
API Documentation OpEnCells

Release v. 0.1

Oct 05, 2020

CONTENTS:

1	Open energy cell simulation	1
1.1	OpEnCells	1
1.2	Introduction	1
1.3	To Dos	1
2	serializable module	3
3	simulatable module	5
4	data_loader module	7
5	load module	11
6	environment module	13
7	photovoltaic module	15
8	power_component module	19
9	power_junction module	21
10	battery module	23
11	Indices and tables	27
	Python Module Index	29

OPEN ENERGY CELL SIMULATION

1.1 OpEnCells

OpEnCellS (Open energy cell simulation) is a open-source modeling framework for the simulation of energy systems. The tool is developed by Fabian Schmid, written in Python and follows an object-oriented modeling approach. It enables a detailed simulation of various renewable energy technologies with a temporal resolution of one minute or one hour. Electricity sector components involves models for:

- Photovoltaic
- Power components
- Batteries

Power component technologies can be:

- Pulse-Width-Modulation (PWM) charge controller
- Maximum Power Point Tracker (MPPT) charge controller
- Battery Management Systems (BMS)

Models of heat sector components involve solarthermal collectors, pipes and stratified heat storages, which are currently under development. Future releases will include further hydrogen and wind turbine systems.

1.2 Introduction

This API documentation describes the technical content of the simulation framework OpEnCells. It contains instructions about how to effectively use the implemented classes and methods. For an example energy system model it is referred to the OpEnCellS full documentation.

1.3 To Dos

- Include Installation instructions in file with command “pip install module” or list of necessary external libraries.
- Clean code: Start method of Simulatable in components necessary ?!
- Include good workflow for sensitivity analysis, where for every sensi run own json is generated and no separate sensitivty class is needed.

SERIALIZABLE MODULE

class serializable.**Serializable** (*file_path=None*)

Bases: object

Methods to make simulation serializable with json format, which makes it possible to automatically store and load components parameters in json file.

Parameters **file_path** (*string*) – File path where to store/load json file.

Note:

- **Example of how to save battery parameters**
 - anaconda prompt: navigate to base folder of simulation
 - open python
 - from components.serializable import Serializable
 - from component.battery import Battery
 - Serializable.save(Battery(None,None,None,None,"file path to battery.json"), "filepath to store new battery.json")
 - Attention json file is very sensible, manipulate it in spyder or sufficated editor as atom.
-

load (*file_path=None*)

Load component parameter from specified json file to make it attributes of component class.

Parameters **file_path** (*string*) – File path where to load json file from.

save (*file_path=None*)

Save component parameter to json file from componnet class attributes.

Parameters **file_path** (*string*) – File path where to store json file.

SIMULATABLE MODULE

class simulatable.**Simulatable** (*childs)

Bases: object

Most central methods to make simulation “simulatable”, which means that timestep proceeds after the calculation of all component performance of the current timestep. Simulatable class - Parent class for simulation class and all component classes.

Parameters ***childs** (*class*) – All classes which shall be simulated

Note:

- **Class needs to be initialized in simulation class with system component as follows:**
 - e.g. **Simulatable.__init__(self, self.env, self.load, self.pv, self.charger, self.power_junction, self.battery_management, self.battery)**

calculate ()

Null methods - stands for for calculation methods of component classes.

Parameters **None** (*None*) –

end ()

End Method, which terminates Simulatable with end method for all childs and sets time index back to zero.

Parameters **None** (*None*) –

start ()

Start Method, which initialize start method for all childs and sets time index to zero.

Parameters **None** (*None*) –

update ()

Method, which updates time index and goes to next simulation step for all childs.

Parameters **None** (*None*) –

DATA_LOADER MODULE

class data_loader.CSV

Bases: object

Relevant methods of CSV loader in order to load csv file of CAMS Radiation Service, MERRA Weather Data or load profile.

Parameters

- **file_name** (*str*) – File path and name of file to be loaded.
- **start** (*ind*) – First timestep of csv file to be loaded.
- **end** (*ind*) – Last timestep of csv file to be loaded.

