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

GA_EMS.py
Example_GA_EMS.py

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

GA_EMS

This library contains the functions used to simulate an energy management system (EMS) using a genetic algorithm controller, for a residential multi-carrier energy system, 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:

- numpy
- math
- pandas
- collections
- matplotlib
- csv

3.1 heat_loss()

This function calculates the heat transfer (in watts) through a building surface due to a temperature difference between the indoor and outdoor environments. It is based on the standard heat transfer equation for conduction and convection.
Formula:

$$\dot{Q} = U \cdot A \cdot (T_{\text{out}} - T_{\text{in}}) \tag{3.1}$$

Arguments:

- **U [float]:** Overall heat transfer coefficient in $W/m^2 \cdot K$.
- **Area [float]:** Surface area in square meters (m^2).
- **T_in [float]:** Indoor temperature in Kelvin (K).
- **T_out [float]:** Outdoor temperature in Kelvin (K).

3.2 new_house_Temperature()

This function updates the indoor temperature of a house based on the net thermal energy balance over a given time step. It accounts for thermal losses and gains from various sources such as heat pumps (HP), thermal energy storage systems (TESS), and solar collectors (SC).
Formula:

$$T_{\text{new}} = T_0 + \frac{dt}{mc_T} \cdot \left(\dot{Q}_{\text{Losses}} + \dot{Q}_{\text{HP}} + \dot{Q}_{\text{TESS}} + \dot{Q}_{\text{SC}}\right) \tag{3.2}$$

Arguments:

- **T_0 [float]:** Initial indoor temperature in Kelvin (K).
- **Qdot_Losses [float]:** Heat losses from the house in watts (W).
- **mc_T [float]:** Thermal mass of the house (J/K).
- **Qdot_HP [float, optional]:** Heat input from the heat pump (W, default is 0).
- **Qdot_TESS [float, optional]:** Heat input from the thermal energy storage system (W, default is 0).
- **Qdot_SC [float, optional]:** Heat input from the solar collector (W, default is 0).
- **dt [float, optional]:** Time step in seconds (default is 3600 s).

3.3 update_TESS()

This function updates the temperature of a thermal energy storage system (TESS) based on its operational state and thermal interactions with the environment and other components such as solar collectors and heat pumps.
Operational Logic:

- If TESS is active and above its minimum operating temperature, it discharges heat to the network.
- If below minimum temperature, it only charges.
- If above maximum temperature, it only discharges.
- If within operating range, it can both charge and discharge.

Formula:

$$T_{\text{new}} = T_0 + \frac{dt}{m \cdot c} \cdot (Q_{\text{SC}} \cdot \eta + Q_{\text{HP}} \cdot \eta - Q_{\text{SD}} - Q_{\text{TESS}}) \tag{3.3}$$

Arguments:

- **active [bool]:** Whether TESS is active.
- **T_0 [float]:** Initial TESS temperature (K).
- **T_soil [float]:** Soil temperature (K).
- **Qdot_SC [float, optional]:** Heat from solar collector (W, default is 0).
- **Qdot_HP [float, optional]:** Heat from heat pump (W, default is 0).

- **Tmax [float, optional]:** Maximum operating temperature (K, default is 368).
- **Tmin [float, optional]:** Minimum operating temperature (K, default is 323).
- **mdot [float, optional]:** Mass flow rate (kg/s, default is 0.1).
- **T_network [float, optional]:** Network temperature (K, default is 313).
- **Qdot_SD [float, optional]:** Heat loss to soil (W, default is 100).
- **efficiency [float, optional]:** Efficiency factor (default is 0.8).
- **m [float, optional]:** Mass of TESS (kg, default is 4000).
- **c [float, optional]:** Specific heat capacity (J/kg·K, default is 4200).
- **dt [float, optional]:** Time step in seconds (default is 3600).

3.4 Qdot_SolarCollector()

This function calculates the thermal power output (in watts) from a solar collector based on solar irradiance, collector area, and efficiency. It assumes a constant efficiency and linear scaling with irradiance.
Formula:

$$\dot{Q}_{\mathrm{SC}} = \begin{cases} A \cdot \eta_{\mathrm{SC}} \cdot G/dt, & \text{if active} \\ 0, & \text{otherwise} \end{cases} \tag{3.4}$$

Arguments:

- **active [bool]:** Whether the solar collector is active.
- **G [float]:** Solar irradiance in W/m^2.
- **A [float, optional]:** Collector area in m^2 (default is 6).
- **SC_eff [float, optional]:** Collector efficiency (default is 0.45).
- **dt [float, optional]:** Time step in seconds (default is 3600).

