valid Interval**

$\geq 0$

**Related indicators**

*Aggregated flow (annual\_total\_flow) | Peak flow (peak\_flow)*

### Aggregated flow (annual\_total\_flow)

**Definition**

Dispatch of the asset over a year, aggregated generation, demand or throughput.

**Type**

Numeric

**Unit**

kWh

**Valid Interval**

$\geq 0$

**Related indicators**

*Average flow (average\_flow) | Peak flow (peak\_flow)*

### Energy demand (total\_demand)

**Definition**

Demand of energy in local energy system over a the project lifetime.

**Type**

Numeric

**Unit**

kWh, kWheq, ...

**Valid Interval**

$\geq 0$

**Related indicators**

None

### Energy export (total\_feedin)

**Definition**

Feed-in of energy into external grid.

**Type**

Numeric

**Unit**

kWh, kWheq, ...

**Valid Interval**

$\geq 0$

**Related indicators**

*Onsite energy fraction (onsite\_energy\_fraction)*



**Energy import (total\_consumption\_from\_energy\_provider)****Definition**

Aggregated energy imports into the local energy system from the provider.

**Type**

Numeric

**Unit**

kWh, kWheq, ...

**Valid Interval**

$\geq 0$

**Related indicators**

None

**Total non-renewable local generation (total\_internal\_non-renewable\_generation)****Definition**

Aggregated amount of non-renewable energy generated within the energy system.

**Type**

Numeric

**Unit**

kWheq

**Valid Interval**

$\geq 0$

**Related indicators**

*Total local generation (total\_internal\_generation) | Total renewable local generation (total\_internal\_renewable\_generation)*

**Total renewable local generation (total\_internal\_renewable\_generation)****Definition**

Aggregated amount of renewable energy generated within the energy system.

**Type**

Numeric

**Unit**

kWheq

**Valid Interval**

$\geq 0$

**Related indicators**

*Total local generation (total\_internal\_generation) | Total non-renewable local generation (total\_internal\_non-renewable\_generation)*

### Total local generation (total\_internal\_generation)

**Definition**

Aggregated amount of energy generated within the energy system.

**Type**

Numeric

**Unit**

kWheq

**Valid Interval**

$\geq 0$

**Related indicators**

*Total non-renewable local generation (total\_internal\_non-renewable\_generation) | Total renewable local generation (total\_internal\_renewable\_generation)*

### Energy excess (total\_excess)

**Definition**

Excess of energy, ie. unused energy in local energy system.

**Type**

Numeric

**Unit**

kWh, kWheq, ...

**Valid Interval**

$\geq 0$

**Related indicators**

None

### Total renewable energy use (total\_renewable\_energy\_use)

**Definition**

Aggregated amount of renewable energy used within the energy system (ie. Including local generation and external supply).

**Type**

Numeric

**Unit**

kWheq

**Valid Interval**

$\geq 0$

**Related indicators**

*Total non-renewable energy use (total\_non-renewable\_energy\_use)*

**Total non-renewable energy use (total\_non-renewable\_energy\_use)****Definition**

Aggregated amount of non-renewable energy used within the energy system (ie. Including local generation and external supply).

**Type**

Numeric

**Unit**

kWheleq

**Valid Interval**

$\geq 0$

**Related indicators**

*Total renewable energy use (total\_renewable\_energy\_use)*

**Renewable share of local generation (renewable\_share\_of\_local\_generation)****Definition**

The renewable share of local generation describes how much of the energy generated locally is produced from renewable sources. It does not take into account the consumption from energy providers.

**Type**

Numeric

**Unit**

Factor

**Valid Interval**

[0,1]

**Related indicators**

*Renewable factor (renewable\_factor)*

The renewable share of local generation describes how much of the energy generated locally is produced from renewable sources. It does not take into account the consumption from energy providers.

The renewable share of local generation for each sector does not utilize energy carrier weighting but has a limited, single-vector view:

$$REGen_v = \frac{\sum_i E_{rgen,i}}{\sum_j E_{gen,j}}$$

with  $v$ : Energy vector

$rgen$ : Renewable generation

$gen$ : Renewable and non-renewable generation

$i, j$ : Asset 1,2,...

For the system-wide share of local renewable generation, energy carrier weighting is used:

$$REGen = \frac{\sum_i E_{rgen,i} \cdot w_i}{\sum_j E_{gen,j} \cdot w_j}$$

with  $rgen$ : Renewable generation

$gen$ : Renewable and non-renewable generation

$i, j$ : Assets 1,2,...

$w_i, w_j$ : Energy carrier weighting factor for output of asset  $i/j$

**Example**

An energy system is composed of a heat and an electricity side. Following are the energy flows:

- 100 kWh from a local PV plant
- 0 kWh local generation for the heat side

This results in:

- A single-vector renewable share of local generation of 0% for the heat sector.
- A single-vector renewable share of local generation of 100% for the electricity sector.
- A system-wide renewable share of local generation of 100%.

**Renewable factor (renewable\_factor)****Definition**

Describes the share of the energy influx to the local energy system that is provided from renewable sources. This includes both local generation as well as consumption from energy providers.

**Type**

Numeric

**Unit**

Factor

**Valid Interval**

[0,1]

**Related indicators**

*Renewable share of local generation (renewable\_share\_of\_local\_generation) | Onsite energy fraction (onsite\_energy\_fraction) | Onsite energy matching (onsite\_energy\_matching)*

Describes the share of the energy influx to the local energy system that is provided from renewable sources. This includes both local generation as well as consumption from energy providers.

$$RF = \frac{\sum_i E_{rgen,i} \cdot w_i + RES \cdot E_{grid}}{\sum_j E_{gen,j} \cdot w_j + \sum_k E_{grid}(k) \cdot w_k}$$

with *rgen*: Renewable generation

*gen*: Renewable and non-renewable generation

*i, j*: Assets 1,2,...

*RES*: Renewable energy share of energy provider

*k*: Energy provider 1,2,...

*w<sub>i</sub>, w<sub>j</sub>, w<sub>k</sub>*: Energy carrier weighting factor for output of asset i/j/k

**Example**

An energy system is composed of a heat and an electricity side. Following are the energy flows:

- 100 kWh from a local PV plant
- 0 kWh local generation for the heat side
- 100 kWh consumption from the electricity provider, who has a renewable factor of 50%

Again, the heat sector would have a renewable factor of 0% when considered separately, and the electricity side would have an renewable factor of 75%. This results in a system-wide renewable share of:

$$RF = \frac{100kWh(el) \cdot \frac{kWh(elec)}{kWh(el)} + 50kWh(el) \cdot \frac{kWh(elec)}{kWh(el)}}{200kWh(el) \cdot \frac{kWh(elec)}{kWh(el)}} = 3/4 = 75 \%$$

The renewable factor, just like the *Renewable share of local generation (renewable\_share\_of\_local\_generation)*, cannot indicate how much renewable energy is used in each of the sectors. In the future, it might be possible to get a clearer picture of the flows between the sectors with the proposed *degree of sector-coupling*.

### Degree of sector-coupling (DSC)

To assess how much an optimized multi-vector energy system makes use of the potential of sector-coupling, it is planned to introduce the degree of sector-coupling in the future. This level of interconnection is to be calculated with the ratio of energy flows in between the sectors (ie. those, where energy carriers are converted to another energy carrier) to the energy demand supplied:

$$DSC = \frac{\sum_{i,j} E_{conversion}(i,j) \cdot w_i}{\sum_i E_{demand}(i) \cdot w_i}$$

with  $i, j$ : Electricity, H2...

---

**Note:** This feature is currently not implemented.

---

### Onsite energy fraction (onsite\_energy\_fraction)

#### Definition

Onsite energy fraction is also referred to as self-consumption. It describes the fraction of all locally generated energy that is consumed by the system itself.

#### Type

Numeric

#### Unit

Factor

#### Valid Interval

[0,1]

#### Related indicators

*Onsite energy matching (onsite\_energy\_matching)*

Onsite energy fraction is also referred to as “self-consumption”. It describes the fraction of all locally generated energy that is consumed by the system itself. (see [1] and [2]).

An OEF close to zero shows that only a very small amount of locally generated energy is consumed by the system itself. It is at the same time an indicator that a large amount is fed into the grid instead. A OEF close to one shows that almost all locally produced energy is consumed by the system itself.

$$OEF = \frac{\sum_i (E_{generation}(i) - E_{gridfeedin}(i)) \cdot w_i}{\sum_i E_{generation}(i) \cdot w_i}$$

$OEF \in [0,1]$

**Onsite energy matching (onsite\_energy\_matching)****Definition**

The onsite energy matching is also referred to as self-sufficiency. It describes the fraction of the total demand that can be covered by the locally generated energy.

**Type**

Numeric

**Unit**

Factor

**Valid Interval**

[0,1]

**Related indicators**

*Onsite energy fraction (onsite\_energy\_fraction) | Energy export (total\_feedin)*

The onsite energy matching is also referred to as “self-sufficiency”. It describes the fraction of the total demand that can be covered by the locally generated energy (see [1] and [2]).

An OEM close to zero shows that very little of the demand can be covered by locally produced energy. An OEM close to one shows that almost all of the demand can be covered with locally generated energy. Per definition OEM cannot be greater than 1 because the excess generated energy would automatically be fed into the grid or an excess sink.

$$OEM = \frac{\sum_i (E_{generation}(i) - E_{gridfeedin}(i) - E_{excess}(i)) \cdot w_i}{\sum_i E_{demand}(i) \cdot w_i}$$

$OEM \in [0,1]$

---

**Note:** The feed into the grid should only be positive.

---

**Degree of Autonomy (degree\_of\_autonomy)****Definition**

A degree of autonomy close to zero shows high dependence on the energy provider, while a degree of autonomy of 1 represents an autonomous or net-energy system and a degree of autonomy higher 1 a surplus-energy system.

**Type**

Numeric

**Unit**

Factor

**Valid Interval**

[0,1]

**Related indicators**

*Energy demand (total\_demand)*

The degree of autonomy describes the overall energy consumed minus the energy consumed from the grid divided by the overall energy consumed. Adapted from this definition [3].

A degree of autonomy close to zero shows high dependence on the grid operator, while a degree of autonomy of one represents an autonomous system. Note that this key parameter indicator does not take into account the outflow from the system to the grid operator (also called feedin). As above, we apply a weighting based on Electricity Equivalent.

$$DegreeofAutonomy = \frac{\sum_i E_{demand,i} \cdot w_i - \sum_j E_{consumption,provider,j} \cdot w_j}{\sum_i E_{demand,i} \cdot w_i}$$

## Degree of Net Zero Energy (degree\_of\_nze)

### Definition

The degree of net zero energy describes the ability of an energy system to provide its aggregated annual demand through local sources.

### Type

Numeric

### Unit

Factor

### Valid Interval

$\geq 0$

### Related indicators

*Energy export (total\_feedin) | Energy import (total\_consumption\_from\_energy\_provider)*

The degree of net zero energy describes the ability of an energy system to provide its aggregated annual demand through local sources. For that, the balance between local generation as well as consumption from and feed-in towards the energy provider is compared. In a net zero energy system, demand can be supplied by energy import, but then local energy generation must provide an equally high energy export of energy in the course of the year. In a plus energy system, the export exceeds the import, while local generation can supply all demand (from an aggregated perspective). To calculate the degree of NZE, the margin between grid feed-in and grid consumption is compared to the overall demand.

Some definitions of NZE systems require that the local demand is solely covered by locally generated renewable energy. In MVS this is not the case - all locally generated energy is taken into consideration. For information about the share of renewables in the local energy system checkout *Renewable share of local generation (renewable\_share\_of\_local\_generation)*.

A degree of NZE lower than 1 shows that the energy system can not reach a net zero balance, and indicates by how much it fails to do so, while a degree of NZE of 1 represents a net zero energy system and a degree of NZE higher 1 a plus-energy system.

As above, we apply a weighting based on Electricity Equivalent.

$$\text{Degree of NZE} = 1 + \frac{\sum_i (E_{\text{gridfeedin}}(i) - E_{\text{gridconsumption}}(i)) \cdot w_i}{\sum_i E_{\text{demand},i} \cdot w_i}$$

## 3.6.5 Environmental KPI

### Total GHG emissions (total\_emissions)

#### Definition

Total greenhouse gas emissions in kg.

#### Type

Numeric

#### Unit

kg GHG<sub>eq</sub>

#### Valid Interval

$\geq 0$

#### Related indicators

*Renewable factor (renewable\_factor) | Specific GHG per electricity equivalent (specific\_emissions\_per\_electricity\_equivalent)*

The total emissions of the MES in question are calculated with all aggregated energy flows from the generation assets including energy providers and their subsequent emission factor:

$$Total\_emissions = \sum_i E_{gen}(i) \cdot emission\_factor(i)$$

with  $i$ : generation assets 1,2,...

The emissions of each generation asset and provider are also calculated and displayed separately in the outputs of MVS.

### Specific GHG per electricity equivalent (specific\_emissions\_per\_electricity\_equivalent)

**Definition**

Specific GHG emissions per supplied electricity equivalent.

**Type**

Numeric

**Unit**

kg GHGeq/kWh

**Valid Interval**

$\geq 0$

**Related indicators**

*Total GHG emissions (total\_emissions)*

The specific emissions per electricity equivalent of the MES are calculated in  $\text{kg/kWh}_{\text{elec}}$ :

$$Specific\_emissions = \frac{Total\_emissions}{total\_demand_{elec}}$$

Emissions can be of different nature: CO2 emissions, CO2 equivalents, greenhouse gases, ...

Currently the emissions do not include life cycle emissions of energy conversion or storage assets, nor are they calculated separately for the energy sectors. For the latter, the problem of the assignment of assets to sectors arises e.g. emissions caused by an electrolyser would be counted to the electricity sector although you might want to count it for the H2 sector, as the purpose of the electrolyser is to feed the H2 sector. Therefore, we will have to verify whether or not we can apply the energy carrier weighting also for this KPI.

## 3.6.6 Files

### Bar chart of optimal capacities

**Definition**

A bar chart to compare the optimized additional capacities for each asset to be installed in the energy system. Please be aware that the units of the capacities may differ.

**Type**

Figure

**Unit**

kWh, kWp, kW, ...

**Valid Interval**

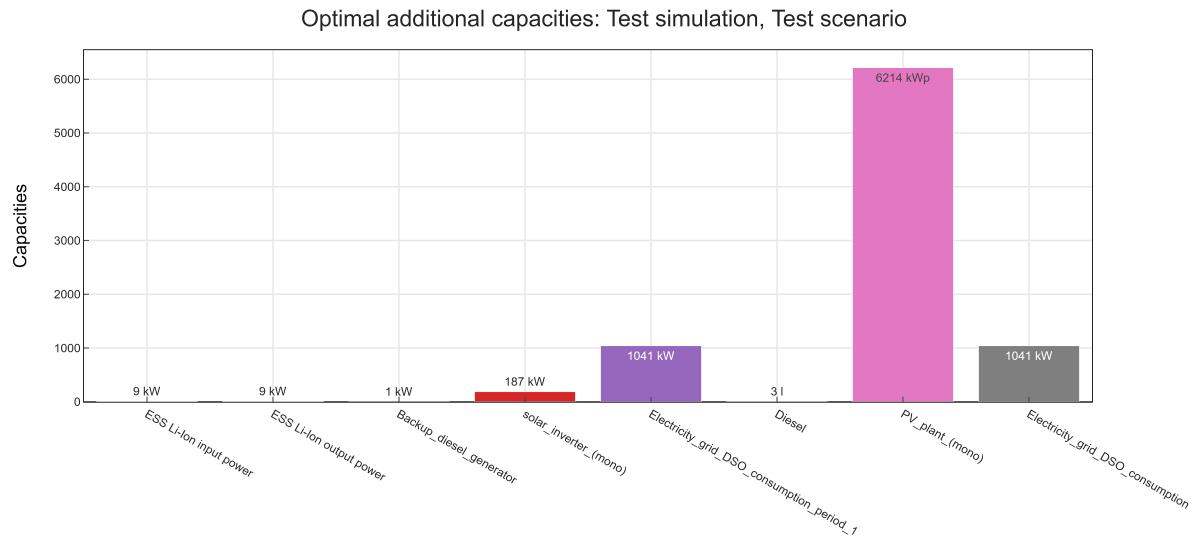
NA

**Related indicators**

*Optimal additional capacity (optimizedAddCap)*



An example of a bar chart of recommended additional asset capacities is shown below. Note that currently kWp are displayed on the same scale as kW (or kWh or gkH2), which is not ideal.



### Pie charts of cost parameters

#### Definition

Displays the share of individual asset costs on different economical parameters of the overall system

#### Type

Figure

#### Unit

Percentages

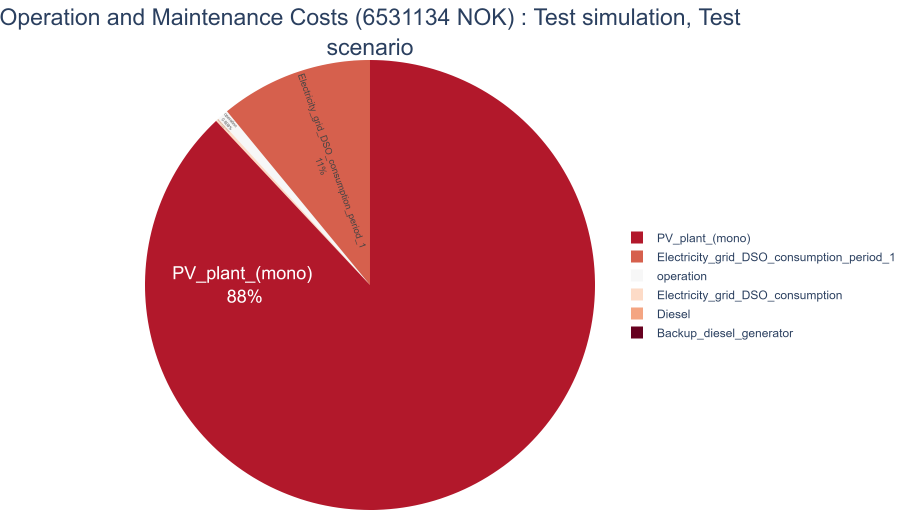
#### Valid Interval

NA

#### Related indicators

*Net Present Costs (NPC) (costs\_total) | Operation, maintenance and dispatch costs (costs\_om\_total)*  
*| Annuity (annuity\_total)*

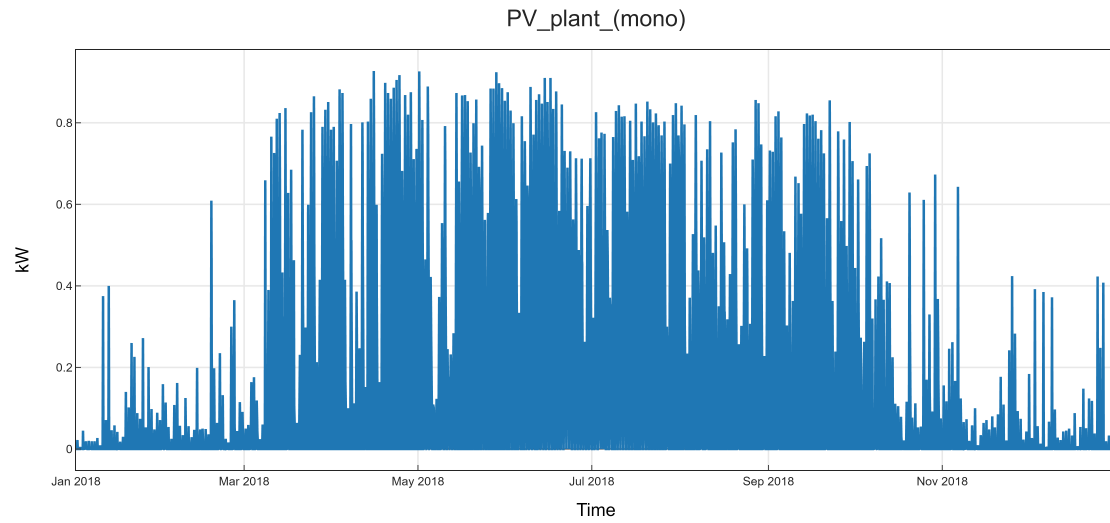
An exemplary pie chart is displayed below, in this case for the operation and management costs of an energy system.



Input timeseries

- Definition**  
Vizualization of timeseries provided as input data, eg. PV generation timeseries.
- Type**  
Figure
- Unit**  
kW, kgH2, ...
- Valid Interval**  
NA
- Related indicators**  
None

An example of the graph created from the timeseries, eg. specific generation timeseries, provided by the input files is shown below.



### Dispatch of all assets on a bus

#### Definition

Visualization of the dispatch of all assets of a specific energy bus, ie. all inflows and outflows of a specific bus. Generated for every single energy bus in the energy system. If relevant, a plot of the state of charge is also displayed.

#### Type

Figure

#### Unit

kW, kgH<sub>2</sub>, ...

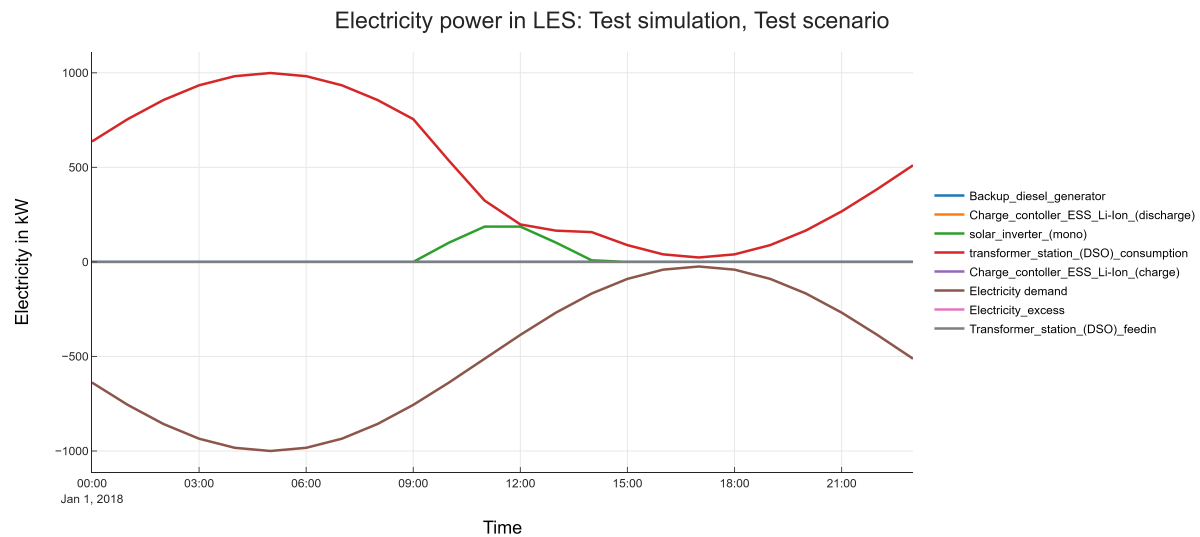
#### Valid Interval

NA

#### Related indicators

*Dispatch of an asset (flow)*

An example of the graph displaying the asset dispatch on a specific bus is shown below.



## Energy system model visualization

### Definition

Plot of the energy system model in oemof-solph topology. This graph also includes the automatically generated components, ie. the sub-assets of energy providers and an energy excess sink on each energy bus. The model therefore appears different than in the Energy Planning Application (EPA).

### Type

Figure

### Unit

NA

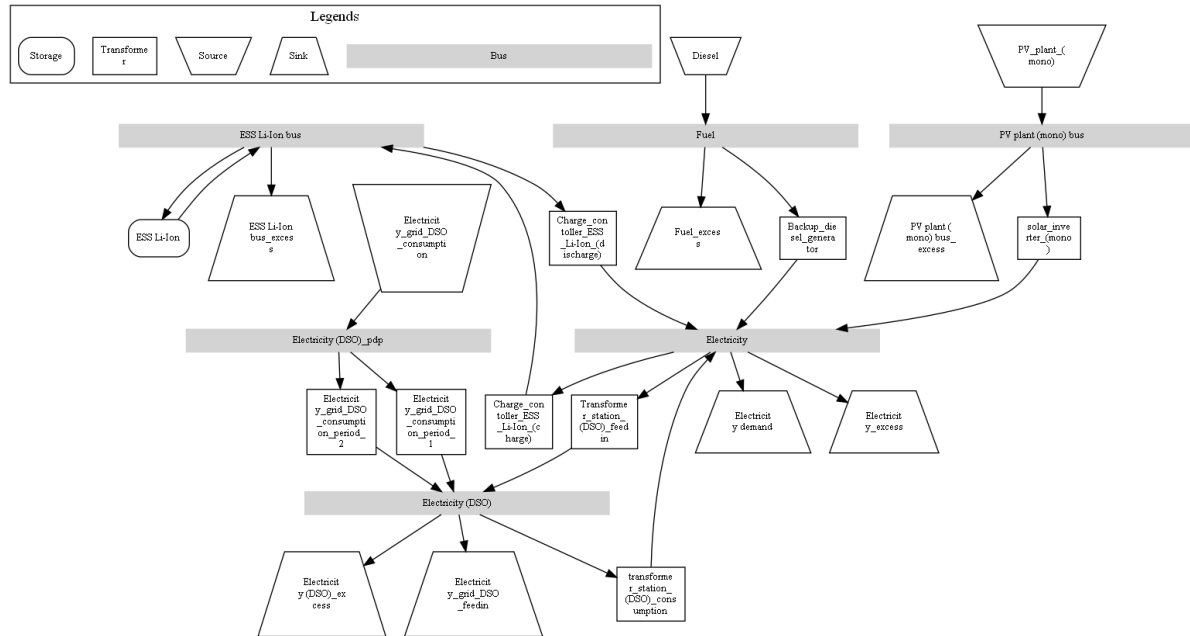
### Valid Interval

NA

### Related indicators

None

An example of the created energy system model graphs is shown below.



## Excel file with all KPI

### Definition

Excel sheet with all calculated KPI, both for the individual assets, the sectors and the overall energy system.

### Type

Excel file

### Unit

NA

### Valid Interval

NA

### Related indicators

None

The file is named `scalars.xlsx`. An example is shown below.

	A	B	C	D	E	F	
1		label	costs_total	costs_om_total	costs_investment_over_lifetime	costs_upfront_in_year_zero	re
2	0	ESS Li-Ion storage capacity	22335.79108	0	22335.79108	0	
3	0	ESS Li-Ion input power	0	0	0	0	
4	0	ESS Li-Ion output power	0	0	0	0	
5	0	Backup_diesel_generator	855.66656	57.34961	798.31695	600	
6	0	Charge_controller_ESS_Li-Ion_(charge)	0	0	0	0	
7	0	Charge_controller_ESS_Li-Ion_(discharge)	0	0	0	0	
8	0	Transformer_station_(DSO)_feedin	0	0	0	0	
9	0	solar_inverter_(mono)	745247.9518	0	745247.9518	560114.34	
10	0	transformer_station_(DSO)_consumption	0	0	0	0	
11	0	Electricity_grid_DSO_consumption_period_1	716153.2061	716153.2061	0	0	
12	0	Electricity_grid_DSO_consumption_period_2	0	0	0	0	
13	0	Diesel	533.87269	533.87269	0	0	
14	0	PV_plant_(mono)	45809980.79	5747672.619	40062308.17	44749746.18	
15	0	Electricity_grid_DSO_consumption	13955.34084	13955.34084	0	0	
16	0	Electricity demand	0	0	0	0	
17	0	ESS Li-Ion bus_excess	0	0	0	0	
18	0	Electricity_excess	0	0	0	0	
19	0	Electricity (DSO)_excess	0	0	0	0	
20	0	Fuel_excess	0	0	0	0	
21	0	PV plant (mono) bus_excess	0	0	0	0	
22	0	Electricity_grid_DSO_feedin	0	0	0	0	

## Excel file with dispatch timeseries

### Definition

Excel sheet with the dispatch of all assets of the energy system. Each tab represents one energy bus.

### Type

Excel file

### Unit

NA

### Valid Interval

NA

### Related indicators

*Dispatch of an asset (flow) | Dispatch of all assets on a bus*

The file is named `timeseries_all_busses.xlsx`. An example is shown below.

	A	B	C	D	E	F	G	H	I
		Backup_diesel_ generator	Charge_controll er_ESS_Li- lon_(discharge)	solar_inverter_ (mono)	transformer_st ation_(DSO)_co nsumption	Charge_controll er_ESS_Li- lon_(charge)	Electricity demand	Electricity_exce ss	Transformer_st ation_(DSO)_fe edin
1									
2	2018-01-01 00:00:00	1	0	0	637	0	-638	0	0
3	2018-01-01 01:00:00	1	0	0	755	0	-756	0	0
4	2018-01-01 02:00:00	1	0	0	856	0	-857	0	0
5	2018-01-01 03:00:00	1	0	0	934	0	-935	0	0
6	2018-01-01 04:00:00	1	0	0	982	0	-983	0	0
7	2018-01-01 05:00:00	1	0	0	999	0	-1000	0	0
8	2018-01-01 06:00:00	1	0	0	982	0	-983	0	0
9	2018-01-01 07:00:00	1	0	0	934	0	-935	0	0
10	2018-01-01 08:00:00	1	0	0	856	0	-857	0	0
11	2018-01-01 09:00:00	1	0	0	755	0	-756	0	0
12	2018-01-01 10:00:00	1	0	101.83897	535.16103	0	-638	0	0
13	2018-01-01 11:00:00	1	0	186.70478	324.29522	0	-512	0	0
14	2018-01-01 12:00:00	1	0	186.70478	198.29522	0	-386	0	0
15	2018-01-01 13:00:00	1	0	101.83897	165.16103	0	-268	0	0
16	2018-01-01 14:00:00	1	0	8.4865809	157.51342	0	-167	0	0
17	2018-01-01 15:00:00	1	0	0	88.7	0	-89.7	0	0
18	2018-01-01 16:00:00	1	0	0	40	0	-41	0	0
19	2018-01-01 17:00:00	1	0	0	23.4	0	-24.4	0	0
20	2018-01-01 18:00:00	1	0	0	40	0	-41	0	0

## MVS logfile

### Definition

Logfile of the MVS simulation including a number of log entries: Debug, information, warning and error messages. Helpful to debug the energy system simulation.

### Type

Logfile

### Unit

NA

### Valid Interval

NA

### Related indicators

None

The file is named `mvs_logfile.log`. An example is shown below.

```

mvs_logfile.log x
1 *****
2 2021-04-28 15:16:01,053 - INFO - logger - Path for logging: D:\PycharmProjects\mvs_eland\MVS_outputs\mvs_logfile.log
3 2021-04-28 15:16:01,053 - INFO - A0_initialization -
4
5 Multi-Vector Simulation Tool (MVS) V0.5.6dev
6 Version: 2021-03-04
7 Part of the toolbox of H2020 project "E-LAND", Integrated multi-vector management system for Energy isLANDs
8 Coded at: Reiner Lemoine Institute (Berlin)
9 Reference: https://zenodo.org/record/4610237
10
11
12 2021-04-28 15:16:01,054 - DEBUG - cli - Accessing script: A1_csv_to_json
13 2021-04-28 15:16:01,054 - INFO - A1_csv_to_json - loading and converting all csv's from tests/inputs/csv_elements into one json
14 2021-04-28 15:16:01,054 - DEBUG - A1_csv_to_json - Loading input data from csv: constraints
15 2021-04-28 15:16:01,059 - DEBUG - A1_csv_to_json - From file constraints following assets are added to the energy system:
16 2021-04-28 15:16:01,059 - DEBUG - A1_csv_to_json - Loading input data from csv: economic_data
17 2021-04-28 15:16:01,062 - DEBUG - A1_csv_to_json - From file economic_data following assets are added to the energy system:
18 2021-04-28 15:16:01,062 - DEBUG - A1_csv_to_json - Loading input data from csv: energyBusses
19 2021-04-28 15:16:01,066 - DEBUG - A1_csv_to_json - From file energyBusses following assets are added to the energy system:
20 2021-04-28 15:16:01,066 - DEBUG - A1_csv_to_json - Loading input data from csv: energyConsumption
21 2021-04-28 15:16:01,068 - WARNING - A1_csv_to_json - The parameter type_asset in the file tests/inputs/csv_elements\energyConsumpt
22 2021-04-28 15:16:01,070 - DEBUG - A1_csv_to_json - From file energyConsumption following assets are added to the energy system:

```

## Simulation report

**Definition**

Automatically generated simulation report, including the most important input data as well as all output data. The *html* can be browsed interactively, while the *pdf* can be shared with partners.

**Type**

html or pdf

**Unit**

NA

**Valid Interval**

NA

**Related indicators**

None

MVS has a feature to automatically *generate a PDF report* that contains the main elements from the input data as well as the simulation results' data. The report can also be viewed as a web app on the browser, which provides some interactivity.

MVS version number, the branch ID and the simulation date are provided as well in the report, under the MVS logo. A commit hash number is provided at the end of the report in order to prevent the erroneous comparing results from simulations using different versions.