Note:

- Class is parent class of MeteoIrradiation, MeteoWeather and LoadDemand.
-

get_column (*i*)

Extracts specific column of loaded Pandas Dataframe by read_csv().

Parameters **i** (*int*) – Column to be extracted from Pandas Dataframe __data_set.

Returns __data_set – Pandas Series with specified column in case loaded pandas Dataframe has multiple columns.

Return type *Pandas.Series*

read_csv (*file_name, start, end*)

Loads the csv file and stores it in parameter __data_set

Parameters

- **file_name** (*str*) – File path and name of file to be loaded.
- **start** (*int*) – First timestep of csv file to be loaded.
- **end** (*int*) – Last timestep of csv file to be loaded.

Returns __data_set – Pandas Dataframe with extracted data rows.

Return type *Pandas.DataFrame*

class data_loader.LoadDemand

Bases: *data_loader.CSV*

Relevant method to load load profile from csv file.

Parameters **None** (*None*) –

Returns `day_profile` – [Wh] Pandas series of load profile specifies load demand per tiestep in watthours.

Return type `pandas.series float`

Note:

- Implemented method is usually integrated in load class to directly load load profile data.
-

get_day_profile()

Returns day load profile

class `data_loader.MeteoIrradiation`

Bases: `data_loader.CSV`

Relevant methods to extract data from CAMS Radiation Service Dataset.

Parameters `None` (*None*) –

Returns

- **time** (`pandas.series int`) – Timestamp of loaded irradiation dataset.
- **irradiance_toa** (`pandas.series float`) – [Wh/m2] Irradiation on horizontal plane at the top of atmosphere.
- **ghi_clear_sky** (`pandas.series float`) – [Wh/m2] Clear sky global irradiation on horizontal plane at ground level.
- **bhi_clear_sky** (`pandas.series float`) – [Wh/m2] Clear sky beam irradiation on horizontal plane at ground level.
- **dhi_clear_sky** (`pandas.series float`) – [Wh/m2] Clear sky diffuse irradiation on horizontal plane at ground level.
- **bni_clear_sky** (`pandas.series float`) – [Wh/m2] Clear sky beam irradiation on mobile plane following the sun at normal incidence.
- **ghi** (`pandas.series float`) – [Wh/m2] Global irradiation on horizontal plane at ground level.
- **bhi** (`pandas.series float`) – [Wh/m2] Beam irradiation on horizontal plane at ground level.
- **dhi** (`pandas.series float`) – [Wh/m2] Diffuse irradiation on horizontal plane at ground level.
- **bni** (`pandas.series float`) – [Wh/m2] Beam irradiation on mobile plane following the sun at normal incidence.
- **reliability** (`pandas.series float`) – [1] Proportion of reliable data in the summarization (0-1).

Note:

- Implemented methods are usually integrated in other classes to directly load irradiance data.
 - Examples are the classes `environment`, `load`, `photovoltaic` or `battery`.
 - **Cams irradiance datasets can be downloaded at:**
 - <http://www.soda-pro.com/web-services/radiation/cams-radiation-service>
-

get_bhi()

Returns [Wh/m2] Beam irradiation on horizontal plane at ground level.

get_bhi_clear_sky()

Returns [Wh/m2] Clear sky beam irradiation on horizontal plane at ground level.

get_bni()

Returns [Wh/m2] Beam irradiation on mobile plane following the sun at normal incidence.

get_bni_clear_sky()

Returns [Wh/m2] Clear sky beam irradiation on mobile plane following the sun at normal incidence.

get_dhi()

Returns [Wh/m2] Diffuse irradiation on horizontal plane at ground level.

get_dhi_clear_sky()

Returns [Wh/m2] Clear sky diffuse irradiation on horizontal plane at ground level.

get_ghi()

Returns [Wh/m2] Global irradiation on horizontal plane at ground level.

get_ghi_clear_sky()

Returns [Wh/m2] Clear sky global irradiation on horizontal plane at ground level.

get_irradiance_toa()

Returns [Wh/m2] Irradiation on horizontal plane at the top of atmosphere.

get_reliability()

Returns [1] Proportion of reliable data in the summarization (0-1).

get_time()

Returns Timestamp of loaded irradiation dataset.

class data_loader.MeteoWeather

Bases: *data_loader.CSV*

Relevant methods to extract data from MERRA Dataset.

