



EMB3Rs

Heat and Cold matching platform

D3.4 - The EMB3RS Techno-Economic Optimization Module

AUTHORS : SHRAVAN KUMAR, JAGRUTI THAKUR,
FRANCESCO GARDUMI

DATE : 26.10.2022



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement



Technical References

Project Acronym	EMB3Rs
Project Title	User-driven Energy-Matching & Business Prospection Tool for Industrial Excess Heat / Cold Reduction, Recovery and Redistribution
Project Coordinator	Mafalda Silva
Project Duration	September 2019 – May 2023 (45 months)

Deliverable No.	3.4 - Techno-economic Optimisation Module including User and System Manuals
Dissemination level ¹	PU
Work Package	WP3 – Platform Creation - Modules Development, Integration & Lab-scale Validation
Task	T3.1 – Individual Module Development
Lead beneficiary	KTH
Contributing beneficiary(ies)	
Due date of deliverable	01/02/2023
Actual submission date	26/10/2022

¹ PU = Public

PP = Restricted to other programme participants (including the Commission Services)

RE = Restricted to a group specified by the consortium (including the Commission Services)

CO = Confidential, only for members of the consortium (including the Commission Services)

Document history

V	Date	Beneficiary	Author
1	2022-04-14	KTH	Shravan Kumar, Jagruti Thakur, Francesco Gardumi
2	2022-10-26	KTH	Shravan Kumar, Jagruti Thakur, Francesco Gardumi
3			
3			



Summary

This deliverable presents the Techno-Economic Optimisation (TEO) module and includes the User and System Manuals.

The user manual for these modules will guide and help the user understand and navigate the module, as part of the integrated platform. The user manual is developed for an average user and includes examples and infographics to promote the user-friendliness of the platform.

The system manual is designed for the advanced user. It is more comprehensive and includes the description of all the inputs and the variables, along with the coding specifications and a detailed presentation of all functionalities.

It describes the functioning and use of the module. The EMB3RS project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 847121. This module is part of a larger assessment toolbox called the 'EMB3RS platform'.



Disclaimer

Any dissemination of results must indicate that it reflects only the author's view and that the Agency and the European Commission are not responsible for any use that may be made of the information it contains.



Table of Contents

TECHNICAL REFERENCES	2
DOCUMENT HISTORY	2
SUMMARY	3
DISCLAIMER	4
TABLE OF CONTENTS	5
1 INTRODUCTION	7
1.1 EMB3RS PROJECT	7
1.1.1 Core functionalities (CF) module	8
1.1.2 GIS module	8
1.1.3 TEO Module	8
1.1.4 Market Module	9
1.1.5 Business Module	9
1.2 MODULE DEVELOPMENT TIMELINE	9
2 SYSTEM MANUAL	11
2.1 PURPOSE AND SCOPE	11
2.2 MAIN FEATURES OF THE TEO MODULE	11
2.3 GENERAL MODULE ARCHITECTURE	12
2.4 MODULE REQUIREMENTS	13
2.5 DETAILED MODULE DESCRIPTION	14
2.5.1 Structure of the TEO module	14
2.5.2 Inputs and Outputs	41
2.5.3 Contributions and requirements for the knowledge base	47
2.5.4 Functioning of the TEO module	49
2.5.5 Interaction with other modules	52
2.5.6 Pre-Conditions	52
2.5.7 Input data error handling	53
2.6 REPORTS	56
2.6.1 Contribution to the main Simulation report	56
2.6.2 Detailed Module report and configuration	57
3 USER MANUAL	59
3.1 INTRODUCTION TO THE TEO	59
3.2 MAIN FEATURES OF THE TEO MODULE	60
3.3 USER INPUTS	61
3.4 SIMULATION	66
3.4.1 Actors	66
3.4.2 Pre-Conditions	67



3.4.3	Basic Flow for the user	67
3.4.4	User-defined constraints	73
3.5	RUNNING A TEST CASE USING THE STANDALONE VERSION OF THE TEO	74
3.5.1	Description of the test case	74
3.5.2	Data and instructions to run the model	76
3.5.3	Note on the solvers	77
3.5.4	Results from the Test Case	78
4	REFERENCES	83
5	APPENDIX 1	84
	EQUATIONS OF THE TEO CODE	84



1 Introduction

The EMB3Rs platform aims to provide quantitative insights into different options for the potential use of excess heat and cold. This is intended to enable easier and faster identification of potentially interesting options for the use of excess heat and cold. To do this, the platform must be able to present the technical and economic situation in sufficient detail and provide relevant indicators for decision-making. The Techno-Economic Optimization (TEO) module identifies the most cost-effective combinations of technologies for the use and transport of excess heat and cold (HC) from specific sources to specific sinks. The user (representing the generator of excess heat - i.e., the source - or a demand point - i.e., the sink) wants to evaluate the options for using the excess HC generated to meet the heating/cooling needs of one or more known/specified sinks. The goal of the optimization is to find the most cost-effective combination of technologies and the best match between sources and sinks that will meet the demand, considering the constraints imposed by regulation, heat availability, load profiles, and techno-economic characteristics of the technologies, investment plans, etc.

