Manual for Mixed-Integer Programming (MIP) Model for Energy System Analysis

Arkadii Kolchin and Sami Tarvainen

31 August 2023

Contents

1 Introduction

The rapid transition to sustainable energy systems has brought hydrogen production to the forefront of research, policy-making, and industrial applications. The complexity and multi-dimensional nature of hydrogen production necessitate an advanced analytical tool that can encompass various technologies, scenarios, and locations. This manual is dedicated to introducing the Mixed-Integer Programming (MIP) model designed to analyze, optimize, and visualize hydrogen production systems.

The model incorporates a wide range of features, allowing for flexibility in scenario management, expandability to different technologies, advanced optimization techniques, and comprehensive visual reporting. This manual offers step-by-step guidance to utilize the full potential of the model, from basic applications to advanced research functionalities.

2 Overview of the Model

2.1 Purpose and Scope

The Mixed-Integer Programming (MIP) model for Energy System Analysis, focusing primarily on hydrogen and thermal production, aims to provide a comprehensive and scalable framework for analyzing various energy scenarios. Besides hydrogen and thermal production, the model is also adaptable for implementing other resources, thereby offering a flexible tool for various users.

2.2 Key Features and Functions

Flexible Scenario Management

- **Support for Various Technologies**: Accommodates different technologies for power generation, hydrogen production, and storage.
- **Adaptation to Various Geographical Locations**: Customizable for different locations, considering weather patterns, regulations, and local demand.
- **Time-Scaling**: Can be run for different time periods, from hourly to yearly, depending on the scenario.

Expandable Structure

- **Modular Design**: Easy addition or modification of technologies due to module-based components.
- **Integration with Other Energy Systems**: Could be part of a larger energy system analysis.

Advanced Optimization Features

- **Mixed-Integer Programming (MIP) Framework**: Enables modeling of discrete decisions alongside continuous variables.
- **Multi-Objective Optimization**: Optional function for simultaneous optimization of multiple objectives.

Visual Results and Reporting Tools

- **Modular Visualizer**: Input parameters and visualize results through graphs, charts, and maps for the requirements of specific scenarios.
- **Interactive Dashboard**: User-friendly interface to design and run scenarios, based on Jupyter Notebooks.
- **Exportable Reports**: Generate detailed reports in various formats (PDF, Excel, etc.).

Educational and Research Orientations

- **Tutorials, Guides, and Task Templates**: Comprehensive resources for students and researchers.
- **Customizable for Research Purposes**: Tailorable for specific research needs and requirements.

3 Getting started

3.1 Installation and setup

- Install the required library for linear programming using: `pip install mip`
- Download the energy model source code to a directory of your choosing. The source code is available at https://version.aalto.fi/gitlab/ehirvijo/aesm. Ensure that the required modules like `powerPlants`, `electrolyzer`, etc., are properly imported.

3.2 Running Modelling Examples

We have added a few modelling examples to help you get started. Short instructions for the modelling examples are available in the READ.me file and expanded instructions are available in the Jupyter Notebook.

Here are the steps how you can access the Jupyter Notebook instructions:

- Launch Jupyter Notebook. This can be easily done from Anaconda Promt by entering command Jupyter Notebook. You might need to install Anaconda if this is not already installed. Note that Jupyter is automatically installed with Anaconda. There are good installation and setup instructions also available on the internet, for instance https://www.youtube.com/watch?v=HW29067qVWk&t=1620s.
- Launch file UI.ipynb with Jupyter Notebook and follow the instructions there.
- Ensure that your input files are properly formatted and located in the correct directories.

4 Program Components

4.1 File Structure Overview

- `model.py` The `model.py` file houses the core optimization model for the electricity, heat, and hydrogen production system. It utilizes various technology classes to construct and solve the optimization problem. The primary class within this module is the `ModelRunner`, which is responsible for setting up the optimization problem, adding constraints, defining the objective function, and running the optimization. It establishes the demand-supply balance, integrates different technologies, and solves the model to minimize costs.

- `technologies` Package

  The `technologies` package comprises multiple modules representing different types of technologies involved in the system. Each module contains one or multiple technology classes that define specific attributes, variables, constraints, and objectives for a given technology. There are modules for generation of electricity, hydrogen and thermal energy, as well as the means for storing them.

- `main.py`: This module integrates all components of the modeling system. It processes user input, runs specific energy scenarios using the `ModelRunner` class, and visualizes the results. After computing results, it provides both a console report and an exported file containing the results of the scenario analysis. The module is driven by YAML-based scenario definitions.

- `scenario.py`: This module handles demand definitions and scenario construction. It encompasses a class for electricity, hydrogen, and heat demands, and a class to define scenarios based on these demands and specific technologies.

- `user_input.py`: This is an older module dedicated to creating and defining scenarios, which are configured through by hardcoding methods that entail particular demand profiles and technological configurations. Instead of this legacy module using YAML scenario formatting is recommended, nonetheless the module the module might still be useful.

- `yamled.py` the module that runs improved method of YAML-based definition and extraction of scenarios. With the integration of this module, scenarios can be easily defined in an external `.yaml` file, facilitating a more user-friendly interface for large and intricate simulations.

- `visualizer.py` This module plots the model results (see Subsection 7.1).

- `resultExport.py` This module exports the model results in CSV format to the folder `results` (see Subsection 7.2).

- `data` Folder for the input data. The data files used in the example cases are already included in the folder.
- `results` Model results are exported in this folder.