Parameters **None** (*None*) –

Returns

- **date** (*pandas.seriesint*) – Date with format YYYY-MM-DD.
- **time** (*pandas.series int*) – Time of day with format HH-MM.
- **temperature** (*pandas.series float*) – [K] Ambient temperature at 2 m above ground.
- **humidity** (*pandas.series float*) – [%] Relative humidity at 2 m above ground.
- **wind_speed** (*pandas.series float*) – [m/s] Wind speed at 10 m above ground.
- **wind_direction** (*pandas.series float*) – [°] Wind direction at 10 m above ground (0 means from North, 90 from East. . .).
- **rainfall** (*pandas.series float*) – [mm] Rainfall (= rain depth in mm).
- **snowfall** (*pandas.series float*) – [kg/m2] Snowfall.
- **snow_depth** (*pandas.series float*) – [m] Snow depth.

Note:

- Implemented methods are usually integrated in other classes to directly load weather data.
- Examples are the classes environment, load, photovoltaic or battery.
- **MERRA datasets can be downloaded at:**

– <http://www.soda-pro.com/web-services/meteo-data/merra>

get_air_pressure()

Returns Pressure (hPa); Pressure at ground level

get_date()

Returns Date. format YYYY-MM-DD

get_humidity()

Returns Relative humidity (%); Relative humidity at 2 m above ground

get_rainfall()

Returns Rainfall (kg/m²); Rainfall (= rain depth in mm)

get_snow_depth()

Returns Snow depth (m); Snow depth

get_snowfall()

Returns Snowfall (kg/m²); Snowfall

get_temperature()

Returns Temperature (K); Temperature at 2 m above ground

get_time()

Returns Time of day. format HH-MM

get_wind_direction()

Returns Wind direction (deg); Wind direction at 10 m above ground (0 means from North, 90 from East. ...)

get_wind_speed()

Returns Wind speed (m/s); Wind speed at 10 m above ground

LOAD MODULE

class load.Load

Bases: *simulatable.Simulatable*

Relevant methods to define the simulation load profile power.

Parameters **None** (*None*) –

Note:

- Class data_loader is integrated and its method LoadDemand() to integrate csv load.
 - This method is called externally before the central method simulate() of the class simulation is called.
-

calculate ()

Extracts power flow of load profile for each timestep in order to make class simulatable..

Parameters **None** (*None*) –

Returns **power** – [W] Load power flow of timestep in watts.

Return type *float*

ENVIRONMENT MODULE

class environment.**Environment** (*timestep, system_orientation, system_location*)

Bases: object

Relevant methods for the calculation of the global irradiation and sun position.

Parameters

- **timestep** (*int*) – [s] Simulation timestep in seconds.
- **system_orientation** (*floats*) – [°] Tuple of floats defining the system orientation with system azimuth and inclination.
- **system_location** (*floats*) – [°] Tuple of floats defining the system location coordinates with longitude and latitude.

Note:

- **System orientation (tuple of floats)**
 - 1. tuple entry system azimuth in degrees [°]. Panel azimuth from north (0°=north, 90°=east, 180°=south, 270°=west).
 - 2. tuple entry system inclination in degrees [°]. Panel tilt from horizontal.
- **System location (tuple of floats)**
 - 1. tuple entry system longitude in degrees [°]. Positive east of prime meridian, negative west of prime meridian.
 - 2. tuple entry system latitude in degrees [°]. Positive north of equator, negative south of equator.

load_data ()

Loads and calculates all relevant environment data, including total, beam, sky, ground irradiation (pvlib), temperature in [K] and windspeed data in [m/s] sun position (pvlib) and angle of inclination (pvlib).

