Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Department of Electrical and Sustainable Energy
Group of DC Systems, Energy Conversion and Storage

User Manual

MPPT_PO_controller
PeakShaving_Curtailment
Example_MPPT
Example_PeakShaving_Curtailment

Author:
Dr. Joel Alpízar-Castillo

Version 1.0
31 July, 2025

Contents

Chapter 1

Introduction

This document provides a description of the functions used to simulate the residential multi-carrier energy systems, together with a low-voltage distribution networks in the dissertation [1]. Each chapter provides a description of each of the functions used, including a brief mathematical explanation and details of the arguments. In addition, example files are also provided.

Note that all the functions and example files were tested in Sypder 5.5.1, using Python 3.12.7, and Windows 10.

The FLEXINet project was carried out with a Top Sector Energy subsidy from the Ministry of Economic Affairs and Climate, carried out by the Netherlands Enterprise Agency (RVO). The specific subsidy for this project concerns the MOOI subsidy round 2020.

Chapter 2

csvreader

This library was not created for the FLEXINet project, but as part of the MSc thesis [2]. Nevertheless, as it is constantly used, its functions are also detailed. The library itself is stored in the repository in [3].

For this library to be used, the following module must be installed:

- itertools
- pandas
- numpy

2.1 read_data Class

The `read_data` class provides a lightweight interface for reading and processing CSV files without relying on full-featured data frames. It allows conversion of CSV data into lists of columns, rows, matrices, or arrays.

Constructor

```
read_data(csv='', delim=';', address='CSVs\\')
```

Arguments:

- **csv [str]:** Name of the CSV file, including extension (e.g., `"data.csv"`).
- **delim [str, optional]:** Delimiter used in the CSV file (default is `';'`).
- **address [str, optional]:** Path to the folder containing the CSV file (default is `'CSVs
  '`).

Attributes:

- **df:** The raw data frame read from the CSV.
- **nrows:** Number of rows in the data.
- **ncols:** Number of columns in the data.

Methods

data2cols()

Converts the data frame into a list of columns. Each column is stored as a list in `self.col`.

data2rows()

Converts the data frame into a list of rows. Each row is stored as a list in `self.row`.

data2matrix()

Converts the data frame into a NumPy matrix and stores it in `self.mat`.

data2array()

Converts the data frame into a NumPy array and stores it in `self.ar`.

Chapter 3

MPPT_PO_Controller

This library contains the functions used to simulate maximum power point-tracker, used for power curtailment, as described in [4]. The library itself is stored in the repository in [5].

For this library to be used, the following modules must be installed:

- math
- matplotlib

In addition, the following libraries must be used:

- csvreader.py

3.1 PV_degradation()

This function models the linear degradation of photovoltaic (PV) panel performance over time. It estimates the relative power output of a PV module after a given number of years, assuming a linear decline from an initial to a final performance level.

Formula:

$$P(t) = \left(\frac{p_f - p_0}{\Delta t} \right) \cdot t + p_0 \tag{3.1}$$

where:

- $P(t)$ is the relative power output after t years,
- p_0 is the initial performance (default 0.975),
- p_f is the final performance after Δt years (default 0.8 after 25 years).

Arguments:

- **year [float]:** Number of years since installation.
- **p0 [float, optional]:** Initial performance ratio (default is 0.975).
- **pf [float, optional]:** Final performance ratio after degradation period (default is 0.8).
- **dt_years [int, optional]:** Total degradation period in years (default is 25).

3.2 PV_Current_Power()

This function calculates the current and power output of a photovoltaic (PV) module based on its voltage, solar irradiance, and temperature, following the method proposed in [6]. It uses a physics-based model derived from semiconductor characteristics and manufacturer data.
Model Basis:

- Based on the single-diode model of a solar cell.
- Incorporates temperature-dependent short-circuit current and open-circuit voltage.
- Uses Newton-Raphson iteration to solve the implicit current-voltage relationship.

Formula

$$I\left(V_{\mathrm{PV}}, G, T\right) = I_L\left(G, T\right) - I_0 \exp\left[\frac{q\left(V_{\mathrm{PV}} + I R_s\right)}{n\,k\,T} - 1\right], \tag{3.2}$$

with

$$I_L\left(G, T\right) = I_{L_{T_1}}\left(G\right)\left[1 + K_0\left(T - T_1\right)\right], \tag{3.3}$$

$$I_{L_{T_1}}\left(G\right) = \frac{G\, I_{\mathrm{SC}_{T_1}}}{G_{\mathrm{ref}}}, \tag{3.4}$$

$$K_0 = \frac{I_{\mathrm{SC}_{T_2}} - I_{\mathrm{SC}_{T_1}}}{T_2 - T_1} I_0 = I_{0_{(T_1)}} \left(\frac{T}{T_1}\right)^{\frac{3}{n}} \exp\left[\frac{-qV_g}{n\,k}\left(\frac{1}{T} - \frac{1}{T_1}\right)\right], \tag{3.5}$$

$$I_{0_{(T_1)}} = \frac{I_{\mathrm{SC}_{T_1}}}{\exp\left(\frac{qV_{\mathrm{OC}_{T_1}}}{n\,k\,T_1} - 1\right)}, \tag{3.6}$$

$$R_S = -\frac{\mathrm{d}V_{\mathrm{PV}}}{\mathrm{d}I_{V_{\mathrm{OC}}}} - \frac{1}{X_V} \tag{3.7}$$

$$X_V = I_{0_{(T_1)}} \left(\frac{q}{n\,k\,T_1}\right) \exp\left(\frac{qV_{\mathrm{OC}_{T_1}}}{n\,k\,T_1}\right) \tag{3.8}$$

Arguments:

- **V [float]:** Voltage across the PV module in volts.
- **G [float]:** Solar irradiance in W/m^2 (1 Sun = 1000 W/m^2).
- **T [float]:** Ambient temperature in degrees Celsius.

Notes:

- The function uses constants such as Boltzmann's constant, electron charge, diode quality factor, and bandgap voltage.
- Manufacturer data for short-circuit current and open-circuit voltage at reference temperatures is embedded.
- The output is computed iteratively for accuracy.

3.3 PV_curves()

This function generates the current-voltage (I-V) and power-voltage (P-V) curves of a photovoltaic (PV) module under specified irradiance and temperature conditions. It can also return the maximum power point (MPP) or the full curve data.
Process:

- Iterates over a voltage range to compute current and power using the `PV_Current_Power()` model.
- Optionally plots the I-V and P-V curves.
- Returns either the maximum power point or the full curve data.

Arguments:

- **G [float]:** Solar irradiance in W/m^2.
- **T [float]:** Ambient temperature in degrees Celsius.
- **Vmin [float, optional]:** Minimum voltage for the curve (default is 0 V).
- **Vmax [float, optional]:** Maximum voltage for the curve (default is 55 V).
- **dV [float, optional]:** Voltage step size (default is 0.1 V).
- **reuturn_lists [bool, optional]:** If `True`, returns full I-V and P-V curve data (default is `False`).
- **plots [bool, optional]:** If `True`, displays plots of the I-V and P-V curves (default is `False`).

3.4 Curtailment_voltage()

This function determines the voltage at which a photovoltaic (PV) module should operate to limit its power output to a specified curtailed value. It uses the P-V curve to find the closest voltage corresponding to the curtailed power.
Process:

- Generates the P-V curve using the `PV_curves()` function.
- Searches for the voltage at which the PV power output is equal to or just above the curtailed power.
- If the curtailed power exceeds the maximum available power, the voltage at the maximum power point is returned.

Arguments:

- **P_curtailed [float]:** Target curtailed power in watts.
- **G [float]:** Solar irradiance in W/m^2.
- **T [float]:** Ambient temperature in degrees Celsius.
- **Vmin [float, optional]:** Minimum voltage for the search range (default is 0 V).

- **Vmax [float, optional]:** Maximum voltage for the search range (default is 60 V).
- **dV [float, optional]:** Voltage step size (default is 0.1 V).
- **reutrn_lists [bool, optional]:** Passed to `PV_curves()` to return full curve data (default is `True`).
- **plots [bool, optional]:** If `True`, plots the I-V and P-V curves (default is `False`).

3.5 PO_algorithm()

This function implements the Perturb and Observe (P&O) algorithm for Maximum Power Point Tracking (MPPT) in photovoltaic (PV) systems. It adjusts the operating voltage of the PV module to maximize power output based on the observed change in power.
Logic:

- If the current power is greater than the previous power:
  - Increase voltage if current voltage is greater than previous.
  - Decrease voltage otherwise.
- If the current power is less than the previous power:
  - Decrease voltage if current voltage is greater than previous.
  - Increase voltage otherwise.
- If the power is unchanged, maintain the current voltage.

Arguments:

- **V_module [float]:** Current operating voltage of the PV module.
- **G [float]:** Solar irradiance in W/m^2.
- **T [float]:** Ambient temperature in degrees Celsius.
- **V_ref [float]:** Previous voltage reference.
- **I_ref [float]:** Previous current reference (not used in logic).
- **P_ref [float]:** Previous power reference.
- **Vmin [float, optional]:** Minimum allowable voltage (default is 0 V).
- **Vmax [float, optional]:** Maximum allowable voltage (default is 55 V).
- **dV [float, optional]:** Voltage perturbation step (default is 0.1 V).