4.2 How program logic works?

The linear optimization logic is defined in `model.py` and `Technologies` package. This is what happens between input of the scenario and output of information. The program uses python library mip for Mixed-Integer Linear program (MIPs), if the concept is not familiar it is recommended to introduce yourself to it by watching several videos.

1. **Model Initialization in `model.py`:**
   (a) *Technologies List Reception:* The model starts by receiving a list of technologies. This list can contain objects from various classes: `Technology`, `PowerPlant`, `Storage`, etc.
   (b) *Sorting Technologies:* The model sorts these technologies into separate variables based on their class. This helps in ensuring that each technology is handled according to its specific characteristics and constraints.
2. **Setting Up Variables and Constraints (`setupVarsConstr` function):**
   (a) *Technology Initialization (`add_to_model` function):* The `add_to_model` method is invoked for each technology in the list. This method works recursively:
      i. It starts by calling the `add_to_model` method of the base class, which is `Technology`. This initializes basic attributes such as generation capacity (`C_G`) and calculates the fundamental costs.
      ii. If the technology is an instance of a child class (like `PowerPlant` or `WindPower`), the method calls the `add_to_model` function of the respective child class. This process continues down the inheritance chain until the most specific class's method is executed. Along this chain, each class adds its own specific variables and constraints.
   (b) *Meeting Demand Constraints:* Once all technologies have been added to the model with their respective variables and constraints, the model sets up additional constraints to ensure that the production levels meet the demand for every timeframe. This can be achieved through:
      i. *Direct Generation:* Using technologies like `PowerPlant`.
      ii. *Charging/Discharging Storage:* Leveraging classes like `Storage` to use stored energy.

3. **Setting Objective (`addObjective` function):**

   (a) The model sums up the costs of all technologies. Each technology's cost has been computed and stored in its `Cost` attribute during the `add_to_model` process.

   (b) The primary objective of the model becomes minimizing this total cost, capturing both the operational and investment costs associated with all technologies.

4.2.1 Setting Up Variables and Constraints

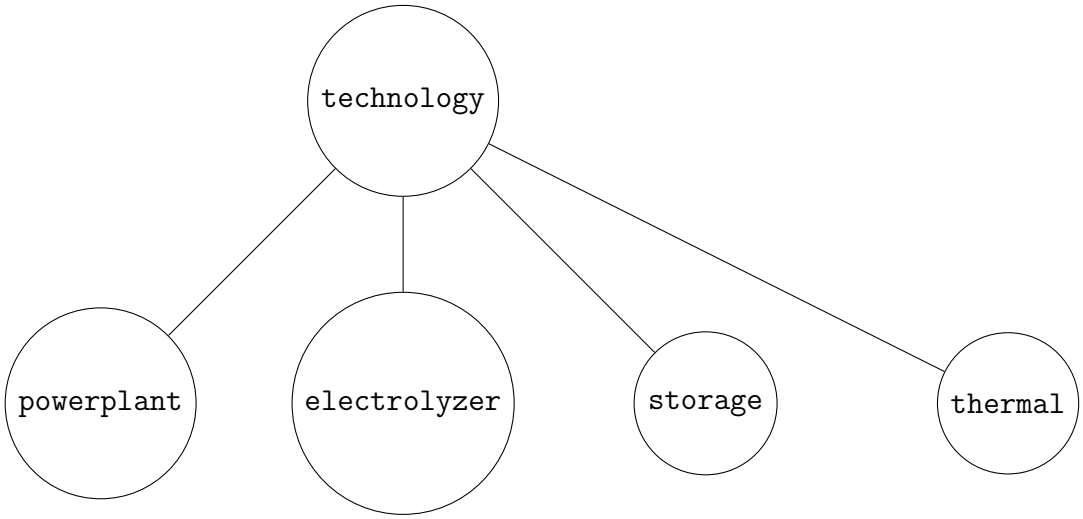
The `setupVarsConstr` method sets up variables and constraints for different technologies and demand-supply balance.

4.2.2 Objective Function

The `addObjective` method defines the objective function, which aims to minimize the sum of expenses across all technologies.

5 Technologies and definitions

This chapter includes description of the linear model, including objective function, variables, parameters, and constraints. The model allows for the modelling of electricity, hydrogen, and heat production. Used technologies are divided to four different categories: electricity, heat and hydrogen production as well as storage. It should be noted that the division is not perfect, as e.g. electrolyzers allow for heat production. The inbuilt technologies are listed below.



Electricity production

- Solar
- Wind
- Nuclear
- Hydro reservoir
- Gas turbine

Heat production

- Heat pumps

Hydrogen production

- Electrolyzer

Storage

- Hydrogen storage
- Electric battery
- Thermal energy storage (e.g. water tank)

5.1 Objective function

The model minimizes the total cost of technology mix:

$$min \sum_i Cost_i \tag{1}$$

where:

- $Cost_i$: Capital, fixed and variable costs of technology i.