Parameters **None** (*None*) –

Returns

- **sun_position_pvlib** (*floats*) – [°] Tuple of floats, defining the sun position with its elevation and azimuth angle.
- **sun_aoi_pvlib** (*floats*) – [°] Angle of incidence of the solar vector on the module surface.
- **sun_irradiance_pvlib** (*floats*) – [W/m²] Plane on array irradiation (total, beam, sky, ground).
- **power** (*float*) – [Wh] Total plane on array irradiation.

Note:

- All models are based on pvlib library version 0.7.1.
 - **Solar position calculator (pvlib)**
 - `pvlib.solarposition.get_solarposition(time, latitude, longitude, altitude=None, pressure=None, method='nrel_numpy', temperature=12, kwargs)`
 - https://pvlib-python.readthedocs.io/en/stable/generated/pvlib.solarposition.get_solarposition.html
 - Further details, compare¹,² and³.
 - **Angle of incident calculator (pvlib)**
 - The angle of incidence of the solar vector and the module surface normal.
 - `pvlib.irradiance.aoi(surface_tilt, surface_azimuth, solar_zenith, solar_azimuth)`
 - <https://pvlib-python.readthedocs.io/en/stable/generated/pvlib.irradiance.aoi.html>
 - **Total, beam, sky diffuse and ground reflected in-plane irradiance**
 - Calculated using the specified sky diffuse irradiance model (pvlib).
 - `pvlib.irradiance.get_total_irradiance(surface_tilt, surface_azimuth, solar_zenith, solar_azimuth, dni, ghi, dhi, dni_extra=None, airmass=None, albedo=0.25, surface_type=None, model='isotropic', model_perez='allsitescomposite1990', kwargs)`
 - https://pvlib-python.readthedocs.io/en/stable/generated/pvlib.irradiance.get_total_irradiance.html
 - **Class data_loader**
 - Integrated and its method `MeteoIrradiation()` and `MeteoWeather()` to integrate csv weather data.
 - This method is called externally before the central method `simulate()` of the class simulation is called.
-

¹ I. Reda and A. Andreas, Solar position algorithm for solar radiation applications. Solar Energy, vol. 76, no. 5, pp. 577-589, 2004.

² I. Reda and A. Andreas, Corrigendum to Solar position algorithm for solar radiation applications. Solar Energy, vol. 81, no. 6, p. 838, 2007.

³ NREL SPA code: <http://redc.nrel.gov/solar/codesandalgorithms/spa/>

PHOTOVOLTAIC MODULE

class photovoltaic.**Photovoltaic** (*timestep, peak_power, controller_type, env, file_path=None*)
Bases: *serializable.Serializable, simulatable.Simulatable*

Relevant methods for the calculation of photovoltaic performance.

Parameters

- **timestep** (*int*) – [s] Simulation timestep in seconds.
- **peak_power** (*int*) – [Wp] Installed PV peak power.
- **controller_type** (*string*) – Type of charge controller PWM or MPPT.
- **environment** (*class*) – To get access to solar irradiation, ambient temperature and wind-speed.
- **file_name** (*json*) – To load component parameters (optional).

Note:

- Photovoltaic with MPPT assumption or fixed voltage (PWM) is possible.
 - Corresponding charge controller type must be considered manually.
 - **Models are based on pvlib library version 0.7.1.**
 - compare <https://pvlib-python.readthedocs.io/en/stable/api.html>
 - **Photovoltaic model parameters based on SAM library.**
 - compare <https://sam.nrel.gov/photovoltaic/pv-sub-page-2.html>
-

calculate ()

Calculates and extracts all photovoltaic performance parameters from implemented methods.

Parameters **None** (*None*) –

Returns

- **temperature** (*float*) – [K] Photovoltaic cell temperature, equals temperature_cell.
- **power** (*float*) – [W] Photovoltaic overall power of specified installed array specified by parameter peak_power.
- **peak_power_current** (*float*) – [W] Photovoltaic current peak power assuming power degradation by implemented method photovoltaic_aging()
- **state_of_destruction** (*float*) – [-] Photovoltaic state of destruction as fraction of current and nominal peak power.