3.6 PO_algorithm_curtailment()

This function extends the Perturb and Observe (P&O) algorithm for Maximum Power Point Tracking (MPPT) to include power curtailment. It adjusts the operating voltage of a photovoltaic (PV) module to approach a target curtailed power level while still following the P&O logic.
Logic:

- If the current power equals the curtailed power, maintain the current voltage.
- If the current power is closer to the curtailed power than the previous reference:
  - Increase voltage if current voltage is greater than reference.
  - Decrease voltage otherwise.
- If the current power is further from the curtailed power than the reference:
  - Decrease voltage if current voltage is greater than reference.
  - Increase voltage otherwise.

Arguments:

- **V_module [float]:** Current operating voltage of the PV module.
- **G [float]:** Solar irradiance in W/m^2.
- **T [float]:** Ambient temperature in degrees Celsius.
- **V_ref [float]:** Previous voltage reference.
- **I_ref [float]:** Previous current reference (not used in logic).
- **P_ref [float]:** Previous power reference.
- **P_curtailment [float]:** Target curtailed power in watts.
- **Vmin [float, optional]:** Minimum allowable voltage (default is 0 V).
- **Vmax [float, optional]:** Maximum allowable voltage (default is 55 V).
- **dV [float, optional]:** Voltage perturbation step (default is 0.1 V).

3.7 MPPT()

This function simulates the Maximum Power Point Tracking (MPPT) process for a photovoltaic (PV) module using the Perturb and Observe (P&O) algorithm. It optionally includes power curtailment and generates time series data for voltage, current, and power over a specified duration.
Process:

- Runs the P&O algorithm iteratively over a time interval.
- If curtailment is specified, uses the modified algorithm `PO_algorithm_curtailment()`.
- Tracks and stores voltage, current, and power at each time step.
- Optionally plots the time evolution of these quantities.

Arguments:

- **V_module [float]:** Initial operating voltage of the PV module.
- **G [float]:** Solar irradiance in W/m^2.
- **T [float]:** Ambient temperature in degrees Celsius.
- **P_curtailment [float, optional]:** Target curtailed power (default is 0, meaning no curtailment).
- **V_ref, I_ref, P_ref [float, optional]:** Initial reference values for voltage, current, and power (default is 0).
- **dV [float, optional]:** Voltage perturbation step (default is 0.01 V).
- **f [float, optional]:** Sampling frequency in Hz (default is 4000 Hz).
- **dt [float, optional]:** Duration of simulation in hours (default is 1/60 hours).
- **Vmin [float, optional]:** Minimum allowable voltage (default is 0 V).
- **Vmax [float, optional]:** Maximum allowable voltage (default is 55 V).
- **plots [bool, optional]:** If `True`, plots voltage, current, and power over time (default is `False`).

Chapter 4

Peakshaving_Curtailment

This library contains the functions used to estimate the degradation of a switch used in a boost converter due to power curtailment, as described in [4]. The library itself is stored in the repository in [5].

For this library to be used, the following modules must be installed:

- math
- rainflow
- matplotlib
- numpy
- statistics
- matplotlib
- csv

In addition, the following libraries must be used:

- MPPT_PO_controller

4.1 Yearly_Curtailed_Power()

This function processes a registry of yearly photovoltaic (PV) power data and optionally downsamples it to a lower frequency. It is typically used to prepare PV power data for further analysis or visualization.
Process:

- If sampling is enabled, the function selects every `sampling_frequency`-th data point from each year's PV power profile.
- Optionally converts power units from kilowatts to watts.

Arguments:

- **P_PV_registry [list]:** A list of yearly PV power profiles. Each profile is a list of power values.

- **Sampling [bool, optional]:** Whether to apply downsampling (default is `True`).
- **sampling_frequency [int, optional]:** Interval at which to sample the data (default is 4).
- **kW [bool, optional]:** If `True`, converts power values from kW to W (default is `True`).

4.2 Yearly_Voltage()

This function computes the yearly voltage profiles of a photovoltaic (PV) system based on curtailed power values, irradiance, and ambient temperature. It uses the `Curtailment_voltage()` function to determine the operating voltage required to achieve the curtailed power.
Process:

- Optionally downsamples the PV power registry.
- For each time step, calculates the voltage required to deliver the curtailed power using irradiance and temperature data.
- Scales the voltage by the number of modules and series connections.

Arguments:

- **P_PV_registry [list]:** List of yearly PV power profiles.
- **G [list]:** Global irradiance values corresponding to the PV data.
- **T_amb [list]:** Ambient temperature values corresponding to the PV data.
- **Sampling [bool, optional]:** Whether to apply downsampling (default is `True`).
- **sampling_frequency [int, optional]:** Sampling interval (default is 4).
- **n_modules [int, optional]:** Number of PV modules per string (default is 5).
- **n_series [int, optional]:** Number of series-connected strings (default is 1).

4.3 duty_cycle()

This function calculates the duty cycle of a DC-DC converter in a photovoltaic (PV) system. The duty cycle represents the fraction of time the switch is on during one switching period and is essential for estimating current and power flow in the converter.
Formula:

$$D = 1 - \frac{V_{\text{PV}}}{V_{\text{bus}}} \tag{4.1}$$

Arguments:

- **V_PV [float]:** Voltage of the PV module in volts.
- **V_bus [float, optional]:** Bus voltage of the converter (default is 400 V).

4.4 i_av_switch()

This function calculates the average current through the switch (typically an IGBT or MOSFET) in a DC-DC converter of a photovoltaic (PV) system. It uses the duty cycle and power output of the PV module to estimate the current.
Formula:

$$\bar{i}_{\text{switch}} = \frac{P_{\text{PV}} \cdot D}{V_{\text{PV}}} \tag{4.2}$$

where:

- P_{PV} is the power output of the PV module,
- D is the duty cycle,
- V_{PV} is the PV module voltage.

Arguments:

- **V_PV [float]:** Voltage of the PV module in volts.
- **P_PV [float]:** Power output of the PV module in watts.
- **V_bus [float, optional]:** Bus voltage of the converter (default is 400 V).

4.5 i_av_diode()

This function calculates the average current through the freewheeling diode in a DC-DC converter of a photovoltaic (PV) system. It complements the current through the switch and is based on the duty cycle and power output of the PV module.
Formula:

$$\bar{i}_{\text{diode}} = \frac{P_{\text{PV}} \cdot (1 - D)}{V_{\text{PV}}} \tag{4.3}$$

where:

- P_{PV} is the power output of the PV module,
- D is the duty cycle,
- V_{PV} is the PV module voltage.

Arguments:

- **V_PV [float]:** Voltage of the PV module in volts.
- **P_PV [float]:** Power output of the PV module in watts.
- **V_bus [float, optional]:** Bus voltage of the converter (default is 400 V).

4.6 i_RMS_switch()

This function calculates the root mean square (RMS) current through the switch (e.g., IGBT or MOSFET) in a DC-DC converter of a photovoltaic (PV) system. The RMS current is used to estimate conduction and switching losses in power electronics.
Formula:

$$i_{\mathrm{RMS}} = \sqrt{D\left(\frac{P_{\mathrm{PV}}}{V_{\mathrm{PV}}}\right)^2 + \frac{1}{3}\left(\frac{V_{\mathrm{PV}} \cdot D}{2fL}\right)^2} \tag{4.4}$$

where:

- D is the duty cycle,
- P_{PV} is the PV power output,
- V_{PV} is the PV voltage,
- f is the switching frequency,
- L is the inductance.

Arguments:

- **V_PV [float]:** Voltage of the PV module in volts.
- **P_PV [float]:** Power output of the PV module in watts.
- **V_bus [float, optional]:** Bus voltage of the converter (default is 400 V).
- **f [float, optional]:** Switching frequency in Hz (default is 20 kHz).
- **L [float, optional]:** Inductance in henries (default is 1.45 mH).

4.7 i_RMS_diode()

This function calculates the root mean square (RMS) current through the freewheeling diode in a DC-DC converter of a photovoltaic (PV) system. The RMS current is used to estimate thermal and conduction losses in the diode.
Formula:

$$i_{\mathrm{RMS}} = \sqrt{(1-D)\left(\frac{P_{\mathrm{PV}}}{V_{\mathrm{PV}}}\right)^2 + \frac{1}{3}\left(\frac{V_{\mathrm{PV}} \cdot D}{2fL}\right)^2} \tag{4.5}$$

where:

- D is the duty cycle,
- P_{PV} is the PV power output,
- V_{PV} is the PV voltage,
- f is the switching frequency,
- L is the inductance.

Arguments:

- **V_PV [float]:** Voltage of the PV module in volts.
- **P_PV [float]:** Power output of the PV module in watts.
- **V_bus [float, optional]:** Bus voltage of the converter (default is 400 V).
- **f [float, optional]:** Switching frequency in Hz (default is 20 kHz).
- **L [float, optional]:** Inductance in henries (default is 1.45 mH).