5.2 Electricity production

5.2.1 Solar PV

Parameters:

- $CAPEX_{G,i}$: Capital expenditures per installed generation capacity for solar technology i [$e.g.\ EUR/MW_e$]
- R_i: Expected capital return of solar technology i [decimal number]
- $FC_{G,i}$: Fixed costs per installed generation capacity for solar technology i [$e.g.\ EUR/MW_e$]
- $VC_{G,i}$: Variable costs per unit of electricity for solar technology i [$e.g.\ EUR/MWh_e$]
- AF_{it}: Availability factor of solar technology i at time t [$\in [0,1]$]
- $C_{Gmin,i}$: Minimum required capacity of solar technology i [$e.g.\ MW_e$]
- $C_{Gmax,i}$: Maximum capacity potential of solar technology i [$e.g.\ MW_e$]

Variables:

- $C_{G,i}$: Installed generation capacity of solar technology i [$e.g.\ MW_e$]
- $G_{elec,it}$: Generation of electricity with solar technology i at time t [$e.g.\ MWh_e$]
- $Curtail_{it}$: Curtailment of solar technology i at time t [$e.g.\ MWh_e$]

Objective:

$$Cost_i = (CAPEX_{G,i}R_i + FC_{G,i})C_{G,i} + \sum_t VC_{G,i}G_{elec,it} \tag{2}$$

Constraints:
The sum of electricity generation and curtailment is limited by the installed capacity and availability factor.

$$G_{elec,it} + Curtail_{it} = AF_{it}C_{G,i} \tag{3}$$

The installed capacity is limited by the maximum capacity potential and the minimum required capacity.

$$C_{G,i} \geq C_{Gmin,i} \tag{4}$$

$$C_{G,i} \leq C_{Gmax,i} \tag{5}$$

5.2.2 Wind

Formulation of the wind power is equivalent to solar power. Both onshore and offshore wind power plants can be modeled with this framework.

Parameters:

- $CAPEX_{G,i}$: Capital expenditures per installed generation capacity for wind technology i [$e.g.\ EUR/MW_e$]
- R_i: Expected capital return of wind technology i [decimal number]
- $FC_{G,i}$: Fixed costs per installed generation capacity for wind technology i [$e.g.\ EUR/MW_e$]
- $VC_{G,i}$: Variable costs per unit of electricity for wind technology i [$e.g.\ EUR/MWh_e$]
- AF_{it}: Availability factor of wind technology i at time t [$\in [0,1]$]
- $C_{Gmin,i}$: Minimum required capacity of wind technology i [$e.g.\ MW_e$]
- $C_{Gmax,i}$: Maximum capacity potential of wind technology i [$e.g.\ MW_e$]

Variables:

- $C_{G,i}$: Installed generation capacity of wind technology i [$e.g.\ MW_e$]
- $G_{elec,it}$: Generation of electricity with wind technology i at time t [$e.g.\ MWh_e$]
- $Curtail_{it}$: Curtailment of wind technology i at time t [$e.g.\ MWh_e$]

Objective:

$$Cost_i = (CAPEX_{G,i}R_i + FC_{G,i})C_{G,i} + \sum_t VC_{G,i}G_{elec,it} \tag{6}$$

Constraints:
The sum of electricity generation and curtailment is limited by the installed capacity and availability factor.

$$G_{elec,it} + Curtail_{it} = AF_{it}C_{G,i} \tag{7}$$

The installed capacity is limited by the maximum capacity potential and the minimum required capacity.

$$C_{G,i} \geq C_{Gmin,i} \tag{8}$$
$$C_{G,i} \leq C_{Gmax,i} \tag{9}$$

5.2.3 Nuclear

Parameters:

- $CAPEX_{G,i}$: Capital expenditures per installed generation capacity for nuclear technology i [*e.g.* EUR/MW_e]
- R_i: Expected capital return of nuclear technology i [decimal number]
- $FC_{G,i}$: Fixed costs per installed generation capacity for nuclear technology i [*e.g.* EUR/MW_e]
- $VC_{G,i}$: Variable costs per unit of electricity for nuclear technology i, like fuel. [*e.g.* EUR/MWh_e]
- $RampU_i$: Maximum ramp-up of electricity generation per installed capacity $C_{G,i}$ [$\in (0, 1]$]
- $RampD_i$: Maximum ramp-down of electricity generation per installed capacity $C_{G,i}$ [$\in (0, 1]$]
- $AF_{max,i}$: Maximum availability factor for nuclear technology i. For example, Olkiluoto 1 and 2 have maintenance break of three weeks, which implies average maximum availability factor of 49 / 52 = 0.94. [$\in (0, 1]$]
- $AF_{min,i}$: Minimum availability factor for nuclear technology i [$\in [0, 1]$]

Variables:

- $C_{G,i}$: Installed generation capacity of nuclear technology i [*e.g.* MW_e]
- $G_{elec,it}$: Generation of electricity with nuclear technology i at time t [*e.g.* MWh_e]

Objective:

$$Cost_i = (CAPEX_{G,i}R_i + FC_{G,i})C_{G,i} + \sum_t VC_{G,i}G_{elec,it} \tag{10}$$

Constraints:
Electricity generation is limited by the minimum and maximum availability factors.

$$G_{elec,it} \leq AF_{max,i}C_{G,i} \tag{11}$$
$$G_{elec,it} \geq AF_{min,i}C_{G,i} \tag{12}$$

Change in electricity production cannot exceed the maximum ramp-up / ramp-down time.

$$G_{elec,it} - G_{elec,it-1} \leq C_{G,i}RampU_i \tag{13}$$
$$G_{elec,it-1} - G_{elec,it} \leq C_{G,i}RampD_i \tag{14}$$

5.2.4 Hydro Reservoir

Parameters:

- $CAPEX_{G,i}$: Capital expenditures per installed generation capacity for technology i [$e.g.\ EUR/MW_e$]
- $CAPEX_{S,i}$: Capital expenditure per installed hydro reservoir storage capacity for technology i [$e.g.\ EUR/MWh_e$]
- R_i: Expected capital return of hydro technology i [decimal number]
- $FC_{G,i}$: Fixed costs per installed generation capacity for technology i [$e.g.\ EUR/MW_e$]
- $FC_{S,i}$: Fixed costs per per installed reservoir storage capacity for technology i [$e.g.\ EUR/MW_e$]
- $Inflow_{it}$: Inflow of energy to the reservoir i at time t [$e.g.\ MWh_e$]
- $Outflow_{min,it}$: Minimum outflow of energy from the reservoir i at time t [$e.g.\ MWh_e$]
- $C_{GMax,i}$: Maximum generation capacity for technology i [$e.g.\ MW_e$]
- $C_{SMax,i}$: Maximum reservoir storage capacity for technology i [$e.g.\ MWh_e$]

Variables:

- $C_{G,i}$: Installed generation capacity for technology i [$e.g.\ MW_e$]
- $C_{S,i}$: Installed reservoir storage capacity for technology i [$e.g.\ MWh_e$]
- $G_{elec,it}$: Generation of electricity with hydro technology i at time t [$e.g.\ MW_e$]
- S_{it}: Hydro reservoir storage level for technology i at time t [$e.g.\ MWh_e$]
- $Outflow_{it}$: Outflow of energy from the reservoir i at time t [$e.g.\ MW_e$]
- $Bypass_{it}$: Energy flow bypassed by the generator i at time t [$e.g.\ MW_e$]

Objective:

$$Cost_i = (CAPEX_{G,i}R_i + FC_{G,i})C_{G,i} + (CAPEX_{S,i}R_i + FC_{S,i})C_{S,i} \quad (15)$$

Constraints:
Electricity generation is limited by the installed capacity.

$$G_{elec,it} \leq C_{G,i} \quad (16)$$

Reservoir storage level is determined by the energy inflow and outflow and limited by the installed reservoir storage capacity.

$$S_{it+1} = S_{it} + Inflow_{it} - Outflow_{it} \tag{17}$$
$$S_{it} \leq C_{S,i} \tag{18}$$

The installed generation capacity or the reservoir storage capacity cannot exceed the maximum capacity.

$$C_{G,i} \leq C_{GMax,i} \tag{19}$$
$$C_{S,i} \leq C_{SMax,i} \tag{20}$$

The outflow equals the electricity generation and energy bypass. The outflow must exceed the minimum outflow.

$$Outflow_{it} = G_{elec,it} + Bypass_{it} \tag{21}$$
$$Outflow_{it} \geq Outflow_{min,it} \tag{22}$$

Initial and final reservoir storage levels are set to be equal to make the storage cyclical.

$$S_{it=1} = S_{it=last\ time\ step}. \tag{23}$$

5.2.5 Gas Turbine

Parameters:

- $CAPEX_{G,i}$: Capital expenditures per installed generation capacity for technology i [$e.g.\ EUR/MW_e$]
- R_i: Expected capital return of technology i [decimal number]
- $FC_{G,i}$: Fixed costs per installed generation capacity for technology i [$e.g.\ EUR/MW_e$]
- $VC_{G,i}$: Variable costs per unit of electricity for technology i [$e.g.\ EUR/MWh_e$]
- $\eta_{turbine,i}$: Turbine efficiency in terms of lower heating value (LHV) for turbine technology i [$\in (0, 945]$]

Variables:

- $C_{G,i}$: Installed capacity of gas turbine technology i [$e.g.\ MW_e$]
- $G_{elec,it}$: Generation of hydrogen with gas turbine technology i at time t [$e.g.\ MWh_e$]
- $Conv_{H2ToElec,it}$: Hydrogen converted to electricity with turbine technology i at time t [$e.g.\ kg\ H_2$]

Objective:

$$Cost_i = (CAPEX_{G,i} R_i + FC_{G,i}) C_{G,i} + \sum_t VC_{G,i} G_{elec,it} \tag{24}$$

Constraints:
Electricity generation is limited by the installed capacity.

$$G_{elec,it} \leq C_{G,i} \tag{25}$$

Hydrogen consumption is determined by the electrical efficiency and hourly electricity generation.

$$Conv_{H2ToElec,it} = \frac{G_{elec,it}}{LHV_{H2}\ \eta_{turbine,i}} \tag{26}$$

5.3 Heat production

5.3.1 Heat pump

Parameters:

- $CAPEX_{G,i}$: Capital expenditures per installed generation capacity for heat pump technology i [*e.g.* EUR/MW_e]
- R_i: Expected capital return of heat pump technology i [decimal number]
- $FC_{G,i}$: Fixed costs per installed generation capacity for heat pump technology i [*e.g.* EUR/MW_e]
- $VC_{G,i}$: Variable costs per unit of heat produced for heat pump technology i [*e.g.* EUR/MWh_{th}]
- $COP_{max,i}$: Maximum coefficient of performance for heat pump technology i at time t [decimal number]
- $COP_{relative,it}$: COP value at time t per $COP_{max,i}$ [$\in (0, 1]$]

Variables:

- $C_{G,i}$: Installed capacity of heat pump technology i in terms of electric power [*e.g.* MW_e]
- $G_{heat,it}$: Generation of heat with heat pump technology i at time t [*e.g.* MWh_{th}]
- $Conv_{ElecToHeat,it}$: Electricity converted to heat at time t [*e.g.* MW_e]