It includes several tables with project data, simulation settings, the various demands supplied by the user, the various components of the system and the optimization results such as the energy flows and the costs. The report also provides several plots which help to visualize the flows and costs.

Please, refer to the [report section](#) for more information on how to setup and use this feature, or type

`mvs_report -h`

in your terminal or command line.

A screenshot of the example report header is displayed below. The full exemplary report can be accessed on github in `docs/model/images/example_simulation_report.pdf`.





## Information

MVS Release: 0.5.6dev (2021-03-04)

Simulation date: 2021-05-07

### Simulation data after pre-processing (JSON)

**Definition**

This file includes all data that is used to setup the energy system model, including all the pre-processing performed within *the module C0*. It is mostly used by developers.

**Type**

JSON

**Unit**

NA

**Valid Interval**

NA

**Related indicators**

None

### Simulation data and results (JSON)

**Definition**

This file includes all simulation data and also results of the energy system optimization. With `mvs_report` this file can be used to create a report without re-simulating the energy system. This file is also only used by developers, and also the file used to provide the EPA with the simulation results.

**Type**

JSON

**Unit**

NA

**Valid Interval**

NA

**Related indicators**

None

## 3.7 Validation methodology

MVS is validated using three validation methods: conceptual model validation, model verification and operational validity.

**Conceptual model validation** consists of looking into the underlying theories and assumptions. Therefore, the conceptual validation scheme includes a comprehensive review of the generated equations by the *oemof-solph* python library (see *Economic Dispatch* and *Energy Balance Equation*) and the *components' models*. Next step is to try and adapt them to a sector coupled *example with specific constraints*. Tracing and examining the flowchart is also considered as part of this validation type, which is presented in *Multi-vector simulator*. The aim is to assess the reasonability of the model behavior through pre-requisite knowledge; this technique is known as face validity.

**Model validation** is related to computer programming and looks into whether the code is a correct representation of the conceptual model. To accomplish this, static testing methods are used to validate the output with respect to an input. Unit tests and integration tests, using proof of correctness techniques, are integrated within the code. Unit tests target a single unit such as an individual component, while integration tests target more general parts such as entire modules. Both test types are implemented using *pytest* for the MVS, their evaluation is automatized and they are executed with each change of the MVS. The unit tests are further described in *Unit and integration tests*.

**Operational validity** assesses the model's output with respect to the required accuracy. In order to achieve that, several validation techniques are used, namely:

- **Graphical display**, which is the use of model generated or own graphs for result interpretation. Graphs are simultaneously used with other validation techniques to inspect the results. This technique was regularly applied within the MVS developing process, especially with the help of real use cases from the E-LAND pilot sites.
- **Benchmark testing**, through which scenarios are created with different constraints and component combinations, and the output is calculated and compared to the expected one to evaluate the performance of the model. The applied benchmark tests are described in *Benchmark tests*.
- **Extreme scenarios** (e.g., drastic meteorological conditions, very high costs, etc.) are created to make sure the simulation runs through and check if the output behavior is still valid by the use of graphs and qualitative analysis.
- **Comparison to other validated model**, which compares the results of a case study simulated with the model at hand to the results of a validated optimization model in order to identify the similarities and differences in results. Further information are provided in *Comparison to other models*
- **Sensitivity analysis**, through which input-output transformations are studied to show the impact of changing the values of some input parameters. An example is provided in *Sensitivity analysis verification tests*

Additionally to the presented validation tests, a couple of input verification tests are implemented in the pre-processing module *C0* and a number of output verification tests in *E4* (see *Automatic output verification*).

The validation process of the MVS was identified and defined within the master thesis (*El Mir, 2020*). The evaluation of extreme scenarios and sensitivity analysis was conducted for that thesis only, they are not repeated for each MVS release.

### 3.7.1 Unit and integration tests

To make sure that the MVS works correctly from a programming perspective, its functions need to be tested by unit tests, while its modules should be tested with integration tests. To automatize the testing process, the tests are implemented as `pytest` functions, which also allows to test the test coverage with `coveralls`. They also ensure that the tested existing functionalities do not cease to work properly as new code is introduced in the continuous development of MVS, as each proposed pull request must first pass all existing tests. The unit and integration tests can be found in the folder `tests` of the [MVS github repository](#). Each of the files represent tests for one of the codebase modules (e.g., `A0`, `A1`, `B0`, etc.), and are named respectively: as such, the test file for the codebase module `C2_economic_functions` is named `test_C2_economic_functions`.

As of MVS release 0.5.5, the unit tests covered 74% of the code lines of the MVS. The *benchmark tests*, which also include integration tests, increase this coverage to 91%.

### 3.7.2 Benchmark tests

A benchmark is a point of reference against which results are compared to assess the operational validity of a model. Benchmark tests are also automated like unit and integration tests, hence it is necessary to check that they are always passing for any implemented changes in the model. A list of selected benchmark tests, which cover several features and functionalities of the MVS, are listed here below. The list is not exhaustive, some additional tests are provided in `tests`.

- **Electricity Grid + PV** ([data/pytest](#)): Maximum use of PV to serve the demand and the rest is compensated from the grid
- **Electricity Grid + PV + Battery** ([data/pytest](#)): Reduced excess energy compared to Grid + PV scenario to charge the battery
- **Electricity Grid + Diesel Generator** ([data/pytest](#)): The diesel generator is only used if its LCOE is less than the grid price
- **Electricity Grid + Battery** ([data/pytest](#)): The grid is only used to feed the load
- **Electricity Grid + Battery + Peak Demand Pricing** ([data/pytest](#)): Battery is charged at times of peak demand and used when demand is larger
- **Electricity Grid (Price as Time Series) + Heat Pump + Heat Grid** ([data/pytest](#)): Heat pump is used when  $\text{electricity\_price}/\text{COP}$  is less than the heat grid price
- **Maximum emissions constraint**: Grid + PV + Diesel Generator ([data: set 1, set 2, set 3/pytest](#)): Emissions are limited by constraint, more PV is installed to reduce emissions. For RE share of 100 % in grid, more electricity from the grid is used
- **Parser** converting an energy system model from EPA to MVS ([data/pytest](#))
- **Stratified thermal energy storage** ([data/pytest](#)): With fixed thermal losses absolute and relative reduced storage capacity only if these losses apply
- **Net zero energy (NZE) constraint**: Grid + PV and Grid + PV + Heat Pump ([data set 1, set 2, set 3, set 4/pytest](#)): Degree of NZE  $\geq 1$  when constraint is used and degree of NZE  $< 1$  when constraint is not used.

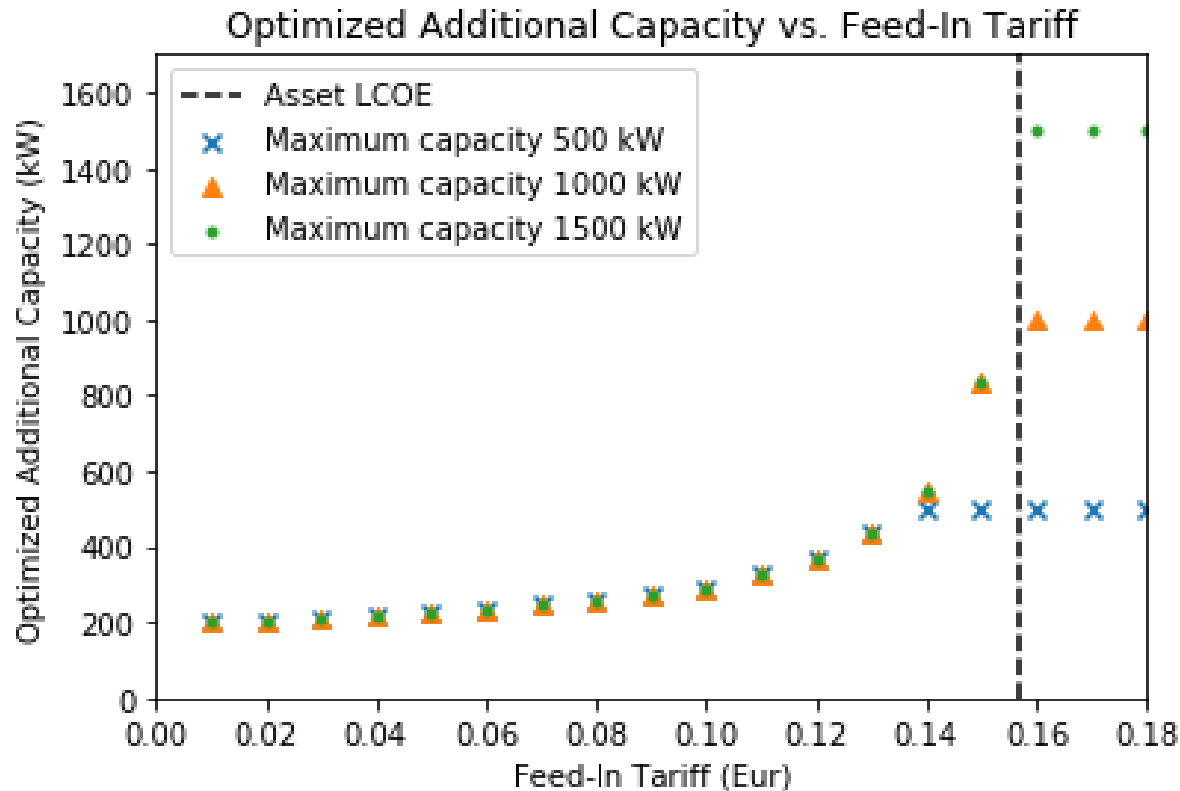
---

**Note:** Benchmark test input data is available in the codebase folders within `tests/benchmark_test_inputs`. It can also be used as simple example cases to get to know the MVS. The benchmark test assertions are provided as `pytest`s in a number of files in `tests` with the naming convention `test_benchmark_*`.

---

### 3.7.3 Sensitivity analysis verification tests

Sensitivity analysis can determine whether a model behaves as expected regarding changes of the model inputs. For the MVS, a sensitivity analysis was performed in (El Mir, 2020, p. 54f) regarding the effect of changing the value of the feed-in tariff (FIT), combined with an energy generation asset with constant marginal costs of generation less than the electricity price. Below graph visualizes the relation of installed PV capacity and FIT, indicating that a FIT larger than the marginal costs of generation leads to an installation of the maximum allowed capacity (maximumCap):



The graph underlines the use of the graphical displays validation technique for model verification. It is not an automated output of the MVS, but indicates that such tests would also be appropriate to translate into benchmark tests.

Other input-output transformations that could be used for sensitivity analysis tests are:

- **Fuel price or generator efficiency variation** around a point where the fuel price or generator efficiency is equal to electricity price or transformer efficiency of the electricity grid.
- **Peak demand price variation** around a point where generator dispatch could avoid consumption from the grid at times of peak demand, thus avoiding peak demand pricing expenditures

### 3.7.4 Comparison to other models

A comparison of the results of different models regarding an identical reference system is a validation method that is commonly used. However, one model cannot absolutely validate another model or claim that one is better than the other. This is why the focus should rather be on testing the correctness, appropriateness and accuracy of a model vis-à-vis its purpose. Since the MVS is an open source tool, it is important to use a validated model for comparison, but also similar open source tools like urbs and Calliope for instance. The following two articles list some of the models that could be used for comparison to the MVS: (*Ringkjøb, 2018*) and (*Bloess, 2017*). A thorough comparison to other models able to perform optimizations for sector-coupled energy systems is something that should be performed in the future.

So far, the MVS has been compared to HOMER for a sector coupled energy system combining electricity and hydrogen sectors. This comparison was able to highlight the similarities and differences between the two optimization models. On the electricity side, most of the values are comparable and within the same range. The differences mainly arise on the hydrogen part in terms of investment into electrolyzer capacity, i.e. the component linking the two sectors, as well as related values. The calculation of the levelized cost of a certain energy carrier appear very different, which, however, was expected due to the *energy carrier weighting approach*: Using this, the costs of the energy system are attributed to the different energy sectors based on their respective share of the total electricity equivalent demand (compare *Costs attributed to a specific sector (attributed\_costs)*, *Levelized costs of electricity equivalent (levelized\_costs\_of\_electricity\_equivalent)*), whereas Homer is likely to assign the costs of the assets based on their output energy carrier. Details regarding the comparison drawn between the two models can be found in (*El Mir, 2020, p. 55ff*).

### 3.7.5 Automatic output verification

In addition to the aforementioned validation tests, a couple of verification tests are implemented. They serve as a safeguard against issues that indicate noteworthy misbehaviour of the model, and are tested with each MVS execution. Some of the issues are fatal issues that the users need to be protected against, others indicate possible unrealistic system optimization (and input) results. The tests are provided in the MVS codebase module *E4\_verification*.

Following test serves as an alert to the energy system modeler to check their inputs again:

- **Excessive excess generation:** Certain combinations of inputs can lead to excessive excess generation on a bus, for example if PV panels itself are very cheap compared to electricity input, while inverter capacity is very expensive. The test *E4.detect\_excessive\_excess\_generation\_in\_bus* notifies to user of optimal but overly high excess generation of a bus within the energy system. Excess generation is defined to be excessive, if the ratio of total outflows to total inflows is less than 90%. The test is applied to each bus individually. The user should check the inputs again and potentially define a *maximumCap* for the generation asset at the root of the problem.

Following tests ensure that introduced constraints were applied correctly:

- **Adherence to maximum emissions constraint:** With the *maximum emission constraint* the user can define the maximum allowed emissions in the energy mix of the optimized energy system. The test *E4.maximum\_emissions\_test* runs a verifies that the constraint is adhered to.
- **Adherence to minimal renewable share constraint:** Test *E4.minimal\_renewable\_share\_test* makes sure that the user-defined constraint of the *minimal share of renewables* in the energy mix of the optimized system is respected.
- **Adherence to net zero energy constraint:** If the user activated the *net zero energy constraint*, the test *E4.net\_zero\_energy\_constraint\_test* makes sure that the optimized energy system adheres to it.
- **Adherence to realistic SOC values:** Test *E4.verify\_state\_of\_charge* makes sure that the timeseries of the state of charge (SOC) values for storages in the energy system simulation results are within the valid interval of  $[0, 1]$ . A SOC value out of bounds is physically not feasible, but can occur when the optimized storage capacity is so marginal that it is in the range of the *precision limit* of the MVS.

---

**Note:** If there is an ERROR displayed in the log file (or the automatic report), the user should follow the instructions of the error message. Some will require the user to check and adapt their input data, others will indicate serious misbehaviour. A WARNING in the log file (or the automatic report) is important information about the performed system optimization which the user should be aware of.

---

## 3.8 E-LAND requirements of the MVS

### 3.8.1 Functional Requirements

#### FUN-MVS-01 - Solving an energy system optimization model

**Description**

The MVS shall solve an energy system planning optimization problem and provide the optimal sizing of individual assets.

**Rationale**

Basic operation of MVS.

**Priority**

HIGH

**Progress**

Done

**Progress message**

The MVS can solve energy system planning optimization problems and identify the optimal additional capacities of chosen assets. In-code validation checks, unit tests and benchmark tests were added to ensure that the simulation runs smoothly and correctly.

**Notes**

As with all simulation tools, there are always possibilities to improve the tool, specifically to address current limitations (comp. *Limitations*).

#### FUN-MVS-02 - Automatic setting up of an energy system optimization model

**Description**

The MVS should accept modelling parameters regarding the LES in a specific format.

**Rationale**

Currently MVS supports the Oemof model. The rationale is to support external entities or users with no experience in Oemof, by automatically generating the respective Oemof model for the agreed format

**Priority**

HIGH

**Progress**

Done

**Progress message**

The MVS accepts simulation data provided as csv files and automatically sets up an energy system. It also supports the integration into the Energy Planning Application (EPA) by providing a parser for the interfaces of the two tools.

### FUN-MVS-03 - Manual setting up an energy system optimization model

**Description**

The MVS shall support adding specific components/constraints from a set of options to an energy system optimization model.

**Rationale**

Basic operation of MVS

**Priority**

LOW

**Progress**

Done

**Progress message**

It is possible to add as many components as needed to the energy model that is to be simulated with the MVS. They can be divided into following asset types:

- *Energy providers*
- *Energy production*
- *Energy consumption*
- *Energy conversion*
- *Energy storage*

Details on how to model different assets are included in the model assumptions (see *Component models*).

**Notes**

Energy excess sinks are automatically added by the MVS to enable energy system optimization and do not have to be added by the energy system planners.

In the future, it may be possible to add energy shortage sources, which would allow energy systems with a defined annual supply shortage. While this mostly will not result in an energy system many operators would require, it would also have benefits for the debugging of energy systems, as infeasible energy systems would be easier to evaluate and specific debug messages could be displayed.

### FUN-MVS-04 - Optimisation Results

**Description**

The MVS shall provide the results of the optimisation process upon completion of calculation in a specific format, which include at least information related to asset costs (CAPEX and OPEX), sizes, as well as aggregated energy flows and overall system performance (autonomy, renewable share, losses).

**Rationale**

Basic operation of MVS.

**Priority**

HIGH

**Progress**

Done

**Progress message**

The results of the MVS simulation are post-processed, and result in numerous key performance indicators (KPI). Following information is calculated for an economic evaluation of the energy system:

- Capital and operational expenditures (capex, opex) per asset, both as annuities as well as present costs. This includes also the first-time investment costs (FIC), the replacement costs minus residual values, and the costs for asset dispatch (equations compare *Economic Dispatch*).
- *NPC* and annuity of the whole energy system
- *Levelized cost of energy (LCOE)* of the energy system, in electricity equivalent
- *Levelized cost of an energy carrier* in electricity equivalent (LCOE<sub>eq</sub>) for each energy carrier in the energy system
- *Levelized cost of asset dispatch*, calculated from the annuity of an asset and their throughput

Additionally, a number of technical parameters are calculated both the energy system and the individual energy vectors:

- Dispatch, *aggregated energy flows* as well as *peak flows* of each asset
- *Renewable share*
- *Renewable share of local generation*
- *Degree of autonomy*
- *Degree of net zero energy*
- *Onsite Energy Matching (OEM)*
- *Onsite Energy Fraction (OEF)*
- Annual excess energy
- *Annual GHG<sub>eq</sub> emissions* and specific emissions per electricity equivalent

#### Notes

Currently in discussion is the implementation of a so-called *degree of sector-coupling* (see issue 702). This is a novel key performance indicator and would be integrated in addition to above mentioned parameters.

### FUN-MVS-05 - Production Assets

#### Description

The MVS should consider a diverse type of production assets in the energy model i.e. PV, BESS, CHP, Thermal Storage

#### Rationale

Enable support of multi-vector production and storage assets.

#### Priority

HIGH

#### Progress

In-progress

#### Progress message

The MVS is able to simulate a wide range of assets:

- *PV plants, wind plants*
- *Battery Electricity Storage Systems (BESS)*, via generic storage object
- *Thermal storages*, via generic or thermal storage object
- *Power plants* as simple generators.
- And many more (see below)



**ToDo**

A CHP with fix ratio between the heat and electricity output can already be simulated, but has not been tested. For a CHP with a variable ration between those two outputs, we need to add the specific CHP asset to the possible inputs.

**FUN-MVS-06 - Assets of Energy Conversion****Description**

The MVS should consider assets which convert energy from one vector to another i.e. CHP, geothermal conversion (heat pump)

**Rationale**

Integration of the multi-vector approach in the MVS.

**Priority**

LOW

**Progress**

Done

**Progress message**

The MVS already covers generic conversion assets. How the generic definition can be applied to the individual assets is explained [here](#). This includes

- *Electric transformers*
- *Power plants (Condensing power plants and Combined heat and power)*
- *Heat pumps and Heating, Ventilation, and Air Conditioning (HVAC) assets*
- *Electrolyzers*

**ToDo**

A CHP with a variable share of heat and electricity output is currently not implemented. It could be added as a new oemof asset type.

When using two conversion objects to emulate a bidirectional conversion assets (eg. charge controllers, bi-directional inverters), their capacity should be interdependent. This is currently not the case, as explained in the [limitations](#).

**FUN-MVS-07 - Optimisation goal****Description**

The optimisation process should take into account: Increasing the degree of autonomy of the LES, system costs minimization, and CO2 emissions reduction. Optional extension of the MVS is to allow for multi-objective optimisation.

**Rationale**

Different optimisation goal shall be supported for covering the different perspectives of the possible end-users.

**Priority**

HIGH

**Progress**

Done

**Progress message**

In general, the MVS aims to minimize the energy supply cost of the local energy system. Additionally, a number of [constraints](#) can be activated:

- *Minimal renewable share constraint*
- *Minimal degree of autonomy*
- *Maximum GHG emission constraint*
- *Net zero energy constraint*
- *Limited maximum capacities of assets to be optimized*

## FUN-MVS-08 - Electricity cost model

### Description

The MVS model shall be provided with data defining electricity grid supply regarding: a) kWh prices (both import and export from/to the grid), b) kWh/h prices (time series of prices), c) Constraints of the interconnection with the main grid (e.g. substation capacity)

### Rationale

Information necessary for building the MVS Multi-vector Model.

### Priority

HIGH

### Progress

Done

### Progress message

The different constraints regarding the electricity DSO can be considered:

- a) The energy price as well as the feed-in tariff of a DSO can be provided as a time series (see *Time series: time\_series folder*)
- b) *Peak demand pricing* can be considered
- c) The transformer station limitation can, but does not have to, be added.

## FUN-MVS-09 - Load profiles

### Description

The MVS model shall be provided with annual electric/thermal demand profiles (hourly values) for each load in the LES.

### Rationale

Information necessary for building the MVS Multi-vector Model.

### Priority

HIGH

### Progress

Done

### Progress message

The MVS can be provided with a variable number of energy consumption profiles, that can be connected to variable busses. Details on how this works can be found in *these instructions*.

## FUN-MVS-10 - DH cost model

### Description

For calculations involving district heating, the MVS model shall support data on thermal distribution network supply, concerning: a) kWh prices (both import and export from/to the grid), b) kWh/h prices (time series of prices), c) optional: thermal power cap (e.g. maximum allowable feed-in per day)

### Rationale

Information necessary for building the MVS Multi-vector Model.

### Priority

HIGH

### Progress

Done

### Progress message

Same as for *FUN-MVS-08 - Electricity cost model*.

## FUN-MVS-11 - PV data

### Description

For calculations involving PV assets, the MVS model shall be provided with data on PV assets:

a) At minimum: Precise location (latitude and longitude), b) Optionally: performance indicators for new PV systems (efficiency - constant or time series, module technology, performance ratio), historical/tracked data (energy generated by existing PV systems, weather data), Inverter efficiency

### Rationale

Information necessary for building the MVS Multi-vector Model.

### Priority

HIGH

### Progress

Done for option (b), no automatization (minimal requirement met)

### Progress message

To simulate a PV plant, the MVS model requires following data from the end-user:

- (Historical) Specific PV generation profile (in kWh/kWp)
- Inverter efficiencies can be considered with an additional energyConversion asset

### ToDo

To ease the data input for the end-user, more processing could be included here (option a)). For example, the `feedinlib` could be used to fetch the specific PV generation profiles with following data:

- Longitude and latitude
- Module or efficiency
- Performance ratio

This could also be implemented in the EPA.

## FUN-MVS-12 - Battery data

### Description

For calculations involving battery assets, the MVS model shall be provided with data on Battery Energy Storage Systems (BESS): a) Battery type (e.g. lead-acid, lithium ion) b. Technical parameters: C-rate, max and min state of charge (SOC), max. depth of discharge (DOD), roundtrip efficiency (constant or time series), c. Inverter efficiency (optional), d. historical/tracked data from existing BESS

### Rationale

Information necessary for building the MVS Multi-vector Model.

### Priority

HIGH

### Progress

Done for option (b), no default inputs, no historical data (minimal requirement met)

### Progress message

For the MVS, the type of the BESS does not matter. Important are the technical parameters:

- *C-rate*
- *Maximum* and *minimum* state of charge (SOC), whereas the latter is inverse to the maximum depth of discharge (DOD)
- Charge- and discharge (constant or time series, equivalent to roundtrip-efficiency) as well as self-discharge rate (comp. *efficiency*)
- It is possible to define *soc\_initial*, the initial storage charge at the beginning of the optimization period, which is most important for short-term optimizations.
- An inverter or charge controller can be defined by defining an additional energyConversion asset

### Notes

It may be preferable for the end-users to have default input values for different battery types (option a)), which is not implemented. This would best be addressed in the EPA with a database of default values, but is currently not being discussed.

Historical dispatch data of batteries is not considered, as the MVS is supposed to determine the optimal dispatch rather than only the performance of a current energy system with determined operational schedules.

## FUN-MVS-13 - CHP data

### Description

For calculations involving CHP assets, the MVS model shall be provided with efficiency factors (electric/thermal)

### Rationale

Information necessary for building the MVS Multi-Vector Model.

### Priority

LOW

### Progress

In progress (minimal requirement met)

### Progress message

A simple CHP model is already included in the MVS (compare *Condensing power plants and Combined heat and power (CHP)*). It considers a fix ratio between thermal and electric output.

**ToDo**

For a variable ratio between heat and electricity output, a new, specific oemof asset would need to be added to the MVS.

**FUN-MVS-14 - Thermal storage data****Description**

For calculations involving Thermal Storage assets, the MVS model shall be provided with: a) Charging and discharging efficiencies, b. Max/Min SOC, initial SOC

**Rationale**

Information necessary for building the MVS Multi-Vector Model.

**Priority**

LOW

**Progress**

Done

**Progress message**

It is possible to simulate thermal storage assets with the MVS. Their model is analogous to the BESS, which fulfills the requirement. They are defined by:

- *C-rate*
- *Maximum* and *minimum* state of charge (SOC)
- Charge- and discharge (constant or time series, equivalent to roundtrip-efficiency) as well as self-discharge rate (comp. *efficiency*)
- It is possible to define *soc\_initial*, the initial storage charge at the beginning of the optimization period, which is most important for short-term optimizations.

Adding another level of detail, it is possible to model a *Stratified thermal energy storage*, with additional parameters *fixed\_thermal\_losses\_relative* and *fixed\_thermal\_losses\_absolute*.

**FUN-MVS-15 - Autonomous operation data****Description**

The MVS model shall be provided with information on the autonomous operation of the LES i.e. minimum/maximum time of autonomy for specific time intervals.

**Rationale**

Information necessary for building the MVS Multi-vector Model

**Priority**

HIGH

**Progress**

Done

**Progress message**

This requirement is addressed by the *degree of autonomy constraint*. It is related to the aggregated demand of the energy system and the required consumption from the grid (comp. *DOA*), and not minimum or maximum time of autonomous operation.

**Notes**

A constraint of time-related autonomous operation is not possible in the current MVS, as it would introduced a mixed-integer constraint, which would extend simulation times too much. It would be possible in the future to add KPI that quantify the behaviour.

### FUN-MVS-16 - Economic data

#### Description

The MVS model shall be provided with information on economic assumptions per asset: CAPEX/kW and OPEX/kWh (constant or time series), lifetime (years), Weighted Average Cost of Capital (WACC).

#### Rationale

Information necessary for building the Multi-vector Model.

#### Priority

HIGH

#### Progress

Done

#### Progress message

The MVS receives economic data from the end-user. This includes:

- *Specific investment costs of assets (CAPEX/kW)*
- *Dispatch price of assets*
- *Specific annual operation and management costs (OPEX/kWh, constant or time series))*
- *Currency*
- *Tax*
- *Weighted Average Cost of Capital (WACC)*
- *Lifetime of the project*
- *Lifetime of assets*

### FUN-MVS-17 - Constraints

#### Description

The MVS model shall be provided with constraints of the optimisation problem: a) Operating reserve provided by the battery (i.e. redundancy, availability), b. Sizing constraints, c. Cost constraints

#### Rationale

Information necessary for building the Multi-vector Model.

#### Priority

HIGH

#### Progress

In progress

#### Progress message

To address the sizing constraint, the attribute `maximumCap` was introduced. This will limit the optimized capacity, even if this results in higher energy supply costs.

A cost constraint is for now disregarded, as always the cheapest supply solution is identified. Limiting the overall NPC would result in infeasible solutions and a termination of the MVS. Cost constraints considering specific technologies can be covered by adapting the `maximumCap`.

#### **ToDo**

It was decided at the beginning of the project that the operating reserve constraint may be developed in cooperation with the end-users. This constraint would still need to be defined with the stakeholders.

### **3.8.2 Non-Functional Requirements**

#### **NF-MVS-01 - MVS pre-processing tools for LES optimization model input**

##### **Description**

The MVS should support Python-Pandas DataFrames as parameterization input for the LES model

##### **Scope**

Interface, Usability

##### **Metric**

Y/N

##### **Verification and Measurement**

The requirement is validated by observing the system under test when an operator attempts to input/modify the model parameters.

##### **Target**

User can adjust input parameters without any further support

##### **Progress**

Done

##### **Progress message**

Internally, the MVS uses dictionaries (*dict*) in combination with pandas (*pd.DataFrame*) to set up the energy system model. However, for data exchange with the end-user the input files, ie. the csv or json file is essential. To be able to use all features of the MVS, the user should consider the terminal-based MVS with csv input files. For a more comfortable and interactive usage, the end user can use the MVS though the user interface of the Energy Planning Application (EPA). Here, the data format becomes irrelevant for the user.

#### **NF-MVS-02 - MVS post-processing tools for LES optimization model output/results**

##### **Description**

The MVS should provide results aggregation, reports, and plots

##### **Scope**

User Interface, Usability

##### **Metric**

Y/N

##### **Verification and Measurement**

The requirement is validated by observing the system under test when an operator attempts to access the output results.

##### **Target**

User can extract the results in a way that can be directly used for the users purpose

### Progress

Done

### Progress message

The post-processing of results ensures that important KPI can be provided for the energy system optimization. There are three output formats of the MVS:

- For the end-user of the standalone application, an *automatic report* is generated that makes scenario evaluation easy
- For a developer of the standalone application, the results are also provided as excel files and pngs
- For the EPA, the results are provided in a json format to be displayed interactively in their environment

### Notes

Improving the outputs is a continuing task. Following improvements can be considered in the future:

- Move all KPI connected to the individual energy vectors into a separate table and display in the report
- Add-on requested by end-users: Cash flow projections

## NF-MVS-03 - Communication interface between MVS and ESB

### Description

Communication functionality must be included so that ESB can send requests to MVS and vice versa. This assures that all requests can be coordinated through one platform (e.g. ESB).

### Scope

User Interface, Usability

### Metric

Y/N

### Verification and Measurement

Send a set of different requests from ESB to MVS and count received requests. Do vice versa.

### Target

Send/receive requests that can be processed without information loss

### Progress

Done

### Progress message

After discussion, there is no direct interface of the ESB and the MVS. The MVS is a standalone application that must be usable without the ESB. To ease end-user use, the EPA (Energy Planning Application) is developed. It sends inputs in json format to the MVS, and receives a json file with the results back. Parsers are coded to allow a translation of the different formats of the MVS and the EPA.

### Notes

The EPA development is a continuous process, and currently the MVS has more features than the EPA. Mainly, the EPA does not feature:

- Any constraints of the MVS
- GHG emission calculation
- Set of energy assets of different energy vectors (as EPA explicitly names the assets)



**NF-MVS-04 - Unit commitment time step restriction****Description**

Energy flows between selected components (Unit commitment) are simulated in hourly timesteps.

**Scope**

Performance

**Metric**

Timestamps

**Verification and Measurement**

Subtract 2-time steps.

**Target**

Timestep width of 1 hour

**Progress**

Done

**Progress message**

The MVS can be run for a variable number of days. The time series have to be provided on an hourly basis.

**Notes**

A wish from the end-users was a finer resolution of eg. 15-minute time steps. This possibility still has to be explored.

**NF-MVS-05 - Interface for technical parameters and model****Description**

Technical parameters are reflected in component modelling

**Scope**

Performance

**Metric**

Technical variable in energy system model object

**Verification and Measurement**

Technical variable in ESM object being not NAN.

**Target**

N/A

**Progress**

Done

**Progress message**

The MVS uses the input parameters to compile the component models. This is also tested using pytests and benchmark tests.

## NF-MVS-06 - Interface for economic parameters and model

### Description

Cost parameters are reflected in component modelling

### Scope

Interface

### Metric

Cost variable in energy system model object

### Verification and Measurement

Cost variable in ESM object being not NAN.

### Target

N/A

### Progress

Done

### Progress message

The MVS uses the input parameters to compile the component models. This is also tested using pytests and benchmark tests.