4.8 get_currents()

This function computes the average and RMS currents through both the switch and the diode in a DC-DC converter for a photovoltaic (PV) system over time. It processes time series data of PV voltage and power to evaluate current profiles for thermal and electrical analysis.
Process:

- Iterates over each year or scenario in the PV voltage and power registries.
- For each time step, calculates:
  - Average switch current using `i_av_switch()`.
  - RMS switch current using `i_RMS_switch()`.
  - Average diode current using `i_av_diode()`.
  - RMS diode current using `i_RMS_diode()`.

Arguments:

- **V_PV_registry [list]:** List of voltage profiles (one per year or scenario).
- **P_PV_registry [list]:** List of power profiles corresponding to the voltage profiles.
- **n_series [int, optional]:** Number of series-connected modules (default is 1).
- **V_bus [float, optional]:** Bus voltage of the converter (default is 400 V).
- **f [float, optional]:** Switching frequency in Hz (default is 20 kHz).
- **L [float, optional]:** Inductance in henries (default is 1.45 mH).

4.9 conduction_loss_IGBT()

This function estimates the conduction power loss in an Insulated Gate Bipolar Transistor (IGBT) based on the average and RMS current values. It is used in power electronics thermal modeling to evaluate the heat generated by the IGBT during operation.
Formula:

$$P_{\text{cond}} = V_T \cdot i_{\text{av}} + R_{CE} \cdot i_{\text{RMS}}^2 \tag{4.6}$$

Arguments:

- **i_av [float]:** Average current through the IGBT in amperes.
- **i_RMS [float]:** Root mean square (RMS) current through the IGBT in amperes.
- **R_CE [float, optional]:** Equivalent conduction resistance of the IGBT in ohms (default is 0.0856 Ω).
- **VT [float, optional]:** Threshold voltage of the IGBT in volts (default is 1.198 V).

4.10 energy_losses_IGBT()

This function estimates the energy loss per switching event in an Insulated Gate Bipolar Transistor (IGBT) based on empirical parameters and the average and RMS current values. It is typically used in the calculation of switching losses in power electronics systems.
Formula:

$$E_{\text{loss}} = \frac{1}{1000}\left(a + b \cdot i_{\text{av}} + c \cdot i_{\text{RMS}}^2\right) \tag{4.7}$$

Arguments:

- **i_av [float]:** Average current through the IGBT in amperes.
- **i_RMS [float]:** Root mean square (RMS) current through the IGBT in amperes.
- **dynamics [list of 3 floats, optional]:** Empirical coefficients $[a, b, c]$ used in the energy loss model (default is $[0.0195, 0.011, 0.0005]$).

4.11 switching_losses_IGBT()

This function calculates the total switching power losses in an Insulated Gate Bipolar Transistor (IGBT) during operation. It uses the energy loss per switching event and scales it by the switching frequency and voltage conditions.
Formula:

$$P_{\text{sw}} = f \cdot E_{\text{loss}} \cdot \frac{V_{\text{bus}}}{V_{\text{nom}}} \tag{4.8}$$

$$E_{\text{loss}} = \frac{1}{1000}\left(a + b \cdot i_{\text{av}} + c \cdot i_{\text{RMS}}^2\right) \tag{4.9}$$

Arguments:

- **i_av [float]:** Average current through the IGBT in amperes.
- **i_RMS [float]:** Root mean square (RMS) current through the IGBT in amperes.
- **f [float, optional]:** Switching frequency in hertz (default is 20 kHz).
- **V_bus [float, optional]:** DC bus voltage in volts (default is 400 V).
- **V_nom [float, optional]:** Nominal voltage used for scaling in volts (default is 600 V).

4.12 Calc_HeatSink_R()

This function estimates the required thermal resistance of a heatsink for an IGBT and diode pair, based on their maximum power losses and ambient temperature. It ensures that the junction temperature does not exceed the maximum allowable limit.
Formula:

$$R_{\text{ch,IGBT}} = \frac{T_{j,\text{max}} - Q_{\text{IGBT}} \cdot R_{\text{jc,IGBT}} - T_a}{Q_{\text{IGBT}} + Q_{\text{diode}}} \tag{4.10}$$

$$R_{\text{ch,diode}} = \frac{T_{j,\text{max}} - Q_{\text{diode}} \cdot R_{\text{jc,diode}} - T_a}{Q_{\text{IGBT}} + Q_{\text{diode}}} \tag{4.11}$$

$$R_{\text{ch}} = \min(R_{\text{ch,IGBT}}, R_{\text{ch,diode}}) \tag{4.12}$$

Arguments:

- **IGBT_losses_Registry [list of lists]:** Time series of IGBT power losses in watts.
- **Diode_losses_Registry [list of lists]:** Time series of diode power losses in watts.
- **T_amb [list]:** Ambient temperature in kelvin.
- **R_jc_IGBT [float, optional]:** Junction-to-case thermal resistance of the IGBT in °C/W (default is 1.7).
- **R_jc_diode [float, optional]:** Junction-to-case thermal resistance of the diode in °C/W (default is 4).
- **T_j_max [float, optional]:** Maximum allowable junction temperature in °C (default is 175).

4.13 Foster_impedance()

This function calculates the transient thermal impedance of a semiconductor device using the Foster network model. It is commonly used in thermal modeling to estimate the junction-to-case thermal impedance over time.
Formula:

$$Z_{\mathrm{jc}}(t) = \sum_{i=1}^{n} R_i \left(1 - e^{-t/\tau_i}\right) \tag{4.13}$$

Arguments:

- **R_i [list of floats, optional]:** Thermal resistances of the Foster model elements in °C/W (default is $[0.29566, 0.25779, 0.19382, 0.05279]$).
- **tau_i [list of floats, optional]:** Thermal time constants of the Foster model elements in seconds (default is $[6.478 \cdot 10^{-2}, 6.12 \cdot 10^{-3}, 4.679 \cdot 10^{-4}, 6.45 \cdot 10^{-5}]$).
- **t [float, optional]:** Time in seconds at which the impedance is evaluated (default is 100 s).

4.14 switch_total_losses_IGBT()

This function computes the total power losses in an Insulated Gate Bipolar Transistor (IGBT) due to both conduction and switching effects over a time series. It combines the conduction loss and switching loss models for each time step.
Formula:

$$P_{\mathrm{total}} = P_{\mathrm{cond}} + P_{\mathrm{sw}} = V_T \cdot i_{\mathrm{av}} + R_{CE} \cdot i_{\mathrm{RMS}}^2 + f \cdot E_{\mathrm{loss}} \cdot \frac{V_{\mathrm{bus}}}{V_{\mathrm{nom}}} \tag{4.14}$$

Arguments:

- **i_av_switch_registry [list of lists]:** Time series of average current values through the IGBT in amperes.
- **i_RMS_switch_registry [list of lists]:** Time series of RMS current values through the IGBT in amperes.
- **R_CE [float, optional]:** Equivalent conduction resistance of the IGBT in ohms (default is 0.0856 Ω).

- **VT [float, optional]:** Threshold voltage of the IGBT in volts (default is 1.198 V).
- **f [float, optional]:** Switching frequency in hertz (default is 20 kHz).
- **V_bus [float, optional]:** DC bus voltage in volts (default is 400 V).
- **V_nom [float, optional]:** Nominal voltage used for scaling in volts (default is 600 V).

4.15 switch_temperature_IGBT()

This function estimates the junction temperature of an Insulated Gate Bipolar Transistor (IGBT) based on its power losses and thermal impedance model. It supports both Foster and simplified thermal resistance models.
Formula (Foster model):

$$T_j = (P_{\text{cond}} + P_{\text{sw}}) \cdot (Z_{\text{jc}}(t) + R_{\text{tch}}) + T_{\text{amb}} \tag{4.15}$$

Formula (Simplified model):

$$T_j = (P_{\text{cond}} + P_{\text{sw}}) \cdot (R_{\text{th}}[0] + R_{\text{th}}[1]) + T_{\text{amb}} \tag{4.16}$$

Arguments:

- **T_amb [float]:** Ambient temperature in degrees Celsius.
- **i_av [float]:** Average current through the IGBT in amperes.
- **i_RMS [float]:** RMS current through the IGBT in amperes.
- **R_CE [float, optional]:** Conduction resistance of the IGBT in ohms (default is 0.0856 Ω).
- **VT [float, optional]:** Threshold voltage of the IGBT in volts (default is 1.198 V).
- **f [float, optional]:** Switching frequency in hertz (default is 20 kHz).
- **V_bus [float, optional]:** DC bus voltage in volts (default is 400 V).
- **V_nom [float, optional]:** Nominal voltage in volts (default is 600 V).
- **R_th [list of 2 floats, optional]:** Simplified thermal resistances $[R_{\text{jc}}, R_{\text{ch}}]$ in °C/W (default is $[1.7, 8]$).
- **R_tch [float, optional]:** Case-to-heatsink thermal resistance in °C/W (default is 60).
- **R_i [list of floats, optional]:** Foster model thermal resistances (default is $[0.29566, 0.25779, 0.19382, 0.05279]$).
- **tau_i [list of floats, optional]:** Foster model time constants in seconds (default is $[6.478 \cdot 10^{-2}, 6.12 \cdot 10^{-3}, 4.679 \cdot 10^{-4}, 6.45 \cdot 10^{-5}]$).
- **t [float, optional]:** Time in seconds for Foster model evaluation (default is 100 s).