Objective:

$$Cost_i = (CAPEX_{G,i}R_i + FC_{G,i})C_{G,i} + \sum_t VC_{G,i}G_{heat,it} \tag{27}$$

Constraints:
Electricity usage is limited by the installed capacity.

$$Conv_{ElecToHeat,it} \leq C_{G,i} \tag{28}$$

Electricity consumption is determined by heat generation, maximum COP and relative COP.

$$Conv_{ElecToHeat,it} = \frac{G_{heat,it}}{COP_{max,i}COP_{relative,it}} \tag{29}$$

5.4 Hydrogen production

5.4.1 Electrolyzer

Parameters:

- $CAPEX_{G,i}$: Capital expenditures per installed generation capacity for electrolyzer technology i [$e.g.$ EUR/MW_e]
- R_i: Expected capital return of electrolyzer technology i [decimal number]
- $FC_{G,i}$: Fixed costs per installed generation capacity for electrolyzer technology i [$e.g.$ EUR/MW_e]
- $VC_{G,i}$: Variable costs per unit of hydrogen for electrolyzer technology i [$e.g.$ $EUR/kg\ H2$]
- $RampU_i$: Maximum ramp-up of hydrogen generation per installed capacity $C_{G,i}$ [$\in (0, 1]$]
- $RampD_i$: Maximum ramp-down of hydrogen generation per installed capacity $C_{G,i}$ [$\in (0, 1]$]
- $\eta_{elec,i}$: Electrical efficiency in terms of lower heating value (LHV) for electrolyzer technology i [$\in (0, 1.058]$]
- $\eta_{heat,i}$: The share of excess heat utilized compared to the total waste heat for electrolyzer technology i [$\in [0, 1]$]

Variables:

- $C_{G,i}$: Installed capacity of electrolyzer technology i in terms of electric power [$e.g.$ MW_e]
- $G_{H2,it}$: Generation of hydrogen with electrolyzer technology i at time t [$e.g.$ MWh_e]

- $G_{heat,it}$: Generation of heat with electrolyzer technology i at time t [*e.g.* MWh_{th}]
- $Conv_{ElecToH2AndHeat,it}$: Electricity converted to hydrogen and heat at time t [*e.g.* MW_e]

Objective:

$$Cost_i = (CAPEX_{G,i}R_i + FC_{G,i})C_{G,i} + VC_{G,i}G_{H2,i} \tag{30}$$

Constraints:
Production cannot exceed installed capacity. The constraint accounts for the fact that generation is in terms of kilograms and capacity is defined in terms of electric power.

$$\frac{G_{H2,it}LHV_{H2}}{\eta_{elec,i}} \leq C_{G,i} \tag{31}$$

Change in hydrogen production cannot exceed the maximum ramp-up / ramp-down time.

$$(G_{H2,it} - G_{H2,it-1})\frac{LHV_{H2}}{\eta_{elec,i}} \leq C_{G,i}RampU_i \tag{32}$$

$$(G_{H2,it-1} - G_{H2,it})\frac{LHV_{H2}}{\eta_{elec,i}} \leq C_{G,i}RampD_i \tag{33}$$

Electricity consumption is determined by the electrical efficiency and hourly hydrogen production.

$$Conv_{ElecToH2AndHeat,it} = \frac{G_{H2,it}LHV_{H2}}{\eta_{elec,i}} \tag{34}$$

Heat production is determined by hydrogen production, efficiency of waste heat utilization, lower heating value of hydrogen, and electrical efficiency. Theoretical maximum for electrical efficiency (in terms of LHV) is 105.8%, in which case no heat is generated and all electricity is converted to hydrogen.

$$G_{heat,it} = G_{H2,it}\ \eta_{heat,i}\ LHV_{H2}(\frac{1}{\eta_{elec,i}} - \frac{1}{1.058}) \tag{35}$$

5.5 Storage

Subclasses:

- Hydrogen storage
- Electric battery
- Thermal energy storage

Parameters:

- $CAPEX_{G,i}$: Capital expenditures of the power capacity of storage technology i [$e.g.\ EUR/MW_e$]
- $CAPEX_{S,i}$: Capital expenditures of the storage capacity of storage technology i [$e.g.\ EUR/MWh_e$]
- R_i: Expected capital return of storage technology i [decimal number]
- $FC_{G,i}$: Fixed cost of the power capacity of storage technology i [$e.g.\ EUR/MW_e$]
- $FC_{S,i}$: Fixed cost of the storage capacity of storage technology i [$e.g.\ EUR/MWh_e$]
- $\eta_{Charge,i}$: Charge efficiency of storage technology i [$\in (0,1]$]
- $\eta_{Discharge,i}$: Discharge efficiency of storage technology i [$\in (0,1]$]
- $\eta_{SelfDischarge,i}$: Self-discharge rate of storage technology i [$\in [0,1)$]
- $\alpha_{pump,i}$: Electricity consumption of hydrogen pump per hydrogen charged [$e.g.\ MWh_e/kg\ H_2$](only hydrogen storage)
- DOD: depth of discharge, tells how much of the battery is usable that will not overly discharge the battery (only for electricity storage)

Variables:

- $C_{G,i}$: Installed power capacity of storage technology i [$e.g.\ MW_e$]
- $C_{S,i}$: Installed storage capacity of storage technology i [$e.g.\ MWh_e$]
- S_{it}: Energy storage level of technology i at time t [$e.g.\ MWh_e$]
- $Charge_{it}$: Charge of energy into storage i at time t [$e.g.\ MWh_e$]
- $Discharge_{it}$: Discharge of energy from storage i at time t [$e.g.\ MWh_e$]
- ΔS_{it}: Effect of the change in storage on the energy balance [$e.g.\ kg\ H_2$]
- $Conv_{ElecToH2,it}$: Electricity consumed by hydrogen pump i at time t [$e.g.\ MWh_e$] (only hydrogen storage)

Objective:

$$Cost_i = (CAPEX_{G,i}R_i + FC_{G,i})C_{G,i} + (CAPEX_{S,i}R_i + FC_{S,i})C_{S,i} \quad (36)$$

Constraints:
Storage level is limited by the installed capacity.

$$S_{it} \leq C_{S,i} \quad (37)$$

Storage level at $t+1$ is determined by the storage level at time t, self-discharge rate as well as charge and discharge at time t.

$$S_{it+1} = S_{it}(1 - \eta_{SelfDischarge,i}) + Charge_{it} - Discharge_{it} \tag{38}$$

The effect of charge and discharge and their corresponding efficiencies on the energy balance is:

$$\Delta S_{it} = \frac{Charge_{it}}{\eta_{Charge,i}} - Discharge_{it}\ \eta_{Discharge,i}. \tag{39}$$

Change in storage is limited by the maximum power capacity.

$$\Delta S_{it} \leq C_{G,i} \tag{40}$$
$$\Delta S_{it} \geq -C_{G,i} \tag{41}$$

Initial and final storage levels are set to be equal to make the storage cyclical.

$$S_{it=1} = S_{it=last\ time\ step}. \tag{42}$$

Electricity consumption of hydrogen compression is determined by the charge rate and the pump's specific electricity consumption.

$$Conv_{ElecToH2,it} = Charge_{it}\alpha_{pump,i} \tag{43}$$

5.6 Scenario

Scenario should include a demand object and a list of technologies also it has the following constants:

- $nSmpl$: the amount of time frames during simulation. The interpretation is of them is left for the user, they can be viewed as hours, day, month or anything else.
- LHV_H2: Lower heating value of hydrogen
- $name$: Name of the scenario

5.7 Demand and Supply

Parameters:

- $D_{elec,t}$: Base demand of electricity at time t (does not account for heat or H_2 production) [$e.g.\ MWh_e$]
- $D_{heat,t}$: Demand of heat at time t [$e.g.\ MWh_{th}$]
- $D_{H2,t}$: Demand of hydrogen at time t [$e.g.\ kg\ H_2$]

Variables (inherited from other classes):

- $Conv_{ElecToH2AndHeat,it}$: Electricity converted to hydrogen and heat with technology i at time t [$e.g.\ MWh_e$]
- $Conv_{ElecToH2,it}$: Electricity consumed by hydrogen pump i at time t [$e.g.\ MWh_e$]
- $Conv_{H2ToElec,it}$: Hydrogen converted to electricity with technology i at time t [$e.g.\ kg\ H_2$]
- $G_{elec,it}$: Generation of electricity with technology i at time t [$e.g.\ MWh_e$]
- $G_{H2,it}$: Generation of hydrogen with technology i at time t [$e.g.\ kg\ H_2$]
- $G_{heat,t}$: Generation of heat with technology i at time t [$e.g.\ MW_e$]
- $\Delta S_{elec,it}$: Change in electricity storage level of storage technology i at time t [$e.g.\ MWh_e$]
- $\Delta S_{H2,it}$: Change in hydrogen storage level of storage technology i at time t [$e.g.\ kg\ H_2$]
- $\Delta S_{heat,it}$: Change in heat storage level of storage technology i at time t [$e.g.\ MWh_{th}$]

Constraints:
Electricity, hydrogen and heat supply equals demand. If storage is added to the model, supply-demand balance is defined as:

$$\sum_i G_{elec,it} - \sum_i \Delta S_{elec,it} - \sum_i Conv_{ElecToH2AndHeat,it} - \sum_i Conv_{ElecToHeat,it} - D_{elec,t} = 0 \quad (44)$$

$$\sum_i G_{H2,it} - \sum_i \Delta S_{H2,it} - \sum_i Conv_{H2ToElec,it} - D_{H2,t} = 0 \quad (45)$$

$$\sum_i G_{heat,it} - \sum_i \Delta S_{heat,it} - D_{heat,t} = 0. \quad (46)$$

If no storage is added, generation needs to exceed demand.

$$\sum_i G_{elec,it} - \sum_i Conv_{ElecToH2AndHeat,it} - \sum_i Conv_{ElecToH2,it} - \sum_i Conv_{ElecToHeat,it} - D_{elec,t} \geq 0 \quad (47)$$

6 User Input

The `Scenario` Class: is a cornerstone of the simulation environment, serving as a structure to define and manage different energy scenarios. This module encompasses functionalities for managing demand profiles, instantiating technologies, and ensuring the cohesive interaction of different components during a simulation run. Let's delve into its main components and functionalities.

6.1 Using user_input.py for Hardcoded Scenarios

The `user_input.py` module serves as a legacy tool that enables users to formulate scenarios directly within the Python script. It is now being overshadowed by the more advanced YAML-based methodologies. While robust, the hardcoded method lacks the agility and versatility offered by the YAML-based approach. It is still useful for specific cases or for users favoring a Python-focused method.