## API REFERENCE

- **Documentation:** *Modules and functions*
- **Getting involved:** *Contributing guidelines and protocols*
- **Academic references:** *Publications and Bibliography*
- **Using or modifying MVS:** *License* | *How to cite MVS*
- **Getting help:** *Know issues and workaround* | *Report a bug or issue*

### 4.1 Code documentation

#### 4.1.1 Util functions

##### Module `data_parser`

This module defines all functions to convert formats between EPA and MVS - Define similar parameters mapping between the EPA and MVS in `MAP_EPA_MVS` and `MAP_MVS_EPA` - Define which fields are expected in asset list of EPA for various assets' groups in `EPA_ASSET_KEYS` - Convert MVS to EPA - Convert EPA to MVS

`multi_vector_simulator.utils.data_parser.convert_epa_params_to_mvs(epa_dict)`

Convert the EPA output parameters to MVS input parameters

**Parameters**

**`epa_dict`** (*dict*) – parameters from EPA user interface

**Returns**

**`dict_values`** – MVS json file, generated from EPA inputs, to be provided as MVS input

**Return type**

`dict`

##### Notes

- **For *simulation\_settings*:**
  - parameter *TIMESTEP* is parsed as unit-value pair
  - *OUTPUT\_LP\_FILE* is set to *False* by default
- For *project\_data*: parameter *SCENARIO\_DESCRIPTION* is defined as placeholder string.
- *fix\_cost* is not required, default value will be set if it is not provided.

- For missing asset group **CONSTRAINTS** following parameters are added:
  - MINIMAL\_RENEWABLE\_FACTOR: 0
  - MAXIMUM\_EMISSIONS: None
  - MINIMAL\_DEGREE\_OF\_AUTONOMY: 0
  - NET\_ZERO\_ENERGY: False
- **ENERGY\_STORAGE** assets:
  - Optimize cap written to main asset and removed from subassets
  - Units defined automatically (assumed: electricity system)
  - SOC\_INITIAL: None
  - THERM\_LOSSES\_REL: 0
  - THERM\_LOSSES\_ABS: 0
- If **TIMESERIES** parameter in asset dictionary: Redefine unit, value and label.
- **ENERGY\_PROVIDERS**:
  - Auto-define unit as kWh(el)
  - INFLOW\_DIRECTION=OUTFLOW\_DIRECTION
  - Default value for EMISSION\_FACTOR added
- **ENERGY\_CONSUMPTION**:
  - DSM is *False*
  - DISPATCHABILITY is FALSE
- **ENERGY\_PRODUCTION**:
  - Default value for EMISSION\_FACTOR added
  - DISPATCHABILITY is always *False*, as no dispatchable fuel assets possible right now. Must be tackled by EPA.

`multi_vector_simulator.utils.data_parser.convert_mvs_params_to_epa(mvs_dict, verbatim=False)`

Convert the MVS output parameters to EPA format

**Parameters**

**mvs\_dict** (*dict*) – output parameters from MVS

**Returns**

**epa\_dict** – epa parameters

**Return type**

*dict*

## Helper functions

Util functions that are useful throughout the MVS

Including: - `find_value_by_key()`: Finds value of a key in a nested dictionary.

`multi_vector_simulator.utils.helpers.find_value_by_key(data, target, result=None)`

Finds value of a key in a nested dictionary.

### Parameters

- **data** (*dict*) – Dict to be searched for target key
- **target** (*str*) – Key for which the value should be found in data
- **result** (*None, value or list*) – Only provided if function loops in itself

### Returns

- *value if the key is only once in data*
- *list of values if it appears multiple times.*

`multi_vector_simulator.utils.helpers.get_item_if_list(list_or_float, index)`

`multi_vector_simulator.utils.helpers.get_length_if_list(list_or_float)`

`multi_vector_simulator.utils.helpers.peak_demand_bus_name(dso_name: str, feedin: bool = False)`

Name for auto created bus related to peak demand pricing period

`multi_vector_simulator.utils.helpers.peak_demand_transformer_name(dso_name: str, peak_number: Optional[int] = None, feedin: bool = False)`

Name for auto created bus related to peak demand pricing period

`multi_vector_simulator.utils.helpers.translates_epa_strings_to_mvs_readable(folder_name, file_name)`

This function translates the json file generated by the EPA to a file readable by the MVS. This is necessary as there are some parameter names whose string representative differs in both tools.

### Parameters

- **folder\_name** (*path*) – Path to the folder with the json file
- **file\_name** (*json file name with extension*) – Json to be converted

### Returns

- *Stores converted json file to current dict*
- *Usage – `import multi_vector_simulator.utils.helpers as helpers`  
`helpers.translates_epa_strings_to_mvs_readable("./epa_benchmark", "epa_benchmark.json-original")`*

## 4.1.2 Initialization

### Module A0 - Initialization

Module A0\_initialization defines functions to parse user inputs to the MVS simulation.

- Display welcome message with current version number
- Parse command line arguments and set default values for MVS parameters if not provided
- Check that all necessary files and folder exist
- Create output directory
- Define screen logging depth

Usage from root of repository:

```
python mvs_tool.py [-h] [-i [PATH_INPUT_FOLDER]] [-ext [{json,csv}]] [-o [PATH_OUTPUT_
↪FOLDER]]
[-log [{debug,info,error,warning}]] [-f [OVERWRITE]] [-pdf [PDF_REPORT]] [-png [SAVE_
↪PNG]]
```

Usage when multi-vector-simulator is installed as a package:

```
mvs_tool [-h] [-i [PATH_INPUT_FOLDER]] [-ext [{json,csv}]] [-o [PATH_OUTPUT_FOLDER]]
[-log [{debug,info,error,warning}]] [-f [OVERWRITE]] [-pdf [PDF_REPORT]] [-png [SAVE_
↪PNG]]
```

Process MVS arguments

**optional arguments:**

- h, --help** show this help message and exit
- i [PATH\_INPUT\_FOLDER]**  
path to the input folder
- ext [{json,csv}]**  
type (json or csv) of the input files (default: 'json')
- o [PATH\_OUTPUT\_FOLDER]**  
path to the output folder for the simulation's results
- log [{debug,info,error,warning}]**  
level of logging in the console
- f [OVERWRITE]**  
overwrite the output folder if True (default: False)
- pdf [PDF\_REPORT]**  
generate a pdf report of the simulation if True (default: False)
- png [SAVE\_PNG]**  
generate png figures of the simulation in the output\_folder if True (default: False)

`multi_vector_simulator.A0_initialization.check_input_folder(path_input_folder, input_type)`  
Enforces the rules for the input folder and files

There should be a single json file for config (described under JSON\_FNAME) in case input\_type is equal to JSON\_EXT. There should be a folder with csv files (name of folder given by CSV\_ELEMENTS) in case input\_type is equal to CSV\_EXT.

#### Parameters

- **path\_input\_folder** – path to input folder
- **input\_type** – of of JSON\_EXT or CSV\_EXT

#### Returns

the json filename which will be used as input of the simulation

```
multi_vector_simulator.A0_initialization.check_output_folder(path_input_folder,
                                                             path_output_folder, overwrite)
```

Enforces the rules for the output folder

An error is raised if the path\_output\_folder already exists, unless overwrite is set to True. The path\_output\_folder is created if not existing and the content of path\_input\_folder is copied in a folder named INPUTS\_COPY.

#### Parameters

- **path\_input\_folder** – path to input folder
- **path\_output\_folder** – path to output folder
- **overwrite** – boolean indicating what to do if the output folder exists already

#### Returns

the path to the folder stored in the output folder as copy of the input folder

```
multi_vector_simulator.A0_initialization.mvs_arg_parser()
```

Create a command line argument parser for MVS

Usage from root of repository:

```
python mvs_tool.py [-h] [-i [PATH_INPUT_FOLDER]] [-ext [{json,csv}]] [-o [PATH_
↪OUTPUT_FOLDER]]
[-log [{debug,info,error,warning}]] [-f [OVERWRITE]] [-pdf [PDF_REPORT]] [-png↪
↪[SAVE_PNG]]
[--version]
```

Usage when multi-vector-simulator is installed as a package:

```
mvs_tool [-h] [-i [PATH_INPUT_FOLDER]] [-ext [{json,csv}]] [-o [PATH_OUTPUT_FOLDER]]
[-log [{debug,info,error,warning}]] [-f [OVERWRITE]] [-pdf [PDF_REPORT]] [-png↪
↪[SAVE_PNG]]
[--version]
```

Process MVS arguments

**optional arguments:**

**-h, --help** show this help message and exit

**-i [PATH\_INPUT\_FOLDER]**  
path to the input folder

**-ext** [{json,csv}]  
type (json or csv) of the input files (default: 'json')

**-o** [PATH\_OUTPUT\_FOLDER]  
path to the output folder for the simulation's results

**-log** [{debug,info,error,warning}]  
level of logging in the console

**-f** [OVERWRITE]  
overwrite the output folder if True (default: False)

**-pdf** [PDF\_REPORT]  
generate a pdf report of the simulation if True (default: False)

**-png** [SAVE\_PNG]  
generate png figures of the simulation in the output\_folder if True (default: False)

**--version** show program's version number and exit

**Returns**  
parser

```
multi_vector_simulator.A0_initialization.process_user_arguments(path_input_folder=None,  
                                                                input_type=None,  
                                                                path_output_folder=None,  
                                                                overwrite=None,  
                                                                pdf_report=None,  
                                                                display_output=None,  
                                                                save_png=None,  
                                                                lp_file_output=False,  
                                                                welcome_text=None)
```

Process user command from terminal inputs. If inputs provided as arguments of the function, they will overwrite the command line arguments.

#### Parameters

- **path\_input\_folder** – Describes path to inputs folder (command line “-i”)
- **input\_type** – Describes type of input to expect (command line “-ext”)
- **path\_output\_folder** – Describes path to folder to be used for terminal output (command line “-o”) Must not exist before
- **overwrite** – (Optional) Can force tool to replace existing output folder (command line “-f”)
- **pdf\_report** – (Optional) Can generate an automatic pdf report of the simulation's results (Command line “-pdf”)
- **save\_png** – (Optional) Can generate png figures with the simulation's results (Command line “-png”)
- **display\_output** – (Optional) Determines which messages are used for terminal output (command line “-log”) Allowed values are “debug”: All logging messages, “info”: All informative messages and warnings (default), “warning”: All warnings, “error”: Only errors,
- **lp\_file\_output** – Save linear equation system generated as lp file
- **welcome\_text** – Text to be displayed

**Returns**  
a dict with these arguments as keys (except welcome\_text which is replaced by label)



`multi_vector_simulator.A0_initialization.report_arg_parser()`

Create a command line argument parser for MVS

Usage when multi-vector-simulator is installed as a package:

```
mvs_report [-h] [-i [PATH_SIM_OUTPUT]] [-o [REPORT_PATH]] [-pdf]
```

Process mvs report command line arguments

**optional arguments:**

**-h, --help** show this help message and exit

**-pdf [PRINT\_REPORT]**  
print the report as pdf (default: False)

**-i [OUTPUT\_FOLDER]**  
path to the simulation result json file 'json\_with\_results.json'

**-o [REPORT\_PATH]**  
path to save the pdf report

**Returns**  
parser

## Module A1 - Csv to json

Convert csv files to json file as input for the simulation.

The default input csv files are stored in “/inputs/elements/csv”. Otherwise their path is provided by the user.

The user can change parameters of the simulation of of the energy system in the csv files.

Storage: The “energyStorage.csv” contains information about all storages. For each storage there needs to be another file named exactly after each storage-column in the “energyStorage.csv” file. For the default file this is “storage\_01”, “storage\_02” etc. Please stick to this convention.

The function “create\_input\_json()” reads all csv files that are stored in the given input folder (input\_directory) and creates one json input file for mvs\_tool.

Functions of this module (that need to be tested) - read all necessary input files (*REQUIRED\_CSV\_FILES*) from input folder - display error message if *CSV\_FNAME* already in input folder - read all parameters in from csv files - parse parameter that is given as a timeseries with input file name and header - parse parameter that is given as a list

- check that parameter that is given as a list results and subsequent other parameters to be given as list e.g. if we have two output flows in conversion assets there should be two efficiencies to operational costs (this is not implemented in code yet)
- only necessary parameters should be transferred to json dict, error message with additional parameters
- parse data from csv according to intended types - string, boolean, float, int, dict, list!

`multi_vector_simulator.A1_csv_to_json.add_storage_components(storage_filename, input_directory, storage_label)`

Creates json dict from storage csv.

Loads the csv of a the specific storage listed as column in “energyStorage.csv”, checks for complete set of parameters, adds a label and creates a json dictionary.

**Parameters**

- **storage\_filename** (*str*) – file name excl. extension, given by the parameter ‘file\_name’ in “energyStorage.csv
- **input\_directory** (*str*) – path to the input directory where *storage\_filename* is located
- **storage\_label** (*str*) – Label of storage

**Notes**

Tested with: - test\_add\_storage\_components\_label\_correctly\_added()

**Returns**

dictionary containing the storage parameters

**Return type**

dict

`multi_vector_simulator.A1_csv_to_json.check_storage_file_is_csv(storage_file)`

Checks that the storage file name defined in *energyStorage.csv* has ending *.csv*.

**Parameters**

**storage\_file** (*str*) – Defined storage file name

**Returns**

- **If test fails** (*MissingCsvEndingError(ValueError), else:*)
- **storage\_file** (*str*) – Storage file name without ending ‘.csv’

`multi_vector_simulator.A1_csv_to_json.conversion(value, asset_dict, row, param, asset, filename=)`

This function converts the input given in the csv to the dict used in the MVS.

When using json files, they are already provided parsed like this functions output.

**Parameters**

- **value** (*Misc.*) – Value to be parsed
- **asset\_dict** (*dict*) – Dict of asset that is to be filled with data
- **row** –
- **param** (*str*) – Parameter that is currently parsed
- **asset** –
- **filename** –

`multi_vector_simulator.A1_csv_to_json.create_input_json(input_directory, pass_back=True)`

Convert csv files to json file as input for the simulation.

Looks at all csv-files in *input\_directory* and compile the information they contain into a json file. The json file is then saved within the *input\_directory* with the filename *CSV\_FNAME*. While reading the csv files, it is checked, whether all required parameters for each component are provided. Missing parameters will return a warning message.

**Parameters**

- **input\_directory** – path of the directory where the input csv files can be found
- **str** – path of the directory where the input csv files can be found
- **pass\_back** – if True the final json dict is returned. Otherwise it is only saved

- **bool** – if True the final json dict is returned. Otherwise it is only saved
- **optional** – if True the final json dict is returned. Otherwise it is only saved

**Return type**

None or dict

```
multi_vector_simulator.A1_csv_to_json.create_json_from_csv(input_directory, filename,
                                                         parameters=None,
                                                         asset_is_a_storage=False)
```

One csv file is loaded and it's parameters are checked. The csv file is then converted to a dictionary; the name of the csv file is used as the main key of the dictionary. Exceptions are made for the files ["economic\_data", "project", "project\_data", "simulation\_settings", "constraints"], here no main key is added. Another exception is made for the file "energyStorage". When this file is processed, the according "storage" files (names of the "storage" columns in "energyStorage" are called and added to the energyStorage Dictionary.

**Parameters**

- **input\_directory** (*str*) – path of the directory where the input csv files can be found
- **filename** (*str*) – name of the input file that is transformed into a json, without extension
- **parameters** (*list*) – List of parameters names that are required
- **asset\_is\_a\_storage** (*bool*) – default value is False. If the function is called by `add_storage_components()` the parameter is set to True

**Returns**

the converted dictionary

**Return type**

dict

**Notes**

Tested with: - `test_default_values_storage_without_thermal_losses()` - `test_default_values_storage_with_thermal_losses()`

### 4.1.3 Data input

**Module B0 - Data input json**

```
multi_vector_simulator.B0_data_input_json.convert_from_json_to_special_types(a_dict,
                                                                              prev_key=None,
                                                                              time_index=None)
```

Convert the field values of the mvs result json file which are not simple types.

The function is recursive to explore all nested levels

**Parameters**

- **a\_dict** (*variable*) – In the recursion, this is either a dict (moving down one nesting level) or a field value
- **prev\_key** (*str*) – The previous key of the dict in the recursive loop

**Returns**

- *The original dictionary, with the serialized instances of pandas.Series,*

- *pandas.DatetimeIndex, pandas.DataFrame, numpy.array converted back to their original form*

`multi_vector_simulator.B0_data_input_json.convert_from_special_types_to_json(o)`

This converts all data stored in dict\_values that is not compatible with the json format to a format that is compatible.

**Parameters**

**o** – Any type. Object to be converted to json-storable value.

**Returns**

json-storable value.

**Return type**

type

`multi_vector_simulator.B0_data_input_json.load_json(path_input_file, path_input_folder=None, path_output_folder=None, move_copy=False, flag_missing_values=True, set_default_values=False)`

Opens and reads json input file and parses it to dict of input parameters.

**Parameters**

- **path\_input\_file** (*str*) – The path to the json file created from csv files
- **path\_input\_folder** (*str, optional*) – The path to the directory where the input CSVs/JSON files are located. Default: 'inputs/'.
- **path\_output\_folder** (*str, optional*) – The path to the directory where the results of the simulation such as the plots, time series, results JSON files are saved by MVS E-Lands. Default: 'MVS\_outputs/'
- **move\_copy** (*bool, optional*) – if this is set to True, the path\_input\_file will be moved to the path\_output\_folder Default: False
- **flag\_missing\_values** (*bool*) – if True, raise MissingParameterError for each missing required parameter
- **set\_default\_values** (*bool*) – if True, set the default value of a missing required parameter which is listed in KNOWN\_EXTRA\_PARAMETERS

**Return type**

dict of all input parameters of the MVS E-Lands simulation

`multi_vector_simulator.B0_data_input_json.retrieve_date_time_info(simulation_settings)`

Updates simulation settings by all time-related parameters. - START\_DATE - END\_DATE - TIME\_INDEX - PERIODS

**Parameters**

**simulation\_settings** (*dict*) – Simulation parameters of the input data

**Return type**

Update simulation\_settings by start date, end date, timeindex, and number of simulation periods

## Notes

Function tested with `test_retrieve_datetimeindex_for_simulation()`

### 4.1.4 Data pre-processing and validity checks

#### Module C0 - Data processing

Module C0 prepares the data read from csv or json for simulation, ie. pre-processes it. - Verify input values with C1 - Identify energyVectors and write them to `project_data/sectors` - Create an excess sink for each bus - Process `start_date/simulation_duration` to `pd.datetimeindex` (future: Also consider `timesteplengths`) - Add economic parameters to json with C2 - Calculate “simulation annuity” used in oemof model - Add demand sinks to energyVectors (this should actually be changed and demand sinks should be added to bus relative to `input_direction`, also see issue #179) - Translate `input_directions/output_directions` to bus names - Add missing cost data to automatically generated objects (eg. DSO transformers) - Read timeseries of assets and store into json (differ between one-column csv, multi-column csv) - Read timeseries for parameter of an asset, eg. efficiency - Parse list of inputs/outputs, eg. for chp - Define dso sinks, sources, transformer stations (this will be changed due to bug #119), also for peak demand pricing - Add a source if a conversion object is connected to a new `input_direction` (bug #186) - Define all necessary energyBusses and add all assets that are connected to them specifically with asset name and label - Multiply *maximumCap* of non-dispatchable sources by `max(timeseries(kWh/kWp))` as the *maximumCap* is limiting the flow but we want to limit the installed capacity (see issue #446)

`multi_vector_simulator.C0_data_processing.add_a_transformer_for_each_peak_demand_pricing_period(dict_values, dict_dso, dict_availability_timeseries)`

Adds transformers that are supposed to model the `peak_demand_pricing` periods for each period. This is changed compared to MVS 0.3.0, as there a peak demand pricing period was added by adding a source, not a transformer.

#### Parameters

- **dict\_values** (*dict*) – dict with all simulation parameters
- **dict\_dso** (*dict*) – dict with all info on the specific dso at hand
- **dict\_availability\_timeseries** (*dict*) – dict with all availability timeseries for each period

#### Returns

- **list\_of\_dso\_energyConversion\_assets** (*list*) – List of names of newly added energy conversion assets,
- *Updated dict\_values with a transformer for each peak demand pricing period*

## Notes

Tested by: - C0.test\_add\_a\_transformer\_for\_each\_peak\_demand\_pricing\_period\_1\_period - C0.test\_add\_a\_transformer\_for\_each\_peak\_demand\_pricing\_period\_2\_periods

`multi_vector_simulator.C0_data_processing.add_asset_to_asset_dict_for_each_flow_direction(dict_values, dict_asset, asset_key)`

Add asset to the asset dict of the busses connected to the `INPUT_DIRECTION` and `OUTPUT_DIRECTION` of the asset.

#### Parameters

- **dict\_values** (*dict*) – All simulation information
- **dict\_asset** (*dict*) – All information of the current asset
- **asset\_key** (*str*) – Key that calls the dict\_asset from dict\_values[asset\_group][key]

**Return type**

Updated dict\_values, with dict\_values[ENERGY\_BUSSES] now including asset dictionaries for each asset connected to a bus.

**Notes**

Tested with: - C0.test\_add\_asset\_to\_asset\_dict\_for\_each\_flow\_direction()

```
multi_vector_simulator.C0_data_processing.add_asset_to_asset_dict_of_bus(bus, dict_values,  
                                                                           asset_key,  
                                                                           asset_label)
```

Adds asset key and label to a bus defined by *energyBusses.csv* Sends an error message if the bus was not included in *energyBusses.csv*

**Parameters**

- **dict\_values** (*dict*) – Dict of all simulation parameters
- **bus** (*str*) – A bus label
- **asset\_key** (*str*) – Key with which a dict\_asset would be called from dict\_values[groups][key]
- **asset\_label** (*str*) – Label of the asset

**Returns**

- Updated dict\_values[ENERGY\_BUSSES] by adding an asset to the busses` ASSET\_DICT
- **EnergyBusses now has following keys** (*LABEL*, *ENERGY\_VECTOR*, *ASSET\_DICT*)

**Notes**

Tested with: - C0.test\_add\_asset\_to\_asset\_dict\_of\_bus() - C0.test\_add\_asset\_to\_asset\_dict\_of\_bus\_ValueError()

```
multi_vector_simulator.C0_data_processing.add_assets_to_asset_dict_of_connected_busses(dict_values)
```

This function adds the assets of the different asset groups to the asset dict of ENERGY\_BUSSES. The asset groups are: ENERGY\_CONVERSION, ENERGY\_PRODUCTION, ENERGY\_CONSUMPTION, ENERGY\_PROVIDERS, ENERGY\_STORAGE

**Parameters**

**dict\_values** (*dict*) – Dictionary with all simulation information

**Return type**

Extends dict\_values[ENERGY\_BUSSES] by an asset\_dict that includes all connected assets.

## Notes

Tested with: - C0.test\_add\_assets\_to\_asset\_dict\_of\_connected\_busses()

`multi_vector_simulator.C0_data_processing.add_economic_parameters(economic_parameters)`

Update economic parameters with annuity factor and CRF

### Parameters

**economic\_parameters** (*dict*) – Economic parameters of the simulation

### Return type

Updated economic parameters

## Notes

Function tested with test\_add\_economic\_parameters()

`multi_vector_simulator.C0_data_processing.add_version_number_used(simulation_settings)`

Add version number to simulation settings

### Parameters

**simulation\_settings** (*dict*) – Dict of simulation settings

### Returns

- Updated dict simulation\_settings with *VERSION\_NUM* equal to local version number.
- *This version number will be added to the json output files.*
- The automatic report generated in *F0* references the version number and date on its own accord.

`multi_vector_simulator.C0_data_processing.all(dict_values)`

Function executing all pre-processing steps necessary :param dict\_values All input data in dict format

:return Pre-processed dictionary with all input parameters

`multi_vector_simulator.C0_data_processing.apply_function_to_single_or_list(function,  
parameter,  
**kwargs)`

Applies function to a paramter or to a list of parameters and returns resut

### Parameters

- **function** (*func*) – Function to be applied to a parameter
- **parameter** (*float/str/boolean or list*) – Parameter, either float/str/boolean or list to be evaluated
- **kwargs** – Miscellaneous arguments for function to be called

### Return type

Processed parameter (single) or list of processed para<meters

`multi_vector_simulator.C0_data_processing.change_sign_of_feedin_tariff(dict_feedin_tariff, dso)`

Change the sign of the feed-in tariff. Additionally, prints a logging.warning in case of the feed-in tariff is entered as negative value in 'energyProviders.csv'.

#todo This only works if the feedin tariff is not defined as a timeseries :param dict\_feedin\_tariff: Dict of feedin tariff with Unit-value pair :type dict\_feedin\_tariff: dict :param dso: Name of the energy provider :type dso: str

**Returns**

**dict\_feedin\_tariff** – Dict of feedin tariff, to be used as input to C0.define\_sink

**Return type**

dict

**Notes**

Tested with: - C0.test\_change\_sign\_of\_feedin\_tariff\_positive\_value() - C0.test\_change\_sign\_of\_feedin\_tariff\_negative\_value()  
- C0.test\_change\_sign\_of\_feedin\_tariff\_zero()

`multi_vector_simulator.C0_data_processing.compute_timeseries_properties(dict_asset)`

Compute peak, aggregation, average and normalize timeseries

**Parameters**

**dict\_asset** (*dict*) – dict of all asset parameters, must contain TIMESERIES key

**Returns**

- *None*
- *Add TIMESERIES\_PEAK, TIMESERIES\_TOTAL, TIMESERIES\_AVERAGE and TIMESERIES\_NORMALIZED*
- *to dict\_asset*

**Notes**

Function tested with - C0.test\_compute\_timeseries\_properties\_TIMESERIES\_in\_dict\_asset() -  
C0.test\_compute\_timeseries\_properties\_TIMESERIES\_not\_in\_dict\_asset()

`multi_vector_simulator.C0_data_processing.define_auxiliary_assets_of_energy_providers(dict_values, dso_name)`

Defines all sinks and sources that need to be added to model the transformer using assets of energyConsumption, energyProduction and energyConversion.

**Parameters**

- **dict\_values** (*dict*) – All simulation parameters
- **dso\_name** (*str*) – the name of the energy provider asset

**Return type**

Updated dict\_values

**Notes**

This function is tested with following pytests: - C0.test\_define\_auxiliary\_assets\_of\_energy\_providers()  
- C0.test\_determine\_months\_in\_a\_peak\_demand\_pricing\_period\_not\_valid() -  
C0.test\_determine\_months\_in\_a\_peak\_demand\_pricing\_period\_valid() - C0.test\_define\_availability\_of\_peak\_demand\_pricing\_a  
- C0.test\_define\_availability\_of\_peak\_demand\_pricing\_assets\_monthly() - C0.test\_define\_availability\_of\_peak\_demand\_pricing  
- C0.test\_add\_a\_transformer\_for\_each\_peak\_demand\_pricing\_period\_1\_period() -  
C0.test\_add\_a\_transformer\_for\_each\_peak\_demand\_pricing\_period\_2\_periods() -  
C0.test\_define\_transformer\_for\_peak\_demand\_pricing() - C0.test\_define\_source() -  
C0.test\_define\_source\_exception\_unknown\_bus() - C0.test\_define\_source\_timeseries\_not\_None() -  
C0.test\_define\_source\_price\_not\_None\_but\_with\_scalar\_value() - C0.test\_define\_sink() -> incomplete -



C0.test\_change\_sign\_of\_feedin\_tariff\_positive\_value() - C0.test\_change\_sign\_of\_feedin\_tariff\_negative\_value()  
 - C0.test\_change\_sign\_of\_feedin\_tariff\_zero()

multi\_vector\_simulator.C0\_data\_processing.define\_availability\_of\_peak\_demand\_pricing\_assets(*dict\_values*,  
*num-*  
*ber\_of\_pricing*  
*months\_in\_a\_p*

Determined the availability timeseries for the later to be defined dso assets for taking into account the peak demand pricing periods.

#### Parameters

- **dict\_values** (*dict*) – All simulation inputs
- **number\_of\_pricing\_periods** (*int*) – Number of pricing periods in a year. Valid: 1,2,3,4,6,12
- **months\_in\_a\_period** (*int*) – Duration of a period

#### Returns

**dict\_availability\_timeseries** – Dict with all availability timeseries for each period

#### Return type

dict

multi\_vector\_simulator.C0\_data\_processing.define\_energy\_vectors\_from\_busses(*dict\_values*)

Identifies all energyVectors used in the energy system by looking at the defined energyBusses. The EnergyVectors later will be used to distribute costs and KPI amongst the sectors

#### Parameters

**dict\_values** (*dict*) – All input data in dict format

#### Return type

Update dict[PROJECT\_DATA] by included energyVectors (LES\_ENERGY\_VECTOR\_S)

#### Notes

Function tested with - C1.test\_define\_energy\_vectors\_from\_busses

multi\_vector\_simulator.C0\_data\_processing.define\_excess\_sinks(*dict\_values*)

Define energy excess sinks for each bus

#### Parameters

**dict\_values** (*dict*) – All simulation parameters

#### Return type

Updates dict\_values

multi\_vector\_simulator.C0\_data\_processing.define\_missing\_cost\_data(*dict\_values*, *dict\_asset*)

#### Parameters

- **dict\_values** –
- **dict\_asset** –

#### Returns

multi\_vector\_simulator.C0\_data\_processing.define\_sink(*dict\_values*, *asset\_key*, *price*,  
*inflow\_direction*, *energy\_vector*, *\*\*kwargs*)

This automatically defines a sink for an oemof-sink object. The sinks are added to the energyConsumption assets.

**Parameters**

- **dict\_values** (*dict*) – All information of the simulation
- **asset\_key** (*str*) – label of the asset to be generated
- **price** (*float*) – Price of dispatch of the asset
- **inflow\_direction** (*str*) – Direction from which energy is provided to the sink
- **kwargs** (*Misc*) – Common parameters: -

**Returns**

- *Updates dict\_values[ENERGY\_BUSSES] if outflow\_direction not in it*
- *Updates dict\_values[ENERGY\_CONSUMPTION] with a new sink*

**Notes**

Examples: - Used to define excess sinks for all energyBusses - Used to define feed-in sink for each DSO

The pytests for this function are not complete. It is started with: - C0.test\_define\_sink() and only the assertion messages are missing

```
multi_vector_simulator.C0_data_processing.define_source(dict_values, asset_key, outflow_direction,  
                                                         energy_vector, emission_factor,  
                                                         price=None, timeseries=None)
```

Defines a source with default input values. If kwargs are given, the default values are overwritten.

**Parameters**

- **dict\_values** (*dict*) – Dictionary to which source should be added, with all simulation parameters
- **asset\_key** (*str*) – key under which the asset is stored in the asset group
- **energy\_vector** (*str*) – Energy vector the new asset should belong to
- **emission\_factor** (*dict*) – Dict with a unit-value pair of the emission factor of the new asset
- **price** (*dict*) – Dict with a unit-value pair of the dispatch price of the source. The value can also be defined though FILENAME and HEADER, making the value of the price a timeseries. Default: None
- **timeseries** (*pd.DataFrame*) – Timeseries defining the availability of the source. Currently not used. Default: None

**Returns**

- *Updates dict\_values[ENERGY\_BUSSES] if outflow\_direction not in it*
- *Standard source defined as*

## Notes

The pytest for this function is not complete. It is started with:

- C0.test\_define\_source() - C0.test\_define\_source\_exception\_unknown\_bus()
- C0.test\_define\_source\_price\_not\_None\_but\_with\_scalar\_value() Missing:
- C0.test\_define\_source\_price\_not\_None\_but\_timeseries(), ie. value defined by FILENAME and HEADER

```
multi_vector_simulator.C0_data_processing.define_transformer_for_peak_demand_pricing(dict_values,
                                                                                     dict_dso,
                                                                                     trans-
                                                                                     former_name,
                                                                                     time-
                                                                                     series_availability)
```

Defines a transformer for peak demand pricing in energyConversion

### Parameters

- **dict\_values** (*dict*) – All simulation parameters
- **dict\_dso** (*dict*) – All values connected to the DSO
- **transformer\_name** (*str*) – label of the transformer to be added
- **timeseries\_availability** (*pd.Series*) – Timeseries of transformer availability. Introduced to cover peak demand pricing.

### Return type

Updated dict\_values with newly added transformer asset in the energyConversion asset group.

```
multi_vector_simulator.C0_data_processing.determine_dispatch_price(dict_values, price, source)
```

This function needs to be re-evaluated.

### Parameters

- **dict\_values** –
- **price** –
- **source** –

```
multi_vector_simulator.C0_data_processing.determine_months_in_a_peak_demand_pricing_period(number_of_pricing_periods,
                                                                                          sim-
                                                                                          u-
                                                                                          la-
                                                                                          tion_period_length)
```

Check if the number of peak demand pricing periods is valid. Warns user that in case the number of periods exceeds 1 but the simulation time is not a year, there could be an unexpected number of timeseries considered. Raises error if number of peak demand pricing periods is not valid.