4.16 get_switch_temperature_IGBT()

This function computes the junction temperature of an Insulated Gate Bipolar Transistor (IGBT) over time for multiple operating profiles. It applies either a Foster thermal impedance model or a simplified thermal resistance model to each time step in the input current and temperature profiles.
Formula:

$$T_j(t) = (P_{\text{cond}} + P_{\text{sw}}) \cdot R_{\text{th}} + T_{\text{amb}} \tag{4.17}$$

Arguments:

- **T_amb [list]:** Ambient temperature profile in degrees Celsius.
- **i_av_switch_registry [list of lists]:** Time series of average current values through the IGBT in amperes.
- **i_RMS_switch_registry [list of lists]:** Time series of RMS current values through the IGBT in amperes.
- **R_CE [float, optional]:** Conduction resistance of the IGBT in ohms (default is 0.0856 Ω).
- **VT [float, optional]:** Threshold voltage of the IGBT in volts (default is 1.198 V).
- **f [float, optional]:** Switching frequency in hertz (default is 20 kHz).
- **V_bus [float, optional]:** DC bus voltage in volts (default is 400 V).
- **V_nom [float, optional]:** Nominal voltage in volts (default is 600 V).
- **R_th [list of 2 floats, optional]:** Simplified thermal resistances $[R_{\text{jc}}, R_{\text{ch}}]$ in °C/W (default is $[1.7, 8]$).
- **R_tch [float, optional]:** Case-to-heatsink thermal resistance in °C/W (default is 39).
- **R_i [list of floats, optional]:** Foster model thermal resistances (default is $[0.29566, 0.25779, 0.19382, 0.05279]$).
- **tau_i [list of floats, optional]:** Foster model time constants in seconds (default is $[6.478 \cdot 10^{-2}, 6.12 \cdot 10^{-3}, 4.679 \cdot 10^{-4}, 6.45 \cdot 10^{-5}]$).
- **t [float, optional]:** Time in seconds for Foster model evaluation (default is 100 s).

4.17 conduction_loss_MOSFET()

This function calculates the conduction power loss in a Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) based on the RMS current and the drain-to-source resistance. It is used in thermal and efficiency analysis of power electronic systems.
Formula:

$$P_{\text{cond}} = R_{\text{DS}} \cdot i_{\text{RMS}}^2 \tag{4.18}$$

Arguments:

- **i_RMS [float]:** Root mean square (RMS) current through the MOSFET in amperes.
- **R_DS [float, optional]:** Drain-to-source on-state resistance in ohms (default is 0.7 Ω).

4.18 switching_losses_MOSFET()

This function calculates the switching power losses in a Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) based on the RMS current, switching frequency, and voltage conditions. It uses a simplified linear model based on rise and fall times.
Formula:

$$P_{\text{sw}} = \frac{1}{2} \cdot V_{\text{DS}} \cdot i_{\text{RMS}} \cdot f \cdot (t_r + t_f) \tag{4.19}$$

Arguments:

- **i_RMS [float]:** Root mean square (RMS) current through the MOSFET in amperes.
- **V_DS [float, optional]:** Drain-to-source voltage in volts (default is 400 V).
- **f [float, optional]:** Switching frequency in hertz (default is 20 kHz).
- **t_r [float, optional]:** Rise time of the switching event in seconds (default is 40 ns).
- **t_f [float, optional]:** Fall time of the switching event in seconds (default is 35 ns).

4.19 switch_temperature_MOSFET()

This function estimates the junction temperature of a Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) based on its conduction and switching losses, as well as the thermal resistances of the case and heatsink.
Formula:

$$T_j = \left(R_{\text{DS}} \cdot i_{\text{RMS}}^2 + \frac{1}{2} \cdot V_{\text{DS}} \cdot i_{\text{RMS}} \cdot f \cdot (t_r + t_f)\right) \cdot (R_{\text{chc}} + R_{\text{cha}}) + T_{\text{amb}} \tag{4.20}$$

Arguments:

- **T_amb [float]:** Ambient temperature in degrees Celsius.
- **i_RMS [float]:** Root mean square (RMS) current through the MOSFET in amperes.
- **V_DS [float, optional]:** Drain-to-source voltage in volts (default is 400 V).
- **f [float, optional]:** Switching frequency in hertz (default is 20 kHz).
- **R_DS [float, optional]:** Drain-to-source on-state resistance in ohms (default is 0.7 Ω).
- **t_r [float, optional]:** Rise time of the switching event in seconds (default is 40 ns).
- **t_f [float, optional]:** Fall time of the switching event in seconds (default is 35 ns).
- **R_chc [float, optional]:** Case-to-heatsink thermal resistance in °C/W (default is 2.5).
- **R_cha [float, optional]:** Heatsink-to-ambient thermal resistance in °C/W (default is 62.5).

4.20 get_switch_temperature_MOSFET()

This function computes the junction temperature of a Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) over time for multiple operating profiles. It applies a thermal model that includes both conduction and switching losses, scaled by the thermal resistances of the case and heatsink.
Formula:

$$T_j(t) = \left(R_{\mathrm{DS}} \cdot i_{\mathrm{RMS}}^2 + \frac{1}{2} \cdot V_{\mathrm{DS}} \cdot i_{\mathrm{RMS}} \cdot f \cdot (t_r + t_f) \right) \cdot (R_{\mathrm{chc}} + R_{\mathrm{cha}}) + T_{\mathrm{amb}} \tag{4.21}$$

Arguments:

- **T_amb [list]:** Ambient temperature profile in degrees Celsius.
- **i_RMS_switch_registry [list of lists]:** Time series of RMS current values through the MOSFET in amperes.
- **V_DS [float, optional]:** Drain-to-source voltage in volts (default is 400 V).
- **f [float, optional]:** Switching frequency in hertz (default is 20 kHz).
- **R_DS [float, optional]:** Drain-to-source on-state resistance in ohms (default is 0.7 Ω).
- **t_r [float, optional]:** Rise time of the switching event in seconds (default is 40 ns).
- **t_f [float, optional]:** Fall time of the switching event in seconds (default is 35 ns).
- **R_chc [float, optional]:** Case-to-heatsink thermal resistance in °C/W (default is 2.5).
- **R_cha [float, optional]:** Heatsink-to-ambient thermal resistance in °C/W (default is 62.5).

4.21 get_t_on()

This function calculates the on-time duration of a switching device in a DC-DC converter based on the input and bus voltages and the switching frequency. It is used in thermal cycling and lifetime estimation models for power electronics.
Formula:

$$t_{\mathrm{on}} = \frac{V_{\mathrm{bus}} - V_{\mathrm{PV}}}{f \cdot V_{\mathrm{bus}}} \tag{4.22}$$

Arguments:

- **V_PV [float]:** Input voltage from the photovoltaic source in volts.
- **V_bus [float, optional]:** DC bus voltage in volts (default is 400 V).
- **f [float, optional]:** Switching frequency in hertz (default is 20 kHz).

4.22 Bayerer_method()

This function estimates the lifetime of an Insulated Gate Bipolar Transistor (IGBT) using the Bayerer lifetime model. It calculates the number of cycles to failure based on thermal cycling parameters such as temperature swing, minimum temperature, and on-time duration.
Formula:

$$N_f = A \cdot \Delta T^{b_1} \cdot \exp\left(\frac{b_2}{T_{\min}}\right) \cdot t_{\text{on}}^{b_3} \cdot I_b^{b_4} \cdot V_b^{b_5} \cdot D^{b_6} \tag{4.23}$$

Arguments:

- **DT [float]:** Temperature swing (ΔT) in kelvin.
- **T_min [float]:** Minimum junction temperature during the cycle in kelvin.
- **t_on [float]:** On-time duration of the switching cycle in seconds.

Constants:

- $A = 9.34 \cdot 10^{14}$
- $b_1 = -4.416$, $b_2 = 1285$, $b_3 = -0.463$
- $b_4 = -0.716$, $b_5 = -0.761$, $b_6 = -0.5$
- $I_b = 10\,\text{A}$, $V_b = 6\,\text{V}$, $D = 0.45 \cdot 10^{-3}$

4.23 damage()

This function estimates the accumulated damage in an Insulated Gate Bipolar Transistor (IGBT) due to thermal cycling, using the rainflow counting algorithm and the Bayerer lifetime model. It optionally plots histograms of thermal cycle characteristics.
Procedure:

1. Extract thermal cycles from the junction temperature profile using rainflow counting.
2. For each cycle, compute:
   - Temperature swing ΔT
   - Minimum temperature $T_{\min}$
   - On-time duration t_{on}
3. Estimate damage per cycle using the Bayerer model:

$$D = \sum \frac{1}{N_f(\Delta T, T_{\min}, t_{\text{on}})} \tag{4.24}$$

Arguments:

- **T_Switch [list]:** Time series of IGBT junction temperatures in kelvin.
- **t_on_list [list, optional]:** List of on-time durations for each thermal cycle in seconds (default is 0, computed internally).
- **plot_histogram [bool, optional]:** If `True`, plots histograms of ΔT, $T_{\min}$, and t_{on} (default is `True`).
- **dt [float, optional]:** Time step of the temperature profile in seconds (default is 3600 s).