1. **Defining Scenario and Demand:** Scenarios are defined using the 'Scenario' class, which requires a technology list ('techList'), a demand profile, and other parameters like the scenario name.

   (a) Instantiate the 'Demand' class and set up the demand profiles.

   (b) Use the `readExcel`, `hourlyProfile` or other appropriate methods to load demand data into the scenario's demand object. This allows the demand to be modified or updated based on the specific scenario.

   (c) Instantiate the 'Scenario' class by passing in the configured demand, a scenario name, the sample size (nSmpl), and the lower heating value of hydrogen (e.g. if MWh is used as the unit for electricity production, input LHV_H2 = 0.0333 [MWh / kg H_2]).

   For instance:

```
demand = Demand()
scenarioAU = Scenario(techList=[], demand=demand,
    scenarioName="Scenario Australia", nSmpl=8760, LHV_H2=0.0333)
scenarioAU.demand.D_BaseElec = scenarioAU.readExcel(demandFileAU,
    "B")


```

2. **Adding Technologies:** Technologies can be added to a scenario's 'techList'. Each technology module has its own classes and required parameters that can be instantiated with specific parameters and can be found in chapter "Technologies". Once an instance of a technology is created, it's appended to the scenario's 'techList'.

   For example:

```
    scenarioAU.techList.append(pp.SolarPower(CAPEX_G = 761000,
        R= 0.0640, ...))

```

3. **Running Scenarios:** After setting up the scenario, it should be added to the ‘scenariosToRun‘ list. Once added, the scenario will be executed when functions or scripts that utilize the ‘scenariosToRun‘ list are called.

```
    scenariosToRun = [scenarioAU]

```

6.2 Writing YAML Scenarios: A Guide

YAML (YAML Ain't Markup Language) is a human-readable data serialization format. When combined with the capabilities of `yamled.py`, YAML becomes a powerful tool for defining energy scenarios. This guide will walk you through the steps of creating a comprehensive scenario using the YAML format.

6.2.1 Basic Structure

A scenario in YAML begins with the scenario name, followed by key-value pairs that detail its components:

```
Scenario Name:
  key1: value1
  key2: value2
```

6.2.2 Defining General Attributes

Typically, a scenario starts with general attributes such as the scenario name, number of samples, and LHV_H2:

```
Scenario Nuclear:
  name: "Scenario Nuclear"
  nSmpl: 10
  LHV_H2: 0.0333
```

6.2.3 Configuring Demand

For defining demand, you need to specify the type of demand and its associated profile:

```
  demand:
    D_Elec:
      get_value_mode: "Sin"
      value: 1000
```

The `get_value_mode` can be set to any setting in `hourlyProfile` from `scenario` like "Sin" or "Const", or "Excel", allowing for flexibility in defining demand patterns.

6.3 Adding Technologies

The ‘technologies‘ section allows for the configuration of multiple technologies within a scenario:

```
technologies:
  TechnologyName:
    type: "path.to.technologyClass"
    attribute1: value
    attribute2: value
```

For instance, if defining a nuclear power plant:

```
Nuce:
  type: "technologies.powerPlants.Nuclear"
  CAPEX_G: 10
  R: 0.5
  ...
```

Remember, each technology should have its type defined, and the subsequent attributes will depend on the type of technology being described.

6.4 Using Excel Data

If you're sourcing data from Excel, ensure that you specify the file path, the column, and optionally, the sheet name:

```
get_value_mode: Excel
file: "./data/some_data.xlsx"
column: B
sheet: OptionalSheetName
```

6.5 Final Notes

1. Indentation Matters: YAML is sensitive to indentation. Ensure consistent spacing (preferably two spaces) for nested items.
2. Strings: For values that are strings, enclose them in double quotes, e.g., ‘name: "Scenario Nuclear"‘.
3. Lists: In YAML, lists are defined with a hyphen (dash) followed by a space.
4. Comments: You can add comments in YAML using the 'hash' symbol. Comments are not processed and can be useful for adding notes or explanations.

In conclusion, creating scenarios using YAML is both concise and intuitive. Combined with the `yamled.py` module, this approach streamlines the process of setting up and executing energy scenarios. Ensure that your ‘.yaml‘ file adheres to the structure and guidelines detailed above for a smooth simulation experience.

6.6 Defining Technologies

Different technologies can be defined and customized within the powerPlant, electrolyzer, and other related classes, by following already implemented classes in Technologies chapter and package. First write a make a formal definition, write all appropriate formulas and then find the class the the most closely resemble the desired technology. create new class with inheritance and add or modify variables and constraints according to your needs.

7 Visual Results and Reporting Tools

7.1 Visualizing Results

`visualizer.py` is the module for results visualization. The module includes some predefined functions for basic visualizations, including electricity, heat, and hydrogen supply and consumption plots as well as functions for storage level visualization.

The module is called in `main.py` by calling `plotScenario`. The user can pass `mode="subplot"` or `mode="separate"` as an input argument to define whether plots are combined into one figure or displayed separately.

In the `plotScenario` function, the user can define in the `plotting_tasks` list all the displayed plot types. Additionally, the user should define here:

- `x`: Index used in the x-axis. x: range from zero to sample size (main type). x1: range from zero to sample size + 1 (hourly storage levels require one additional data point).
- `coordinates`: (rowIndex, colIndex) of the subplot (e.g. (2,0) = (third row, first column)).