### Parameters

- **number\_of\_pricing\_periods** (*int*) – Defined in csv, is number of pricing periods within a year
- **simulation\_period\_lenght** (*int*) – Defined in csv, is number of days of the simulation

### Returns

**months\_in\_a\_period** – Number of months that make a period, will be used to determine availability of dso assets

### Return type

float

`multi_vector_simulator.CO_data_processing.energyConsumption(dict_values, group)`

**Parameters**

- **dict\_values** –
- **group** –

**Returns**

`multi_vector_simulator.CO_data_processing.energyConversion(dict_values, group)`

Add lifetime capex (incl. replacement costs), calculate annuity (incl. om), and simulation annuity to each asset

**Parameters**

- **dict\_values** –
- **group** –

**Returns**

`multi_vector_simulator.CO_data_processing.energyProduction(dict_values, group)`

**Parameters**

- **dict\_values** –
- **group** –

**Returns**

`multi_vector_simulator.CO_data_processing.energyProviders(dict_values, group)`

**Parameters**

- **dict\_values** –
- **group** –

**Returns**

`multi_vector_simulator.CO_data_processing.energyStorage(dict_values, group)`

**Parameters**

- **dict\_values** –
- **group** –

**Returns**

`multi_vector_simulator.CO_data_processing.evaluate_lifetime_costs(settings, economic_data, dict_asset)`

Evaluates specific costs of an asset over the project lifetime. This includes: - LIFETIME\_PRICE\_DISPATCH (C2.determine\_lifetime\_price\_dispatch) - LIFETIME\_SPECIFIC\_COST - LIFETIME\_SPECIFIC\_COST\_OM - ANNUITY\_SPECIFIC\_INVESTMENT\_AND\_OM - SIMULATION\_ANNUITY

The DEVELOPMENT\_COSTS are not processed here, as they are not necessary for the optimization.

**Parameters**

- **settings** (*dict*) – dict of simulation settings, including: - EVALUATED\_PERIOD
- **economic\_data** (*dict*) – dict of economic data of the simulation, including - project duration (PROJECT\_DURATION) - discount factor (DISCOUNTFACTOR) - tax (TAX) - CRF - ANNUITY\_FACTOR

- **dict\_asset** (*dict*) – dict of all asset parameters, including - SPECIFIC\_COSTS - SPECIFIC\_COSTS\_OM - LIFETIME

**Returns**

- *Updates asset dict with*
- - *LIFETIME\_PRICE\_DISPATCH (C2.determine\_lifetime\_price\_dispatch)*
- - *LIFETIME\_SPECIFIC\_COST*
- - *LIFETIME\_SPECIFIC\_COST\_OM*
- - *ANNUITY\_SPECIFIC\_INVESTMENT\_AND\_OM*
- - *SIMULATION\_ANNUITY*
- - *SPECIFIC\_REPLACEMENT\_COSTS\_INSTALLED*
- - *SPECIFIC\_REPLACEMENT\_COSTS\_OPTIMIZED*

**Notes**

Tested with: - test\_evaluate\_lifetime\_costs\_adds\_all\_parameters() - Test\_Economic\_KPI.test\_benchmark\_Economic\_KPI\_C2\_E2  
`multi_vector_simulator.C0_data_processing.get_timeseries_multiple_flows(settings, dict_asset, file_name, header)`

**Parameters**

- **dict\_asset** –
- **asset** (*dictionary of the*) –
- **file\_name** –
- **series** (*name of the file to read the time*) –
- **header** –
- **provided** (*name of the column where the timeseries is*) –

`multi_vector_simulator.C0_data_processing.process_all_assets(dict_values)`

defines dict\_values['energyBusses'] for later reference

Processes all assets of the energy system by evaluating them, performing economic pre-calculations and validity checks.

**Parameters**

**dict\_values** (*dict*) – All simulation inputs

**Returns**

**dict\_values** – Updated dict\_values with pre-processes assets, including economic parameters, busses and auxiliary assets like excess sinks and all assets connected to the energyProviders.

**Return type**

dict

## Notes

Tested with: - test\_C0\_data\_processing.test\_process\_all\_assets\_fixcost()

```
multi_vector_simulator.C0_data_processing.process_maximum_cap_constraint(dict_values, group,
                                                                    asset,
                                                                    subasset=None)
```

Processes the maximumCap constraint depending on its value.

- If MaximumCap not in asset dict: MaximumCap is None
- If MaximumCap < installedCap: invalid, MaximumCapValueInvalid raised
- If MaximumCap == 0: invalid, MaximumCap is None
- If group == energyProduction and filename not in asset\_dict (dispatchable assets): pass
- If group == energyProduction and filename in asset\_dict (non-dispatchable assets): MaximumCapNormalized == MaximumCap\*peak(timeseries), MaximumAddCapNormalized == MaximumAddCap\*peak(timeseries)

### Parameters

- **dict\_values** (*dict*) – dictionary of all assets
- **group** (*str*) – Group that the asset belongs to (str). Used to acces sub-asset data and for error messages.
- **asset** (*str*) – asset name
- **subasset** (*str or None*) – subasset name. Default: None.

## Notes

Tested with: - test\_process\_maximum\_cap\_constraint\_maximumCap\_undefined() -  
test\_process\_maximum\_cap\_constraint\_maximumCap\_is\_None() - test\_process\_maximum\_cap\_constraint\_maximumCap\_is\_int()  
- test\_process\_maximum\_cap\_constraint\_maximumCap\_is\_float() - test\_process\_maximum\_cap\_constraint\_maximumCap\_is\_0()  
- test\_process\_maximum\_cap\_constraint\_maximumCap\_is\_int\_smaller\_than\_installed\_cap() -  
test\_process\_maximum\_cap\_constraint\_group\_is\_ENERGY\_PRODUCTION\_fuel\_source() -  
test\_process\_maximum\_cap\_constraint\_group\_is\_ENERGY\_PRODUCTION\_non\_dispatchable\_asset() -  
test\_process\_maximum\_cap\_constraint\_subasset()

### Returns

- *Updates the asset dictionary.*
- *\* Unit of MaximumCap is asset unit*

```
multi_vector_simulator.C0_data_processing.process_normalized_installed_cap(dict_values,
                                                                    group, asset,
                                                                    subasset=None)
```

Processes the normalized installed capacity value based on the installed capacity value and the chosen timeseries.

### Parameters

- **dict\_values** (*dict*) – dictionary of all assets
- **group** (*str*) – Group that the asset belongs to (str). Used to acces sub-asset data and for error messages.
- **asset** (*str*) – asset name

- **subasset** (*str* or *None*) – subasset name. Default: None.

## Notes

Tested with: - test\_process\_normalized\_installed\_cap()

### Return type

Updates the asset dictionary with the normalizedInstalledCap value.

```
multi_vector_simulator.C0_data_processing.receive_timeseries_from_csv(settings, dict_asset,
                                                                    input_type,
                                                                    is_demand_profile=False)
```

### Parameters

- **settings** –
- **dict\_asset** –
- **type** –

### Returns

```
multi_vector_simulator.C0_data_processing.replace_nans_in_timeseries_with_0(timeseries,
                                                                              label)
```

Replaces nans in the timeseries (if any) with 0

### Parameters

- **timeseries** (*pd.Series*) – demand or resource timeseries in dict\_asset (having nan value(s) if any), also of parameters that are not defined as scalars but as timeseries
- **label** (*str*) – Contains user-defined information about the timeseries to be printed into the eventual error message

### Returns

**timeseries** – timeseries without NaN values

### Return type

pd.Series

## Notes

Function tested with - C0.test\_replace\_nans\_in\_timeseries\_with\_0()

```
multi_vector_simulator.C0_data_processing.treat_multiple_flows(dict_asset, dict_values,
                                                                parameter)
```

This function consider the case a technical parameter on the json file has a list of values because multiple inputs or outputs busses are considered. :param dict\_values: :param dictionary of current values of the asset: :param parameter: :param usually efficiency. Different efficiencies will be given if an asset has multiple inputs or outputs busses: :param : :param so a list must be considered.:

## Module C1 - Verification

Module C1 is used to validate the input data compiled in A1 or read in B0.

In A1/B0, the input parameters were parsed to str/bool/float/int. This module tests whether the parameters are in correct value ranges: - Display error message when wrong type - Display error message when outside defined range - Display error message when feed-in tariff > electricity price (would cause loop, see #119)

`multi_vector_simulator.C1_verification.all_valid_intervals(name, value, title)`

Checks whether *value* of *name* is valid.

Checks include the expected type and the expected range a parameter is supposed to be inside.

### Parameters

- **name** –
- **value** –
- **title** –

### Returns

`multi_vector_simulator.C1_verification.check_efficiency_of_storage_capacity(dict_values)`

Raises error or logs a warning to help users to spot major change in PR #676.

In #676 the *efficiency of storage capacity* in *'storage\_\*.csv'* was defined as the storages' efficiency/ability to hold charge over time. Before it was defined as loss rate. This function raises an error if efficiency of 'storage capacity' of one of the storages is 0 and logs a warning if efficiency of 'storage capacity' of one of the storages is <0.2.

### Parameters

**dict\_values** (*dict*) – Contains all input data of the simulation.

## Notes

Tested with: - test\_check\_efficiency\_of\_storage\_capacity\_is\_0 - test\_check\_efficiency\_of\_storage\_capacity\_is\_btw\_0\_and\_02  
- test\_check\_efficiency\_of\_storage\_capacity\_is\_greater\_02

### Returns

- *Indirectly, raises error message in case of efficiency of 'storage capacity' is 0*
- *and logs warning message in case of efficiency of 'storage capacity' is <0.2.*

`multi_vector_simulator.C1_verification.check_emission_factor_of_providers(dict_values)`

Logs a logging.warning message in case the grid has a renewable share of 100 % but an emission factor > 0.

This would affect the optimization if a maximum emissions constraint is used. Additionally, it effects the KPIs connected to emissions.

### Parameters

**dict\_values** (*dict*) – Contains all input data of the simulation.

### Returns

- *Indirectly, logs a logging.warning message in case the grid has a renewable share*
- *of 100 % but an emission factor > 0.*



## Notes

Tested with: - C1.test\_check\_emission\_factor\_of\_providers\_no\_warning\_RE\_share\_lower\_1()  
 - C1.test\_check\_emission\_factor\_of\_providers\_no\_warning\_emission\_factor\_0() -  
 C1.test\_check\_emission\_factor\_of\_providers\_warning()

`multi_vector_simulator.C1_verification.check_energy_system_can_fulfill_max_demand(dict_values)`

Helps to do oemof-solph termination debugging: Logs a logging.warning message if the aggregated installed capacity and maximum capacity (if applicable) of all conversion, generation and storage assets connected to one bus is smaller than the maximum demand. The check is applied to each bus of the energy system. Check passes when the potential peak supply is larger then or equal to the peak demand on the bus, or if the maximum capacity of an asset is set to None when optimizing.

### Parameters

**dict\_values** (*dict*) – Contains all input data of the simulation.

### Returns

- Indirectly, logs a logging.warning message if the installed and maximum capacities of
- conversion/generation/storage assets are less than the maximum demand, for each bus.

## Notes

Tested with: - test\_check\_energy\_system\_can\_fulfill\_max\_demand\_sufficient\_capacities()  
 - test\_check\_energy\_system\_can\_fulfill\_max\_demand\_no\_maximum\_capacity()  
 - test\_check\_energy\_system\_can\_fulfill\_max\_demand\_insufficient\_capacities() -  
 test\_check\_energy\_system\_can\_fulfill\_max\_demand\_with\_storage() - test\_check\_energy\_system\_can\_fulfill\_max\_demand\_suffic  
 - test\_check\_energy\_system\_can\_fulfill\_max\_demand\_insufficient\_dispatchable\_production -  
 test\_check\_energy\_system\_can\_fulfill\_max\_demand\_sufficient\_non\_dispatchable\_production -  
 test\_check\_energy\_system\_can\_fulfill\_max\_demand\_insufficient\_non\_dispatchable\_production -  
 test\_check\_energy\_system\_can\_fulfill\_max\_demand\_fails\_mvs\_runthrough

`multi_vector_simulator.C1_verification.check_feasibility_of_maximum_emissions_constraint(dict_values)`

Logs a logging.warning message in case the maximum emissions constraint could lead into an unbound problem.

If the maximum emissions constraint is used it is checked whether there is any production asset with zero emissions that has a capacity to be optimized without maximum capacity constraint. If this is not the case a warning is logged.

### Parameters

**dict\_values** (*dict*) – Contains all input data of the simulation.

### Returns

- Indirectly, logs a logging.warning message in case the maximum emissions constraint
- is used while no production with zero emissions is optimized without maximum capacity.

## Notes

Tested with: - C1.test\_check\_feasibility\_of\_maximum\_emissions\_constraint\_no\_warning\_no\_constraint() -  
C1.test\_check\_feasibility\_of\_maximum\_emissions\_constraint\_no\_warning\_although\_emission\_constraint()  
- C1.test\_check\_feasibility\_of\_maximum\_emissions\_constraint\_maximumcap() -  
C1.test\_check\_feasibility\_of\_maximum\_emissions\_constraint\_optimizeCap\_is\_False() -  
C1.test\_check\_feasibility\_of\_maximum\_emissions\_constraint\_no\_zero\_emission\_asset()

`multi_vector_simulator.C1_verification.check_feedin_tariff_vs_energy_price(dict_values)`

Raises error if feed-in tariff > energy price of any asset in 'energyProvider.csv'. This is not allowed, as oemof otherwise is subjected to an unbound and unrealistic problem, eg. one where the owner should consume electricity to feed it directly back into the grid for its revenue.

### Parameters

**dict\_values** (*dict*) – Contains all input data of the simulation.

### Returns

- *Indirectly, raises error message in case of feed-in tariff > energy price of any*
- *asset in 'energyProvider.csv'.*

## Notes

Tested with: - C1.test\_check\_feedin\_tariff\_vs\_energy\_price\_greater\_energy\_price() -  
C1.test\_check\_feedin\_tariff\_vs\_energy\_price\_not\_greater\_energy\_price()

`multi_vector_simulator.C1_verification.check_feedin_tariff_vs_levelized_cost_of_generation_of_production`

Raises error if feed-in tariff > levelized costs of generation for energy asset in ENERGY\_PRODUCTION with capacity to be optimized and no maximum capacity constraint.

This is not allowed, as oemof otherwise may be subjected to an unbound problem, ie. a business case in which an asset should be installed with infinite capacities to maximize revenue.

In case of a set maximum capacity or no capacity optimization logging messages are logged.

### Parameters

**dict\_values** (*dict*) – Contains all input data of the simulation.

### Returns

- *Raises error message in case of feed-in tariff > levelized costs of generation for energy asset of any*
- *asset in ENERGY\_PRODUCTION*

## Notes

Tested with: - C1.test\_check\_feedin\_tariff\_vs\_levelized\_cost\_of\_generation\_of\_production\_non\_dispatchable\_not\_greater\_costs()  
- C1.test\_check\_feedin\_tariff\_vs\_levelized\_cost\_of\_generation\_of\_production\_non\_dispatchable\_greater\_costs()  
- C1.test\_check\_feedin\_tariff\_vs\_levelized\_cost\_of\_generation\_of\_production\_dispatchable\_higher\_dispatch\_price()  
- C1.test\_check\_feedin\_tariff\_vs\_levelized\_cost\_of\_generation\_of\_production\_dispatchable\_lower\_dispatch\_price()  
- C1.test\_check\_feedin\_tariff\_vs\_levelized\_cost\_of\_generation\_of\_production\_non\_dispatchable\_greater\_costs\_with\_maxcap()  
- C1.test\_check\_feedin\_tariff\_vs\_levelized\_cost\_of\_generation\_of\_production\_non\_dispatchable\_greater\_costs\_dispatch\_mode()

This test does not cover cross-sectoral invalid feedin tariffs. Example: If there is very cheap electricity generation but a high H2 feedin tariff, then it might be a business case to install a large Electrolyzer, and the simulation would fail. In that case one should set bounds to the solution.

`multi_vector_simulator.C1_verification.check_for_label_duplicates(dict_values)`

This function checks if any LABEL provided for the energy system model in `dict_values` is a duplicate. This is not allowed, as oemof can not build a model with identical labels.

**Parameters**

**dict\_values** (*dict*) – All simulation inputs

**Returns**

pass or error message

**Return type**

DuplicateLabels

`multi_vector_simulator.C1_verification.check_for_sufficient_assets_on_busses(dict_values)`

Validation check for busses, to make sure a sufficient number of assets is connected.

Each bus has to have 3 or more assets connected to it. The reasoning is that each bus needs: - One asset for inflow into the bus - One asset for outflow from the bus - One energy excess asset Note, however, that this test does not check whether the assets actually serve that function, so there might be false negatives: The test can for example pass, if there are two output assets, one excess asset but no input asset, which would represent a non-sensical combination.

On the bus created for the peak demand pricing function (name includes *DSO\_PEAK\_DEMAND\_SUFFIX*) no excess sinks are added, and therefore the rule does not have to be applied to this bus.

**Parameters**

**dict\_values** (*dict*) – All simulation parameters

**Return type**

Logging error message if test fails

## Notes

This function is tested with: - `test_C1_verification.test_check_for_sufficient_assets_on_busses_example_bus_passes()`  
- `test_C1_verification.test_check_for_sufficient_assets_on_busses_example_bus_fails()` -  
`test_C1_verification.test_check_for_sufficient_assets_on_busses_skipped_for_peak_demand_pricing_bus()`

`multi_vector_simulator.C1_verification.check_if_energy_vector_is_defined_in_DEFAULT_WEIGHTS_ENERGY_CARR`

Raises an error message if an energy vector is unknown.

It then needs to be added to the `DEFAULT_WEIGHTS_ENERGY_CARRIERS` in `constants.py`

**Parameters**

- **energy\_carrier** (*str*) – Name of the energy carrier
- **asset\_group** (*str*) – Name of the asset group
- **asset** (*str*) – Name of the asset

**Return type**

None

### Notes

Tested with: - test\_check\_if\_energy\_vector\_is\_defined\_in\_DEFAULT\_WEIGHTS\_ENERGY\_CARRIERS\_pass()  
- test\_check\_if\_energy\_vector\_is\_defined\_in\_DEFAULT\_WEIGHTS\_ENERGY\_CARRIERS\_fails()

`multi_vector_simulator.C1_verification.check_if_energy_vector_of_all_assets_is_valid(dict_values)`

Validates for all assets, whether 'energyVector' is defined within DEFAULT\_WEIGHTS\_ENERGY\_CARRIERS and within the energyBusses.

#### Parameters

**dict\_values** (*dict*) – All input data in dict format

### Notes

Function tested with - test\_add\_economic\_parameters() - test\_check\_if\_energy\_vector\_of\_all\_assets\_is\_valid\_fails  
- test\_check\_if\_energy\_vector\_of\_all\_assets\_is\_valid\_passes

`multi_vector_simulator.C1_verification.check_input_values(dict_values)`

#### Parameters

**dict\_values** –

#### Returns

`multi_vector_simulator.C1_verification.check_non_dispatchable_source_time_series(dict_values)`

Raises error if time series of non-dispatchable sources are not between [0, 1].

#### Parameters

**dict\_values** (*dict*) – Contains all input data of the simulation.

#### Returns

- *Indirectly, raises error message in case of time series of non-dispatchable sources*
- *not between [0, 1].*

`multi_vector_simulator.C1_verification.check_time_series_values_between_0_and_1(time_series)`

Checks whether all values of *time\_series* in [0, 1].

#### Parameters

**time\_series** (*pd.Series*) – Time series to be checked.

#### Returns

True if values of *time\_series* within [0, 1], else False.

#### Return type

bool

`multi_vector_simulator.C1_verification.lookup_file(file_path, name)`

Checks whether file specified in *file\_path* exists.

If it does not exist, a FileNotFoundError is raised.

#### Parameters

- **file\_path** – File name including path of file that is checked.
- **name** – Something referring to which component the file belongs. In `get_timeseries_multiple_flows()` the label of the asset is used.

#### Returns

## Module C2 - Economic preprocessing

Module C2 performs the economic pre-processing of MVS' input parameters. It includes basic economic formulas.

Functionalities: - Calculate annuity factor - calculate crf depending on year - calculate specific lifetime capex, considering replacement costs and residual value of the asset - calculate annuity from present costs - calculate present costs based on annuity - calculate effective fuel price cost, in case there is a annual fuel price change (this functionality still has to be checked in this module)

`multi_vector_simulator.C2_economic_functions.annuity(present_value, crf)`

Calculates the annuity which is a fixed stream of payments incurred by investments in assets

### Parameters

- **present\_value** (*float*) – current equivalent value of a set of future cash flows for an asset
- **crf** (*float*) – ratio used to calculate the present value of an annuity

### Returns

**annuity** – annuity, i.e. payment made at equal intervals

### Return type

float

### Notes

Tested with `test_annuity()`

`multi_vector_simulator.C2_economic_functions.annuity_factor(project_life, discount_factor)`

Calculates the annuity factor, which in turn is used to calculate the present value of annuities (instalments)

### Parameters

- **project\_life** (*int*) – time period over which the costs of the system occur
- **discount\_factor** (*float*) – weighted average cost of capital, which is the after-tax average cost of various capital sources

### Returns

**annuity\_factor** – financial value “annuity factor”. Dividing a present cost by the annuity factor returns its annuity, multiplying an annuity with the annuity factor returns its present value

### Return type

float

### Notes

$$annuityfactor = \frac{1}{discountfactor} - \frac{1}{discountfactor \cdot (1 + discountfactor)^{projectlife}}$$

`multi_vector_simulator.C2_economic_functions.capex_from_investment(investment_t0, lifetime, project_life, discount_factor, tax, age_of_asset, asset_label=)`

Calculates the capital expenditures, also known as CapEx.

CapEx represent the total funds used to acquire or upgrade an asset. The specific capex is calculated by taking into account all future cash flows connected to the investment into one unit of the asset. This includes reinvestments, operation and management costs, dispatch costs as well as a deduction of the residual value at project end.

The residual value is calculated with a linear depreciation of the last investment, ie. as a even share of the last investment over the lifetime of the asset. The remaining value of the asset is translated in a present value and then deducted.

#### Parameters

- **investment\_t0** (*float*) – first investment at the beginning of the project made at year 0
- **lifetime** (*int*) – time period over which investments and re-investments can occur. can be equal to, longer or shorter than project\_life
- **project\_life** (*int*) – time period over which the costs of the system occur
- **discount\_factor** (*float*) – weighted average cost of capital, which is the after-tax average cost of various capital sources
- **tax** (*float*) – compulsory financial charge paid to the government
- **age\_of\_asset** (*int*) – age since asset installation in year
- **asset\_label** (*str*) – name of the asset

#### Returns

- **specific\_capex** (*float*) – Specific capital expenditure for an asset over project lifetime
- **specific\_replacement\_costs\_optimized** (*float*) – Specific replacement costs for the asset capacity to be optimized, needed for E2
- **specific\_replacement\_costs\_already\_installed** (*float*) – replacement costs per unit for the currently already installed assets, needed for E2

#### Notes

Tested with - test\_capex\_from\_investment\_lifetime\_equals\_project\_life() -  
test\_capex\_from\_investment\_lifetime\_smaller\_than\_project\_life() - test\_capex\_from\_investment\_lifetime\_bigger\_than\_project\_li  
multi\_vector\_simulator.C2\_economic\_functions.crf(project\_life, discount\_factor)

Calculates the capital recovery ratio used to determine the present value of a series of equal payments (annuity)

#### Parameters

- **project\_life** – time period over which the costs of the system occur
- **discount\_factor** – weighted average cost of capital, which is the after-tax average cost of various capital sources

#### Returns

capital recovery factor, a ratio used to calculate the present value of an annuity

multi\_vector\_simulator.C2\_economic\_functions.determine\_lifetime\_price\_dispatch(dict\_asset,  
eco-  
nomic\_data)

Determines the price of dispatch of an asset LIFETIME\_PRICE\_DISPATCH and updates the asset info.

It takes into account the asset's future expenditures due to dispatch. Depending on the price data provided, another function is executed.

#### Parameters

- **dict\_asset** (*dict*) – Data of an asset
- **economic\_data** (*dict*) – Economic data, including CRF and ANNUITY\_FACTOR

**Return type**

Updates asset dict

**Notes**

Tested with - test\_determine\_lifetime\_price\_dispatch\_as\_int() - test\_determine\_lifetime\_price\_dispatch\_as\_float()  
 - test\_determine\_lifetime\_price\_dispatch\_as\_list() - test\_determine\_lifetime\_price\_dispatch\_as\_timeseries ()

```
multi_vector_simulator.C2_economic_functions.get_lifetime_price_dispatch_list(dispatch_price,
                                                                              eco-
                                                                              nomic_data)
```

Determines the lifetime dispatch price in case that the dispatch price is a list.

The dispatch\_price can be a list when for example if there are two input flows to a component, eg. water and electricity. There should be a lifetime\_price\_dispatch for each of them.

$$lifetime\_price\_dispatch_i = DISPATCH\_PRICE_i \cdot ANNUITY\_FACTOR \forall i$$

with  $i$  for all list entries

**Parameters**

- **dispatch\_price** (*list*) – Dispatch prices of the asset as a list
- **economic\_data** (*dict*) – Economic data

**Returns**

**lifetime\_price\_dispatch** – List of floats of lifetime dispatch price that the asset will be updated with

**Return type**

list

**Notes**

Tested with - test\_determine\_lifetime\_price\_dispatch\_as\_list() - test\_get\_lifetime\_price\_dispatch\_list()

```
multi_vector_simulator.C2_economic_functions.get_lifetime_price_dispatch_one_value(dispatch_price,
                                                                              eco-
                                                                              nomic_data)
```

Lifetime dispatch price is a scalar value that is calculated with the annuity

By doing this, the operational expenditures, in the simulation only taken into account for a year, can be compared to the investment costs.

$$lifetime\_price\_dispatch = DISPATCH\_PRICE \cdot ANNUITY\_FACTOR$$

**Parameters**

- **dispatch\_price** (*float or int*) – dispatch\_price of the asset
- **economic\_data** (*dict*) – Economic data

**Returns**

**lifetime\_price\_dispatch** – Float that the asset dict is to be updated with

**Return type**

float

## Notes

Tested with - test\_determine\_lifetime\_price\_dispatch\_as\_int() - test\_determine\_lifetime\_price\_dispatch\_as\_float()  
- test\_get\_lifetime\_price\_dispatch\_one\_value()

`multi_vector_simulator.C2_economic_functions.get_lifetime_price_dispatch_timeseries(dispatch_price, eco-  
nomic_data)`

Calculates the lifetime price dispatch for a timeseries. The dispatch\_price can be a timeseries, eg. in case that there is an hourly pricing. .. math:

$$\text{lifetime\_price\_dispatch}(t) = \text{DISPATCH\_PRICE}(t) \cdot \text{ANNUITY\_FACTOR} \text{ for all } t$$

### Parameters

- **dispatch\_price** (`pandas.Series`) – Dispatch price as a timeseries (eg. electricity prices)
- **economic\_data** (`dict`) – Dict of economic data

### Returns

**lifetime\_price\_dispatch** – Lifetime dispatch price that the asset will be updated with

### Return type

float

## Notes

### Tested with

- test\_determine\_lifetime\_price\_dispatch\_as\_timeseries()
- test\_get\_lifetime\_price\_dispatch\_timeseries()

`multi_vector_simulator.C2_economic_functions.get_replacement_costs(age_of_asset,  
project_lifetime,  
asset_lifetime,  
first_time_investment,  
discount_factor,  
asset_label="")`

Calculating the replacement costs of an asset

### Parameters

- **age\_of\_asset** (`int`) – Age in years of an already installed asset
- **project\_lifetime** (`int`) – Project duration in years
- **asset\_lifetime** (`int`) – Lifetime of an asset in years
- **first\_time\_investment** (`float`) – Investment cost of an asset to be installed
- **discount\_factor** (`float`) – Discount factor of a project
- **asset\_label** (`str`) – name of the asset

### Returns

- Per-unit replacement costs of an asset. If `age_asset == 0`, they need to be added to the lifetime\_specific\_costs of the asset.



- If *age\_asset > 0*, it will be needed to calculate the future investment costs of a previously installed asset.

`multi_vector_simulator.C2_economic_functions.present_value_from_annuity(annuity,  
annuity_factor)`

Calculates the present value of future instalments from an annuity

#### Parameters

- **annuity** (*float*) – payment made at equal intervals
- **annuity\_factor** (*float*) – financial value

#### Returns

**present\_value** – present value of future payments from an annuity

#### Return type

float

`multi_vector_simulator.C2_economic_functions.simulation_annuity(annuity, days)`

Scaling annuity to timeframe Updating all annuities above to annuities “for the timeframe”, so that optimization is based on more adequate costs. Includes `project_cost_annuity`, `distribution_grid_cost_annuity`, `main-grid_extension_cost_annuity` for consistency even though these are not used in optimization.

#### Parameters

- **annuity** (*float*) – Annuity of an asset
- **days** (*int*) – Days to be simulated

#### Return type

Simulation annuity that considers the lifetime cost for the optimization of one year duration.

#### Notes

Tested with - `test_simulation_annuity_week` - `test_simulation_annuity_year`

## 4.1.5 Building the energy system model

### Module D0 - Model building

Functional requirements of module D0: - measure time needed to build model - measure time needed to solve model - generate energy system model for oemof - create dictionary of components so that they can be used for constraints and some - raise warning if component not a (in mvs defined) oemof model type - add all energy conversion, energy consumption, energy production, energy storage devices model - plot network graph - at constraints to remote model - store lp file (optional) - start oemof simulation - process results by giving them to the next function - dump oemof results - add simulation parameters to dict values

**class** `multi_vector_simulator.D0_modelling_and_optimization.model_building`

**adding\_assets\_to\_energysystem\_model**(*dict\_model*, *model*, *\*\*kwargs*)

#### Parameters

- **dict\_values** (*dict*) – dict of simulation data
- **dict\_model** – Updated list of assets in the oemof energy system model
- **model** (*oemof.solph.network.EnergySystem*) – Model of oemof energy system

**initialize()**

Initialization of oemof model

**Parameters**

**dict\_values** (*dict*) – dictionary of simulation

**Return type**

oemof energy model (oemof.solph.network.EnergySystem), dict\_model which gathers the assets added to this model later.

**plot\_networkx\_graph(model, save\_energy\_system\_graph=False)**

Plots a graph of the energy system if that graph is to be displayed or stored.

**Parameters**

- **dict\_values** (*dict*) – All simulation inputs
- **model** (*oemof.solph.network.EnergySystem*) – oemof-solph object for energy system model
- **save\_energy\_system\_graph** (*bool*) – if True, save the graph in the mvs output folder  
Default: False

**Return type**

None

**plot\_sankey\_diagramm(model, save\_energy\_system\_graph=False)**

Prepare a sankey diagram of the simulated energy model

**Parameters**

- **dict\_values** (*dict*) – All simulation inputs
- **model**: *oemof.solph.network.EnergySystem*  
oemof-solph object for energy system model
- **save\_energy\_system\_graph** (*bool*) – if True, save the graph in the mvs output folder  
Default: False

**simulating(model, local\_energy\_system)**

Initiates the oemof-solph simulation, accesses results and writes main results into dict

If an error is encountered in the oemof solver, mvs should not be allowed to continue, otherwise other errors related to the uncomplete simulation result might occur and it will be more obscure to the endusers what went wrong.

A MVS error is raised if the oemof solver warning states explicitly that “termination condition infeasible”, otherwise the oemof solver warning is re-raised as an error.

**Parameters**

- **dict\_values** (*dict*) – All simulation inputs
- **model** (*object*) – oemof-solph object for energy system model
- **local\_energy\_system** (*object*) – pyomo object storing all constraints of the energy system model

**Return type**

Updated model with results, main results (flows, assets) and meta results (simulation)

**store\_lp\_file**(*local\_energy\_system*)

Stores linear equation system generated with pyomo as an “lp file”.

**Parameters**

- **dict\_values** (*dict*) – All simulation input data
- **local\_energy\_system** (*object*) – pyomo object including all constraints of the energy system

**Return type**

Nothing.

```
multi_vector_simulator.D0_modelling_and_optimization.run_oemof(dict_values,
                                                                save_energy_system_graph=False,
                                                                return_les=False)
```

Creates and solves energy system model generated from excel template inputs. Each component is included by calling its constructor function in D1\_model\_components.

**Parameters**

- **values** (*dict*) – Includes all dictionary values describing the whole project, including costs, technical parameters and components. In C0\_data\_processing, each component was attributed with a certain in/output bus.
- **save\_energy\_system\_graph** (*bool*) – if set to True, saves a local copy of the energy system’s graph
- **return\_les** (*bool*) – if set to True, the return also includes the local\_energy\_system in third position

**Return type**

saves and returns oemof simulation results

```
class multi_vector_simulator.D0_modelling_and_optimization.timer
```

**initialize**()

Starts a timer

**stop**(*start*)

Ends timer and adds duration of simulation to dict\_values :param dict\_values: Dict of simulation including SIMULATION\_RESULTS key :type dict\_values: dict :param start: start time of timer :type start: times-tamp

**Return type**

Simulation time in dict\_values

## Module D1 - Oemof components

Module D1 includes all functions that are required to build an oemof model with adaptable components.