- **replacement** (*float*) – [s] Time of replacement in case state_of_destruction equals 1.

Note:

- **Method mainly extracts parameters by calling implemented methods:**
 - photovoltaic_aging()
 - photovoltaic_state_of_destruction()
-

load_data ()

Calculates all photovoltaic performance parameters by calling all methods based on pvlib.

Parameters **None** (*None*) –

Returns

- **temperature_cell** (*float*) – [K] Photovoltaic cell temperature, by calling method photovoltaic.temperature().
 - **power_module** (*float*) – [W] Photovoltaic module power of specified module, by calling methods photovoltaic.power_mppt() or photovoltaic_pwm().
-

Note:

- Photovoltaic power/temperature is calculated using pvlib library.
 - This library computes parameters not step by step but all in one for env data.
 - Method is equal to environment class, where all env data is loaded all in one.
 - Other parameters are calculated step by step in the method photovoltaic.calculate().
-

photovoltaic_aging ()

Calculates photovoltaic power degradation and current peak power in Watt [W] assuming constant power degradation.

Parameters **None** (-) –

Returns **peak_power_current** – [Wp] Photovoltaic current peak power in watt peak.

Return type *float*

photovoltaic_power_mppt ()

Calculates the Photovoltaic Maximum Power.

Parameters **None** (-) –

Returns **power** – [W] DC photovoltaic mpp power in watts.

Return type *float*

Note:

- **Model is based on NREL's PVWatts DC power model⁶.**
 - pvlib.pvsystem.pvwatts_dc(g_poa_effective, temp_cell, pdc0, gamma_pdc, temp_ref=25.0).
 - https://pvlib-python.readthedocs.io/en/stable/generated/pvlib.pvsystem.pvwatts_dc.html.

⁶

A. P. Dobos, "PVWatts Version 5 Manual" http://pvwatts.nrel.gov/downloads/pvwatts_v5.pdf (2014).

photovoltaic_power_pwm()

Calculates the photovoltaic power at given voltage through the VI curve determination with the single diode model and gets parameter for single diode model.

Parameters **None** (-) –

Returns

- **photocurrent** (*float*) – [A] Light-generated current in amperes
- **saturation_current** (*float*) – [A] Diode saturation current in amperes
- **resistance_series** (*float*) – [Ohm] Series resistance in ohms
- **resistance_shunt** (*float*) – [Ohm] Shunt resistance in ohms
- **nNsVth** (*float*) – (numeric) The product of the usual diode ideality factor (n, unitless), number of cells in series (Ns), and cell thermal voltage at specified effective irradiance and cell temperature.
- **current** (*np.ndarray/scalar*) – [A] Photovoltaic current in amperes at given voltage level.

Note:

- **To construct VI curve to determine power at given voltage level.**
 - Is based on model by Jain et al.².
 - `pvlib.pvsystem.i_from_v(resistance_shunt, resistance_series, nNsVth, voltage, saturation_current, photocurrent, method='lambertw')`
 - https://pvlib-python.readthedocs.io/en/stable/generated/pvlib.pvsystem.i_from_v.html#pvlib-pvsystem-i-from-v
- **Get values to construct single diode model.**
 - Is based on five parameter model, by De Soto et al. described in³.
 - Five values for the single diode equation at effective irradiance and cell temperature can be obtained by calling `calcparams_desoto`.
 - `pvlib.pvsystem.calcparams_desoto(effective_irradiance, temp_cell, alpha_sc, a_ref, I_L_ref, I_o_ref, R_sh_ref, R_s, EgRef=1.121, dEgdT=-0.0002677, irradiance_ref=1000, temp_ref=25)`
 - https://pvlib-python.readthedocs.io/en/stable/generated/pvlib.pvsystem.calcparams_desoto.html
 - results are returned as tuple of above listed returns.
- **To access pvlib module database and get parameters**
 - `modules_list = pvlib.pvsystem.retrieve_sam('CECMod') # 'SandiaMod'`
 - `module = modules_list['NuvoSun_FL0912_100']`
- Further references are⁴ and⁵.