4.24 get_lifetime()

This function estimates the lifetime of an Insulated Gate Bipolar Transistor (IGBT) based on its thermal cycling profile. It uses the damage accumulation calculated from the Bayerer model and optionally plots the damage and relative lifetime as a function of curtailment thresholds.
Procedure:

1. For each temperature profile in `T_Switch_registry`, compute damage using the `damage()` function.
2. If `t_on_registry` is provided, use it to refine the damage estimation.
3. Compute lifetime as the inverse of damage:

$$L = \frac{1}{D} \tag{4.25}$$

4. Optionally, plot damage and normalized lifetime.

Arguments:

- **T_Switch_registry [list of lists]:** Time series of IGBT junction temperatures in kelvin for multiple scenarios.
- **P_Grid_min_list [list, optional]:** List of curtailment thresholds in kilowatts, used for plotting (default is 0).
- **t_on_registry [list of lists, optional]:** On-time durations corresponding to each temperature profile (default is 0).
- **plotting [bool, optional]:** If `True`, plots damage and relative lifetime (default is `True`).
- **return_Lifetime [bool, optional]:** If `True`, returns the computed lifetime values (default is `True`).

4.25 plot_I_T_day()

This function generates plots of the average switch current and junction temperature of the IGBT over a selected day for multiple curtailment thresholds. It helps visualize the thermal and electrical behavior of the switching device under different operating conditions.
Plots:

- Average switch current $\bar{i}_s$ vs. time
- Junction temperature T_j vs. time

Arguments:

- **i_switch_registry [list of lists]:** Time series of average switch currents in amperes for different curtailment thresholds.
- **T_IGBT [list of lists]:** Time series of IGBT junction temperatures in kelvin for different curtailment thresholds.
- **P_Grid_min_list [list]:** List of curtailment thresholds in kilowatts.

- **i_plot [list, optional]:** Indices of the curtailment thresholds to be plotted (default is $[0, 5, 10, 15, 20]$).
- **day [int, optional]:** Day of the year to be plotted (default is 172).
- **dt [float, optional]:** Time step in hours (default is 1).

4.26 BESS_degradation_NMC()

This function estimates the degradation of a lithium-ion battery with Nickel Manganese Cobalt (NMC) chemistry due to cycling and calendar aging. It uses a semi-empirical model based on depth of discharge (DoD), mean state of charge (SoC), temperature, and charge/discharge throughput.
Procedure:

1. Compute depth of discharge (DoD) and mean SoC.
2. Estimate the number of equivalent full cycles.
3. Calculate charge and discharge throughput.
4. Apply the degradation model to compute the remaining capacity.

Arguments:

- **SoC_0 [float]:** Initial state of charge (0 to 1).
- **SoC_f [float]:** Final state of charge (0 to 1).
- **Capacity_BESS [float]:** Current capacity of the battery in kWh.
- **Capacity_BESS_BOL [float, optional]:** Beginning-of-life capacity in kWh (default is 3.36).
- **T [float, optional]:** Battery temperature in kelvin (default is 293 K).
- **dt [float, optional]:** Time step in hours (default is 0.25 h).

4.27 BESS_degradation_LFP()

This function estimates the degradation of a lithium iron phosphate (LFP) battery using one of three empirical models: Olmos, Vermeer, or Wang. It accounts for cycling and calendar aging based on state of charge (SoC), temperature, and usage patterns.
Models:

- **Olmos:** Empirical model based on DoD, mean SoC, and throughput.
- **Vermeer:** Physics-informed model using current, voltage, and temperature.
- **Wang:** Empirical model using temperature, current rate, and Ah throughput.

Arguments:

- **SoC_0 [float]:** Initial state of charge (0 to 1).

- **SoC_f [float]:** Final state of charge (0 to 1).
- **Capacity_BESS [float]:** Current battery capacity in kWh.
- **Capacity_BESS_BOL [float, optional]:** Beginning-of-life capacity in kWh (default is 3.36).
- **T [float, optional]:** Battery temperature in kelvin (default is 293 K).
- **V [float, optional]:** Battery voltage in volts (default is 62.7 V).
- **model [str, optional]:** Degradation model to use: `'Olmos'`, `'Vermeer'`, or `'Wang'` (default is `'Wang'`).
- **dt [float, optional]:** Time step in hours (default is 0.25 h).

4.28 BESS_perm_min()

This function calculates the minimum permissible power that a Battery Energy Storage System (BESS) can deliver or absorb based on its current state of charge (SoC), maximum allowable SoC, and system constraints. It ensures that the BESS does not exceed its operational limits during charging or discharging.
Formula:

$$P_{\text{perm,min}} = \text{clip}\left(\frac{C_{\text{BESS}} \cdot (\text{SoC} - \text{SoC}_{\text{max}})}{\Delta t}, -P_{\text{BESS,max}}, P_{\text{BESS,max}}\right) \tag{4.26}$$

Arguments:

- **SoC [float]:** Current state of charge of the BESS (0 to 1).
- **Capacity_BESS [float, optional]:** Total capacity of the BESS in kWh (default is 3.36).
- **SoCmax [float, optional]:** Maximum allowable state of charge (default is 0.9).
- **P_BESS_max [float, optional]:** Maximum power rating of the BESS in kW (default is 1.28).
- **dt [float, optional]:** Time step in hours (default is 0.25).

4.29 BESS_perm_max()

This function calculates the maximum permissible power that a Battery Energy Storage System (BESS) can deliver or absorb based on its current state of charge (SoC), minimum allowable SoC, and system constraints. It ensures that the BESS does not violate its lower SoC limit during operation.
Formula:

$$P_{\text{perm,max}} = \text{clip}\left(\frac{C_{\text{BESS}} \cdot (\text{SoC} - \text{SoC}_{\text{min}})}{\Delta t}, -P_{\text{BESS,max}}, P_{\text{BESS,max}}\right) \tag{4.27}$$

Arguments:

- **SoC [float]:** Current state of charge of the BESS (0 to 1).
- **Capacity_BESS [float, optional]:** Total capacity of the BESS in kWh (default is 3.36).

- **SoCmin [float, optional]:** Minimum allowable state of charge (default is 0.2).
- **P_BESS_max [float, optional]:** Maximum power rating of the BESS in kW (default is 1.28).
- **dt [float, optional]:** Time step in hours (default is 0.25).

4.30 update_BESS_PeakShaving()

This function simulates the operation of a Battery Energy Storage System (BESS) for peak shaving. It determines the BESS power output, updated state of charge (SoC), and grid power exchange based on load, PV generation, and system constraints.
Logic:

- If SoC is above the maximum, the BESS can only discharge.
- If SoC is between the minimum and maximum, the BESS can charge or discharge.
- If SoC is below the minimum, the BESS can only charge.
- Charging/discharging is limited by power and SoC constraints.

Arguments:

- **SoC_0 [float]:** Initial state of charge (0 to 1).
- **P_Load [float]:** Load power demand in kW.
- **P_PV [float]:** PV power generation in kW.
- **SoCmax [float, optional]:** Maximum allowable SoC (default is 0.9).
- **SoCmin [float, optional]:** Minimum allowable SoC (default is 0.2).
- **P_BESS_max [float, optional]:** Maximum BESS power in kW (default is 1.28).
- **P_Grid_max [float, optional]:** Maximum grid import power in kW (default is 1.5).
- **P_Grid_min [float, optional]:** Minimum grid export power in kW (default is -1.5).
- **Capacity_BESS [float, optional]:** BESS capacity in kWh (default is 3.36).
- **charge_efficiency [float, optional]:** Charging efficiency (default is 0.943).
- **discharge_efficiency [float, optional]:** Discharging efficiency (default is 0.943).
- **P_SD [float, optional]:** Self-discharge rate per time step (default is 0).
- **dt [float, optional]:** Time step in hours (default is 0.25).

4.31 yearly_PeakShaving()

This function simulates the operation of a Battery Energy Storage System (BESS) over one year for peak shaving applications. It updates the BESS state of charge, power exchange with the grid, and degradation over time, based on PV generation and load profiles.
Procedure:

1. Scale PV generation by degradation factor.
2. For each time step:
   - Update BESS state using `update_BESS_PeakShaving()`.
   - Optionally apply degradation model to update capacity.
3. Track energy flows and thermal variables.