3.5 update_BESS()

This function updates the state of charge (SoC) of a battery energy storage system (BESS) based on the power exchanged with the battery, the load, and the PV generation. It accounts for charging and discharging efficiencies and self-discharge losses.
Formula:

$$E_{\mathrm{new}} = \begin{cases} E_0 - \frac{P_{\mathrm{BESS}} \cdot dt}{\eta_{\mathrm{discharge}}}, & \text{if } P_{\mathrm{BESS}} > 0 \\ E_0 - P_{\mathrm{BESS}} \cdot dt \cdot \eta_{\mathrm{charge}}, & \text{if } P_{\mathrm{BESS}} \leq 0 \end{cases} \tag{3.5}$$

$$\mathrm{SoC}_{\mathrm{new}} = \frac{E_{\mathrm{new}} \cdot (1 - P_{\mathrm{SD}})}{\mathrm{Capacity}_{\mathrm{BESS}}} \tag{3.6}$$

Arguments:

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

- **P_BESS [float]:** Power exchanged with the battery (kW). Positive for discharge, negative for charge.
- **P_Load [float]:** Electrical load (kW).
- **P_PV [float, optional]:** PV generation (kW, default is 0).
- **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 battery power (kW, default is 1.28).
- **P_Grid_max [float, optional]:** Maximum grid power (kW, default is 0).
- **Capacity_BESS [float, optional]:** Battery 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).

3.6 BESS_perm_min()

This function calculates the minimum permissible power exchange (in kW) for a battery energy storage system (BESS) based on its current state of charge (SoC). It ensures that the battery does not exceed its maximum allowable state of charge during charging.
Formula:

$$P_{\text{min}} = \text{clip}\left(\frac{\text{Capacity}_{\text{BESS}} \cdot (\text{SoC} - \text{SoC}_{\text{max}})}{dt}, -P_{\text{BESS,max}}, P_{\text{BESS,max}}\right) \tag{3.7}$$

Arguments:

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

3.7 BESS_perm_max()

This function calculates the maximum permissible power exchange (in kW) for a battery energy storage system (BESS) based on its current state of charge (SoC). It ensures that the battery does not fall below its minimum allowable state of charge during discharging.
Formula:

$$P_{\max} = \text{clip}\left(\frac{\text{Capacity}_{\text{BESS}} \cdot (\text{SoC} - \text{SoC}_{\min})}{dt}, -P_{\text{BESS,max}}, P_{\text{BESS,max}} \right) \tag{3.8}$$

Arguments:

- **SoC [float]:** Current state of charge (0–1).
- **Capacity_BESS [float, optional]:** Battery capacity in kWh (default is 3.36).
- **SoCmin [float, optional]:** Minimum allowable SoC (default is 0.2).
- **P_BESS_max [float, optional]:** Maximum battery power (kW, default is 1.28).
- **dt [float, optional]:** Time step in hours (default is 0.25).

3.8 HP_Power()

This function calculates the electrical power input and thermal power output of a heat pump (HP) based on its activation status. It assumes a fixed coefficient of performance (COP) and input power.
Formula:

$$P_{\text{in}} = \text{constant input power (default: 2.7 kW)} \tag{3.9}$$
$$\text{COP} = \text{coefficient of performance (default: 4.1)} \tag{3.10}$$
$$Q_{\text{HP}} = P_{\text{in}} \cdot \text{COP} \tag{3.11}$$

Arguments:

- **active [bool]:** Whether the heat pump is active.
- **P_in [float, optional]:** Electrical input power in watts (default is 2700 W).
- **COP [float, optional]:** Coefficient of performance (default is 4.1).

3.9 House_Thermal_Losses()

This function estimates the total thermal losses (in watts) from a residential building due to transmission through windows, walls, and roof. It also calculates the total thermal capacity of the building envelope and indoor air, which is used to model temperature dynamics.
Returns:

- Total heat loss from:
  - Windows
  - Walls

  - Roof

- Total thermal mass of the building (J/K)

Arguments:

- **T_0_in [float]:** Indoor temperature in Kelvin (K).
- **T_amb [float]:** Outdoor ambient temperature in Kelvin (K).