² A. Jain, A. Kapoor, "Exact analytical solutions of the parameters of real solar cells using Lambert W-function", Solar Energy Materials and Solar Cells, 81 (2004) 269-277.

³ W. De Soto et al., "Improvement and validation of a model for photovoltaic array performance", Solar Energy, vol 80, pp. 78-88, 2006.

⁴ System Advisor Model web page. <https://sam.nrel.gov>.

⁵ A. Dobos, "An Improved Coefficient Calculator for the California Energy Commission 6 Parameter Photovoltaic Module Model", Journal of Solar Energy Engineering, vol 134, 2012.

photovoltaic_state_of_destruction()

Calculates the photovoltaic state of destruction (SoD) and time of component replacement according to end of life criteria.

Parameters **None** (-) –

Returns

- **state_of_destruction** (*float*) – [1] Photovoltaic State of destruction with SoD=1 representing a broken component.
- **replacement** (*int*) – [s] Time of photovoltaic component replacement in seconds.

Note:

- Replacement time is only set in timeseries array in case of a replacement, otherwise entry is 0.
 - In case of replacement `current_peak_power` is reset to nominal power.
-

photovoltaic_temperature()

Calculates photovoltaic cell temperature with the Sandia PV Array Performance Model integrated in pvlib.

Parameters **None** (-) –

Returns **temperature_cell** – [K] Photovoltaic cell temperature (ATTENTION: not C as specified in pvlib)

Return type *float*

Note:

- The Sandia PV Array Performance Model is based on¹.
 - **pvlib.temperature.sapm_cell** is called with
 - `pvlib.temperature.sapm_cell(poa_global, temp_air, wind_speed, a, b, deltaT, ir-rad_ref=1000)`
 - compare, https://pvlib-python.readthedocs.io/en/stable/generated/pvlib.temperature.sapm_cell.html
 - **For numerical values of different module configurations, call**
 - `pvlib.temperature.TEMPERATURE_MODEL_PARAMETERS`
-

¹ King, D. et al, 2004, “Sandia Photovoltaic Array Performance Model”, SAND Report 3535, Sandia National Laboratories, Albuquerque, NM.

POWER_COMPONENT MODULE

```
class power_component.Power_Component (timestep, power_nominal, input_link,  
                                         file_path=None)
```

Bases: *serializable.Serializable*, *simulatable.Simulatable*

Relevant methods for the calculation of power components performance.

Parameters

- **timestep** (*int*) – [s] Simulation timestep in seconds.
- **power_nominal** (*int*) – [W] Nominal power of power component in watt.
- **input_link** (*class*) – [-] Class of component which supplies input power.
- **file_path** (*json*) – To load components parameters (optional).

Note:

- Model is based on method by Sauer and Schmid¹.
 - **Model can be used for all power components with a power dependent efficiency.**
 - e.g. Charge controllers, BMS, power inverters...
-

calculate()

Calculates all power component performance parameters from implemented methods. Decides whether input_power or output_power method is to be called

Parameters *None* (*None*) –

Returns

- **efficiency** (*float*) – [1] Component efficiency
- **power** (*float*) – [W] Component input/output power in watts.
- **state_of_destruction** (*float*) – [-] Component state of destruction.
- **replacement** (*float*) – [s] time of replacement in case state_of_destruction equals 1.

calculate_efficiency_input (*input_link_power*)

Calculates power component efficiency, dependent on Power Output eff(P_out).

Parameters *None* (-) –

Returns **efficiency** – [W] Component efficiency.

¹ D. U. Sauer and H. Schmidt, "Praxisgerechte Modellierung und Abschätzung von Wechselrichter-Wirkungsgraden", in 9. Internationales Sonnenforum - Tagungsband I, 1994, pp. 550–557

Return type *float*

Note:

- Calculated power output is NEGATIVE but fuction can only handle Positive value.
 - Therefore first abs(), at the end -
-

calculate_efficiency_output (*input_link_power*)

Calculates power component efficiency, dependent on Power Input eff(P_in).