- add transformer objects (fix, to be optimized)
- add source objects (fix, to be optimized, dispatchable, non-dispatchable)
- add sink objects (fix, to be optimized, dispatchable, non-dispatchable)
- add storage objects (fix, to be optimized)
- add multiple input/output busses if required for each of the assets
- add oemof component parameters as scalar or time series values

```
class multi_vector_simulator.D1_model_components.CustomBus(*args, **kwargs)
```

```
multi_vector_simulator.D1_model_components.bus(model, name, **kwargs)
```

Adds bus *name* to *model* and to ‘busses’ in *kwargs*.

### Notes

Tested with: - test\_bus\_add\_to\_empty\_dict() - test\_bus\_add\_to\_not\_empty\_dict()

```
multi_vector_simulator.D1_model_components.check_optimize_cap(model, dict_asset, func_constant,  
                                                                func_optimize, **kwargs)
```

Defines a component specified in *dict\_asset* with fixed capacity or capacity to be optimized.

### Parameters

- **model** (*oemof.solph.network.EnergySystem object*) – See the oemof documentation for more information.
- **dict\_asset** (*dict*) – Contains information about the asset like (not exhaustive): efficiency, installed capacity (‘installedCap’), information on the busses the asset is connected to (f.e. ‘inflow\_direction’, ‘outflow\_direction’).
- **func\_constant** (*func*) – Function to be applied if optimization not intended.
- **func\_optimize** (*func*) – Function to be applied if optimization not intended.
- **oemof** (*Required are busses and a dictionary belonging to the respective*) –
- **asset.** (*type of the*) –
- **busses** (*dict, optional*) –
- **sinks** (*dict, optional*) –
- **sources** (*dict, optional*) –
- **transformers** (*dict, optional*) –
- **storages** (*dict, optional*) –

### Returns

- Indirectly updated *model* and dict of asset in *kwargs* with the component object.
- *TODOs*
- *~~~~~*
- *Might be possible to drop non invest optimization in favour of invest optimization if max\_capacity*
- *attributes ie. are set to 0 for fix (but less beautiful, and in case of generator even blocks nonconvex opt.).*

## Notes

Tested with: - test\_check\_optimize\_cap\_raise\_error()

`multi_vector_simulator.D1_model_components.chp(model, dict_asset, **kwargs)`

Defines a chp component specified in *dict\_asset*.

Depending on the 'value' of 'optimizeCap' in *dict\_asset* the chp is defined with a fixed capacity or a capacity to be optimized. The chp has single input and multiple output busses.

### Parameters

- **model** (*oemof.solph.network.EnergySystem object*) – See the oemof documentation for more information.
- **dict\_asset** (*dict*) – Contains information about the chp like (not exhaustive): efficiency, installed capacity ('installedCap'), information on the busses the chp is connected to ('in\_flow\_direction', 'outflow\_direction'), beta coefficient.
- **busses** (*dict*) –
- **sinks** (*dict, optional*) –
- **sources** (*dict, optional*) –
- **transformers** (*dict*) –
- **storages** (*dict, optional*) –
- **extractionTurbineCHP** (*dict, optional*) –

## Notes

The transformer has either multiple input or multiple output busses.

The following functions are used for defining the chp: \* *chp\_fix()* \* *chp\_optimize()* for investment optimization

Tested with: - test\_chp\_fix\_cap() - test\_chp\_optimize\_cap() - test\_chp\_missing\_beta()  
 - test\_chp\_wrong\_beta\_formatting() - test\_chp\_wrong\_efficiency\_formatting() -  
 test\_chp\_wrong\_outflow\_bus\_energy\_vector()

### Return type

Indirectly updated *model* and dict of asset in *kwargs* with chp object.

`multi_vector_simulator.D1_model_components.chp_fix(model, dict_asset, **kwargs)`

Extraction turbine chp from Oemof solph. Extraction turbine must have one input and two outputs .. rubric::

Notes

Tested with: - test\_to\_be\_written()

### Return type

Indirectly updated *model* and dict of asset in *kwargs* with the extraction turbine component.

`multi_vector_simulator.D1_model_components.chp_optimize(model, dict_asset, **kwargs)`

Extraction turbine chp from Oemof solph. Extraction turbine must have one input and two outputs .. rubric::

Notes

Tested with: - test\_to\_be\_written()

### Return type

Indirectly updated *model* and dict of asset in *kwargs* with the extraction turbine component.

`multi_vector_simulator.D1_model_components.sink(model, dict_asset, **kwargs)`

Defines a sink component specified in *dict\_asset*.

Depending on the 'value' of 'optimizeCap' in *dict\_asset* the sink is defined with a fixed capacity or a capacity to be optimized. If a time series is provided for the sink (key 'timeseries' in *dict\_asset*) it is defined as a non dispatchable sink, otherwise as dispatchable sink. The sink has multiple or a single input bus depending on the type of the key 'inflow\_direction' in *dict\_asset*.

#### Parameters

- **model** (*oemof.solph.network.EnergySystem object*) – See the oemof documentation for more information.
- **dict\_asset** (*dict*) – Contains information about the storage like (not exhaustive): efficiency, installed capacity ('installedCap'), information on the busses the sink is connected to ('inflow\_direction'),
- **busses** (*dict*) –
- **sinks** (*dict*) –
- **sources** (*dict, optional*) –
- **transformers** (*dict, optional*) –
- **storages** (*dict, optional*) –

#### Notes

The following functions are used for defining the sink: \* [`sink\_non\_dispatchable\(\)`](#) \* [`sink\_dispatchable\(\)`](#)

Tested with: - `test_sink_non_dispatchable_single_input_bus()` - `test_sink_non_dispatchable_multiple_input_busses()`  
- `test_sink_dispatchable_single_input_bus()` - `test_sink_dispatchable_multiple_input_busses()`

`multi_vector_simulator.D1_model_components.sink_dispatchable_optimize(model, dict_asset, **kwargs)`

Define a dispatchable sink.

The dispatchable sink is capacity-optimized, without any costs connected to the capacity of the asset. Applications of this asset type are: Feed-in sink, excess sink.

See [`sink\(\)`](#) for more information, including parameters.

#### Notes

Tested with: - `test_sink_dispatchable_single_input_bus()` - `test_sink_dispatchable_multiple_input_busses()`

#### Return type

Indirectly updated *model* and dict of asset in *kwargs* with the sink object.

`multi_vector_simulator.D1_model_components.sink_non_dispatchable(model, dict_asset, **kwargs)`

Defines a non dispatchable sink.

See [`sink\(\)`](#) for more information, including parameters.

## Notes

Tested with: - test\_sink\_non\_dispatchable\_single\_input\_bus() - test\_sink\_non\_dispatchable\_multiple\_input\_busses()

### Return type

Indirectly updated *model* and dict of asset in *kwargs* with the sink object.

`multi_vector_simulator.D1_model_components.source(model, dict_asset, **kwargs)`

Defines a source component specified in *dict\_asset*.

Depending on the 'value' of 'optimizeCap' in *dict\_asset* the source is defined with a fixed capacity or a capacity to be optimized. If a time series is provided for the source (key 'timeseries' in *dict\_asset*) it is defined as a non dispatchable source, otherwise as dispatchable source. The source has multiple or a single output bus depending on the type of the key 'inflow\_direction' in *dict\_asset*.

### Parameters

- **model** (*oemof.solph.network.EnergySystem object*) – See the oemof documentation for more information.
- **dict\_asset** (*dict*) – Contains information about the storage like (not exhaustive): efficiency, installed capacity ('installedCap'), information on the busses the sink is connected to ('inflow\_direction'),
- **busses** (\* *We should actually not allow multiple output*) –
- **sinks** (*dict*) –
- **sources** (*dict, optional*) –
- **transformers** (*dict, optional*) –
- **storages** (*dict, optional*) –
- **TODOS** –
- **^ ^ ^ ^ ^ ^** –
- **busses** –
- **then** (*probably - because a pv would*) –
- **example** (*feed in twice as much as solar\_gen\_specific for*) –
- **#121** (*see issue*) –

## Notes

The following functions are used for defining the source: \* [source\\_dispatchable\\_fix\(\)](#)  
 \* [source\\_dispatchable\\_optimize\(\)](#) \* [source\\_non\\_dispatchable\\_fix\(\)](#) \*  
[source\\_non\\_dispatchable\\_optimize\(\)](#)

Tested with: - test\_source\_non\_dispatchable\_optimize() - test\_source\_non\_dispatchable\_fix() -  
 test\_source\_dispatchable\_optimize\_normalized\_timeseries() - test\_source\_dispatchable\_optimize\_timeseries\_not\_normalized\_tin  
 - test\_source\_dispatchable\_fix\_normalized\_timeseries() - test\_source\_dispatchable\_fix\_timeseries\_not\_normalized\_timeseries()

`multi_vector_simulator.D1_model_components.source_dispatchable_fix(model, dict_asset, **kwargs)`

Defines a dispatchable source with a fixed capacity.

See [source\(\)](#) for more information, including parameters.

## Notes

Tested with: - test\_source\_dispatchable\_fix\_normalized\_timeseries() - test\_source\_dispatchable\_fix\_timeseries\_not\_normalized\_t

### Return type

Indirectly updated *model* and dict of asset in *kwargs* with the source object.

```
multi_vector_simulator.D1_model_components.source_dispatchable_optimize(model, dict_asset,  
                                                                           **kwargs)
```

Defines a dispatchable source with a fixed capacity.

See [source\(\)](#) for more information, including parameters.

## Notes

Tested with: - test\_source\_dispatchable\_optimize\_normalized\_timeseries() -  
test\_source\_dispatchable\_optimize\_timeseries\_not\_normalized\_timeseries()

### Returns

Indirectly updated *model* and dict of asset in *kwargs* with the source object.

```
multi_vector_simulator.D1_model_components.source_non_dispatchable_fix(model, dict_asset,  
                                                                           **kwargs)
```

Defines a non dispatchable source with a fixed capacity.

See [source\(\)](#) for more information, including parameters.

## Notes

Tested with: - test\_source\_non\_dispatchable\_fix()

### Return type

Indirectly updated *model* and dict of asset in *kwargs* with the source object.

```
multi_vector_simulator.D1_model_components.source_non_dispatchable_optimize(model,  
                                                                              dict_asset,  
                                                                              **kwargs)
```

Defines a non dispatchable source with a capacity to be optimized.

See [source\(\)](#) for more information, including parameters.

## Notes

Tested with: - test\_source\_non\_dispatchable\_optimize()

### Return type

Indirectly updated *model* and dict of asset in *kwargs* with the source object.

```
multi_vector_simulator.D1_model_components.storage(model, dict_asset, **kwargs)
```

Defines a storage component specified in *dict\_asset*.

Depending on the 'value' of 'optimizeCap' in *dict\_asset* the storage is defined with a fixed capacity or a capacity to be optimized.

### Parameters



- **model** (*oemof.solph.network.EnergySystem object*) – See the oemof documentation for more information.
- **dict\_asset** (*dict*) – Contains information about the storage like (not exhaustive): efficiency, installed capacity ('installedCap'), information on the busses the storage is connected to ('inflow\_direction', 'outflow\_direction'),
- **busses** (*dict*) –
- **sinks** (*dict, optional*) –
- **sources** (*dict, optional*) –
- **transformers** (*dict, optional*) –
- **storages** (*dict*) –

### Notes

The following functions are used for defining the storage: \* [storage\\_fix\(\)](#) \* [storage\\_optimize\(\)](#)

Tested with: - [test\\_storage\\_optimize\(\)](#) - [test\\_storage\\_fix\(\)](#)

`multi_vector_simulator.D1_model_components.storage_fix(model, dict_asset, **kwargs)`

Defines a storage with a fixed capacity.

See [storage\(\)](#) for more information, including parameters.

### Notes

Tested with: - [test\\_storage\\_fix\(\)](#)

#### Return type

Indirectly updated *model* and dict of asset in *kwargs* with the storage object.

`multi_vector_simulator.D1_model_components.storage_optimize(model, dict_asset, **kwargs)`

Defines a storage with a capacity to be optimized.

See [storage\(\)](#) for more information, including parameters.

### Notes

Tested with: - [test\\_storage\\_optimize\(\)](#) - [test\\_storage\\_optimize\\_investment\\_minimum\\_0\\_float\(\)](#) - [test\\_storage\\_optimize\\_investment\\_minimum\\_0\\_time\\_series\(\)](#) - [test\\_storage\\_optimize\\_investment\\_minimum\\_1\\_rel\\_float\(\)](#) - [test\\_storage\\_optimize\\_investment\\_minimum\\_1\\_abs\\_float\(\)](#) - [test\\_storage\\_optimize\\_investment\\_minimum\\_1\\_rel\\_times\\_series\(\)](#) - [test\\_storage\\_optimize\\_investment\\_minimum\\_1\\_abs\\_times\\_series\(\)](#)

#### Return type

Indirectly updated *model* and dict of asset in *kwargs* with the storage object.

`multi_vector_simulator.D1_model_components.transformer(model, dict_asset, **kwargs)`

Defines a transformer component specified in *dict\_asset*.

Depending on the 'value' of 'optimizeCap' in *dict\_asset* the transformer is defined with a fixed capacity or a capacity to be optimized. The transformer has multiple or single input or output busses depending on the types of keys 'inflow\_direction' and 'outflow\_direction' in *dict\_asset*.

#### Parameters

- **model** (*oemof.solph.network.EnergySystem object*) – See the oemof documentation for more information.
- **dict\_asset** (*dict*) – Contains information about the transformer like (not exhaustive): efficiency, installed capacity ('installedCap'), information on the busses the transformer is connected to ('inflow\_direction', 'outflow\_direction').
- **busses** (*dict*) –
- **sinks** (*dict, optional*) –
- **sources** (*dict, optional*) –
- **transformers** (*dict*) –
- **storages** (*dict, optional*) –

## Notes

The transformer has either multiple input or multiple output busses.

The following functions are used for defining the transformer: \* [transformer\\_constant\\_efficiency\\_fix\(\)](#)  
\* [transformer\\_constant\\_efficiency\\_optimize\(\)](#)

Tested with: - [test\\_transformer\\_optimize\\_cap\\_single\\_busses\(\)](#) - [test\\_transformer\\_optimize\\_cap\\_multiple\\_input\\_busses\(\)](#)  
- [test\\_transformer\\_optimize\\_cap\\_multiple\\_output\\_busses\(\)](#) - [test\\_transformer\\_fix\\_cap\\_single\\_busses\(\)](#) -  
[test\\_transformer\\_fix\\_cap\\_multiple\\_input\\_busses\(\)](#) - [test\\_transformer\\_fix\\_cap\\_multiple\\_output\\_busses\(\)](#)

### Return type

Indirectly updated *model* and dict of asset in *kwargs* with transformer object.

```
multi_vector_simulator.D1_model_components.transformer_constant_efficiency_fix(model,  
                                                                              dict_asset,  
                                                                              **kwargs)
```

Defines a transformer with constant efficiency and fixed capacity.

See [transformer\(\)](#) for more information, including parameters.

## Notes

Tested with: - [test\\_transformer\\_fix\\_cap\\_single\\_busses\(\)](#) - [test\\_transformer\\_fix\\_cap\\_multiple\\_input\\_busses\(\)](#) -  
[test\\_transformer\\_fix\\_cap\\_multiple\\_output\\_busses\(\)](#)

### Return type

Indirectly updated *model* and dict of asset in *kwargs* with the transformer object.

```
multi_vector_simulator.D1_model_components.transformer_constant_efficiency_optimize(model,  
                                                                              dict_asset,  
                                                                              **kwargs)
```

Defines a transformer with constant efficiency and a capacity to be optimized.

See [transformer\(\)](#) for more information, including parameters.

## Notes

Tested with: - test\_transformer\_optimize\_cap\_single\_busses() - test\_transformer\_optimize\_cap\_multiple\_input\_busses()  
- test\_transformer\_optimize\_cap\_multiple\_output\_busses()

### Return type

Indirectly updated *model* and dict of asset in *kwargs* with the transformer object.

## Module D2 - Model constraints

This module gathers all constraints that can be added to the MVS optimisation. we will probably require another input CSV file or further parameters in simulation\_settings.

Future constraints are discussed in issue #133 (<https://github.com/rl-institut/multi-vector-simulator/issues/133>)

constraints should be tested in-code (examples) and by comparing the lp file generated.

`multi_vector_simulator.D2_model_constraints.add_constraints(local_energy_system, dict_values, dict_model)`

Adds all constraints activated in constraints.csv to the energy system model.

Possible constraints: - Minimal renewable factor constraint :param local\_energy\_system: Energy system model generated from oemof-solph for the energy system scenario, including all energy system assets. :type local\_energy\_system: :oemof-solph: <oemof.solph.model> :param dict\_values: All simulation parameters :type dict\_values: dict :param dict\_model: Dictionary including the oemof-solph component assets, which need to be connected with constraints :type dict\_model: dict of :oemof-solph: <oemof.solph.assets>

### Returns

**local\_energy\_system** – Updated object local\_energy\_system with the additional constraints and bounds.

### Return type

**oemof-solph**  
<oemof.solph.model>

## Notes

The constraints can be validated by evaluating the LP file. Additionally, there are validation tests in *E4\_verification\_of\_constraints*.

Tested with: - D2.test\_add\_constraints\_maximum\_emissions() - D2.test\_add\_constraints\_maximum\_emissions\_None()

- D2.test\_add\_constraints\_minimal\_renewable\_share() - D2.test\_add\_constraints\_minimal\_renewable\_share\_is\_0()

- D2.test\_add\_constraints\_net\_zero\_energy\_requirement\_is\_true() - D2.test\_add\_constraints\_net\_zero\_energy\_requirement\_is\_fa

`multi_vector_simulator.D2_model_constraints.constraint_maximum_emissions(model, dict_values, dict_model=None)`

Resulting in an energy system adhering to a maximum amount of emissions.

### Parameters

- **model** – Model to which constraint is added.
- **dict\_values** (*dict*) – All simulation parameters
- **dict\_model** (*None*) – To match other constraint function's format

## Notes

Tested with: - D2.test\_constraint\_maximum\_emissions()

```
multi_vector_simulator.D2_model_constraints.constraint_minimal_degree_of_autonomy(model,
                                                                                    dict_values,
                                                                                    dict_model)
```

Resulting in an energy system adhering to a minimal degree of autonomy.

Please be aware that the minimal degree of autonomy is not applied to each sector individually, but to the overall energy system (via energy carrier weighting).

### Parameters

- **model** – Model to which constraint is added.
- **dict\_values** (*dict*) – All simulation parameters
- **dict\_model** (*dict of :oemof-solph: <oemof.solph.assets>*) – Dictionary including the oemof-solph component assets, which need to be connected with constraints

## Notes

The renewable factor of the whole energy system has to adhere for following constraint:

$$\text{minimaldegreeofautonomy} \cdot \left( \sum \text{localdemand} \cdot \text{weightingfactor} \right) \leq \sum \text{localdemand} \cdot \text{weightingfactor} - \sum \text{consumption}$$

Tested with: - Test\_Constraints.test\_benchmark\_minimal\_degree\_of\_autonomy()

```
multi_vector_simulator.D2_model_constraints.constraint_minimal_renewable_share(model,
                                                                                dict_values,
                                                                                dict_model)
```

Resulting in an energy system adhering to a minimal renewable factor.

Please be aware that the renewable factor that has to adhere to the minimal renewable factor is not the one of one specific sector, but of the overall energy system. This means that eg. 1 kg H2 is produced renewably, it goes into account with a heavier weighting factor than one renewably produced electricity unit.

### Parameters

- **model** – Model to which constraint is added.
- **dict\_values** (*dict*) – All simulation parameters
- **dict\_model** (*dict of :oemof-solph: <oemof.solph.assets>*) – Dictionary including the oemof-solph component assets, which need to be connected with constraints

## Notes

The renewable factor of the whole energy system has to adhere for following constraint:

$$\text{minimalrenewablefactor} \leq \frac{\sum \text{renewablegeneration} \cdot \text{weightingfactor}}{\sum \text{renewablegeneration} \cdot \text{weightingfactor} + \sum \text{non-renewablegeneration} \cdot \text{weightingfactor}}$$

Tested with: - Test\_Constraints.test\_benchmark\_minimal\_renewable\_share\_constraint()

`multi_vector_simulator.D2_model_constraints.constraint_net_zero_energy(model, dict_values, dict_model)`

Resulting in an energy system that is a net zero energy (NZE) or plus energy system.

Please be aware that the NZE constraint is not applied to each sector individually, but to the overall energy system (via energy carrier weighting).

#### Parameters

- **model** – Model to which constraint is added.
- **dict\_values** (*dict*) – All simulation parameters
- **dict\_model** (*dict of :oemof-solph: <oemof.solph.assets>*) – Dictionary including the oemof-solph component assets, which need to be connected with constraints

#### Notes

The constraint reads as follows:

$$\sum_i E_{feedin,DSO}(i) \cdot w_i - E_{consumption,DSO}(i) \cdot w_i \geq 0$$

Tested with: - `Test_Constraints.test_net_zero_energy_constraint()`

`multi_vector_simulator.D2_model_constraints.prepare_constraint_minimal_renewable_share(dict_values, dict_model)`

Prepare creating the minimal renewable factor constraint by processing *dict\_values*

#### Parameters

- **dict\_values** (*dict*) – All simulation parameters
- **dict\_model** (*dict of :oemof-solph: <oemof.solph.assets>*) – Dictionary including the oemof-solph component assets, which need to be connected with constraints

#### Returns

- **renewable\_assets** (*dict*) – Dictionary of all assets with renewable generation in the energy system. Defined by: `oemof_solph_object_asset`, `weighting_factor_energy_carrier`, `renewable_share_asset_flow`, `oemof_solph_object_bus`
- **non\_renewable\_assets** (*dict*) – Dictionary of all assets with renewable generation in the energy system. Defined by: `oemof_solph_object_asset`, `weighting_factor_energy_carrier`, `renewable_share_asset_flow`, `oemof_solph_object_bus`

`multi_vector_simulator.D2_model_constraints.prepare_demand_assets(dict_values, dict_model)`

Perpare demand assets by processing *dict\_values*

Used for the following constraints: - minimal degree of autonomy

#### Parameters

- **dict\_values** (*dict*) – All simulation parameters
- **dict\_model** (*dict of :oemof-solph: <oemof.solph.assets>*) – Dictionary including the oemof-solph component assets, which need to be connected with constraints

## Notes

Tested with: - test\_prepare\_demand\_assets()

### Returns

**demands** – Dictionary of all assets with all demands in the energy system. Defined by: oemof\_solph\_object\_asset, weighting\_factor\_energy\_carrier, oemof\_solph\_object\_bus

### Return type

dict

multi\_vector\_simulator.D2\_model\_constraints.**prepare\_energy\_provider\_consumption\_sources**(*dict\_values*,  
*dict\_model*)

Prepare energy provider consumption sources by processing *dict\_values*.

Used for the following constraints: - minimal degree of autonomy - net zero energy (NZE)

### Parameters

- **dict\_values** (*dict*) – All simulation parameters
- **dict\_model** (*dict of :oemof-solph: <oemof.solph.assets>*) – Dictionary including the oemof-solph component assets, which need to be connected with constraints

## Notes

Tested with: - test\_prepare\_energy\_provider\_consumption\_sources()

### Returns

**energy\_provider\_consumption\_sources** – Dictionary of all assets that are sources for the energy consumption from energy providers in the energy system. Defined by: oemof\_solph\_object\_asset, weighting\_factor\_energy\_carrier, oemof\_solph\_object\_bus

### Return type

dict

multi\_vector\_simulator.D2\_model\_constraints.**prepare\_energy\_provider\_feedin\_sinks**(*dict\_values*,  
*dict\_model*)

Prepare energy provider feedin sinks by processing *dict\_values*.

Used for the following constraints: - net zero energy (NZE)

### Parameters

- **dict\_values** (*dict*) – All simulation parameters
- **dict\_model** (*dict of :oemof-solph: <oemof.solph.assets>*) – Dictionary including the oemof-solph component assets, which need to be connected with constraints

## Notes

Tested with:

- test\_prepare\_energy\_provider\_feedin\_sinks()

### Returns

**energy\_provider\_feedin\_sinks** – Dictionary of all assets that are sinks for the energy feed-in to energy providers in the energy system. Defined by: oemof\_solph\_object\_asset, weighting\_factor\_energy\_carrier, oemof\_solph\_object\_bus

**Return type**  
dict

## 4.1.6 Post-processing and evaluation

### Module E0 - evaluation

Module E0 evaluates the oemof results and calculates the KPI - define dictionary entry for kpi matrix - define dictionary entry for cost matrix - store all results to matrix

`multi_vector_simulator.E0_evaluation.evaluate_dict(dict_values, results_main, results_meta)`

Processes all simulation outputs by evaluating oemof results, asset capacities and dispatch as well as all KPIs.

#### Parameters

- **dict\_values** (*dict*) – simulation parameters
- **results\_main** (*DataFrame*) – oemof simulation results as output by `processing.results()`
- **results\_meta** (*DataFrame*) – oemof simulation meta information as output by `processing.meta_results()`

#### Notes

Tested with: - `test_E0_evaluation.test_evaluate_dict_append_new_fields()` -

`test_E0_evaluation.test_evaluate_dict_important_fields_in_output_dict()` - `test_E0_evaluation.test_evaluate_dict_fields_values_in_`

`multi_vector_simulator.E0_evaluation.initialize_kpi(dict_values)`

Adds basic structure of KPI to dict\_values to gather them later on.

#### Parameters

**dict\_values** (*dict*) – All simulation data, but without any results

#### Return type

Updated dict\_values with KPI structure, made up from KPI\_COST\_MATRIX, KPI\_SCALAR\_MATRIX and KPI\_SCALARS\_DICT.

`multi_vector_simulator.E0_evaluation.process_fixcost(dict_values)`

Adds fix costs of the project to the economic evaluation of the energy system.

#### Parameters

**dict\_values** (*dict*) – All simulation data with inputs and results of the assets

#### Return type

Updated dict\_values with costs attributed in dict values also appended to the dict\_values[KPI] (scalar results)

## Notes

Function is tested with: - `test_E0_evaluation.test_process_fixcost()`

`multi_vector_simulator.E0_evaluation.store_result_matrix(dict_kpi, dict_asset, fix_cost=False)`

Storing results to vector and then result matrix for saving it in csv. Defined value types: Str, bool, None, dict (with key "VALUE"), else (int, float)

### Parameters

- **dict\_kpi** (*dict*) – dictionary with the two kpi groups (costs and scalars), which are pd.DF
- **dict\_asset** (*dict*) – all information known for a specific asset
- **fix\_cost** (*Boolean*) – If `fix_cost` is `True`, then no new row is added to KPI\_SCALAR\_MATRIX, as there are no KPI to update. Costs in KPI\_COST\_MATRIX however are added.

### Return type

Updated `dict_kpi` DF, with new row of kpis of the specific asset.

## Module E1 process results

Module E1 processes the oemof results. - receive time series per bus for all assets - write time series to dictionary - get optimal capacity of optimized assets - add the evaluation of time series

`multi_vector_simulator.E1_process_results.add_info_flows(evaluated_period, dict_asset, flow, type=None, bus_name=None)`

Adds *flow* and total flow amongst other information to *dict\_asset*.

### Parameters

- **evaluated\_period** (*int*) – The number of days simulated with the energy system model.
- **dict\_asset** (*dict*) – Contains information about the asset *flow* belongs to.
- **flow** (*pd.Series*) – Time series of the flow.
- **type** (*str, default: None*) – type of the flow, only exception is "STORAGE\_CAPACITY".
- **bus\_name** (*str or None*) – The name of the current bus (for asset connected to more than one bus)

### Returns

- Indirectly updates *dict\_asset* with the *flow*, the total flow, the annual
- *total flow, the maximum of the flow ('peak\_flow') and the average value of*
- *the flow ('average\_flow'). As Storage capacity is not a flow, an aggregation of the timeseries does not make sense*
- *and the parameters TOTAL\_FLOW, ANNUAL\_TOTAL\_FLOW, PEAK\_FLOW, AVERAGE\_FLOW are added set to None.*



## Notes

Tested with: - E1.test\_add\_info\_flows\_365\_days() - E1.test\_add\_info\_flows\_1\_day() -  
E1.test\_add\_info\_flows\_storage\_capacity()

`multi_vector_simulator.E1_process_results.convert_components_to_dataframe(dict_values)`

Dataframe used for the component table of the report

### Parameters

**dict\_values** (*dict*) – output values of MVS

### Return type

`pandas.DataFrame`

## Notes

### Tested with:

- `test_E1_process_results.test_convert_components_to_dataframe()`

`multi_vector_simulator.E1_process_results.convert_cost_matrix_to_dataframe(dict_values)`

Dataframe used for the cost matrix table of the report

### Parameters

**dict\_values** (*dict*) – output values of MVS

### Return type

`pandas.DataFrame`

`multi_vector_simulator.E1_process_results.convert_costs_to_dataframe(dict_values)`

Dataframe used for the costs piecharts of the report

### Parameters

**dict\_values** (*dict*) – output values of MVS

### Return type

`pandas.DataFrame`

`multi_vector_simulator.E1_process_results.convert_demand_to_dataframe(dict_values,  
sector_demands=None)`

Dataframe used for the demands table of the report

### Parameters

- **dict\_values** (*dict*) – output values of MVS
- **sector\_demands** (*str*) – Name of the sector of the energy system whose demands must be returned as a df by this function Default: None

### Return type

`pandas.DataFrame`

`multi_vector_simulator.E1_process_results.convert_kpi_sector_to_dataframe(dict_values)`

Processes the sector KPIs so that they can be included in the report

### Parameters

**dict\_values** (*dict*) – output values of MVS

### Returns

**kpi\_sectors\_dataframe** – Dataframe to be displayed as a table in the report

**Return type**

pandas.DataFrame

**Notes**

Currently, as the KPI\_UNCOUPLED\_DICT does not hold any units, the table printed in the report is unit-less.

`multi_vector_simulator.E1_process_results.convert_scalar_matrix_to_dataframe(dict_values)`

Dataframe used for the scalar matrix table of the report

**Parameters**

**dict\_values** (*dict*) – output values of MVS

**Return type**

pandas.DataFrame

`multi_vector_simulator.E1_process_results.convert_scalars_to_dataframe(dict_values)`

Processes the scalar system-wide KPI so that they can be included in the report

**Parameters**

**dict\_values** (*dict*) – output values of MVS

**Returns**

**kpi\_scalars\_dataframe** – Dataframe to be displayed as a table in the report

**Return type**

pandas.DataFrame

**Notes**

Currently, as the KPI\_SCALARS\_DICT does not hold any units, the table printed in the report is unit-less.

`multi_vector_simulator.E1_process_results.cut_below_micro(value, label)`

Function trims results of oemof optimization to positive values and rounds to 0, if within a certain precision threshold (of  $-10^{-6}$ )

Oemof termination is dependent on the simulation settings of oemof solph. Thus, it can terminate the optimization if the results are with certain bounds, which can sometimes lead to negative decision variables (capacities, flows). Negative values do not make sense in this context. If the values are between  $-10^{-6}$  and 0, we assume that they can be rounded to 0, as they result from the precision settings of the solver. In that case the value is overwritten for the further post-processing. This should also avoid SOC timeseries with doubtful values outside of [0,1]. If any value is a higher negative value then the threshold, its value is not changed but a warning raised. Similarly, if a positive decision variable is detected that has a value lower then the threshold, it is assumed that this only happens because of the solver settings, and the values below the threshold are rounded to 0.

**Parameters**

- **value** (*float or pd.Series*) – Decision variable determined by oemof
- **label** (*str*) – String to be mentioned in the debug messages

**Returns**

**value** – Decision variable with rounded values in case that slight negative values or positive values were observed.

**Return type**

float of pd.Series

## Notes

Tested with:

- E1.test\_cut\_below\_micro\_scalar\_value\_below\_0\_larger\_threshold -
- E1.test\_cut\_below\_micro\_scalar\_value\_below\_0\_smaller\_threshold - E1.test\_cut\_below\_micro\_scalar\_value\_0
- E1.test\_cut\_below\_micro\_scalar\_value\_larger\_0 - E1.test\_cut\_below\_micro\_scalar\_value\_larger\_0\_smaller\_threshold
- E1.test\_cut\_below\_micro\_pd\_Series\_below\_0\_larger\_threshold - E1.test\_cut\_below\_micro\_pd\_Series\_below\_0\_smaller\_threshold
- E1.test\_cut\_below\_micro\_pd\_Series\_0 - E1.test\_cut\_below\_micro\_pd\_Series\_larger\_0 -
- E1.test\_cut\_below\_micro\_pd\_Series\_larger\_0\_smaller\_threshold

```
multi_vector_simulator.E1_process_results.get_flow(settings, bus, dict_asset, flow_tuple,
                                                    multi_bus=None)
```

Adds flow of *bus* and total flow amongst other information to *dict\_asset*.

Depending on *direction* the input or the output flow is used.