Arguments:

- **PV_degradation [function]:** Function that returns a degradation factor for the current year.
- **P_Load [list]:** Load power profile in kW.
- **P_PV_av [list]:** Available PV power profile in kW.
- **P_BESS_0 [float]:** Initial BESS power in kW.
- **SoC_BESS_0 [float]:** Initial state of charge (0 to 1).
- **P_Grid_0 [float]:** Initial grid power in kW.
- **E_BESS_0 [float]:** Initial energy stored in the BESS in kWh.
- **t [list]:** Time vector in hours.
- **SoCmax, SoCmin [float, optional]:** SoC limits (default: 0.9, 0.2).
- **P_BESS_max [float, optional]:** Max BESS power in kW (default: 1.28).
- **P_Grid_max, P_Grid_min [float, optional]:** Grid power limits in kW (default: ± 1.5).
- **Capacity_BESS [float, optional]:** Initial BESS capacity in kWh (default: 3.36).
- **Capacity_BESS_BOL [float, optional]:** Beginning-of-life capacity in kWh (default: 3.36).
- **charge_efficiency, discharge_efficiency [float, optional]:** Charging/discharging efficiencies (default: 0.943).
- **P_SD [float, optional]:** Self-discharge rate per time step (default: 0).
- **dt [float, optional]:** Time step in seconds (default: 900 s).
- **t_final [int, optional]:** Number of time steps (default: $365 \times 24 \times 4$).
- **replace_BESS [bool, optional]:** If `True`, resets capacity when it drops below 80% (default: `False`).

4.32 PeakShaving_Projection()

This function simulates the long-term operation of a Battery Energy Storage System (BESS) for peak shaving over multiple years. It accounts for PV degradation, battery degradation, and optional BESS replacement when capacity falls below a threshold.
Procedure:

1. For each year:
   - Simulate BESS operation using `yearly_PeakShaving()`.
   - Update initial conditions for the next year.
   - Apply degradation or replacement logic.
2. Accumulate yearly results for energy flows, SoC, capacity, and thermal variables.

Arguments:

- **P_Load [list]:** Load power profile in kW.
- **P_PV_av [list]:** Available PV power profile in kW.
- **P_BESS_max [float]:** Maximum BESS power in kW.
- **P_Grid_max [float]:** Maximum grid import power in kW.
- **Capacity_BESS_0 [float]:** Initial BESS capacity in kWh.
- **Capacity_BESS_BOL [float]:** Beginning-of-life BESS capacity in kWh.
- **P_BESS_0 [float, optional]:** Initial BESS power in kW (default is 0).
- **SoC_BESS_0 [float, optional]:** Initial state of charge (default is 0.5).
- **P_Grid_0 [float, optional]:** Initial grid power in kW (default is 0).
- **t [list, optional]:** Initial time vector (default is [0]).
- **replace_BESS [bool, optional]:** If `True`, resets BESS capacity when it drops below 80% (default is `False`).
- **simulation_years [int, optional]:** Number of years to simulate (default is 26).

4.33 SoH_estimation()

This function estimates the State of Health (SoH) of a Battery Energy Storage System (BESS) over time. It computes the SoH as a percentage of the beginning-of-life (BOL) capacity for each time step in the simulation.
Formula:

$$\mathrm{SoH}(t) = 100 \cdot \frac{C_{\mathrm{BESS}}(t)}{C_{\mathrm{BESS}}(0)} \tag{4.28}$$

Arguments:

- **Capacity_BESS [list of lists]:** Time series of BESS capacity values in kWh for each simulation year.

4.34 EoL_BESS()

This function estimates the end-of-life (EoL) of a Battery Energy Storage System (BESS) based on its State of Health (SoH) profile. It calculates the total time during which the battery operates above a specified SoH threshold.
Procedure:

1. Flatten the yearly SoH data into a single time series.
2. Count the number of time steps where SoH exceeds 80%.
3. Convert the count into years using the time step duration.

Formula:

$$\text{EoL}_{\text{BESS}} = \frac{\Delta t}{24 \cdot 365} \cdot \sum_i [\text{SoH}_i > 80] \tag{4.29}$$

Arguments:

- **SoH [list of lists]:** Time series of SoH values in percentage for each simulation year.
- **dt [float, optional]:** Time step in hours (default is 0.25).

4.35 Energy_EoL()

This function calculates the total energy discharged by a Battery Energy Storage System (BESS) up to its estimated end-of-life (EoL). It can return either the accumulated discharged energy or the full power profile up to the EoL point.
Procedure:

1. Determine the number of full years and fractional year from the EoL value.
2. Concatenate the BESS power profiles up to the EoL time.
3. If `accumulated` is `True`, sum only the discharged energy (negative power values).

Arguments:

- **P_BESS [list of lists]:** Yearly BESS power profiles in kW.
- **EoL [float]:** Estimated end-of-life in years.
- **dt [float, optional]:** Time step in hours (default is 0.25).
- **accumulated [bool, optional]:** If `True`, returns total discharged energy in kWh; otherwise, returns the full power profile (default is `True`).

4.36 LCoS_calc()

This function calculates the Levelized Cost of Storage (LCoS) for a Battery Energy Storage System (BESS). The LCoS represents the total cost per unit of discharged energy over the system's lifetime, accounting for capital and operational expenditures.
Formula:

$$\text{LCoS} = \frac{\text{CAPEX} + \text{OPEX}}{E_{\text{discharged}}} \tag{4.30}$$

Arguments:

- **Discharged_energy [float]:** Total discharged energy over the system's lifetime in kWh.
- **CAPEX [float, optional]:** Capital expenditure in euros (default is 6500).
- **OPEX [float, optional]:** Operational expenditure in euros (default is 0).

4.37 plot_BESS_PeakShaving()

This function generates a series of plots to visualize the performance of a Battery Energy Storage System (BESS) over a selected time period. It includes plots for PV generation, BESS state of charge, power flows, C-rate, grid exchange, and system health metrics.
Plots:

- PV power available
- BESS state of charge (SoC)
- BESS power output
- BESS C-rate
- Grid power exchange
- Power imbalance between load and PV
- BESS capacity over time
- Annualized grid energy exchange (net and returned)
- State of Health (SoH) of the BESS

Arguments:

- **t [list]:** Time vector in hours.
- **P_Load [list]:** Load power profile in kW.
- **P_PV_av [list of lists]:** Available PV power profile in kW per year.
- **SoC_BESS [list of lists]:** State of charge of the BESS per year.
- **E_BESS [list of lists]:** Energy stored in the BESS per year.
- **P_BESS [list of lists]:** BESS power output per year.
- **P_Grid [list of lists]:** Grid power exchange per year.
- **Capacity_BESS [list of lists]:** BESS capacity over time per year.
- **SoH [list]:** State of Health of the BESS per year.
- **start_day [int, optional]:** Start day of the year to plot (default is 0).
- **end_day [int, optional]:** End day of the year to plot (default is 365).
- **year [int, optional]:** Year index to plot (default is 0).
- **dt [float, optional]:** Time step in seconds (default is 900 s).

4.38 PeakShaving_limit_violations()

This function analyzes violations of peak shaving thresholds by comparing actual grid power demand against predefined limits. It computes both average and maximum violations and optionally visualizes the results using line and 2D histogram plots.
Procedure:

1. For each threshold:
   - Identify instances where grid power exceeds the threshold.
   - Compute average and maximum violations.
2. Optionally, generate:
   - Line plots of average and maximum violations.
   - 2D histogram of violation magnitude vs. threshold.

Arguments:

- **single_profile_P_Grid_EoL [list of lists]:** Grid power profiles at end-of-life for each threshold.
- **P_Grid_max_list [list]:** List of peak shaving thresholds in kW.
- **single_profile_P_BESS_EoL [list of lists]:** BESS power profiles at end-of-life (used for histogram alignment).
- **plotting [bool, optional]:** If `True`, generates plots (default is `True`).
- **return_violations [bool, optional]:** If `True`, returns average and maximum violations (default is `False`).

4.39 PowerCurtailment()

This function simulates a power curtailment strategy for a photovoltaic (PV) system. It ensures that the power injected into the grid does not exceed a specified minimum or maximum threshold by limiting the PV power output accordingly.
Logic:

- If the available PV power is within the allowable grid injection range, no curtailment is applied.
- If the PV power exceeds the allowable grid injection (i.e., would cause the grid power to fall below the minimum threshold), the PV power is curtailed to maintain grid compliance.

Formula:

$$P_{\text{PV}}^{\text{max}} = P_{\text{Load}} - P_{\text{Grid}}^{\text{min}} \tag{4.31}$$

$$\text{If } P_{\text{PV}}^{\text{av}} \leq P_{\text{PV}}^{\text{max}}: \quad \begin{cases} P_{\text{PV}} = P_{\text{PV}}^{\text{av}} \\ P_{\text{Grid}} = P_{\text{Load}} - P_{\text{PV}} \\ P_{\text{PV}}^{\text{curtailed}} = 0 \end{cases} \tag{4.32}$$

$$\text{Else:} \quad \begin{cases} P_{\text{PV}} = P_{\text{PV}}^{\text{max}} \\ P_{\text{Grid}} = P_{\text{Load}} - P_{\text{PV}} \\ P_{\text{PV}}^{\text{curtailed}} = P_{\text{PV}}^{\text{av}} - P_{\text{PV}}^{\text{max}} \end{cases} \tag{4.33}$$

Arguments:

- **P_Load [float]:** Instantaneous power demand from the load in kilowatts (kW).
- **P_PV_av [float]:** Available PV power in kilowatts (kW).
- **P_Grid_min [float, optional]:** Minimum allowable power injection to the grid in kilowatts (default is -1.5 kW).
- **P_Grid_max [float, optional]:** Maximum allowable power injection to the grid in kilowatts (default is 1.5 kW). This parameter is not used in the current implementation.
- **dt [float, optional]:** Time step in hours (default is 0.25 h). This parameter is not used in the current implementation.