Parameters **None** (-) –

Returns **efficiency** – [W] Component efficiency.

Return type *float*

calculate_power_input (*input_link_power*)

Calculates power component input power, dependent on Power Output P_in(P_out).

Parameters **None** (-) –

Returns **power** – [W] Component input power in watts.

Return type *float*

Note:

- Calculated power output is NEGATIVE but fuction can only handle Positive value.
 - Therefore first abs(), at the end -
-

calculate_power_output (*input_link_power*)

Calculates power component output power, dependent on Power Input P_out(P_in).

Parameters **None** (-) –

Returns **power** – [W] Component output power in watts.

Return type *float*

power_component_state_of_destruction ()

Calculates the component state of destruction (SoD) and time of component replacement according to end of life criteria.

Parameters **None** (-) –

Returns

- **state_of_destruction** (*float*) – [1] Component state of destruction.
- **replacement** (*int*) – [s] Time of component replacement in seconds.

POWER_JUNCTION MODULE

class power_junction.**Power_Junction** (*input_link_1*, *input_link_2*, *load*)

Bases: *simulatable.Simulatable*

Relevant methods of the power junction to compute all power input and output flows and the resulting battery power flow.

Parameters

- **input_link_1** (*class*) – [-] Component 1 that provides input power to junction
- **input_link_2** (*class*) – [-] Component 2 that provides input power to junction
- **load** (*class*) – [-] Load class to integrate load power flow

Note:

- Power junction can be enlarged with further input and output power flows.
-

calculate ()

Calculates needed battery power to balance input and output power flows of junction.

Parameters **None** (*None*) –

Returns **power** – [W] Power junction power flow to battery.

Return type *float*

BATTERY MODULE

class battery.**Battery** (*timestep, capacity_nominal_wh, input_link, env, file_path=None*)
Bases: *serializable.Serializable, simulatable.Simulatable*

Relevant methods for the calculation of battery performance.

Parameters

- **timestep** (*int*) – [s] Simulation timestep in seconds.
- **capacity_nominal_wh** (*int*) – [Wh] Installed battery capacity in watthours.
- **input_link** (*class*) – Class of component which supplies input power.
- **environment** (*class*) – To get access to solar irradiation, ambient temperature and wind-speed.
- **file_name** (*json*) – To load component parameters (optional).

Note:

- Different battery technologies can be modeled with this generic model approach.
 - Model parameter need to be loaded and parametrized externally.
 - So far model parameters for lithium-ion phosphate and lead acid batteries exist.
-

battery_aging_calendar ()

Calculates battery calendar aging according to specified float lifetime.

Parameters **None** (-) –

Returns

- **float_life** (*float*) – [a] Battery float lifetime according to battery temperature.
- **float_life_loss** (*float*) – [Wh] Battery absolute capacity loss per timestep due to calendar aging.

Note:

- Model is based on numerical fitting of float lifetime data given in battery datasheet.
- For detailed description of model parametrization, compare³.

³ F.Schmid, F.Behrendt “Optimal Sizing of Solar Home Systems: Charge Controller Technology and Its Influence on System Design” Under development.

References

battery_aging_cycling ()

Calculates battery cycling aging according to micro cycle approach.

Parameters **None** (-) –

Returns

- **cycle_life** (*float*) – [a] Battery cycle lifetime according to last micro cycle.
- **cycle_life_loss** (*float*) – [Wh] Battery absolute capacity loss per timestep due to cycling aging.

Note:

- Cycle life dependent on DoD and temperature if specified
 - The model is described in detailed in⁴ and⁵.
-

battery_charge_discharge_boundary ()

Calculates battery charge/discharge boundaries.

Parameters **None** (-) –

Returns **charge_discharge_boundary** – [1] Battery charge/discharge boundary.