Notes:

- The function uses fixed geometric and material properties for windows, walls, and roof.
- Convective and conductive heat transfer coefficients are hardcoded based on typical values.
- The building is modeled as a single-zone thermal mass.

3.10 Thermal_Electrical_model()

This function simulates the thermal and electrical behavior of a residential energy system over a single time step. It models the interaction between solar collectors (SC), thermal energy storage systems (TESS), heat pumps (HP), photovoltaic (PV) generation, and battery energy storage systems (BESS).
Arguments:

- **T_amb [float]:** Ambient temperature (K).
- **T_0_in [float]:** Initial indoor temperature (K).
- **T_set [float]:** Indoor temperature setpoint (K).
- **T_soil [float]:** Soil temperature (K).
- **P_PV_av [float]:** Available PV power (kW).
- **P_Load [float]:** Electrical load (kW).
- **G [float]:** Solar irradiance (W/m^2).
- **SC_active [bool]:** Whether the solar collector is active.
- **TESS_active [bool]:** Whether the TESS is active.
- **HP_active [bool]:** Whether the heat pump is active.
- **PV_curtail [float]:** PV curtailment factor (0–1).
- **P_BESS [float]:** Power exchanged with the battery (kW).
- **T_0_TESS [float]:** Initial TESS temperature (K).
- **SoC_0_BESS [float]:** Initial state of charge of the battery (0–1).
- **dt [float, optional]:** Time step in hours (default is 0.25).

3.11 date_time_list()

This function generates a list of timestamps over a specified time horizon, starting from a given initial time. It supports two output formats: a single float representing fractional days, or a list of [day, hour, minute] values. This is useful for time series simulations or scheduling tasks in energy management systems.
Formula:

$$\text{time}_i = \text{start\_date\_time} + i \cdot \frac{dt}{24}, \quad \text{for } i = 0, 1, \ldots, \text{horizon} \tag{3.12}$$

Arguments:

- **horizon [int]:** Number of time steps to generate.
- **start_date_time [float]:** Starting time in fractional days.
- **list_format [str, optional]:** Format of the output list. Options are `'single'` for fractional days or `'multiple'` for [day, hour, minute] format (default is `'multiple'`).
- **dt [float, optional]:** Time step in hours (default is 0.25 hours).

3.12 TS_forecast()

This function generates a forecasted time series based on a given input series and selected forecasting method. It supports both simple randomized linear extrapolation and machine learning-based forecasting using pre-trained Random Forest models.
Forecasting Methods:

- `linear_random`: Applies a small random perturbation to each time step, increasing with time.
- `RandomForests`: Uses a pre-trained Random Forest model to predict future values based on timestamp features.

Arguments:

- **TS [list]:** Input time series to be forecasted.
- **start_date_time [float, optional]:** Starting time in fractional days (default is 0).
- **method [str, optional]:** Forecasting method to use. Options are `'linear_random'` or `'RandomForests'` (default is `'linear_random'`).
- **forecast_type [str, optional]:** Type of forecast, used to select the appropriate model (e.g., `'PV'` for photovoltaic power).
- **list_format [str, optional]:** Format of the output timestamps (default is `'multiple'`).
- **dt [float, optional]:** Time step in hours (default is 0.25 hours).

3.13 chromosome()

This function generates a single chromosome (solution candidate) for a genetic algorithm used in energy management optimization. Each chromosome encodes control decisions over a time horizon and includes the evaluation of its performance using a cost function.
Structure:

- Each chromosome consists of a sequence of control variables for each time step, followed by four cost values: electric cost, thermal comfort cost, CO_2 cost, and total cost.
- Control variables per time step include:
  - Solar Collector activation (binary)
  - TESS activation (binary)
  - Heat Pump activation (binary)
  - PV curtailment (continuous, 0 to 1)
  - BESS power (continuous, bounded)

Arguments:

- **costs [list]:** Cost coefficients for energy, thermal comfort, and CO_2.
- **T_amb, T_0_in, T_set, T_soil, P_PV_av, P_Load, G [list]:** Environmental and system input data over the time horizon.
- **T_0_TESS [float]:** Initial temperature of the thermal energy storage system.
- **SoC_0_BESS [float]:** Initial state of charge of the battery energy storage system.
- **SoC_BESS [list]:** State of charge history (used for bounds).
- **horizon [int]:** Number of time steps in the optimization horizon.
- **beta [float, optional]:** Weighting factor for thermal comfort cost (default is 6).
- **theta_E, theta_T, theta_CO2 [float, optional]:** Normalization factors for energy, thermal, and CO_2 costs.