4.40 yearly_Curtailment()

This function simulates a full-year power curtailment scenario for a photovoltaic (PV) system. It applies the curtailment logic at each time step to ensure that the power injected into the grid does not fall below a specified minimum threshold. The function accounts for PV degradation over time.
Workflow:

- The available PV power is degraded according to a degradation factor.
- For each time step, the function applies the `PowerCurtailment()` logic.
- The results are accumulated over the simulation period.

Arguments:

- **PV_degradation [float]:** Degradation factor applied to the available PV power (e.g., 0.98 for 2% degradation).
- **P_Load [list of float]:** Time series of load demand in kilowatts (kW).
- **P_PV_av [list of float]:** Time series of available PV power in kilowatts (kW).
- **P_Grid_0 [float]:** Initial grid power at the start of the simulation.
- **t [list of float]:** Time vector in hours.
- **P_Grid_min [float, optional]:** Minimum allowable power injection to the grid (default is -1.5 kW).
- **dt [float, optional]:** Time step in seconds (default is 900 s, i.e., 15 minutes).
- **t_final [int, optional]:** Total number of time steps in the simulation (default is 365 $\times$ 24 $\times$ 4 = 35040).

4.41 Curtailment_Projection()

This function performs a multi-year simulation of PV power curtailment based on grid injection constraints. It applies the yearly curtailment logic across a specified number of years, accounting for PV degradation over time.
Workflow:

- For each simulation year:
  - The available PV power is degraded using the `PV_degradation()` function.
  - The `yearly_Curtailment()` function is called to compute curtailed PV power and grid injection.
  - Results are stored for each year.
- The available PV power is updated to include curtailed energy for the next year.

Arguments:

- **P_Load [list of float]:** Time series of load demand in kilowatts (kW).
- **P_PV_av [list of float]:** Initial time series of available PV power in kilowatts (kW).
- **P_Grid_0 [float]:** Initial grid power at the start of the simulation.
- **P_Grid_min [float]:** Minimum allowable power injection to the grid in kilowatts (kW).
- **t [list of float, optional]:** Initial time vector in hours (default is `[0]`).
- **simulation_years [int, optional]:** Number of years to simulate (default is 26).

4.42 LCoE_calc()

This function calculates the Levelized Cost of Energy (LCoE) for a photovoltaic (PV) system. The LCoE represents the average cost per kilowatt-hour (kWh) of electricity generated over the system's lifetime, accounting for capital and operational expenditures.
Formula:

$$\text{LCoE} = \frac{\text{CAPEX} + \text{OPEX}}{\text{Discharged\_energy}} \tag{4.34}$$

Arguments:

- **Discharged_energy [float]:** Total energy discharged by the system over its lifetime, in kilowatt-hours (kWh).
- **CAPEX [float, optional]:** Capital expenditure in euros (default is 6500 €).
- **OPEX [float, optional]:** Operational expenditure in euros (default is 0 €).

4.43 replacement_OPEX()

This function estimates the total operational expenditure (OPEX) associated with replacing a system component (e.g., a battery) over a given project lifetime. It accounts for periodic replacements based on the degradation rate and discounts future costs using a specified discount rate.
Workflow:

- Calculates the replacement period based on the degradation rate and expected component life.
- Iteratively adds the discounted cost of each replacement occurring within the project period.

Formula:

$$\text{OPEX} = \sum_{n=1}^{N} \frac{\text{replacement\_cost}}{(1 + \text{discount\_rate})^{\text{replacement\_year}_n}} \tag{4.35}$$

Arguments:

- **degradation [float, optional]:** Degradation factor (default is 1).
- **replacement_cost [float, optional]:** Cost of a single replacement in euros (default is 1250 €).
- **discount_rate [float, optional]:** Discount rate used for present value calculation (default is 0.08).
- **Expected_life [int, optional]:** Expected lifetime of the component in years (default is 15 years).
- **Period [int, optional]:** Total project duration in years (default is 25 years).

4.44 replacement_LCoE()

This function calculates the Levelized Cost of Energy (LCoE) for a photovoltaic (PV) system, including the cost of component replacements over the system's lifetime. It extends the basic LCoE calculation by incorporating discounted replacement costs and inflation-adjusted operational expenses.
Workflow:

- Computes total cost including initial investment, operational costs, maintenance, fuel, and discounted replacement costs.
- Adjusts operational costs for inflation and discounts them to present value.
- Divides total cost by the total energy generated over the project period.

Formula:

$$\text{LCoE} = \frac{\text{Investment} + \text{replacement\_OPEX} + \sum_{y=1}^{\text{Period}} \frac{(\text{Operational}_y + \text{Maintenance}_y + \text{Fuel}_y)\cdot \text{Inflation}_y}{(1+\text{discount\_rate})^y}}{\sum_{y=1}^{\text{Period}} \text{Energy}_y} \tag{4.36}$$

Arguments:

- **degradation [float]:** Degradation factor used to determine replacement frequency.
- **P_PV [list of list of float]:** PV power output per year in kilowatts (kW).
- **Investment [float]:** Initial capital expenditure in euros.
- **Operational [list of float, optional]:** Annual operational costs in euros (default is 0).
- **Maintenance [list of float, optional]:** Annual maintenance costs in euros (default is 0).
- **Fuel [list of float, optional]:** Annual fuel costs in euros (default is 0).
- **Period [int, optional]:** Project duration in years (default is 25).
- **discount_rate [float, optional]:** Discount rate for present value calculation (default is 0.08).
- **inflation_rate [float, optional]:** Annual inflation rate (default is 0.02).
- **replacement_cost [float, optional]:** Cost of each replacement in euros (default is 1250 €).

4.45 plot_LCoE_comparison()

This function generates comparative plots of the Levelized Cost of Energy (LCoE) for a photovoltaic (PV) system under two scenarios: with and without component replacements. It visualizes the impact of system degradation and replacement strategies on the LCoE over a range of curtailment thresholds.
Workflow:

- Computes the LCoE for each scenario using the `LCoE_calc()` function.
- Plots:
  - Base LCoE vs. curtailment threshold.
  - Replacement LCoE vs. curtailment threshold.
  - Relative difference between the two LCoEs.

Arguments:

- **LCoE_list [list of float]:** LCoE values without replacements.
- **Lifetime_norm [list of float]:** Normalized system lifetime values used to estimate degradation.
- **P_PV_registry [list of list of float]:** PV power output per year for each curtailment threshold.
- **P_Grid_min_list [list of float]:** List of curtailment thresholds in kilowatts (kW).
- **Investment [float, optional]:** Initial capital expenditure in euros (default is 4000 €).
- **Operational [list of float, optional]:** Annual operational costs (default is 0).
- **Maintenance [list of float, optional]:** Annual maintenance costs (default is 0).
- **Fuel [list of float, optional]:** Annual fuel costs (default is 0).
- **Period [int, optional]:** Project duration in years (default is 25).

4.46 plot_PowerCurtailment()

This function generates a series of plots to visualize the performance of a photovoltaic (PV) system under a curtailment strategy. It shows how PV power, curtailed energy, grid exchange, and power imbalance evolve over time, along with histograms for demand and imbalance distributions.
Workflow:

- Computes the available PV power as the sum of used and curtailed PV power.
- Plots:
  - Histogram of load demand.
  - Time series of available and curtailed PV power.
  - Grid power exchange.
  - Power imbalance between load and PV.
  - Histogram of power imbalance.
  - Annualized grid energy exchange (net and returned).
  - Annualized curtailed PV energy.

Arguments:

- **t [list of float]:** Time vector in hours.
- **P_Load [list of float]:** Time series of load demand in kilowatts (kW).
- **P_PV [list of list of float]:** PV power used after curtailment per year.
- **P_PV_curtailed [list of list of float]:** Curtailed PV power per year.
- **P_Grid [list of list of float]:** Grid power exchange per year.
- **start_day [int, optional]:** Start day for plotting (default is 0).
- **end_day [int, optional]:** End day for plotting (default is 365).
- **year [int, optional]:** Year index to plot (default is 0).
- **dt [int, optional]:** Time step in seconds (default is 900 s).

4.47 DoD_cycle_counting()

This function analyzes the depth of discharge (DoD) cycles of a battery system using the rainflow counting method. It identifies significant charge-discharge cycles and optionally visualizes their distribution.
Workflow:

- Applies the rainflow algorithm to the state-of-charge (SoC) time series.
- Filters out cycles with a DoD less than 1%.
- Computes the DoD for each valid cycle.