### Parameters

- **settings** (*dict*) – Contains simulation settings from *simulation\_settings.csv* with additional information like the amount of time steps simulated in the optimization ('periods').
- **bus** (*dict*) –  
  
Contains information about a specific bus. Information about the scalars, if they exist, like investment or initial capacity in key 'scalars' (pd.Series) and the flows between the component and the bus(es) in key 'sequences' (pd.DataFrame).
- **dict\_asset** (*dict*) – Contains information about the asset.
- **flow\_tuple** (*tuple*) – Entry of the oemof-solph outputs to be evaluated
- **multi\_bus** (*str or None*) – The name of the current bus (for asset connected to more than one bus)

### Returns

- Indirectly updates *dict\_asset* with the flow of *bus*, the total flow, the annual
- *total flow, the maximum of the flow ('peak\_flow') and the average value of*
- *the flow ('average\_flow').*

```
multi_vector_simulator.E1_process_results.get_optimal_cap(bus, dict_asset, flow_tuple)
```

Retrieves optimized capacity of asset specified in *dict\_asset*.

### Parameters

- **bus** (*dict*) – Contains information about the busses linked to the asset specified in *dict\_asset*. Information about the scalars like investment or initial capacity in key 'scalars' (pd.Series) and the flows between the component and the busses in key 'sequences' (pd.DataFrame).
- **dict\_asset** (*dict*) – Contains information about the asset.
- **flow\_tuple** (*tuple*) – Key of the oemof-solph outputs dict mapping the value to be evaluated

### Returns

- Indirectly updated *dict\_asset* with optimal capacity to be added
- (*'optimizedAddCap'*).
- *TODOS*
- ^^^^

- \* *direction as optimal parameter or with default value None* (*direction is* – not needed if ‘optimizeCap’ is not in *dict\_asset* or if it’s value is False)

`multi_vector_simulator.E1_process_results.get_parameter_to_be_evaluated_from_oemof_results(asset_group, as-set_label)`

Determine the parameter that needs to be evaluated to determine an asset’s optimized flow and capacity.

#### Parameters

- **asset\_group** (*str*) – Asset group to which the evaluated asset belongs
- **asset\_label** (*str*) – Label of the asset, needed for log message

#### Returns

**parameter\_to\_be\_evaluated** – Parameter that will be processed to get the dispatch and capacity of an asset

#### Return type

*str*

### Notes

Tested by: - `test_get_parameter_to_be_evaluated_from_oemof_results()`

`multi_vector_simulator.E1_process_results.get_results(settings, bus_data, dict_asset, asset_group)`

Reads results of the asset defined in *dict\_asset* and stores them in *dict\_asset*.

#### Parameters

- **settings** (*dict*) – Contains simulation settings from *simulation\_settings.csv* with additional information like the amount of time steps simulated in the optimization (‘periods’).
- **bus\_data** (*dict*) – Contains information about all busses in a nested dict. 1st level keys: bus names; 2nd level keys:
  - ‘scalars’: (pd.Series) (does not exist in all dicts) ‘sequences’: (pd.DataFrame) - contains flows between components and busses
- **dict\_asset** (*dict*) – Contains information about the asset.
- **asset\_group** (*str*) – Asset group to which the evaluated asset belongs

#### Return type

Indirectly updates *dict\_asset* with results.

`multi_vector_simulator.E1_process_results.get_state_of_charge_info(dict_asset)`

Adds state of charge timeseries and average value of the timeseries to the storage dict.

#### Parameters

**dict\_asset** (*dict*) – Dict of the asset, specifically including the STORAGE\_CAPACITY

#### Return type

Updated *dict\_asset*

## Notes

Tested with: - `E1.test_get_state_of_charge_info()`

`multi_vector_simulator.E1_process_results.get_storage_results(settings, storage_bus, dict_asset)`

Reads storage results of simulation and stores them in *dict\_asset*.

### Parameters

- **settings** (*dict*) – Contains simulation settings from *simulation\_settings.csv* with additional information like the amount of time steps simulated in the optimization ('periods').
- **storage\_bus** (*dict*) – Contains information about the storage bus. Information about the scalars like investment or initial capacity in key 'scalars' (pd.Series) and the flows between the component and the busses in key 'sequences' (pd.DataFrame).
- **dict\_asset** (*dict*) – Contains information about the storage like capacity, charging power, etc.

### Returns

- Indirectly updates *dict\_asset* with simulation results concerning the
- *storage*.

`multi_vector_simulator.E1_process_results.get_timeseries_per_bus(dict_values, bus_data)`

Reads simulation results of all busses and stores time series.

### Parameters

- **dict\_values** (*dict*) – Contains all input data of the simulation.
- **bus\_data** (*dict* Contains information about all busses in a nested dict.  
) – 1st level keys: bus names; 2nd level keys:  
    'scalars': (pd.Series) (does not exist in all dicts) 'sequences': (pd.DataFrame) - contains  
    flows between components and busses

## Notes

Tested with: - `test_get_timeseries_per_bus_two_timeseries_for_directly_connected_storage()`

#Todo: This is a duplicate of the *E1.get\_flow()* assertions, and thus *E1.cut\_below\_micro* is applied twice for each flow. This should rather be merged into the other functions.

### Return type

Indirectly updated *dict\_values* with 'optimizedFlows' - one data frame for each bus.

`multi_vector_simulator.E1_process_results.get_tuple_for_oemof_results(asset_label, asset_group, bus)`

Determines the tuple with which to access the oemof-solph results

The order of the parameters in the tuple depends on the direction of the flow. If the asset is defined... a) ...by its influx from a bus, the bus has to be named first in the tuple b) ...by its outflux into a bus, the asset has to be named first in the tuple

### Parameters

- **asset\_label** (*str*) – Name of the asset
- **asset\_group** (*str*) – Asset group the asset belongs to
- **bus** (*str*) – Bus that is to be accessed for the asset's information

**Returns**

**flow\_tuple** – Keys to be accessed in the oemof-solph results

**Return type**

tuple of str

**Notes**

Tested with - test\_get\_tuple\_for\_oemof\_results()

`multi_vector_simulator.E1_process_results.get_units_of_cost_matrix_entries(dict_economic, kpi_list)`

Determines the units of the costs KPI to be stored to :class: DataFrame.

**Parameters**

- **dict\_economic** – Economic project data
- **kpi\_list** – List of cost matrix entries

**Returns**

**unit\_list** – List of units for the :class: DataFrame to be created

**Return type**

list

`multi_vector_simulator.E1_process_results.translate_optimizeCap_from_boolean_to_yes_no(optimize_cap)`

Translates the boolean OPTIMIZE\_CAP to a yes-no value for readability of auto report

**Parameters**

**optimize\_cap** (*bool*) – Setting whether asset is optimized or not

**Returns**

**optimize** – If OPTIMIZE\_CAP==True: “Yes”, else “No”.

**Return type**

str

**Notes**

Tested with: - test\_E1\_process\_results.test\_translate\_optimizeCap\_from\_boolean\_to\_yes\_no()

## Module E2 - Economic processing

The module processes the simulation results regarding economic parameters: - calculate lifetime expenditures based on variable energy flows - calculate lifetime investment costs - calculate present value of an asset - calculate revenue - calculate yearly cash flows of whole project for project lifetime (cash flow projection) - calculate fuel price expenditures calculate upfront investment costs - calculate operation management costs (FOM) - calculate upfront investment costs (UIC) - calculate annuity per asset - calculate annuity for the whole project - calculate net present value - calculate levelised cost of energy - calculate levelised cost of energy carriers (electricity, H2, heat)

**exception** `multi_vector_simulator.E2_economics.MissingParametersForEconomicEvaluation`

Warning if one or more parameters are missing for economic post-processing of an asset

`multi_vector_simulator.E2_economics.all_list_in_dict(dict_asset, list)`

Checks if all items of a list are withing the keys of a dictionary

**Parameters**

- **dict\_asset** (*dict*) – Dict with the keys to be evaluated
- **list** (*list*) – List of keys (parameter in strings) that should be in dict

**Returns**

**boolean** – True: All items in keys of the dict False: At least one item is not in keys of the dict

**Return type**

bool

```
multi_vector_simulator.E2_economics.calculate_costs_replacement(specific_replacement_of_initial_capacity,
                                                                spe-
                                                                cific_replacement_of_optimized_capacity,
                                                                initial_capacity,
                                                                optimized_capacity)
```

Calculate (the present value of) the replacement costs over the project lifetime

**Parameters**

- **specific\_replacement\_of\_initial\_capacity** (*float*) – Per-unit replacement costs of an asset that was pre-existing at the location
- **specific\_replacement\_of\_optimized\_capacity** (*float*) – Per-unit replacement costs of an asset that is to be installed
- **initial\_capacity** (*float*) – Initial capacity installed
- **optimized\_capacity** (*float*) – Additional capacity to be installed, as optimized

**Returns**

**costs\_replacements** – Aggregated replacement costs over the project lifetime

**Return type**

float

```
multi_vector_simulator.E2_economics.calculate_costs_upfront_investment(specific_cost, capacity,
                                                                        development_costs)
```

Calculate investment costs of an asset Depending on the *specific\_cost* provided, either the total asset's lifetime investment costs or the upfront investment costs are calculated,

**Parameters**

- **specific\_cost** (*float*) –
  - a) Specific per-unit investment costs of an asset over its lifetime, including all replacement costs
  - b) Specific per-unit investment costs of an asset in year 0
- **capacity** (*float*) – Capacity to be installed
- **development\_costs** (*float*) – Fix development costs, ie. an expense not related to the capacity that is installed. Could be planning costs of the asset.

**Returns**

**costs\_investment** – a) Total investment costs of an asset over its lifetime, including all replacement costs b) Upfront investment costs in year 0

**Return type**

float

```
multi_vector_simulator.E2_economics.calculate_dispatch_expenditures(dispatch_price, flow, asset)
```

Calculate the expenditures connected to an asset due to its dispatch

**Parameters**

- **dispatch\_price** (*float, int or pd.Series*) – Dispatch price of an asset (variable costs), ie. how much has to be paid for each unit of dispatch Raises error if type does not match a) lifetime\_price\_dispatch (taking into account all years of operation) b) price\_dispatch (taking into account one year of operation)
- **flow** (*pd.Series*) – Dispatch of the asset
- **asset** (*str*) – Label of the asset

**Returns**

- *a) Total dispatch expenditures of an asset*
- *b) Annual dispatch expenditures of an asset*

`multi_vector_simulator.E2_economics.calculate_operation_and_management_expenditures(specific_om_cost, in-stalled_capacity, optimized_add_capacity)`

Calculate operation and management expenditures

**Parameters**

- **specific\_om\_cost** (*float*) –
  - a) specific operation and management costs per unit in year 0
  - b) specific operation and management costs per unit for the whole project lifetime
- **installed\_capacity** (*float*) – Capacity installed initially
- **optimized\_add\_capacity** (*float*) – Capacity installed within the optimization scenario

**Returns**

**costs\_operation\_and\_management** – a) Operation and management expenditures in year 0 b) Total operation and management expenditures over the project lifetime

**Return type**

float

`multi_vector_simulator.E2_economics.calculate_total_asset_costs_over_lifetime(costs_investment, cost_operational_expenditures)`

Calculate costs of an asset over whole lifetime

**Parameters**

- **costs\_investment** (*float*) – Investment costs over whole lifetime
- **cost\_operational\_expenditures** (*float*) – Operation and management as well as dispatch expenditures over whole lifetime

**Returns**

**total\_asset\_costs\_over\_lifetime** – costs of an asset over whole lifetime, including upfront investment costs, development costs, replacement costs, operation and management expenditures, dispatch expenditures

**Return type**

float



`multi_vector_simulator.E2_economics.calculate_total_capital_costs(upfront, replacement)`

Calculate total capital expenditures

**Parameters**

- **upfront** (*float*) – Upfront investments at t=0, including development costs
- **replacement** (*float*) – Replacement costs of pre-installed and new assets

**Returns**

**cost\_total\_investment** – Total capital costs

**Return type**

float

`multi_vector_simulator.E2_economics.calculate_total_operational_expenditures(operation_and_management_expenditures, dispatch_expenditures)`

Calculate total expenditures of an asset (operational costs)

**Parameters**

- **operation\_and\_management\_expenditures** (*float*) –
  - a) operation and management expenditures per annum for the installed capacity
  - b) operation and management expenditures for whole project lifetime for the installed capacity
- **dispatch\_expenditures** (*float*) –
  - a) dispatch expenditures per annum for the installed capacity
  - b) dispatch expenditures for whole project lifetime for the installed capacity

**Returns**

**total\_operational\_expenditures** – a) total operational expenditures per annum for installed capacity b) total operational expenditures for whole project lifetime for installed capacity

**Return type**

float

`multi_vector_simulator.E2_economics.get_costs(dict_asset, economic_data)`

Calculates economic KPI of the asset handed to the function

**Parameters**

- **dict\_asset** (*dict*) – Asset to be evaluated. Warning messages in place in case that the asset should not be evaluated.
- **economic\_data** (*dict*) – Economic data of the project

**Returns**

- *Updated dict\_asset with following KPI*
- - *COST\_INVESTMENT*
- - *COST\_UPFRONT*
- - *COST\_REPLACEMENT*
- - *COST\_TOTAL*
- - *COST\_OM*
- - *COST\_DISPATCH*

- - *COST\_OPERATIONAL\_TOTAL*
- - *ANNUITY\_TOTAL*
- - *ANNUITY\_OM*
- *Tested with*
- - *test\_all\_cost\_info\_parameters\_added\_to\_dict\_asset()*
- - *Test\_Economic\_KPI.test\_benchmark\_Economic\_KPI\_C2\_E2()*

`multi_vector_simulator.E2_economics.lcoe_assets(dict_asset, asset_group)`

Calculates the levelized cost of electricity (lcoe) of each asset. [Follow this link for information](docs/MVS\_Outputs.rst)

#### Parameters

- **dict\_asset** (*dict*) – Dictionary defining an asset
- **asset\_group** (*str*) – Defining to which asset group the asset belongs

#### Returns

- *Updates the asset dictionary with the calculated LCOE\_ASSET.*
- **Storages have four values LCOE\_ASSET** (*One for the overall storage including all costs, and one each for the components.*)

#### Notes

$$LCOE\_ASSET = \frac{A}{E_{throughput}}$$

If  $E_{throughput} = 0$ ,  $LCOE\_ASSET = 0$

### Module E3 - Indicator calculation

In module E3 the technical KPI are evaluated: - calculate renewable share - calculate degree of autonomy (DA) - calculate degree of net zero energy (NZE) - calculate total generation of each asset and total\_internal\_generation - calculate total feedin electricity equivalent - calculate energy flows between sectors - calculate degree of sector coupling - calculate onsite energy fraction (OEF) - calculate onsite energy matching (OEM)

`multi_vector_simulator.E3_indicator_calculation.add_degree_of_autonomy(dict_values)`

Determines degree of autonomy and adds KPI to dict\_values

#### Parameters

**dict\_values** (*dict*) – dict with all project information and results, after applying total\_renewable\_and\_non\_renewable\_energy\_origin and total\_demand\_and\_excess\_each\_sector

#### Returns

- *None* – updated dict\_values with the degree of autonomy
- *Tested with*
- - *test\_add\_degree\_of\_autonomy()*

`multi_vector_simulator.E3_indicator_calculation.add_degree_of_net_zero_energy(dict_values)`

Determines degree of net zero energy (NZE) and adds KPI to dict\_values.

**Parameters**

**dict\_values** (*dict*) – dict with all project information and results, after applying total\_renewable\_and\_non\_renewable\_energy\_origin and total\_demand\_and\_excess\_each\_sector

**Returns**

updated dict\_values with the degree of net zero energy

**Return type**

None

**Notes**

As for other KPI, we apply a weighting based on Electricity Equivalent.

Tested with - test\_add\_degree\_of\_net\_zero\_energy()

`multi_vector_simulator.E3_indicator_calculation.add_degree_of_sector_coupling(dict_values)`

Determines the aggregated flows in between the sectors and the Degree of Sector Coupling.

Takes into account the value of different energy carriers.

**Parameters**

**dict\_values** (*dict*) – dictionary with all project inputs and results, specifically the energy-Conversion assets and the outputs.

**Returns**

- *Energy equivalent of total conversion flows*
- .. *math:: - E\_{\{conversion,eq\}} = \sum\_{\{i\}} \{E\_{\{conversion\}}(i) \cdot w\_i\}* with i are conversion assets

`multi_vector_simulator.E3_indicator_calculation.add_levelized_cost_of_energy_carriers(dict_values)`

Adds levelized costs of all energy carriers and overall system to the scalar KPI.

**Parameters**

**dict\_values** (*dict*) – All simulation inputs and results

**Returns**

Updated KPI\_SCALAR\_DICT

**Return type**

Add *ATTRIBUTED\_COSTS* and *LCOeq* for each energy carrier as well as *LCOeq* for overall energy system

**Notes**

**Tested with:**

- test\_E3\_indicator\_calculation.test\_add\_levelized\_cost\_of\_energy\_carriers\_one\_sector()
- test\_E3\_indicator\_calculation.test\_add\_levelized\_cost\_of\_energy\_carriers\_two\_sectors()

`multi_vector_simulator.E3_indicator_calculation.add_onsite_energy_fraction(dict_values)`

Determines onsite energy fraction (OEF), i.e. self-consumption, and adds KPI to dict\_values

**Parameters**

**dict\_values** (*dict*) – dict with all project information and results after applying total\_renewable\_and\_non\_renewable\_energy\_origin

**Returns**

- *None* – updated dict\_values with onsite energy fraction KPI
- *Tested with*
- - *test\_add\_onsite\_energy\_fraction()*

`multi_vector_simulator.E3_indicator_calculation.add_onsite_energy_matching(dict_values)`

Determines onsite energy matching (OEM), i.e. self-sufficiency, and adds KPI to dict\_values

**Parameters**

**dict\_values** (*dict*) – dict with all project information and results after applying total\_renewable\_and\_non\_renewable\_energy\_origin and total\_demand\_and\_excess\_each\_sector and add\_onsite\_energy\_fraction

**Returns**

- *None* – updated dict\_values with onsite energy matching KPI
- *Tested with*
- - *test\_add\_onsite\_energy\_matching()*

`multi_vector_simulator.E3_indicator_calculation.add_renewable_factor(dict_values)`

Determination of renewable share of one sector

**Parameters**

**dict\_values** – dict with all project information and results, after applying add\_total\_renewable\_and\_non\_renewable\_energy\_origin

**Returns**

updated dict\_values with renewable factor of each sector as well as system-wide indicator

**Return type**

type

**Notes**

Updates the KPI with RENEWABLE\_FACTOR for each sector as well as system-wide KPI.

Tested with - *test\_renewable\_factor\_one\_sector* - *test\_renewable\_factor\_two\_sectors* - *TestTechnicalKPI.renewable\_factor\_and\_renewable\_share\_of\_local\_generation()*

`multi_vector_simulator.E3_indicator_calculation.add_renewable_share_of_local_generation(dict_values)`

Determination of renewable share of local energy production

**Parameters**

**dict\_values**

dict with all project information and results, after applying add\_total\_renewable\_and\_non\_renewable\_energy\_origin

**sector**

Sector for which renewable share is being calculated

**:returns:** updated `dict_values` with renewable share of each sector as well as the system-wide KPI  
**:rtype:** type  
**.. rubric:: Notes**  
 Updates the KPI with `RENEWABLE_SHARE_OF_LOCAL_GENERATION` for each sector as well as system-wide KPI.  
 Tested with  
 \* `test_renewable_share_of_local_generation_one_sector()`  
 \* `test_renewable_share_of_local_generation_two_sectors()`  
 \* `TestTechnicalKPI.renewable_factor_and_renewable_share_of_local_generation()`

`multi_vector_simulator.E3_indicator_calculation.add_specific_emissions_per_electricity_equivalent(dict_values)`

Calculates the specific emissions of the energy system per kWh<sub>eq</sub> and adds KPI to `dict_values`.

#### Parameters

**dict\_values** (`dict`) – All simulation inputs and results including `TOTAL_EMISSIONS` calculated in `E3.calculate_emissions_from_flow`.

#### Notes

This function is run after `E3.calculate_emissions_from_flow`.

Tested with: - `E3.test_add_specific_emissions_per_electricity_equivalent()`

#### Returns

Updated `dict_values` with `SPECIFIC_EMISSIONS_ELEQ` in kgCO<sub>2</sub>eq/kWh<sub>eq</sub> (`UNIT_SPECIFIC_EMISSIONS`).

#### Return type

None

`multi_vector_simulator.E3_indicator_calculation.add_total_consumption_from_provider_electricity_equivalent(dict_values)`

Determines the total consumption from energy providers with weighting of electricity equivalent.

#### Parameters

**dict\_values** (`dict`) – dict with all project information and results

#### Returns

updated `dict_values` with KPI : - `TOTAL_CONSUMPTION_FROM_PROVIDERS + electricity`, - `TOTAL_CONSUMPTION_FROM_PROVIDERS + electricity + SUFFIX_ELECTRICITY_EQUIVALENT - TOTAL_CONSUMPTION_FROM_PROVIDERS + SUFFIX_ELECTRICITY_EQUIVALENT`

#### Return type

None

#### Notes

Tested with: - `E3.test_add_total_consumption_from_provider_electricity_equivalent()` - `E3.test_add_total_consumption_from_provider_electricity_equivalent_two_providers_one_energy_carrier()`

`multi_vector_simulator.E3_indicator_calculation.add_total_emissions(dict_values)`

Calculates the total emission of the energy system in kgCO<sub>2</sub>eq/a and adds KPI to `dict_values`.

#### Parameters

**dict\_values** (`dict`) – All simulation inputs and results

**Returns**

Updated *dict\_values* with TOTAL\_EMISSIONS of the energy system in kgCO2eq/a (UNIT\_EMISSIONS).

**Return type**

None

**Notes**

Tested with: - E3.test\_add\_total\_emissions()

`multi_vector_simulator.E3_indicator_calculation.add_total_feedin_electricity_equivalent(dict_values)`

Determines the total grid feed-in with weighting of electricity equivalent.

**Parameters**

**dict\_values** (*dict*) – dict with all project information and results

**Returns**

- *None* – updated dict\_values with KPI : total feedin
- *Tested with*
- - *test\_add\_total\_feedin\_electricity\_equivalent()*
- - *test\_add\_total\_feedin\_electricity\_equivalent\_two\_providers\_one\_energy\_carrier*

`multi_vector_simulator.E3_indicator_calculation.add_total_renewable_and_non_renewable_energy_origin(dict_values)`

Identifies all renewable generation assets and sums up their total generation to total renewable generation

**Parameters**

**dict\_values** – dict with all project input data and results up to E0

**Returns**

Updated dict\_values with total internal/overall renewable and non-renewable energy origin

**Return type**

type

**Notes**

Tested with - test\_total\_renewable\_and\_non\_renewable\_origin\_of\_each\_sector()

`multi_vector_simulator.E3_indicator_calculation.all_totals(dict_values)`

Calculate sum of all cost parameters

**Parameters**

**dict\_values** – dict all input parameters and results up to E0

**Returns**

List of all total cost parameters for the project

**Return type**

type

## Notes

The totals are calculated for following parameters: - costs\_total - costs\_om\_total - costs\_investment\_over\_lifetime - costs\_upfront\_in\_year\_zero - costs\_dispatch - costs\_cost\_om - annuity\_total - annuity\_om

The levelized\_cost\_of\_energy\_of\_asset are dropped from the list, as they do not hold any actual meaning for the whole energy system. The LCOE of the energy system is calculated separately.

`multi_vector_simulator.E3_indicator_calculation.calculate_electricity_equivalent_for_a_set_of_aggregate`

Calculates the electricity equivalent for a dict of aggregated flows and writes it to the KPI

### Parameters

- **dict\_values** (*dict*) – All simulation parameters
- **dict\_of\_aggregated\_flows** (*dict*) – Dict of aggregated flows, with keys of energy carriers.
- **kpi\_name** (*str*) – Name of the KPI to write to the results

### Return type

Updated dict\_values.

`multi_vector_simulator.E3_indicator_calculation.calculate_emissions_from_flow(dict_asset)`

Calculates the total emissions of the asset in 'dict\_asset' in kg per year.

### Parameters

**dict\_asset** (*dict*) – Contains information about the asset.

## Notes

Tested with: - E3.test\_calculate\_emissions\_from\_flow() - E3.test\_calculate\_emissions\_from\_flow\_zero\_emissions

### Returns

Updated *dict\_asset* with TOTAL\_EMISSIONS of the asset in kgCO<sub>2</sub>eq/a (UNIT\_EMISSIONS).

### Return type

None

`multi_vector_simulator.E3_indicator_calculation.equation_degree_of_autonomy(total_consumption_from_energy_provider, total_demand)`

Calculates the degree of autonomy (DA).

The degree of autonomy describes the relation of how much demand is supplied by local generation (as opposed to grid consumption) compared to the total demand of the system.

### Parameters

- **total\_consumption\_from\_energy\_provider** (*float*) – total energy consumption from providers
- **total\_demand** (*float*) – total demand

### Returns

- *float* – degree of autonomy
- .. 
$$DA = \frac{\sum_i \{E_{\{demand\}}(i) \cdot w_i\}}{\sum_i \{E_{\{consumption, provider, j\}}(j) \cdot w_j\}}$$

- **A DA = 0** (*Demand is fully supplied by DSO consumption*)
- **DA = 1** (*System is autonomous, ie. no DSO consumption is necessary*)
- **Notice** (*As above, we apply a weighting based on Electricity Equivalent.*)
- *Tested with*
- *- test\_equation\_degree\_of\_autonomy()*

`multi_vector_simulator.E3_indicator_calculation.equation_degree_of_net_zero_energy(total_feedin, to-grid_consumption, to-demand)`

Calculates the degree of net zero energy (NZE).

In NZE systems import and export of energy is allowed while the balance over one year should be zero, thus the degree of net zero energy would be 1. The Degree of net zero energy indicates how close the system gets to the NZE ideal. If more energy is exported than imported it is plus-energy system (degree of NZE > 1).

#### Parameters

- **total\_feedin** (*float*) – total grid feed-in in electricity equivalents
- **total\_grid\_consumption** (*float*) – total consumption from energy provider in electricity equivalents
- **total\_demand** (*float*) – total demand in electricity equivalents

#### Returns

degree of net zero energy

#### Return type

float

#### Notes

$$\text{Degree of NZE} = 1 + \frac{(\sum_i E_{\text{gridfeedin}}(i) \cdot w_i - E_{\text{gridconsumption}}(i) \cdot w_i)}{\sum_i E_{\text{demand},i} \cdot w_i}$$

Degree of NZE = 1 : System is a net zero energy system, as  $E_{\text{feedin}} = E_{\text{grid\_consumption}}$  Degree of NZE > 1 : system is a plus-energy system, as  $E_{\text{feedin}} > E_{\text{grid\_consumption}}$  Degree of NZE < 1 : system does not reach net zero balance. The degree indicates by how much it fails to do so. Degree of NZE = 0 : system has no internal production, as  $E_{\text{dem}} = E_{\text{grid\_consumption}}$ .

Tested with - test\_equation\_degree\_of\_net\_zero\_energy() - test\_equation\_degree\_of\_net\_zero\_energy\_is\_zero()  
- test\_equation\_degree\_of\_net\_zero\_energy\_is\_one() - test\_equation\_degree\_of\_net\_zero\_energy\_greater\_one()

`multi_vector_simulator.E3_indicator_calculation.equation_degree_of_sector_coupling(total_flow_of_energy_conversion, to-demand_equivalent)`

Calculates degree of sector coupling.

#### Parameters

- **total\_flow\_of\_energy\_conversion\_equivalent** (*float*) – Energy equivalent of total conversion flows
- **total\_demand\_equivalent** (*float*) – Energy equivalent of total energy demand

#### Returns



- *float* – Degree of sector coupling based on conversion flows and energy demands in electricity equivalent.
- .. *math::* –  

$$DSC = \frac{\sum_{i,j} \{E_{\{conversion\}}(i,j) \cdot w_i\}}{\sum_i \{E_{\{demand\}}(i) \cdot w_i\}}$$
with  $i,j \in [Electricity, H2, \dots]$

`multi_vector_simulator.E3_indicator_calculation.equation_levelized_cost_of_energy_carrier`(*cost\_total*,  
*crf*,  
*total\_flow\_energy\_carrier\_eleq*,  
*total\_demand\_electricity\_equivalent*)

Calculates LCOE of each energy carrier of the system.

Based on distributing the NPC of the energy system over the total weighted energy demand of the local energy system. This avoids that a conversion asset has to be defined as being used for a specific sector only, or that an energy production asset (eg. electricity) which is mainly used for powering energy conversion assets for another energy carrier (eg. H2) are increasing the costs of the first energy carrier (electricity), even though the costs should be attributed to the costs of the end-use of generation.

#### Parameters

- **cost\_total** (*float*) –
- **crf** (*float*) –
- **total\_flow\_energy\_carrier\_eleq** (*float*) –
- **total\_demand\_electricity\_equivalent** (*float*) –
- **total\_flow\_energy\_carrier** (*float*) –

#### Returns

- **lcoe\_energy\_carrier** (*float*) – Levelized costs of an energy carrier in a sector coupled system
- **attributed\_costs** (*float*) – Costs attributed to a specific energy carrier

#### Notes

Please refer to the conference paper presented at the CIRED Workshop Berlin (see [readthedocs](#)) for more detail.

The costs attributed to an energy carrier are calculated from the ratio of electricity equivalent of the energy carrier demand in focus to the electricity equivalent of the total demand:

$$attributedcosts = NPC \cdot \frac{Total\ electricity\ equivalent\ of\ energy\ carrier\ demand}{Total\ electricity\ equivalent\ of\ demand}$$

The LCOE sets these attributed costs in relation to the energy carrier demand (in its original unit):

$$LCOE_{energy\ carrier} = \frac{attributedcosts \cdot CRF}{total\ energy\ carrier\ demand}$$

Tested with: - `test_equation_levelized_cost_of_energy_carrier_total_demand_electricity_equivalent_larger_0_total_flow_energy_carrier_eleq`  
- `test_equation_levelized_cost_of_energy_carrier_total_demand_electricity_equivalent_larger_0_total_flow_energy_carrier_is_0`  
- `test_equation_levelized_cost_of_energy_carrier_total_demand_electricity_equivalent_is_0_total_flow_energy_carrier_is_0()`

`multi_vector_simulator.E3_indicator_calculation.equation_onsite_energy_fraction`(*total\_generation*,  
*total\_feedin*)

Calculates onsite energy fraction (OEF), i.e. self-consumption.

OEF describes the fraction of all locally generated energy that is consumed by the system itself.

#### Parameters

- **total\_generation** (*float*) – Energy equivalent of total generation flows
- **total\_feedin** (*float*) – Total feed into the grid

#### Returns

- *float* – Onsite energy fraction.
- .. *math::* –  $OEF = \frac{\sum_i \{E_{\{generation\}}(i) \cdot w_i\} - E_{\{gridfeedin\}}(i) \cdot w_i}{\sum_i \{E_{\{generation\}}(i) \cdot w_i\}}$  &OEF epsilon text{[0,1]}
- *Tested with*
- - *test\_equation\_onsite\_energy\_fraction()*

`multi_vector_simulator.E3_indicator_calculation.equation_onsite_energy_matching`(*total\_generation*,  
*total\_feedin*,  
*total\_excess*,  
*total\_demand*)

Calculates onsite energy matching (OEM), i.e. self-sufficiency.

OEM describes the fraction of the total demand that can be covered by the locally generated energy.

#### Parameters

- **total\_generation** (*float*) – Energy equivalent of total conversion flows
- **total\_feedin** (*float*) – Total feed into the grid
- **total\_excess** (*float*) – Total Excess energy
- **total\_demand** (*float*) – Total demand

#### Returns

- *Onsite energy matching.*
- .. *math::* –  $OEM = \frac{\sum_i \{E_{\{generation\}}(i) \cdot w_i\} - E_{\{gridfeedin\}}(i) \cdot w_i - E_{\{excess\}}(i) \cdot w_i}{\sum_i \{E_{\{demand\}}(i) \cdot w_i\}}$  &OEM epsilon text{[0,1]}
- *Tested with*
- - *test\_equation\_onsite\_energy\_matching()*

`multi_vector_simulator.E3_indicator_calculation.equation_renewable_share`(*total\_res*,  
*total\_non\_res*)

Calculates the renewable share

#### Parameters

- **total\_res** – Renewable generation of a system
- **total\_non\_res** – Non-renewable generation of a system

**Returns**

Renewable share

**Return type**

type

**Notes**

Used both to calculate RENEWABLE\_FACTOR and RENEWABLE\_SHARE\_OF\_LOCAL\_GENERATION.

Equation:

$$RES = \frac{total_{res}}{total_{onres} + total_{res}}$$

The renewable share is relative to generation, but not consumption of energy, the renewable share can not be larger 1. If there is no generation or consumption from a DSO within an energyVector and supply is solely reached by energy conversion from another vector, the renewable share is defined to be zero.