3.14 initial_population()

This function initializes a population of chromosomes for a genetic algorithm. Each chromosome represents a candidate solution for optimizing energy and thermal comfort in a smart home environment. The function also tracks population diversity.
Process:

- Generates a specified number of chromosomes using the `chromosome()` function.
- Each chromosome encodes control decisions and includes cost evaluations.
- Diversity is measured by counting unique chromosome strings.

Arguments:

- **costs [list]:** Cost coefficients for energy, thermal comfort, and CO_2.

- **T_amb, T_0_in, T_set, T_soil, P_PV_av, P_Load, G [list]:** Environmental and system input data over the time horizon.
- **T_0_TESS [float]:** Initial temperature of the thermal energy storage system.
- **SoC_0_BESS [float]:** Initial state of charge of the battery energy storage system.
- **SoC_BESS [list]:** State of charge history (used for bounds).
- **horizon [int]:** Number of time steps in the optimization horizon.
- **individuals [int]:** Number of chromosomes to generate.
- **beta [float, optional]:** Weighting factor for thermal comfort cost (default is 6).
- **theta_E, theta_T, theta_CO2 [float, optional]:** Normalization factors for energy, thermal, and CO_2 costs.

3.15 create_new_population()

This function generates a new population of chromosomes for the next generation in a genetic algorithm. It uses elitism and random selection to form a mating pool, performs crossover to create offspring, and selects survivors for the next generation.
Process:

- Selects elite individuals based on fitness (lowest cost).
- Randomly selects non-elite individuals to maintain diversity.
- Forms a mating pool and shuffles it.
- Generates a new generation using crossover.
- Combines old and new populations and selects survivors randomly.

Arguments:

- **population [array]:** Current population of chromosomes.
- **costs [list]:** Cost coefficients for energy, thermal comfort, and CO_2.
- **T_amb, T_0_in, T_set, T_soil, P_PV_av, P_Load, G [list]:** Environmental and system input data over the time horizon.
- **T_0_TESS [float]:** Initial temperature of the thermal energy storage system.
- **SoC_0_BESS [float]:** Initial state of charge of the battery energy storage system.
- **elite_rate [float, optional]:** Fraction of top-performing individuals to retain (default is 0.05).
- **non_elite_rate [float, optional]:** Fraction of randomly selected non-elite individuals (default is 0.15).
- **beta [float, optional]:** Weighting factor for thermal comfort cost (default is 6).
- **theta_E, theta_T, theta_CO2 [float, optional]:** Normalization factors for energy, thermal, and CO_2 costs.

3.16 create_new_generation()

This function creates a new generation of chromosomes by performing crossover operations on a mating population. It supports both fixed and random gene crossover strategies and evaluates the resulting offspring using a cost function.
Process:

- Selects pairs of chromosomes from the mating population.
- Performs crossover by exchanging selected genes between pairs.
- Evaluates each new chromosome using the cost function.
- Returns a reshaped array of the new generation.

Arguments:

- **mating_population [array]:** Array of chromosomes selected for reproduction.
- **costs [list]:** Cost coefficients for energy, thermal comfort, and CO_2.
- **T_amb, T_0_in, T_set, T_soil, P_PV_av, P_Load, G [list]:** Environmental and system input data over the time horizon.
- **T_0_TESS [float]:** Initial temperature of the thermal energy storage system.
- **SoC_0_BESS [float]:** Initial state of charge of the battery energy storage system.
- **rand_gene_number [bool, optional]:** If `True`, selects a random number of genes for crossover; otherwise, uses half of the genes (default is `False`).
- **beta [float, optional]:** Weighting factor for thermal comfort cost (default is 6).
- **theta_E, theta_T, theta_CO2 [float, optional]:** Normalization factors for energy, thermal, and CO_2 costs.
- **horizon [int, optional]:** Number of time steps in the optimization horizon (default is 0).

3.17 best_individual()

This function identifies the best-performing chromosome from a given population based on the total cost value. It is used in the selection phase of the genetic algorithm to track the most optimal solution found so far.
Process:

- Evaluates the last element of each chromosome, which represents the total cost.
- Returns the chromosome with the minimum total cost.