The user can define additional functions and call them in the `plotting_tasks` if necessary.

7.2 Exporting Results

Results can be exported to a CSV file with the `exportRes` method in `resultExport.py`. The method is called in `main.py` and it exports the objective value as well as the following results if needed:

- Capacity by technology. Capacities are exported if an argument `capacity = True` is passed when calling `exportRes` (default) and no capacities are exported when `capacity = False` is passed as the argument.
- Cost by technology. Costs are exported if an argument `cost = True` is passed when calling `exportRes` (default) and no cost data is exported when `cost = False` is passed as the argument.
- Generation of electricity, hydrogen, and heat by technology by hour. Generation data is exported if an argument `generation = True` is passed when calling `exportRes` (default) and no generation data is exported when `generation = False` is passed as the argument.
- Storage and change in storage by technology by hour. Storage data is exported if an argument `storage = True` is passed when calling `exportRes` (default) and no storage data is exported when `storage = False` is passed as the argument.

8 Appendices

- Glossary of Terms
- References and Additional Resources

8.1 Docs

8.1.1 `main.py`

`main.py` serves as the entry point for the modeling system. It contains the primary workflow from processing the user input to visualizing the results.

1. **Function:** `resultProcessor(model, msg)`:
    - Accepts a model object and a message (scenario name).
    - **Nested Function:** `getRes()`:
        - Provides a concise report on the console about the required capacities of different technologies, storage needs, and the objective value.
        - Has a verbose mode for debugging that displays the value of every MIP variable and the electricity generation.
    - After calling `getRes()`, it exports the results using `exportRes` and plots the scenario outcomes with `visu.plotScenario`.
2. **Execution Sequence**:
    - `scenariosToRun` fetches all the scenarios from the `yamled` module.
    - For each scenario, the following steps occur:
        - The scenario details are printed to the console.
        - A `ModelRunner` instance is created with the scenario specifications.
        - The model is run using the `run_model` method of the runner.
        - Results are processed using the `resultProcessor` function, which both prints results to the console and visualizes them.

In essence, `main.py` provides an interface for users to run predefined scenarios, view results on the console, visualize the outcomes, and obtain exportable results. It is designed to handle multiple scenarios in sequence, making it easy for users to compare various energy strategies.

8.1.2 scenario.py

- `Demand` Class This class manages different types of energy demands:
    - `D_Elec`: Demand for electricity.
    - `D_H2`: Demand for hydrogen.

  - `D_Heat`: Demand for heat.

  Methods:

  - `setDemand(D_Type, arr)`: Set a specific type of demand using an array.

- `Scenario` Class This class facilitates the creation and management of energy scenarios:

  Attributes:

  - `externFiles`: Indicates if external files are being used.
  - `techList`: List of technologies involved in the scenario.
  - `demand`: Demand object.
  - `scenarioName`: Name of the scenario.
  - `nSmpl`: Number of samples.
  - `LHV_H2`: Lower Heating Value of H2.

  Methods:

  - `useExternalFiles(file, col)`: Enables the use of external files for data and determines the number of samples.
  - `readCsv(fname, col, skipRows)`: Reads data from a CSV file.
  - `readExcel(fname, col, start_row, sheet_name)`: Reads data from an Excel file.
  - `hourlyProfile(mode, value)`: Generates an hourly profile based on the mode selected. Supported modes include:
    * `"Const"`: Constant value.
    * `"Rising"`: Increasing linearly.
    * `"Falling"`: Decreasing linearly.
    * `"Sin"`: Sine wave.
    * `"Cos"`: Cosine wave.
    * `"Arch"`: Arch-shaped wave.
    * You can define any other mode

The `scenario.py` module provides the foundational building blocks for defining and manipulating energy scenarios. The `Demand` class encapsulates different energy demands while the `Scenario` class builds upon it by allowing the user to define various scenarios using different technologies and demand profiles. The flexibility to import data from external sources or generate data in different patterns provides the user with a comprehensive toolkit to experiment with different energy strategies.

8.1.3 `yamled.py`

The `yamled.py` module serves as a bridge between human-readable scenario definitions in YAML format and the underlying Python framework. Its primary function is to interpret the contents of a given YAML file and translate them into objects and attributes within the simulation environment. This not only simplifies the scenario definition process but also makes the entire system more modular and maintainable. Let's delve into the key functionalities of `yamled.py`:

1. **Value Extraction:** The function `get_values()` retrieves data from the YAML file, either from an Excel source or by creating creating values with `hourlyProfile` from `scenario` instance.

2. **Technology Creation:** The function `technology_from_conf()` interprets the 'technologies' section of the YAML file. This includes identifying the technology type, extracting parameters, and instantiating the appropriate Python class.

3. **Demand Configuration:** Using the function `create_demand_from_config()`, the module reads demand profiles, either from a predefined mode or from an Excel file, and associates them with a scenario.

4. **Scenario Assembly:** The function `scenario_builder_from_conf()` is where all pieces come together. An empty scenario is first created, populated with technologies and demands, and then returned.

5. **YAML Loading:** Finally, the module features the `load_from_yaml()` function that reads a given YAML file and returns a dictionary of scenarios, with each scenario name as a key.

In summary, the `yamled.py` module streamlines scenario creation by offering a YAML-based interface. This allows for easier, more intuitive definitions, and the potential to define multiple scenarios in one concise file. The associated `test.yaml` file showcases examples of how scenarios can be structured and defined.