Return type *float*

Note:

- The model describes the power-dependent charge and discharge boundaries.
 - For a detailed description of the parametrization approach [2].
-

battery_power ()

Calculates the battery efficiency & charging/discharging power in Watt.

Parameters **None** (-) –

Returns

- **power** (*float*) – [W] Battery charge/discharge power extracted from the battery.
- **efficiency** (*float*) – [1] Battery charge/discharge efficiency.

Note:

- The model describes a power-dependent efficiency.
 - For a detailed description of the parametrization approach [2].
-

⁴ Narayan et al. “A simple methodology for estimating battery lifetimes in Solar Home System design”, IEEE Africon 2017 Proceedings, 2017.
⁵ Narayan et al. “Estimating battery lifetimes in Solar Home System design using a practical modelling methodology, Applied Energy 228, 2018.

battery_state_of_charge ()

Calculates the battery state of charge.

Parameters **None** (-) –

Returns **state_of_charge** – [1] Battery state of charge.

Return type *float*

Note:

- Model is based on simple energy balance using an off-line book-keeping method.
 - Considers charge/discharge terminal power, power losses, self-discharge rate.
 - For a detailed description of the model².
-

battery_state_of_destruction ()

Calculates the battery state of destruction (SoD) and time of component replacement according to end of life criteria.

Parameters **None** (-) –

Returns

- **state_of_destruction** (*float*) – [1] Battery State of destruction with SoD=1 representing a broken component.
 - **replacement** (*int*) – [s] Time of battery component replacement in seconds.
-

Note:

- Replacement time is only set in timeseries array in case of a replacement, otherwise entry is 0.
 - In case of replacement **current_peak_power** is reset to nominal power.
-

battery_temperature ()

Calculates the battery temperature in Kelvin.

Parameters **None** (-) –

Returns **temperature** – [K] Battery temperature in Kelvin.

Return type *float*

Note:

- Thermal model is based on general heat balance and convective heat transport to the environment. - Compare heat balance by¹.
-

battery_voltage ()

Calculates battery voltage dependent on battery power and State of charge.

Parameters **None** (-) –

²

J. Winzer et al. "An open-source modeling tool for multi-objective optimization of renewable nano/micro-off-grid power supply system", Energy, IN REVIEW, 2020

¹ Bernardi, E. Pawlikowski, and J. Newman, 'A General Energy Balance for Battery Systems', J. Electrochem. Soc., vol. 132, no. 1, p. 5, 1985.

Returns **voltage** – [V] Battery voltage level.

Return type *float*

Note:

- The model describes the voltage dependent on charge/discharge power and state of charge.
-

calculate()

Calculates all battery performance parameters from implemented methods.

Parameters **None** (*None*) –

Returns

- **temperature** (*float*) – [K] Battery temperature in Kelvin.
- **power** (*float*) – [W] Battery charge/discharge power extracted from the battery.
- **efficiency** (*float*) – [1] Battery charge/discharge efficiency.
- **state_of_charge** (*float*) – [1] Battery state of charge.
- **charge_discharge_boundary** (*float*) – [1] Battery charge/discharge boundary.
- **capacity_current_wh** (*float*) – [Wh] Battery capacity of current timestep.
- **state_of_health** (*float*) – [1] Battery state of health.
- **state_of_destruction** (*float*) – [1] Battery state of destruction.
- **voltage** (*float*) – [V] Battery voltage level.

Note:

- **Method mainly extracts parameters by calling implemented methods:**
 - battery_temperature()
 - battery_power()
 - battery_state_of_charge()
 - battery_charge_discharge_boundary()
 - battery_voltage()
 - battery_aging_cycling()
 - battery_aging_calendar()
 - It includes the charge discharge algorithm of the battery.
-

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

b

battery, [23](#)

d

data_loader, [7](#)

e

environment, [13](#)

l

load, [11](#)

p

photovoltaic, [15](#)

power_component, [19](#)

power_junction, [21](#)

s

serializable, [3](#)

simulatable, [5](#)