- Optionally plots a histogram of the DoD distribution.

Arguments:

- **SoC [list of float]:** Time series of battery state-of-charge values (normalized between 0 and 1).
- **return_DoD [bool, optional]:** If `True`, returns the list of computed DoD values (default is `False`).
- **plot_histogram [bool, optional]:** If `True`, displays a histogram of DoD values (default is `True`).

4.48 parameters_degradation()

This function extracts cycle parameters from a battery's state-of-charge (SoC) profile using the rainflow counting method. It identifies significant charge-discharge cycles and records their characteristics, which are useful for degradation modeling and lifetime analysis.
Workflow:

- Applies the rainflow algorithm to the SoC time series.
- Filters out cycles with a depth of discharge (DoD) less than 1%.
- Records the following parameters for each valid cycle:
  - Cycle range (amplitude)
  - Mean SoC
  - Cycle count
  - Start and end indices
  - Depth of discharge (DoD)
- Optionally plots the SoC profile with cycle start and end points.

Arguments:

- **SoC [list of float]:** Time series of battery state-of-charge values (normalized between 0 and 1).
- **plot_graphs [bool, optional]:** If `True`, plots the SoC profile with cycle markers (default is `False`).

4.49 P_SoC_2d_histogram()

This function generates a 2D histogram to visualize the joint distribution of battery power and state-of-charge (SoC) for a specific curtailment threshold. It is useful for analyzing the operational behavior of a battery energy storage system (BESS) under different grid constraints.
Workflow:

- Selects the power and SoC profiles corresponding to a specific curtailment threshold.
- Scales SoC values to percentage.

- Plots a 2D histogram using a heatmap to show the frequency of power-SoC combinations.

Arguments:

- **single_profile_P_BESS_EoL [list of list of float]:** BESS power profiles at end-of-life for each curtailment threshold.
- **single_profile_SoC_EoL [list of list of float]:** SoC profiles at end-of-life for each curtailment threshold.
- **P_Grid_max_list [list of float]:** List of curtailment thresholds in kilowatts (kW).
- **threshold [int]:** Index of the curtailment threshold to visualize.
- **P_range [list of float, optional]:** Range of BESS power values for the x-axis (default is [-2.56, 2.56] kW).
- **SoC_range [list of float, optional]:** Range of SoC values for the y-axis in percentage (default is [20, 90]%).
- **vmax_lim [float, optional]:** Maximum value for the heatmap color scale (default is 10).

4.50 Load_histogram()

This function generates a histogram to visualize the distribution of load demand values in a given dataset. It is useful for understanding the frequency and range of power consumption in a system.
Workflow:

- Computes the frequency distribution of the load data.
- Normalizes the histogram to show relative frequencies (percentages).
- Displays the histogram with labeled axes and grid lines.

Arguments:

- **Load [list of float]:** Time series of load demand values in kilowatts (kW).
- **xlabel [str, optional]:** Label for the x-axis (default is `'Demanded power, $P_{L}$, [kW]'`).

4.51 Save_CSV()

This function saves a 2D list (matrix-like structure) to a CSV file. It is useful for exporting simulation results, logs, or any tabular data for further analysis or sharing.
Workflow:

- Opens a file in write mode.
- Writes each sublist of the input variable as a row in the CSV file.

Arguments:

- **variable [list of list]:** The data to be saved. Each sublist represents a row in the CSV file.
- **file_name [str, optional]:** Name of the output CSV file (default is an empty string, which should be replaced by a valid filename).

Chapter 5

Example_MPPT

This example script demonstrates the use of the `MPPT_PO_controller` library to simulate a photovoltaic (PV) system operating under a Perturb and Observe (P&O) Maximum Power Point Tracking (MPPT) algorithm. The simulation includes scenarios with and without power curtailment under varying irradiance conditions. The example file can be found in the repository [5].

Purpose

The script evaluates the dynamic response of a PV system to changes in irradiance and temperature, and how the MPPT controller adapts to maintain optimal power output or enforce curtailment limits.

Usage

To run the example:

1. Ensure the `MPPT_PO_controller` library is available and correctly imported.
2. Configure the following parameters:
   - `f_MPPT` – MPPT switching frequency (e.g., 4 kHz).
   - `dt` – Time step in hours.
   - `G_list`, `T_list` – Lists of irradiance and temperature values.
   - `P_curtailment_list` – List of power curtailment thresholds (or `False` for no curtailment).
3. Execute the script in a Python environment.

Key Variables

- **V_MPPT**, **I_MPPT**, **P_MPPT** – Time series of voltage, current, and power from the MPPT controller.
- **G_ref_plot**, **P_ref_plot** – Reference irradiance and power for plotting.
- **P_curtailment** – Optional power limit applied to the PV output.

Outputs

The script generates the following plots:

- **Voltage vs. Time** – PV module voltage over time.
- **Current vs. Time** – PV module current over time.
- **Power vs. Time** – Comparison of actual PV power and reference/curtailed power.
- **Irradiance vs. Time** – Overlay of irradiance profile for context.

Chapter 6

Example_Peakshaving_Curtailment

This example script demonstrates the use of the `PeakShaving_Curtailment` library to simulate and analyze two energy management strategies: **Peak Shaving** and **Curtailment**. It loads real-world data, configures system parameters, runs multi-year simulations, and visualizes the results through various plots. The example file can be found in the repository [5].

Purpose

The script evaluates the impact of grid constraints on photovoltaic (PV) energy usage, battery energy storage system (BESS) performance, and economic metrics such as Levelized Cost of Energy (LCoE) and Levelized Cost of Storage (LCoS).

Usage

To run the example:

1. Ensure the required CSV files are available:
   - `Tamb_15min.csv` – Ambient temperature data.
   - `P_Load_HP_TESS.csv` – Load demand profile.
   - `Radiation_1min.csv` – Solar irradiance data.
   - `PV_15min.csv` – PV power data.
2. Set the desired simulation mode: `'PeakShaving'` or `'Curtailment'`.
3. Execute the script in a Python environment with required dependencies.

Key Variables

- **P_Load** – Load demand in kW.
- **P_PV_av_0** – Available PV power based on irradiance and module specs.
- **P_Grid_max_list**, **P_Grid_min_list** – Grid injection thresholds for peak shaving and curtailment.
- **simulation_years** – Duration of the simulation (default: 26 years).

- **scale**, **scale_PV**, **scale_BESS** – Scaling factors for system size.
- **Investment**, **Operational**, **Replacement_cost** – Economic parameters.

Outputs

The script generates multiple plots depending on the selected mode:

Peak Shaving Mode

- End-of-life (EoL) of the BESS vs. grid threshold.
- Accumulated energy stored before EoL.
- Energy exchanged with the grid.
- Number of required BESS replacements.
- Levelized Cost of Storage (LCoS).
- Normalized EoL.
- 2D histograms of BESS power vs. SoC.
- Time series plots of BESS operation and grid exchange.

Curtailment Mode

- Total PV energy curtailed vs. grid threshold.
- PV energy used vs. grid threshold.
- LCoE of PV system.
- Energy exchanged with the grid.
- Temperature and current profiles of switching devices.

Bibliography

[1] J. Alpízar-Castillo, "Residential multi-carrier energy storage systems as potential flexibility providers in low-voltage networks: A new player has joined the game," Ph.D. dissertation, Delft University of Technology, 2025. [Online]. Available: `https://repository.tudelft.nl/record/uuid:76075049-7656-4c49-b206-e742a65b6062`.

[2] J. Alpízar-Castillo, "Model predictive control implementation for the ocean grazer wave energy converter with a port-hamiltonian model," M.S. thesis, University of Groningen, Costa Rica Institute of Technology, 2018. [Online]. Available: `https://repositoriotec.tec.ac.cr/handle/2238/10439`.

[3] J. Alpízar-Castillo, *CSV Python Tools*, `https://github.com/jjac13/CSV_Python_Tools`, 2022.

[4] J. Alpízar-Castillo, C. Engström, L. Ramírez-Elizondo, and P. Bauer, "Investigating the effect of power curtailment on the switch of a solar boost converter under residential loads," in *2024 IEEE 21st International Power Electronics and Motion Control Conference (PEMC)*, 2024, pp. 1–6. DOI: `10.1109/PEMC61721.2024.10726347`. [Online]. Available: `https://research.tudelft.nl/en/publications/investigating-the-effect-of-power-curtailment-on-the-switch-of-a-`.

[5] J. Alpízar-Castillo and C. Engstrom, *BESS PEC degradation*, `https://github.com/jjac13/BESS_PEC_degradation`, 2025.

[6] G. Walker, "Evaluating mppt converter topologies using a matlab pv model," *Journal of Electrical and Electronics Engineering, Australia*, vol. 21, Jan. 2001. [Online]. Available: `https://www.researchgate.net/publication/37630747_Evaluating_MPPT_converter_topologies_using_a_Matlab_PV_Model`.