Arguments:

- **population [array]:** A NumPy array where each row is a chromosome and the last column contains the total cost.

3.18 cost_function()

This function evaluates the performance of a chromosome by computing its associated costs over a given time horizon. It calculates the electric cost, thermal comfort cost, and CO_2 emissions cost, and combines them into a total cost using a Euclidean norm.
Cost Components:

- **Electric Cost:** Based on energy prices and power consumption.
- **Thermal Comfort Cost:** Penalizes deviation from the desired indoor temperature.
- **$\mathbf{CO_2}$ Cost:** Based on emissions associated with power usage.
- **Total Cost:** Computed as:

$$\text{Total Cost} = \sqrt{(\text{Electric Cost})^2 + (\text{Thermal Cost})^2 + (\text{CO}_2\ \text{Cost})^2} \tag{3.13}$$

Arguments:

- **chromosome [array]:** Control decisions for each time step.
- **costs [list]:** Cost coefficients for energy, thermal comfort, and CO_2.
- **T_amb, T_0_in, T_set, T_soil, P_PV_av, P_Load, G [list]:** Environmental and system input data over the time horizon.
- **T_0_TESS [float]:** Initial temperature of the thermal energy storage system.
- **SoC_0_BESS [float]:** Initial state of charge of the battery energy storage system.
- **horizon [int]:** Number of time steps in the optimization horizon.
- **beta [float, optional]:** Weighting factor for thermal comfort cost (default is 6).
- **theta_E, theta_T, theta_CO2 [float, optional]:** Normalization factors for energy, thermal, and CO_2 costs.
- **dt [float, optional]:** Time step in seconds (default is 900 seconds).

3.19 GA_Optimization()

This function runs the full genetic algorithm (GA) optimization process for a smart energy management system. It iteratively evolves a population of chromosomes to minimize a multi-objective cost function that includes energy cost, thermal comfort, and CO_2 emissions.
Process:

- Initializes the population (or resumes from the last saved one).
- Evaluates the best candidate in the current population.
- Iteratively generates new populations using selection, crossover, and mutation.
- Tracks the best candidate and stops when no improvement is found over a number of generations.

Arguments:

- **T_amb, T_0_in, T_set, T_soil, P_PV_av, P_Load, G [list]:** Environmental and system input data over the time horizon.
- **T_0_TESS [float]:** Initial temperature of the thermal energy storage system.
- **SoC_0_BESS [float]:** Initial state of charge of the battery energy storage system.
- **SoC_BESS [list]:** State of charge history (used for bounds).
- **horizon [int]:** Number of time steps in the optimization horizon.
- **costs [list, optional]:** Cost coefficients for energy, thermal comfort, and CO_2 (default is [0, 0, 0, 0, 0.13, 0.483]).
- **consecutive_generations [int, optional]:** Number of generations without improvement before stopping (default is 5).
- **individuals [int, optional]:** Number of chromosomes in the population (default is 200).
- **beta [float, optional]:** Weighting factor for thermal comfort cost (default is 6).
- **theta_E, theta_T, theta_CO2 [float, optional]:** Normalization factors for energy, thermal, and CO_2 costs.

3.20 week_graph_TQP()

This function generates a set of time series plots to visualize thermal and electrical performance over a one-week period. It is used to analyze the behavior of indoor temperature, thermal power flows, and energy prices in a smart home energy management system.
Plots:

- **Temperature Plot 1:** Ambient temperature.
- **Temperature Plot 2:** Setpoint and indoor temperature.
- **Thermal Power Plot:** Thermal power from load, solar collector, TESS, and heat pump.
- **Energy Price Plot:** Grid energy price over time.

Arguments:

- **t [list]:** Time vector (not directly used in plotting).
- **T_in [list]:** Indoor temperature time series.
- **T_amb [list]:** Ambient temperature time series.
- **T_set [list]:** Setpoint temperature time series.
- **Qdot_TESS, Qdot_SC, Qdot_HP, Qdot_Load [list]:** Thermal power flows in watts.
- **P_Grid, P_Load, P_HP, P_PV, P_BESS [list]:** Electrical power flows in kilowatts.
- **labels [list]:** X-axis labels for days of the week.
- **Energy_price [list]:** Energy price from the grid in €/kWh.
- **dt [float, optional]:** Time step in hours (default is 0.25).
- **days [int, optional]:** Number of days to display (default is 7).