- renewable share = 1 - all energy in the energy system is of renewable origin
- renewable share < 1 - part of the energy in the system is of renewable origin
- renewable share = 0 - no energy is of renewable origin

Tested with:

- test\_renewable\_share\_equation\_no\_generation()
- test\_renewable\_share\_equation\_below\_1()
- test\_renewable\_share\_equation\_is\_0()
- test\_renewable\_share\_equation\_is\_1()

`multi_vector_simulator.E3_indicator_calculation.total_demand_and_excess_each_sector(dict_values)`

Calculation of the total demand and total excess of each sector

Both in original energy carrier unit and electricity equivalent

**Parameters**

**dict\_values** – dict with all project input data and results up to E0

**Returns**

- *Updated KPI\_SCALARS\_DICT with*
- *- total demand of each energy carrier (original unit)*
- *- total demand of each energy carrier (electricity equivalent)*
- *- total demand in electricity equivalent*
- *- total excess of each energy carrier (original unit)*
- *- total excess of each energy carrier (electricity equivalent)*
- *- total excess in electricity equivalent*

## Notes

Tested with - test\_add\_levelized\_cost\_of\_energy\_carriers\_one\_sector() - test\_add\_levelized\_cost\_of\_energy\_carriers\_two\_sectors  
- TestTechnicalKPI.renewable\_factor\_and\_renewable\_share\_of\_local\_generation()

`multi_vector_simulator.E3_indicator_calculation.weighting_for_sector_coupled_kpi(dict_values, kpi_name)`

Calculates the weighted kpi for a specific kpi\_name both for a single sector and system-wide

### Parameters

- **dict\_values** – dict with all project information and results, including KPI\_UNCOUPLED\_DICT with the specific kpi\_name in question
- **kpi\_name** – str with the kpi which should be weighted

### Returns

Append specific KPI that describes sector-coupled system to dict\_values[KPI][KPI\_SCALARS\_DICT] Appends specific KPI in energy equivalent to each sector to dict\_values[KPI][KPI\_UNCOUPLED\_DICT]

### Return type

type

## Module E4 - Verification of results

- Detect excessive excess generation
- Verify that minimal renewable share constraint is adhered to
- Verify that maximum emission constraint is adhered to
- Verify that net zero energy (NZE) constraint is adhered to

`multi_vector_simulator.E4_verification.detect_excessive_excess_generation_in_bus(dict_values)`

Warning for any bus with excessive excess generation is given.

A logging.warning message is printed when the ratio between total outflows and total inflows of a bus is < 0.9.

### Parameters

**dict\_values** –

### Returns

- - *Nothing if there is no excessive excess generation*
- - *Prints logging.warning message for every bus with excessive excess generation.*

`multi_vector_simulator.E4_verification.maximum_emissions_test(dict_values)`

Tests if maximum emissions constraint was correctly applied.

### Parameters

**dict\_values (dict)** – all input parameters and results up to E0

### Returns

- *Nothing*
- - *Nothing if the constraint is confirmed*
- - *Prints logging.warning message if the difference from the constraint is < 10<sup>-6</sup>.*
- - *Prints a logging.error message if the difference from the constraint is >= 10<sup>-6</sup>.*

## Notes

Tested with: - E4.test\_maximum\_emissions\_test\_passes() - E4.test\_maximum\_emissions\_test\_fails()

`multi_vector_simulator.E4_verification.minimal_constraint_test(dict_values, minimal_constraint, bounded_result)`

Test if minimal constraint was correctly applied

### Parameters

- **dict\_values** (*dict*) – Dict of all simulation information
- **minimal\_constraint** (*str*) – Key to access the value of the minimal bound of parameter subjected to constraint to be tested
- **bounded\_result** (*str*) – Key to access the value of the resulting parameter to be compared to minimal\_bound

### Returns

- *Nothing*
- - *Nothing if the constraint is confirmed*
- - *Prints logging.warning message if the deviation from the constraint is  $< 10^{-6}$ .*
- - *Prints a logging.error message if the deviation from the constraint is  $\geq 10^{-6}$ .*

## Notes

Executed to test - MINIMAL\_DEGREE\_OF\_AUTONOMY vs. RENEWABLE\_FACTOR - MINIMAL\_RENEWABLE\_FACTOR vs. DEGREE\_OF\_AUTONOMY

Tested with: - E4.test\_minimal\_constraint\_test\_passes() - E4.test\_minimal\_constraint\_test\_fails()

`multi_vector_simulator.E4_verification.net_zero_energy_constraint_test(dict_values)`

Tests if net zero energy constraint was correctly applied.

### Parameters

**dict\_values** (*dict*) – all input parameters and results up to E0

### Returns

- *Nothing*
- - *Nothing if the constraint is confirmed*
- - *Prints logging.warning message if the difference from the constraint is  $< 10^{-6}$ .*
- - *Prints a logging.error message if the difference from the constraint is  $\geq 10^{-6}$ .*

## Notes

Tested with: - E4.test\_net\_zero\_energy\_constraint\_test\_fails() - E4.test\_net\_zero\_energy\_constraint\_test\_passes()

`multi_vector_simulator.E4_verification.verify_state_of_charge(dict_values)`

This function checks the state of charge of each storage component It raises warning log messages if the SoC has a physically infeasible value

### Parameters

**dict\_values** (*dict*) – Dictionary with all information regarding the simulation, specifically including the energyStorage assets

**Returns**

- - *Nothing if there are no physically infeasible SoC values for the storage components*
- - *Prints log messages to console and log file if there are physically impossible SoC values*

**Notes**

Tested with: - `test_E4_verification.test_verify_state_of_charge_feasible()` -  
`test_E4_verification.test_verify_state_of_charge_soc_below_zero()` - `test_E4_verification.test_verify_state_of_charge_soc_above`

## 4.1.7 Output

### Module F0 - Output

The model F0 output defines all functions that store evaluation results to file. - Aggregate demand profiles to a total demand profile - Plot all energy flows for both 14 and 365 days for each energy bus - Store timeseries of all energy flows to excel (one sheet = one energy bus) - Execute function: plot optimised capacities as a barchart (F1) - Execute function: plot all annuities as a barchart (F1) - Store scalars/KPI to excel - Process dictionary so that it can be stored to Json - Store dictionary to Json

```
multi_vector_simulator.F0_output.evaluate_dict(dict_values, path_pdf_report=None,  
                                              path_png_figs=None)
```

This is the main function of F0. It calls all functions that prepare the simulation output, ie. Storing all simulation output into excellent files, bar charts, and graphs.

**Parameters**

- **dict\_values** – dict Of all input and output parameters up to F0
- **path\_pdf\_report** ((str)) – if provided, generate a pdf report of the simulation to the given path
- **path\_png\_figs** ((str)) – if provided, generate png figures of the simulation's results to the given path

**Returns**

NA

**Return type**

type

```
multi_vector_simulator.F0_output.parse_simulation_log(path_log_file, dict_values)
```

Gather a log file with several log messages, this function gathers them all and inputs them into the dict with all input and output parameters up to F0

**Parameters**

- **path\_log\_file** (str/None) – path to the mvs log file Default: None
- **dict\_values** – dict Of all input and output parameters up to F0

**Return type**

Updates the results dictionary with the log messages of the simulation

## Notes

This function is tested with: - test\_F0\_output.TestLogCreation.test\_parse\_simulation\_log

`multi_vector_simulator.F0_output.store_as_json(dict_values, output_folder=None, file_name=None)`

Converts dict\_values to JSON format and saves dict\_values as a JSON file or return json

### Parameters

- **dict\_values** ((dict)) – dict to be stored as json
- **output\_folder** ((path)) – Folder into which json should be stored Default None
- **file\_name** ((str)) – Name of the file the json should be stored as Default None

### Returns

- *If file\_name is provided, the json variable converted from the dict\_values is saved under*
- *this file\_name, otherwise the json variable is returned*

`multi_vector_simulator.F0_output.store_scalars_to_excel(dict_values)`

All output data that is a scalar is storage to an excellent file tab. This could for example be economical data or technical data.

### Parameters

**dict\_values** – dict Of all input and output parameters up to F0

### Returns

Excel file with scalar data

### Return type

type

`multi_vector_simulator.F0_output.store_timeseries_all_busses_to_excel(dict_values)`

This function plots the energy flows of each single bus and the energy system and saves it as PNG and additionally as a tab and an Excel sheet.

### Parameters

**dict\_values** – dict Of all input and output parameters up to F0

### Returns

Plots and excel with all timeseries of each bus

### Return type

type

## Module F1 - Plotting

Module F1 describes all the functions that create plots.

- creating graphs for energy flows
- creating bar chart for capacity
- creating pie chart for cost data
- creating network graph for the model brackets only working on Ubuntu

```
class multi_vector_simulator.F1_plotting.ESGraphRenderer(energy_system=None, filepath='network',
                                                         img_format=None, legend=True,
                                                         txt_width=10, txt_fontsize=10, **kwargs)
```

**add\_bus**(*label='Bus', subgraph=None*)

**add\_component**(*label='component', subgraph=None*)

**add\_sink**(*label='Sink', subgraph=None*)

**add\_source**(*label='Source', subgraph=None*)

**add\_storage**(*label='Storage', subgraph=None*)

**add\_transformer**(*label='Transformer', subgraph=None*)

**connect**(*a, b*)

Draw an arrow from node a to node b

**Parameters**

- **a** (*oemof.solph.network.Node*) – An oemof node (usually a Bus or a Component)
- **b** (*oemof.solph.network.Node*) – An oemof node (usually a Bus or a Component)

**render**(*\*\*kwargs*)

Call the render method of the DiGraph instance

**sankey**(*results*)

Return a dict to a plotly sankey diagram

**view**(*\*\*kwargs*)

Call the view method of the DiGraph instance

`multi_vector_simulator.F1_plotting.convert_plot_data_to_dataframe`(*plot\_data\_dict, data\_type*)

**Parameters**

- **plot\_data\_dict** (*dict*) – timeseries for either demand or supply
- **data\_type** (*str*) – one of DEMANDS or RESOURCES

**Returns**

**df** – timeseries for plotting

**Return type**

`pandas:pandas.DataFrame<frame>`,

`multi_vector_simulator.F1_plotting.create_plotly_barplot_fig`(*x\_data, y\_data, plot\_title=None, trace\_name="", legends=None, x\_axis\_name=None, y\_axis\_name=None, file\_name='barplot.png', file\_path=None*)

Create figure for specific capacities barplot

**Parameters**

- **x\_data** (*list, or pandas series*) – The list of abscissas of the data required for plotting.
- **y\_data** (*list, or pandas series, or list of lists*) – The list of ordinates of the data required for plotting.
- **plot\_title** (*str*) – The title of the plot generated. Default: None



- **trace\_name** (*str*) – Sets the trace name. The trace name appear as the legend item and on hover. Default: ""
- **legends** (*list, or pandas series*) – The list of the text written within the bars and on hover below the trace\_name Default: None
- **x\_axis\_name** (*str*) – Default: None
- **y\_axis\_name** (*str*) – Default: None
- **file\_name** (*str*) – Name of the image file. Default: "barplot.png"
- **file\_path** (*str*) – Path where the image shall be saved if not None

**Returns**

**fig** – figure object

**Return type**

plotly.graph\_objs.Figure

```
multi_vector_simulator.F1_plotting.create_plotly_flow_fig(df_plots_data, x_legend=None,
                                                         y_legend=None, plot_title=None,
                                                         color_list=None, file_name='flows.png',
                                                         file_path=None)
```

Generate figure of an asset's flow.

**Parameters**

- **df\_plots\_data** (*pandas.DataFrame*) – dataFrame with timeseries of the asset's energy flow
- **x\_legend** (*str*) – Default: None
- **y\_legend** (*str*) – Default: None
- **plot\_title** (*str*) – Default: None
- **color\_list** (*list of str or list to tuple (hexadecimal or rgb code)*) – list of colors Default: None
- **file\_name** (*str*) – Name of the image file. Default: "flows.png"
- **file\_path** (*str*) – Path where the image shall be saved if not None Default: None

**Returns**

**fig** – figure object

**Return type**

plotly.graph\_objs.Figure

```
multi_vector_simulator.F1_plotting.create_plotly_line_fig(x_data, y_data, plot_title=None,
                                                         x_axis_name=None,
                                                         y_axis_name=None,
                                                         color_for_plot='#0A2342',
                                                         file_path=None)
```

Create figure for generic timeseries lineplots

**Parameters**

- **x\_data** (*list, or pandas series*) – The list of abscissas of the data required for plotting.
- **y\_data** (*list, or pandas series, or list of lists*) – The list of ordinates of the data required for plotting.

- **plot\_title** (*str*) – The title of the plot generated. Default: None
- **x\_axis\_name** (*str*) – Default: None
- **y\_axis\_name** (*str*) – Default: None
- **file\_path** (*str*) – Path where the image shall be saved if not None

**Returns**

figure object

**Return type**

fig plotly.graph\_objs.Figure

```
multi_vector_simulator.F1_plotting.create_plotly_piechart_fig(title_of_plot, names, values,  
                                                             color_scheme=None,  
                                                             file_name='costs.png',  
                                                             file_path=None)
```

Generate figure with piechart plot.

**Parameters**

- **title\_of\_plot** (*str*) – title of the figure
- **names** (*list*) – List containing the labels of the slices in the pie plot.
- **values** (*list*) – List containing the values of the labels to be plotted in the pie plot.
- **color\_scheme** (*instance of the px.colors class of the Plotly express library*) – This parameter holds the color scheme which is palette of colors (list of hex values) to be applied to the pie plot to be created. Default: None
- **file\_name** (*str*) – Name of the image file. Default: “costs.png”
- **file\_path** (*str*) – Path where the image shall be saved if not None Default: None

**Returns**

fig – figure object

**Return type**

plotly.graph\_objs.Figure

```
multi_vector_simulator.F1_plotting.extract_plot_data_and_title(dict_values, df_dem=None)
```

Dataframe used for the plots of demands and resources timeseries in the report

**Parameters**

- **dict\_values** (*dict*) – output values of MVS
- **df\_dem** (*pandas.DataFrame*) – summarized demand information for each demand

**Return type**

pandas.DataFrame

```
multi_vector_simulator.F1_plotting.fixed_width_text(text, char_num=10)
```

Add linebreaks every char\_num characters in a given text.

**Parameters**

- **text** (*obj: 'str'*) – text to apply the linebreaks
- **char\_num** (*obj: 'int'*) – max number of characters in a line before a line break Default: 10

**Returns**

obj – the text with line breaks after every char\_num characters

**Return type**`'str'``multi_vector_simulator.F1_plotting.get_color(idx_line, color_list=None)`

Pick a color within a color list with periodic boundary conditions

**Parameters**

- **idx\_line** (*int*) – index of the line in a plot for which a color is required
- **colors** (*list of str or list to tuple (hexadecimal or rgb code)*) – list of colors Default: None

**Return type**

The color in the color list corresponding to the index modulo the color list length

`multi_vector_simulator.F1_plotting.get_fig_style_dict()``multi_vector_simulator.F1_plotting.plot_instant_power(dict_values, file_path=None)`

Plotting timeseries of instantaneous power for each assets within the energy system

**Parameters**

- **dict\_values** (*dict*) – all simulation input and output data up to this point
- **file\_path** (*str*) – Path where the image shall be saved if not None Default: None

**Returns****multi\_plots** – Dict with html DOM id for the figure as keys and `plotly.graph_objs.Figure` as values**Return type**`dict``multi_vector_simulator.F1_plotting.plot_optimized_capacities(dict_values, file_path=None)`

Plot capacities as a bar chart.

**Parameters**

- **dict\_values** – dict Of all input and output parameters up to F0
- **file\_path** (*str*) – Path where the image shall be saved if not None Default: None

**Return type**Dict with html DOM id for the figure as key and `plotly.graph_objs.Figure` as value`multi_vector_simulator.F1_plotting.plot_piecharts_of_costs(dict_values, file_path=None)`

Plotting piecharts of different cost parameters (ie. annuity, total cost, etc...)

**Parameters**

- **dict\_values** (*dict*) – all simulation input and output data up to this point
- **file\_path** (*str*) – Path where the image shall be saved if not None Default: None

**Returns****pie\_plots** – Dict with html DOM id for the figure as keys and `plotly.graph_objs.Figure` as values**Return type**`dict``multi_vector_simulator.F1_plotting.plot_sankey(dict_values)`

```
multi_vector_simulator.F1_plotting.plot_timeseries(dict_values, data_type='demands',
                                                    sector_demands=None, max_days=None,
                                                    color_list=None, file_path=None)
```

Plot timeseries as line chart.

#### Parameters

- **dict\_values** – dict Of all input and output parameters up to F0
- **data\_type** (*str*) – one of DEMANDS or RESOURCES Default: DEMANDS
- **sector\_demands** (*str*) – Name of the sector of the energy system Default: None
- **max\_days** (*int*) – maximal number of days the timeseries should be displayed for
- **color\_list** (*list of str or list to tuple (hexadecimal or rgb code)*) – list of colors Default: None
- **file\_path** (*str*) – Path where the image shall be saved if not None Default: None

#### Return type

Dict with html DOM id for the figure as key and `plotly.graph_objs.Figure` as value

```
multi_vector_simulator.F1_plotting.save_plots_to_disk(fig_obj, file_name, file_path="", width=None,
                                                    height=None, scale=None)
```

This function saves the plots generated using the Plotly library in this module to the outputs folder.

#### Parameters

- **fig\_obj** (*instance of the classes of the Plotly go library used to generate the plots in this auto-report*) – Figure object of the plotly plots
- **file\_name** (*str*) – The name of the PNG image of the plot to be saved in the output folder.
- **file\_path** (*str*) – Path where the image shall be saved
- **width** (*int or float*) – The width of the picture to be saved in pixels. Default: None
- **height** (*int or float*) – The height of the picture to be saved in pixels. Default: None
- **scale** (*int or float*) – The scale by which the plotly image ought to be multiplied. Default: None

#### Return type

Nothing is returned. This function call results in the plots being saved as .png images to the disk.

## Module F2 - Autoreport

This script generates a report of the simulation automatically, with all the important data.

```
multi_vector_simulator.F2_autoreport.create_app(results_json, path_sim_output=None)
```

Initializes the app and calls all the other functions, resulting in the web app as well as pdf.

This function specifies the layout of the web app, loads the external styling sheets, prepares the necessary data from the json results file, calls all the helper functions on the data, resulting in the auto-report.

#### Parameters

- **results\_json** (*json results file*) – This file is the result of the simulation and contains all the data necessary to generate the auto-report.
- **path\_sim\_output** (*str*) – Path to the mvs simulation's output files' folder Default: output path saved in the result\_json

**Returns**

**app** – This app holds together all the html elements wrapped in Python, necessary for the rendering of the auto-report.

**Return type**

instance of the Dash class within the dash library

`multi_vector_simulator.F2_autoreport.create_demands_section(output_JSON_file, sectors=None)`

This function creates a HTML Div element that holds an entire section with either the demands or the resources

**Parameters**

- **output\_JSON\_file** (*dict*) – Dict with all simulation parameters
- **sectors** (*list*) – List holding the names of sectors of the energy system as strings Default: None

**Return type**

Function call to insert\_subsection() that generates the demands section of the autoreport

`multi_vector_simulator.F2_autoreport.encode_image_file(img_path)`

Encode image files to load them in the dash layout under img html tag

**Parameters**

**img\_path** (*str*) – path to the image file

**Returns**

**encoded\_img** – encoded bytes of the image file

**Return type**

bytes

`multi_vector_simulator.F2_autoreport.insert_body_text(body_of_text)`

This function is for rendering blocks of text within the sub-sections.

**Parameters**

**body\_of\_text** (*str*) – Typically a single-line or paragraph of text.

**Returns**

A html element that renders the paragraph of text in the Dash app layout.

**Return type**

html.P()

`multi_vector_simulator.F2_autoreport.insert_headings(heading_text)`

This function is for creating the headings such as information, input data, etc.

**Parameters**

**heading\_text** (*str*) – Big headings for several sub-sections.

**Returns**

A html element with the heading text encased container.

**Return type**

html.P()

`multi_vector_simulator.F2_autoreport.insert_log_messages(log_dict)`

This function inserts logging messages that arise during the simulation, such as warnings and error messages, into the auto-report.

**Parameters**

**log\_dict** (*dict*) – A dictionary containing the logging messages collected during the simulation.

**Returns**

This html element holds the children html elements that produce the lists of warnings and error messages for both print and screen versions of the auto-report.

**Return type**

html.Div()

`multi_vector_simulator.F2_autoreport.insert_plotly_figure(fig, id_plot=None, print_only=False)`

Insert a plotly figure in a dash app layout

**Parameters**

- **fig** (plotly.graph\_objs.Figure) – figure object
- **id\_plot** (str) – Id of the graph. Should be unique. Default: None
- **print\_only** (bool) – Used to determine if a web version of the plot is to be generated or not. Default: False

**Returns**

Html Div component containing an image for the print-only version and a plotly figure for the online (no-print) app (in the app the user can interact with plotly figure, whereas the image is static).

**Return type**

*dash\_html\_components.Div*

`multi_vector_simulator.F2_autoreport.insert_subsection(title, content, **kwargs)`

Inserts sub-sections within the Dash app layout, such as Input data, simulation results, etc.

**Parameters**

- **title** (str) – This is the title or heading of the subsection.
- **content** (list) – This is typically a list of html elements or function calls returning html elements, that make up the body of the sub-section.
- **kwargs** (Any possible optional arguments such as styles, etc.) –

**Returns**

This returns the sub-section of the report including the title and other information within the sub-section.

**Return type**

html.Div()

`multi_vector_simulator.F2_autoreport.make_dash_data_table(df, title=None)`

Function that creates a Dash DataTable from a Pandas dataframe.

**Parameters**

- **df** (pandas.DataFrame) – This dataframe holds the data from which the dash table is to be created.
- **title** (str) – An optional title for the table. Default: None

**Returns**

This element contains the title of the dash table and the dash table itself encased in a child html.Div() element.

**Return type**

html.Div()

`multi_vector_simulator.F2_autoreport.open_in_browser(app, timeout=600)`

Runs the dash app in a thread and open a browser window

#### Parameters

- **app** (*instance of the Dash class, part of the dash library*) –
- **timeout** (*int or float*) – Specifies the number of seconds that the web app should be open in the browser before timing out.

#### Return type

Nothing, but the web app version of the auto-report is displayed in a browser.

`multi_vector_simulator.F2_autoreport.print_pdf(app=None,  
path_pdf_report='/home/docs/checkouts/readthedocs.org/user_builds/multi-vector-simulator/checkouts/latest/docs/MVS_outputs/out.pdf')`

Runs the dash app in a thread and print a pdf before exiting

#### Parameters

- **app** (*instance of the Dash class of the dash library*) – Default: None
- **path\_pdf\_report** (*str*) – Path where the pdf report should be saved.

#### Return type

None, but saves a pdf printout of the provided app under the provided path

`multi_vector_simulator.F2_autoreport.ready_capacities_plots(dict_values, only_print=False)`

Insert the capacities bar plots in a dash html layout

#### Parameters

- **dict\_values** (*dict*) – Dict with all simulation parameters
- **only\_print** (*bool*) – Setting this value true results in the function creating only the plot for the PDF report, but not the web app version of the auto-report. Default: False

#### Returns

**cap\_plots** – List containing the capacities bar plots dash components

#### Return type

list

`multi_vector_simulator.F2_autoreport.ready_costs_pie_plots(dict_values, only_print=False)`

Insert the pie plots in a dash html layout

#### Parameters

- **dict\_values** (*dict*) – Dict with all simulation parameters
- **only\_print** (*bool*) – Setting this value true results in the function creating only the plot for the PDF report, but not the web app version of the auto-report. Default: False

#### Returns

**pie\_plots** – List containing the cost pie plots dash components

#### Return type

list

`multi_vector_simulator.F2_autoreport.ready_flows_plots(dict_values, only_print=False)`

Generate figure for each assets' flow of the energy system.

#### Parameters

- **dict\_values** (*dict*) – Dict with all simulation parameters
- **only\_print** (*bool*) – Setting this value true results in the function creating only the plot for the PDF report, but not the web app version of the auto-report. Default: False

**Returns**

**multi\_plots** – List containing the assets' timeseries plots as dash components

**Return type**

list

```
multi_vector_simulator.F2_autoreport.ready_sankey_diagram(dict_values, only_print=False)
```

```
multi_vector_simulator.F2_autoreport.ready_timeseries_plots(dict_values, data_type='demands',  
                                                             only_print=False,  
                                                             sector_demands=None)
```

Insert the timeseries line plots in a dash html layout.

**Parameters**

- **dict\_values** (*dict*) – Dict with all simulation parameters
- **data\_type** (*str*) – one of DEMANDS or RESOURCES Default: DEMANDS
- **only\_print** (*bool*) – Setting this value true results in the function creating only the plot for the PDF report, but not the web app version of the auto-report. Default: False
- **sector\_demands** (*str*) – Name of the sector of the energy system Default: None

**Returns**

**plots** – List containing the timeseries line plots dash components

**Return type**

list

## 4.2 Release Notes

TBD (adapt the changelog)

## 4.3 License

The MVS is licensed with the **GNU General Public License v2.0**. The GNU GPL is the most widely used free software license and has a strong copyleft requirement. When distributing derived works, the source code of the work must be made available under the same license. There are multiple variants of the GNU GPL, each with different requirements.

## 4.4 Contributing to MVS

### 4.4.1 Proposed workflow

The workflow is described in the [CONTRIBUTING.md](#) file in the repository.



### 4.4.2 Unit tests (pytest)

When developing code for the MVS please make sure that you always also develop test in `tests`. We integrate those unit tests with `pytest`. Make sure that your tests are as lightweight as possible - this means that you do not always have to run the whole code to test for one feature, but can test a function with a standalone tests. Please refer to the other tests that have already been introduced.

Always aim for the test coverage button on [the main page of the github repository](#) to reach 100%!

When you do have to run the MVS itself for a test, eg. for benchmark tests, please always use the arguments `-f -log warning` to make the test results better readable.

### 4.4.3 Build documentation

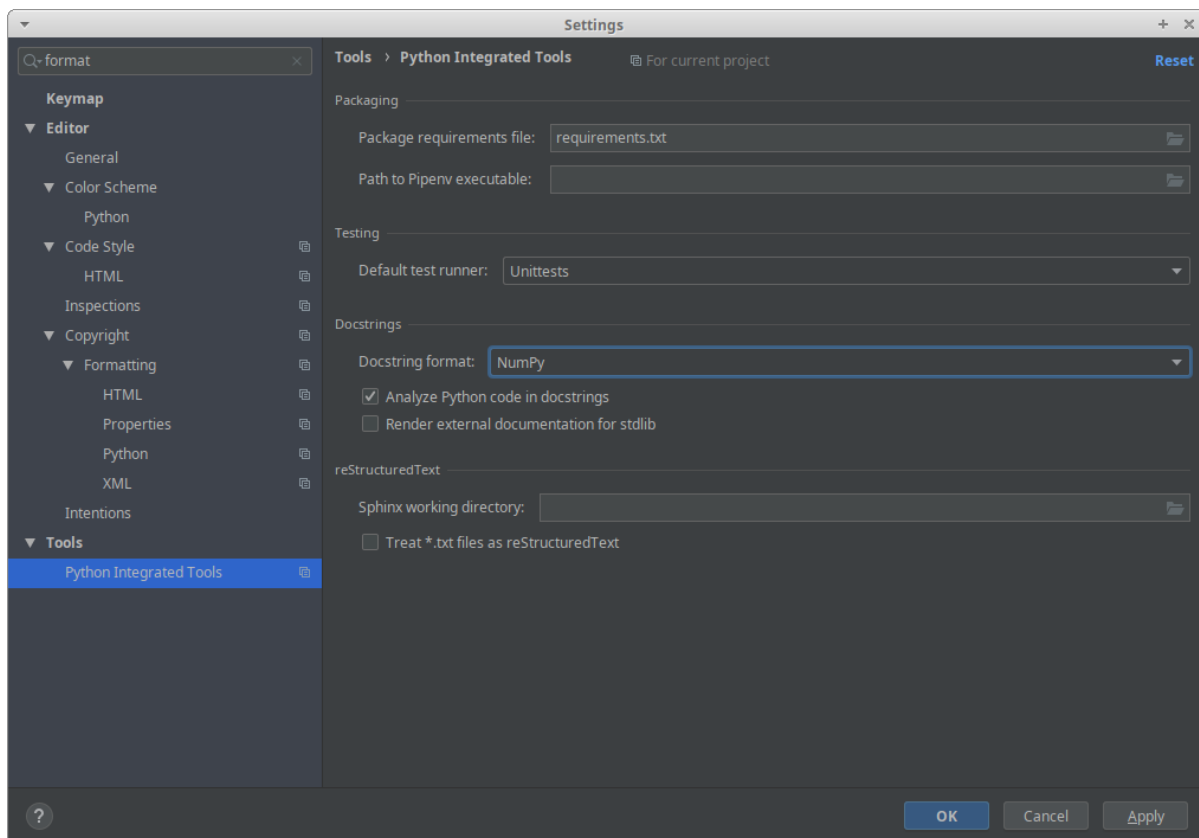
You can build the documentation locally moving inside the `docs/` folder and typing

```
html build
```

into a console, then go to `docs/_build/` and open `index.html` into your favorite browser.

All functions in the code will be automatically documented via their docstrings. Please make sure they follow the [Numpy format](#).

Here is how to set that in pycharm



#### 4.4.4 Format of Docstrings

Please add docstrings for every function you add. As docstrings are a powerful means of documentation we give an example here:

Download: [Example docstring](#)

```
import pandas as pd

def example_function(arg1, argN):
    r"""
    One line no more than 80 character explaining very shortly what the function does

    More detailed explanation about the function,
    can have several lines

    Parameters
    -----
    arg1 : str or list(str)
        description of arg1
        Default: <default value here>.

    ...

    argN : str or list(str)
        description of argN
        Default: <default value here>.

    Returns
    -----
    :class:`pandas.DataFrame<frame>`
        here comes the description
    (In case of no return, you can write what the function changes, e.g. updates
    `variable_x` with `y`.)

    Notes
    -----
    You can cite the references below using [1]_ or [2]_ add maths equations like this:

    .. math:: P=\frac{1}{8}\cdot\rho_{\text{hub}}\cdot d_{\text{rotor}}^2
        \cdot\pi\cdot v_{\text{wind}}^3\cdot cp\left(v_{\text{wind}}\right)

    with:
        P: power [W], :math:`\rho`: density [kg/m3], d: diameter [m],
        v: wind speed [m/s], cp: power coefficient

    You can also indicate here which tests are covering this function:
    This function is tested with:
    - tests.test_example_function()

    References
    -----
    .. [1] paper 1
```

(continues on next page)

(continued from previous page)

```

.. [2] paper 2

Examples
-----
# Here you can write some basic python code that is tested with pytest
>>> import src.C2_economic_functions as e_funcs
>>> CAPEX = e_funcs.capex_from_investment(
...     investment_t0=2200000, lifetime=20, project_life=20,
...     discount_factor=0.1, tax=0.15)
>>> round(CAPEX, 7)
253000.0

"""
return pd.DataFrame(...)

```

## 4.5 Publications and Bibliography

### 4.5.1 Scientific Publications

The MVS is currently under development in the H2020 research project *E-LAND*. Still, there are already some references where additional information can be found regarding its intention, application, and method. More information about E-LAND can be found on the research project website: [E-LAND Horizon 2020. Novel solutions for decarbonized energy islands](#)

#### Articles

- **(Hoffmann, 2020b)**  
Martha M. Hoffmann, Sanket Puranik, Marc Juanpera, José M. Martín-Rapún, Heidi Tuiskula, & Philipp Blechinger: *Investment planning in multi-vector energy systems: Definition of key performance indicators*, Conference paper, presented at the CIRED 2020 Berlin Workshop (CIRED 2020), Berlin / online. 2020. DOI: [10.5281/zenodo.4449918](#)

#### Posters

- **(Hoffmann, 2020a)**  
Martha M. Hoffmann, Sanket Puranik, Marc Juanpera, José M. Martín-Rapún, Heidi Tuiskula, & Philipp Blechinger: *Investment planning in multi-vector energy systems: Definition of key performance indicators*, Conference poster, presented at the CIRED 2020 Berlin Workshop (CIRED 2020), Berlin / online: DOI: [10.5281/zenodo.4449969](#)

## Presentations

- **(Hoffmann, 2020c)**  
*Introducing an os simulation tool for sector-coupled energy sys opt: Multi-Vector Simulator (MVS)* Presentation at Energy Modelling Platform for Europe (EMP-E) 2020, 06. – 08. October 2020, online. Link to session: [Youtube](#)
- **(Hoffmann, 2020d)**  
*Multi-Vector Simulator*, session: *Building on experience: What to take from individual models for the oemof-community*, presentation at oemof developer meeting, 02. - 04. December 2020, online. Link: [conference schedule](#)

## Master thesis

- **(Backhaus, 2021\*)**  
Andra Backhaus: *Analyzing the application of different energy cell sizes as an approach for the integration of decentralized renewable energy sources*. Master of Science Thesis, Albert-Ludwigs Universität Freiburg. To be submitted in June 2021. Written in scope of the H2020 research project [open\\_plan](#)
- **(Gering, 2021\*)**  
Marie-Claire Gering: *Modellierung und Analyse von sektorgekoppelten Energiesystemen mit photovoltaisch betriebenen Wärmepumpen und thermischen Energiespeichern*. Master of Science Thesis, Technische Universität Berlin. To be submitted in June 2021. Written in scope of the research project [GRECO](#).
- **(El Mir, 2020)**  
Ursula El Mir: *Identification of a Validation Method for Open Source Simulation Tools and Application of Said Method to the MVS: Multi-Vector Simulator - Sector Coupled Systems*. Master of Science Thesis, Delft University of Technology. September 2020. Link: <http://resolver.tudelft.nl/uuid:50c283c7-64c9-4470-8063-140b56f18cfe> . Written in scope of the H2020 research project [E-LAND](#)

## 4.5.2 Reference projects

As an publicly developed open-source tool, the MVS can also be used, adapted and improved in other projects. The current projects that an serve as a reference for MVS utilization are listed.

### H2020 research project GRECO and tool pvcompare

Within the [H2020 research project GRECO](#) the model [pvcompare](#) ([github repository](#)) was developed to compare the benefits of different PV technologies in local energy systems in different energy supply scenarios. It uses MVS for optimizing these energy systems and calculating specific KPIs. Functionalities of [pvcompare](#) include among others the calculation of an area potential for PV on roof-tops and facades, heat and electricity demand profiles, PV feed-in time series for different technologies, temperature dependent COPs for heat pumps and pre-calculations for a stratified thermal storage. [pvcompare](#) concentrates on the integration of PV technologies into local energy systems but could easily be enhanced to analyze other conversion technologies. Checkout the [documentation](#) to learn more about [pvcompare](#).

## H2020 research project open\_Plan

The [H2020 research project open\\_plan](#) aims to build an open source tool to plan the design of a single energy cell. It will extend on the existing features of MVS to fulfil the requirements of its pilot projects. The project open\_plan is funded until December 2022, the development of the graphical user interface will take place on the [github repository](#) of open\_plan.

### 4.5.3 Bibliography

This RTD referenced following sources:

- **(Bloess, 2017)**  
Andreas Bloess, Wolf-Peter Schill, Alexander Zerrahn: *Power-to-heat for renewable energy integration: A review of technologies, modeling approaches, and flexibility potentials*. Applied Energy, 2018. DOI: [10.1016/j.apenergy.2017.12.073](#)
- **(Ringkjøb, 2018)**  
Hans-Kristian Ringkjøb, Peter M. Haugan, Ida Marie Solbrekke: *A review of modelling tools for energy and electricity systems with large shares of variable renewables*. Renewable and Sustainable Energy Reviews, 2018. DOI: [10.1016/j.rser.2018.08.002](#)

## 4.6 Cite MVS

If you use the MVS, please cite the tool as

Hoffmann, Martha M., Duc, Pierre-Francois, & Haas, Sabine. (2021, March 4). Multi-Vector Simulator (Version v0.5.5, beta release). Zenodo. DOI: [10.5281/zenodo.4610237](#)

in your reference section.

## 4.7 Troubleshooting

### 4.7.1 Installation

#### Python package “pygraphviz”

The installation of pygraphviz can cause errors. You can circumvent this issue by setting the `simulation_setting`, `plot_nx_graph` to False. If you need to plot the network graphs (set parameter `plot_nx_graph` to True) or run all pytests, check if we already have a solution for your OS/distribution:

**Ubuntu 18.4:** Pygraphviz could not be installed with pip. Solution:

```
sudo apt-get install python3-dev graphviz libgraphviz-dev pkg-config
pip install pygraphviz
```

**Windows 10** Installing via

```
pip install -r requirements.txt
```

results in an error:

error: Microsoft Visual C++ 14.0 is required. Get it with “Build Tools for Visual Studio”: <https://visualstudio.microsoft.com/downloads/>

You can find fixes on [stackoverflow](#) If you have conda installed, activate your environment and run

```
conda install -c alubbock graphviz pygraphviz
```

Then you need to configure the dot command on your computer to be able to use graphviz

```
dot -c
```

### Python package “xlrd”

On **Windows** there can be issues installing xlrd. This could solve your troubles:

1. Delete xlrd from requirements.txt
2. Download the xlrd-1.2.0-py2.py3-none-any.whl file from [here](#).
3. Copy the file to main directory of the project on your laptop
4. Install it manually writing `pip install xlrd-1.2.0-py2.py3-none-any.whl`

### Python package “wkhtmltopdf”

There can be issues installing wkhtmltopdf. Solution can be found on the [packages documentation](#).

### cbc-solver

While with Ubuntu the installation of the cbc solver should work rather well, even when adding it to the environment variables (like described in the installation instructions) can sometimes not work on Windows. This was experienced with Windows 10.

A workaround is to directly put the cbc.exe file into the root of the MVS repository, ie. in the same folder where also the CHANGELOG.md file is located. Python/Oemof/Pyomo then are able to find the solver.

### pyppeteer

If you are using OS X, you might need to install this package separately with conda using:

```
conda install -c conda-forge pyppeteer
```

or

```
conda install -c conda-forge/label/cf202003 pyppeteer
```

More information is available on [their website](#).

## 4.7.2 Error messages and MVS termination

Even though we try to keep the error messages of the MVS clear and concise, there might be a some that are harder to understand. This especially applies to error messages that occur due to the termination of the oemof optimization process.

### `json.decoder.JSONDecodeError`

If the error `json.decoder.JSONDecodeError` is raised, there is a formatting issue with the json file that is used as an input file.

Have you changed the json file manually? Please check for correct formatting, ie. apostrophes, commas, brackets, and so on.

If you have not changed the Json file yourself please consider raising an issue in the github project.

## 4.8 Bug report

Have a look at [Troubleshooting](#) section.

Please submit your bug reports via our [github issues](#)





## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### m

- `multi_vector_simulator.A0_initialization`, 114
- `multi_vector_simulator.A1_csv_to_json`, 117
- `multi_vector_simulator.B0_data_input_json`,  
119
- `multi_vector_simulator.C0_data_processing`,  
121
- `multi_vector_simulator.C1_verification`, 131
- `multi_vector_simulator.C2_economic_functions`,  
136
- `multi_vector_simulator.D0_modelling_and_optimization`,  
141
- `multi_vector_simulator.D1_model_components`,  
143
- `multi_vector_simulator.D2_model_constraints`,  
151
- `multi_vector_simulator.E0_evaluation`, 155
- `multi_vector_simulator.E1_process_results`,  
156
- `multi_vector_simulator.E2_economics`, 162
- `multi_vector_simulator.E3_indicator_calculation`,  
166
- `multi_vector_simulator.E4_verification`, 176
- `multi_vector_simulator.F0_output`, 178
- `multi_vector_simulator.F1_plotting`, 179
- `multi_vector_simulator.F2_autoreport`, 184
- `multi_vector_simulator.utils.data_parser`, 111
- `multi_vector_simulator.utils.helpers`, 112



## INDEX

### A

`add_a_transformer_for_each_peak_demand_pricing_period()` (in module `multi_vector_simulator.C0_data_processing`), 121  
`add_asset_to_asset_dict_for_each_flow_direction()` (in module `multi_vector_simulator.C0_data_processing`), 121  
`add_asset_to_asset_dict_of_bus()` (in module `multi_vector_simulator.C0_data_processing`), 122  
`add_assets_to_asset_dict_of_connected_busses()` (in module `multi_vector_simulator.C0_data_processing`), 122  
`add_bus()` (`multi_vector_simulator.F1_plotting.ESGraphRenderer` method), 179  
`add_component()` (`multi_vector_simulator.F1_plotting.ESGraphRenderer` method), 180  
`add_constraints()` (in module `multi_vector_simulator.D2_model_constraints`), 151  
`add_degree_of_autonomy()` (in module `multi_vector_simulator.E3_indicator_calculation`), 166  
`add_degree_of_net_zero_energy()` (in module `multi_vector_simulator.E3_indicator_calculation`), 166  
`add_degree_of_sector_coupling()` (in module `multi_vector_simulator.E3_indicator_calculation`), 167  
`add_economic_parameters()` (in module `multi_vector_simulator.C0_data_processing`), 123  
`add_info_flows()` (in module `multi_vector_simulator.E1_process_results`), 156  
`add_levelized_cost_of_energy_carriers()` (in module `multi_vector_simulator.E3_indicator_calculation`), 167  
`add_onsite_energy_fraction()` (in module `multi_vector_simulator.E3_indicator_calculation`), 167  
`add_onsite_energy_matching()` (in module `multi_vector_simulator.E3_indicator_calculation`), 168  
`add_renewable_factor()` (in module `multi_vector_simulator.E3_indicator_calculation`), 168  
`add_renewable_share_of_local_generation()` (in module `multi_vector_simulator.E3_indicator_calculation`), 168  
`add_sink()` (`multi_vector_simulator.F1_plotting.ESGraphRenderer` method), 180  
`add_source()` (`multi_vector_simulator.F1_plotting.ESGraphRenderer` method), 180  
`add_specific_emissions_per_electricity_equivalent()` (in module `multi_vector_simulator.E3_indicator_calculation`), 169  
`add_storage()` (`multi_vector_simulator.F1_plotting.ESGraphRenderer` method), 180  
`add_storage_components()` (in module `multi_vector_simulator.A1_csv_to_json`), 117  
`add_total_consumption_from_provider_electricity_equivalent()` (in module `multi_vector_simulator.E3_indicator_calculation`), 169  
`add_total_emissions()` (in module `multi_vector_simulator.E3_indicator_calculation`), 169  
`add_total_feedin_electricity_equivalent()` (in module `multi_vector_simulator.E3_indicator_calculation`), 170  
`add_total_renewable_and_non_renewable_energy_origin()` (in module `multi_vector_simulator.E3_indicator_calculation`), 170  
`add_transformer()` (`multi_vector_simulator.F1_plotting.ESGraphRenderer` method), 180  
`add_version_number_used()` (in module `multi_vector_simulator.C0_data_processing`), 123  
`adding_assets_to_energysystem_model()` (`multi_vector_simulator.D0_modelling_and_optimization.model` method), 141  
`all()` (in module `multi_vector_simulator.C0_data_processing`), 123

all_list_in_dict()	(in module multi_vector_simulator.E2_economics),	162	check_efficiency_of_storage_capacity()	(in module multi_vector_simulator.C1_verification),	
all_totals()	(in module multi_vector_simulator.E3_indicator_calculation),	170	check_emission_factor_of_providers()	(in module multi_vector_simulator.C1_verification),	
all_valid_intervals()	(in module multi_vector_simulator.C1_verification),	132	check_energy_system_can_fulfill_max_demand()	(in module multi_vector_simulator.C1_verification),	
annuity()	(in module multi_vector_simulator.C2_economic_functions),	137	check_feasibility_of_maximum_emissions_constraint()	(in module multi_vector_simulator.C1_verification),	
annuity_factor()	(in module multi_vector_simulator.C2_economic_functions),	137	check_feedin_tariff_vs_energy_price()	(in module multi_vector_simulator.C1_verification),	
apply_function_to_single_or_list()	(in module multi_vector_simulator.C0_data_processing),	123	check_feedin_tariff_vs_levelized_cost_of_generation_of_pro	(in module multi_vector_simulator.C1_verification),	
<b>B</b>			check_for_label_duplicates()	(in module multi_vector_simulator.C1_verification),	
bus()	(in module multi_vector_simulator.D1_model_components),	144	check_for_sufficient_assets_on_busses()	(in module multi_vector_simulator.C1_verification),	
<b>C</b>			calculate_costs_replacement()	(in module multi_vector_simulator.E2_economics),	163
calculate_costs_upfront_investment()	(in module multi_vector_simulator.E2_economics),	163	check_if_energy_vector_is_defined_in_DEFAULT_WEIGHTS_ENERG	(in module multi_vector_simulator.C1_verification),	
calculate_dispatch_expenditures()	(in module multi_vector_simulator.E2_economics),	163	check_if_energy_vector_of_all_assets_is_valid()	(in module multi_vector_simulator.C1_verification),	
calculate_electricity_equivalent_for_a_set_of_aggregated_values()	(in module multi_vector_simulator.E3_indicator_calculation),	171	check_input_folder()	(in module multi_vector_simulator.A0_initialization),	
calculate_emissions_from_flow()	(in module multi_vector_simulator.E3_indicator_calculation),	171	check_input_values()	(in module multi_vector_simulator.C1_verification),	
calculate_operation_and_management_expenditures()	(in module multi_vector_simulator.E2_economics),	164	check_non_dispatchable_source_time_series()	(in module multi_vector_simulator.C1_verification),	
calculate_total_asset_costs_over_lifetime()	(in module multi_vector_simulator.E2_economics),	164	check_optimize_cap()	(in module multi_vector_simulator.D1_model_components),	
calculate_total_capital_costs()	(in module multi_vector_simulator.E2_economics),	164	check_output_folder()	(in module multi_vector_simulator.A0_initialization),	
calculate_total_operational_expenditures()	(in module multi_vector_simulator.E2_economics),	165	check_storage_file_is_csv()	(in module multi_vector_simulator.A1_csv_to_json),	
capex_from_investment()	(in module multi_vector_simulator.C2_economic_functions),	137	check_time_series_values_between_0_and_1()	(in module multi_vector_simulator.C1_verification),	
change_sign_of_feedin_tariff()	(in module multi_vector_simulator.C0_data_processing),	123	chp()	(in module multi_vector_simulator.D1_model_components),	
			chp_fix()	(in module multi_vector_simulator.D1_model_components),	

`multi_vector_simulator.D1_model_components)`, `convert_scalar_matrix_to_dataframe()` (in module `multi_vector_simulator.E1_process_results`),  
145  
`chp_optimize()` (in module `multi_vector_simulator.D1_model_components`), `convert_scalars_to_dataframe()` (in module `multi_vector_simulator.E1_process_results`),  
145  
`compute_timeseries_properties()` (in module `multi_vector_simulator.C0_data_processing`), `create_app()` (in module `multi_vector_simulator.F2_autoreport`), 184  
124  
`connect()` (`multi_vector_simulator.F1_plotting.ESGraphRender`), `create_demands_section()` (in module `multi_vector_simulator.F2_autoreport`), 185  
method), 180  
`constraint_maximum_emissions()` (in module `multi_vector_simulator.D2_model_constraints`), `create_input_json()` (in module `multi_vector_simulator.A1_csv_to_json`),  
151 118  
`constraint_minimal_degree_of_autonomy()` (in module `multi_vector_simulator.D2_model_constraints`), `create_json_from_csv()` (in module `multi_vector_simulator.A1_csv_to_json`),  
152 119  
`constraint_minimal_renewable_share()` (in module `multi_vector_simulator.D2_model_constraints`), `create_plotly_barplot_fig()` (in module `multi_vector_simulator.F1_plotting`), 180  
152  
`constraint_net_zero_energy()` (in module `multi_vector_simulator.D2_model_constraints`), `create_plotly_flow_fig()` (in module `multi_vector_simulator.F1_plotting`), 181  
152  
`conversion()` (in module `multi_vector_simulator.A1_csv_to_json`), `create_plotly_line_fig()` (in module `multi_vector_simulator.F1_plotting`), 181  
118  
`convert_components_to_dataframe()` (in module `multi_vector_simulator.E1_process_results`), `create_plotly_piechart_fig()` (in module `multi_vector_simulator.F1_plotting`), 182  
157  
`convert_cost_matrix_to_dataframe()` (in module `multi_vector_simulator.E1_process_results`), `crf()` (in module `multi_vector_simulator.C2_economic_functions`),  
157 138  
`convert_costs_to_dataframe()` (in module `multi_vector_simulator.E1_process_results`), `CustomBus` (class in `multi_vector_simulator.D1_model_components`),  
157 143  
`convert_demand_to_dataframe()` (in module `multi_vector_simulator.E1_process_results`), `cut_below_micro()` (in module `multi_vector_simulator.E1_process_results`),  
157 158  
`convert_epa_params_to_mvs()` (in module `multi_vector_simulator.utils.data_parser`),  
111  
`convert_from_json_to_special_types()` (in module `multi_vector_simulator.B0_data_input_json`),  
119  
`convert_from_special_types_to_json()` (in module `multi_vector_simulator.B0_data_input_json`),  
120  
`convert_kpi_sector_to_dataframe()` (in module `multi_vector_simulator.E1_process_results`),  
157  
`convert_mvs_params_to_epa()` (in module `multi_vector_simulator.utils.data_parser`),  
112  
`convert_plot_data_to_dataframe()` (in module `multi_vector_simulator.F1_plotting`), 180

**D**  
`define_auxiliary_assets_of_energy_providers()` (in module `multi_vector_simulator.C0_data_processing`),  
124  
`define_availability_of_peak_demand_pricing_assets()` (in module `multi_vector_simulator.C0_data_processing`),  
125  
`define_energy_vectors_from_busses()` (in module `multi_vector_simulator.C0_data_processing`),  
125  
`define_excess_sinks()` (in module `multi_vector_simulator.C0_data_processing`),  
125  
`define_missing_cost_data()` (in module `multi_vector_simulator.C0_data_processing`),  
125  
`define_sink()` (in module `multi_vector_simulator.C0_data_processing`),  
125  
`define_source()` (in module `multi_vector_simulator.C0_data_processing`),  
126

define\_transformer\_for\_peak\_demand\_pricing() [174](#)  
 (in module multi\_vector\_simulator.C0\_data\_processing), [127](#) **ESGraphRenderer** (class in multi\_vector\_simulator.F1\_plotting), [179](#)  
 detect\_excessive\_excess\_generation\_in\_bus() [176](#) evaluate\_dict() (in module multi\_vector\_simulator.E0\_evaluation), [155](#)  
 (in module multi\_vector\_simulator.E4\_verification), [176](#) evaluate\_dict() (in module multi\_vector\_simulator.F0\_output), [178](#)  
 determine\_dispatch\_price() (in module multi\_vector\_simulator.C0\_data\_processing), [127](#) evaluate\_lifetime\_costs() (in module multi\_vector\_simulator.C0\_data\_processing), [128](#)  
 determine\_lifetime\_price\_dispatch() (in module multi\_vector\_simulator.C2\_economic\_functions), [138](#) extract\_plot\_data\_and\_title() (in module multi\_vector\_simulator.F1\_plotting), [182](#)  
 determine\_months\_in\_a\_peak\_demand\_pricing\_period() [127](#) **F**  
 (in module multi\_vector\_simulator.C0\_data\_processing), [127](#) find\_value\_by\_key() (in module multi\_vector\_simulator.utils.helpers), [113](#)  
**E** fixed\_width\_text() (in module multi\_vector\_simulator.F1\_plotting), [182](#)  
 encode\_image\_file() (in module multi\_vector\_simulator.F2\_autoreport), [185](#)  
 energyConsumption() (in module multi\_vector\_simulator.C0\_data\_processing), [127](#) **G**  
 energyConversion() (in module multi\_vector\_simulator.C0\_data\_processing), [128](#) get\_color() (in module multi\_vector\_simulator.F1\_plotting), [183](#)  
 energyProduction() (in module multi\_vector\_simulator.C0\_data\_processing), [128](#) get\_costs() (in module multi\_vector\_simulator.E2\_economics), [165](#)  
 energyProviders() (in module multi\_vector\_simulator.C0\_data\_processing), [128](#) get\_fig\_style\_dict() (in module multi\_vector\_simulator.F1\_plotting), [183](#)  
 energyStorage() (in module multi\_vector\_simulator.C0\_data\_processing), [128](#) get\_flow() (in module multi\_vector\_simulator.E1\_process\_results), [159](#)  
 equation\_degree\_of\_autonomy() (in module multi\_vector\_simulator.E3\_indicator\_calculation), [171](#) get\_item\_if\_list() (in module multi\_vector\_simulator.utils.helpers), [113](#)  
 equation\_degree\_of\_net\_zero\_energy() (in module multi\_vector\_simulator.E3\_indicator\_calculation), [172](#) get\_length\_if\_list() (in module multi\_vector\_simulator.utils.helpers), [113](#)  
 equation\_degree\_of\_sector\_coupling() (in module multi\_vector\_simulator.E3\_indicator\_calculation), [172](#) get\_lifetime\_price\_dispatch\_list() (in module multi\_vector\_simulator.C2\_economic\_functions), [139](#)  
 equation\_levelized\_cost\_of\_energy\_carrier() (in module multi\_vector\_simulator.E3\_indicator\_calculation), [173](#) get\_lifetime\_price\_dispatch\_one\_value() (in module multi\_vector\_simulator.C2\_economic\_functions), [139](#)  
 equation\_onsite\_energy\_fraction() (in module multi\_vector\_simulator.E3\_indicator\_calculation), [173](#) get\_lifetime\_price\_dispatch\_timeseries() (in module multi\_vector\_simulator.C2\_economic\_functions), [140](#)  
 equation\_onsite\_energy\_matching() (in module multi\_vector\_simulator.E3\_indicator\_calculation), [174](#) get\_optimal\_cap() (in module multi\_vector\_simulator.E1\_process\_results), [159](#)  
 equation\_renewable\_share() (in module multi\_vector\_simulator.E3\_indicator\_calculation), [174](#) get\_parameter\_to\_be\_evaluated\_from\_oemof\_results() (in module multi\_vector\_simulator.E1\_process\_results), [160](#)  
 (in module multi\_vector\_simulator.E3\_indicator\_calculation), [174](#) get\_replacement\_costs() (in module multi\_vector\_simulator.C2\_economic\_functions), [140](#)  
 (in module multi\_vector\_simulator.E3\_indicator\_calculation), [174](#) get\_results() (in module multi\_vector\_simulator.E1\_process\_results), [160](#)



get\_state\_of\_charge\_info() (in module *multi\_vector\_simulator.E1\_process\_results*),  
 160  
 get\_storage\_results() (in module *multi\_vector\_simulator.E1\_process\_results*),  
 161  
 get\_timeseries\_multiple\_flows() (in module *multi\_vector\_simulator.C0\_data\_processing*),  
 129  
 get\_timeseries\_per\_bus() (in module *multi\_vector\_simulator.E1\_process\_results*),  
 161  
 get\_tuple\_for\_oemof\_results() (in module *multi\_vector\_simulator.E1\_process\_results*),  
 161  
 get\_units\_of\_cost\_matrix\_entries() (in module *multi\_vector\_simulator.E1\_process\_results*),  
 162  
**I**  
 initialize() (*multi\_vector\_simulator.D0\_modelling\_and\_optimization* module, 143  
 method), 143  
 initialize\_kpi() (in module *multi\_vector\_simulator.E0\_evaluation*), 155  
 initialize() (*multi\_vector\_simulator.D0\_modelling\_and\_optimization* module, 141  
 method), 141  
 insert\_body\_text() (in module *multi\_vector\_simulator.F2\_autoreport*), 185  
 insert\_headings() (in module *multi\_vector\_simulator.F2\_autoreport*), 185  
 insert\_log\_messages() (in module *multi\_vector\_simulator.F2\_autoreport*), 185  
 insert\_plotly\_figure() (in module *multi\_vector\_simulator.F2\_autoreport*), 186  
 insert\_subsection() (in module *multi\_vector\_simulator.F2\_autoreport*), 186  
**L**  
 lcoe\_assets() (in module *multi\_vector\_simulator.E2\_economics*), 166  
 load\_json() (in module *multi\_vector\_simulator.B0\_data\_input\_json*),  
 120  
 lookup\_file() (in module *multi\_vector\_simulator.C1\_verification*),  
 136  
**M**  
 make\_dash\_data\_table() (in module *multi\_vector\_simulator.F2\_autoreport*), 186  
 maximum\_emissions\_test() (in module *multi\_vector\_simulator.E4\_verification*),  
 176  
 minimal\_constraint\_test() (in module *multi\_vector\_simulator.E4\_verification*),  
 177  
 MissingParametersForEconomicEvaluation, 162  
 model\_building (class in *multi\_vector\_simulator.D0\_modelling\_and\_optimization*),  
 141  
 module  
   *multi\_vector\_simulator.A0\_initialization*,  
   114  
   *multi\_vector\_simulator.A1\_csv\_to\_json*,  
   117  
   *multi\_vector\_simulator.B0\_data\_input\_json*,  
   119  
   *multi\_vector\_simulator.C0\_data\_processing*,  
   121  
   *multi\_vector\_simulator.C1\_verification*,  
   131  
   *multi\_vector\_simulator.C2\_economic\_functions*,  
   136  
   *multi\_vector\_simulator.D0\_modelling\_and\_optimization*,  
   141  
   *multi\_vector\_simulator.D1\_model\_components*,  
   143  
   *multi\_vector\_simulator.D2\_model\_constraints*,  
   151  
   *multi\_vector\_simulator.E0\_evaluation*, 155  
   *multi\_vector\_simulator.E1\_process\_results*,  
   156  
   *multi\_vector\_simulator.E2\_economics*, 162  
   *multi\_vector\_simulator.E3\_indicator\_calculation*,  
   166  
   *multi\_vector\_simulator.E4\_verification*,  
   176  
   *multi\_vector\_simulator.F0\_output*, 178  
   *multi\_vector\_simulator.F1\_plotting*, 179  
   *multi\_vector\_simulator.F2\_autoreport*, 184  
   *multi\_vector\_simulator.utils.data\_parser*,  
   111  
   *multi\_vector\_simulator.utils.helpers*, 112  
*multi\_vector\_simulator.A0\_initialization*  
 module, 114  
*multi\_vector\_simulator.A1\_csv\_to\_json*  
 module, 117  
*multi\_vector\_simulator.B0\_data\_input\_json*  
 module, 119  
*multi\_vector\_simulator.C0\_data\_processing*  
 module, 121  
*multi\_vector\_simulator.C1\_verification*  
 module, 131  
*multi\_vector\_simulator.C2\_economic\_functions*  
 module, 136  
*multi\_vector\_simulator.D0\_modelling\_and\_optimization*  
 module, 141

`multi_vector_simulator.D1_model_components`  
    module, 143  
`multi_vector_simulator.D2_model_constraints`  
    module, 151  
`multi_vector_simulator.E0_evaluation`  
    module, 155  
`multi_vector_simulator.E1_process_results`  
    module, 156  
`multi_vector_simulator.E2_economics`  
    module, 162  
`multi_vector_simulator.E3_indicator_calculation`  
    module, 166  
`multi_vector_simulator.E4_verification`  
    module, 176  
`multi_vector_simulator.F0_output`  
    module, 178  
`multi_vector_simulator.F1_plotting`  
    module, 179  
`multi_vector_simulator.F2_autoreport`  
    module, 184  
`multi_vector_simulator.utils.data_parser`  
    module, 111  
`multi_vector_simulator.utils.helpers`  
    module, 112  
`mvs_arg_parser()` (in module  
    *multi\_vector\_simulator.A0\_initialization*),  
    115

## N

`net_zero_energy_constraint_test()` (in module  
    *multi\_vector\_simulator.E4\_verification*), 177

## O

`open_in_browser()` (in module  
    *multi\_vector\_simulator.F2\_autoreport*), 186

## P

`parse_simulation_log()` (in module  
    *multi\_vector\_simulator.F0\_output*), 178  
`peak_demand_bus_name()` (in module  
    *multi\_vector\_simulator.utils.helpers*), 113  
`peak_demand_transformer_name()` (in module  
    *multi\_vector\_simulator.utils.helpers*), 113  
`plot_instant_power()` (in module  
    *multi\_vector\_simulator.F1\_plotting*), 183  
`plot_networkx_graph()`  
    (*multi\_vector\_simulator.D0\_modelling\_and\_optimization.model\_building*  
    method), 142  
`plot_optimized_capacities()` (in module  
    *multi\_vector\_simulator.F1\_plotting*), 183  
`plot_piecharts_of_costs()` (in module  
    *multi\_vector\_simulator.F1\_plotting*), 183  
`plot_sankey()` (in module  
    *multi\_vector\_simulator.F1\_plotting*), 183

`plot_sankey_diagramm()`  
    (*multi\_vector\_simulator.D0\_modelling\_and\_optimization.model\_building*  
    method), 142  
`plot_timeseries()` (in module  
    *multi\_vector\_simulator.F1\_plotting*), 183  
`prepare_constraint_minimal_renewable_share()`  
    (in module *multi\_vector\_simulator.D2\_model\_constraints*),  
    153  
`prepare_demand_assets()` (in module  
    *multi\_vector\_simulator.D2\_model\_constraints*),  
    153  
`prepare_energy_provider_consumption_sources()`  
    (in module *multi\_vector\_simulator.D2\_model\_constraints*),  
    154  
`prepare_energy_provider_feedin_sinks()` (in  
    module *multi\_vector\_simulator.D2\_model\_constraints*),  
    154  
`present_value_from_annuity()` (in module  
    *multi\_vector\_simulator.C2\_economic\_functions*),  
    141  
`print_pdf()` (in module  
    *multi\_vector\_simulator.F2\_autoreport*), 187  
`process_all_assets()` (in module  
    *multi\_vector\_simulator.C0\_data\_processing*),  
    129  
`process_fixcost()` (in module  
    *multi\_vector\_simulator.E0\_evaluation*), 155  
`process_maximum_cap_constraint()` (in module  
    *multi\_vector\_simulator.C0\_data\_processing*),  
    130  
`process_normalized_installed_cap()` (in module  
    *multi\_vector\_simulator.C0\_data\_processing*),  
    130  
`process_user_arguments()` (in module  
    *multi\_vector\_simulator.A0\_initialization*),  
    116

## R

`ready_capacities_plots()` (in module  
    *multi\_vector\_simulator.F2\_autoreport*), 187  
`ready_costs_pie_plots()` (in module  
    *multi\_vector\_simulator.F2\_autoreport*), 187  
`ready_flows_plots()` (in module  
    *multi\_vector\_simulator.F2\_autoreport*), 187  
`ready_sankey_diagram()` (in module  
    *multi\_vector\_simulator.F2\_autoreport*), 188  
`ready_timeseries_plots()` (in module  
    *multi\_vector\_simulator.F2\_autoreport*), 188  
`receive_timeseries_from_csv()` (in module  
    *multi\_vector\_simulator.C0\_data\_processing*),  
    131  
`render()` (*multi\_vector\_simulator.F1\_plotting.ESGraphRenderer*  
    method), 180  
`replace_nans_in_timeseries_with_0()` (in module

`multi_vector_simulator.C0_data_processing`), 149  
`report_arg_parser()` (in module `multi_vector_simulator.A0_initialization`), 117  
`retrieve_date_time_info()` (in module `multi_vector_simulator.B0_data_input_json`), 120  
`run_oemof()` (in module `multi_vector_simulator.D0_modelling_and_optimization`), 143

**S**

`sankey()` (`multi_vector_simulator.F1_plotting.ESGraphRenderer` method), 180  
`save_plots_to_disk()` (in module `multi_vector_simulator.F1_plotting`), 184  
`simulating()` (`multi_vector_simulator.D0_modelling_and_optimization` method), 142  
`simulation_annuity()` (in module `multi_vector_simulator.C2_economic_functions`), 141  
`sink()` (in module `multi_vector_simulator.D1_model_components`), 145  
`sink_dispatchable_optimize()` (in module `multi_vector_simulator.D1_model_components`), 146  
`sink_non_dispatchable()` (in module `multi_vector_simulator.D1_model_components`), 146  
`source()` (in module `multi_vector_simulator.D1_model_components`), 147  
`source_dispatchable_fix()` (in module `multi_vector_simulator.D1_model_components`), 147  
`source_dispatchable_optimize()` (in module `multi_vector_simulator.D1_model_components`), 148  
`source_non_dispatchable_fix()` (in module `multi_vector_simulator.D1_model_components`), 148  
`source_non_dispatchable_optimize()` (in module `multi_vector_simulator.D1_model_components`), 148  
`stop()` (`multi_vector_simulator.D0_modelling_and_optimization.timer` method), 143

**T**

`timer` (class in `multi_vector_simulator.D0_modelling_and_optimization`), 143  
`total_demand_and_excess_each_sector()` (in module `multi_vector_simulator.E3_indicator_calculation`), 175  
`transformer()` (in module `multi_vector_simulator.D1_model_components`), 149  
`transformer_constant_efficiency_fix()` (in module `multi_vector_simulator.D1_model_components`), 150  
`transformer_constant_efficiency_optimize()` (in module `multi_vector_simulator.D1_model_components`), 150  
`translate_optimizeCap_from_boolean_to_yes_no()` (in module `multi_vector_simulator.E1_process_results`), 162  
`translates_epa_strings_to_mvs_readable()` (in module `multi_vector_simulator.utils.helpers`), 113  
`treat_multiple_flows()` (in module `multi_vector_simulator.C0_data_processing`), 131

**V**

`verify_state_of_charge()` (in module `multi_vector_simulator.E4_verification`), 177

**W**

`weighting_for_sector_coupled_kpi()` (in module `multi_vector_simulator.E3_indicator_calculation`), 176