3.21 get_Pareto()

This function visualizes the Pareto front of a population registry from a genetic algorithm. It plots the trade-off between energy cost and thermal comfort cost, and optionally identifies the source configuration of each individual.
Process:

- Extracts cost components (energy, thermal comfort, CO_2, total) from all individuals in the population registry.
- Plots energy cost vs. thermal comfort cost.
- Optionally categorizes individuals based on their heating configuration:
  - No heating
  - Only HP (Heat Pump)
  - Only TESS (Thermal Energy Storage System)
  - HP + TESS
  - Only SC (Solar Collector)
  - SC + HP
  - SC + TESS
  - SC + TESS + HP

Arguments:

- **population_registry [list]:** List of populations across generations, each containing chromosomes with cost evaluations.
- **identify_season [bool, optional]:** Reserved for future use (default is `True`).
- **indentify_source [bool, optional]:** If `True`, categorizes individuals by heating source configuration and plots them separately (default is `False`).

3.22 Save_CSV()

This function saves a given variable (typically a list of lists or a 2D array) to a CSV file. It is used to export simulation results, optimization outputs, or any tabular data for further analysis or record-keeping.
Arguments:

- **variable [list]:** The data to be saved. It should be a list of rows, where each row is a list of values.
- **file_name [str, optional]:** Name of the output CSV file (default is an empty string, which may result in an error if not specified).

Chapter 4

Example_GA_EMS

This example script demonstrates the use of the `GA_EMS` library to simulate a residential energy management system (EMS) controlled by a genetic algorithm (GA). The EMS coordinates multiple energy assets—such as photovoltaic (PV) systems, battery energy storage systems (BESS), heat pumps (HP), thermal energy storage systems (TESS), and solar collectors (SC)—to minimize energy costs, CO_2 emissions, and thermal discomfort. The example file can be found in the repository [5].

Purpose

The script evaluates the performance of a GA-based EMS over a specified time horizon, optimizing the operation of thermal and electrical components in a residential setting. It supports forecasting, cost modeling, and multi-objective optimization.

Usage

To run the example:

1. Ensure the following input files are available:
   - `PV_15min.csv` – PV generation data.
   - `Tamb_15min.csv` – Ambient temperature data.
   - `Load_Profile_15min.csv` – Electrical load profile.
   - `Radiation_1min.csv` – Solar irradiance data.
   - `DA_Prices_15min.csv` – Day-ahead electricity prices.
2. Configure simulation parameters such as:
   - `start_day`, `end_day` – Simulation window.
   - `horizon` – Forecast horizon in time steps.
   - `individuals`, `consecutive_generations` – GA parameters.
   - `optimization_weights` – Weights for energy cost, thermal comfort, and CO_2 emissions.
3. Execute the script in a Python environment.

Key Variables

- **P_Load**, **P_PV_av** – Load and available PV power.
- **T_amb**, **T_in**, **T_TESS** – Ambient, indoor, and TESS temperatures.
- **P_BESS**, **SoC_BESS** – BESS power and state-of-charge.
- **Qdot_HP**, **Qdot_TESS**, **Qdot_SC** – Thermal power from HP, TESS, and SC.
- **Energy_price** – Electricity price time series.

Outputs

The script produces:

- Time series plots of:
  - Indoor and ambient temperatures.
  - Thermal power flows (HP, TESS, SC).
  - Electrical power flows (PV, BESS, grid).
  - State-of-charge of the BESS.
  - Energy prices.
- Boxplots of computation time per time step.
- CSV export of GA results (optional).

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, A. Fu, L. Ramírez-Elizondo, M. Cvetkovic, and P. Bauer, "Multi-carrier energy home energy management system using genetic algorithms and random forest predictions," in *2024 IEEE Energy Conversion Congress and Exposition (ECCE)*, 2024, pp. 1037–1044. DOI: 10.1109/ECCE55643.2024.10861342. [Online]. Available: https://research.tudelft.nl/en/publications/multi-carrier-energy-home-energy-management-system-using-genetic-.

[5] J. Alpízar-Castillo, *GA HEMS Predictive*, https://github.com/jjac13/GA_HEMS_Predictive, 2024.