The main aim of this report is to describe the techno-economic optimisation module of the EMB3RS platform along with its user and system manuals. A detailed description of the structure and functioning of the module is provided along with its main inputs and outputs and instruction to run the module. The report consists of 8 sections. Firstly, an introduction to the TEO module is presented in Section 2. The third section provides the system description of the module. In section 4 the instruction for running a test case using the standalone version of the TEO module is presented. Along with the report, the user and system manuals of the TEO module are also documented as publicly available online sources. The code, test case, metadata and everything that is needed to run TEO is on GitHub [here](#) and an extensive (and continuously updated) description of TEO (including most of the material of this report) is publicly available as ReadTheDocs documentation [here](#).

1.1 EMB3RS project

EMB3RS ("User-driven Energy-Matching & Business Prospection Tool for Industrial Excess Heat/Cold Reduction, Recovery and Redistribution") is a European project funded under the H2020 programme (Grant Agreement No.847121) to develop an open-source tool to match potential sources of excess thermal energy with compatible users of heat and cold. For more information about the EMB3RS project, please visit the [EMB3RS website](#).

Users, such as industries and other sources that produce excess heat, specify the essential parameters, such as their location and the available excess thermal energy. The EMB3Rs platform will then autonomously and intuitively assess the feasibility of



new business scenarios and identify the technical solutions to match these sources with compatible sinks. End-users such as building managers, energy communities or individual consumers will be able to determine the costs and benefits of industrial excess heat and cold utilisation routes and define the requirements for implementing the most promising solutions. The EMB3Rs platform will integrate several analysis modules that will allow a comprehensive study of the feasible technical pathways to recover and use the available excess thermal energy.

Several other modules are part of the EMB3RS platform. Each module will be used to perform a specific task or analysis of excess heat and cold recovery. The models and their primary functionalities are listed below.

1.1.1 Core functionalities (CF) module

The purpose of the CF module is to provide a comprehensive quantification of the energy flows of the EMB3RS platform objects (sinks, sources, and links) and costs associated with different options for excess H/C recovery and use. The other analysis modules (GIS, TEO, MM and BM) to perform simulations according to user specifications use this information. As implemented in M29, the CF module has two main functionalities:

1. Full characterization of objects – e.g., in terms of processes, equipment, building characteristics
2. To carry out a preliminary analysis of available supply and demand - described as a simulation feature within the CF.

1.1.2 GIS module

The purpose of the GIS model within EMB3Rs is to analyse possible network solutions for a given set of sources and sinks as well as an assumption of related network heat/cold losses and costs. The GIS thereby finds such a network solution along with the existing Open Street Map (OSM) Road Network connecting all sources and sinks. It currently outputs a graph/map that lets the user check the specifications of every single pipe element from the network found and a table that illustrates all source/sink specific losses, costs, network length and installed pipe capacity.

1.1.3 TEO Module

The TEO module identifies the least-cost combinations of technologies for using and conveying excess heating or cooling (HC) from defined sources to defined sinks. The user (representing the excess heat producer - i.e., source – or a demand point – i.e., sink) wants to evaluate the least-cost options of utilising excess HC generated to meet the heating/cooling demand for one or more known/assumed sinks. The objective of the optimisation is to find the least-cost mix of technologies (in terms of installed capacities – typically, in power units) and match between sources and sinks (in terms of energy flows) that satisfy the demands under constraints dictated by regulation,



availability of heat, load profiles, techno-economic characteristics of technologies, investment plans.

1.1.4 Market Module

The Market Module (MM) will provide the user with economic and fairness indicators like energy transaction, market price, social welfare, and fairness among prices. This will be done by short-term and long-term market analyses that simulate various market structures and incorporate business conditions and network models. The MM will consider the existing Pool market as well as new forms of a decentralized market based on peer-to-peer and community systems. The modelling of heat/cold sources and sinks will include flexibility, offering price and business preferences.

1.1.5 Business Module

Business Model Module evaluates various business models for DHC which incorporate excess heat. This is done by calculating matrices like Net Present Value (NPV), Levelized Cost of Heat (LCOH) and Internal Rate of Return (IRR) under different ownership structures and market frameworks.

1.2 Module development timeline

The techno-economic optimization module has been developed in various stages during the course of the EMB3RS project. Firstly, the prototype version of the module was developed. This version was tested using a very simple test case. The prototype was further enhanced with several added functionalities to the final standalone version of the module. The main activities and the timeline of module development (until M30) are shown in Figure 1. After this, the bulk of the work developed focused on supporting the integration on the platform.

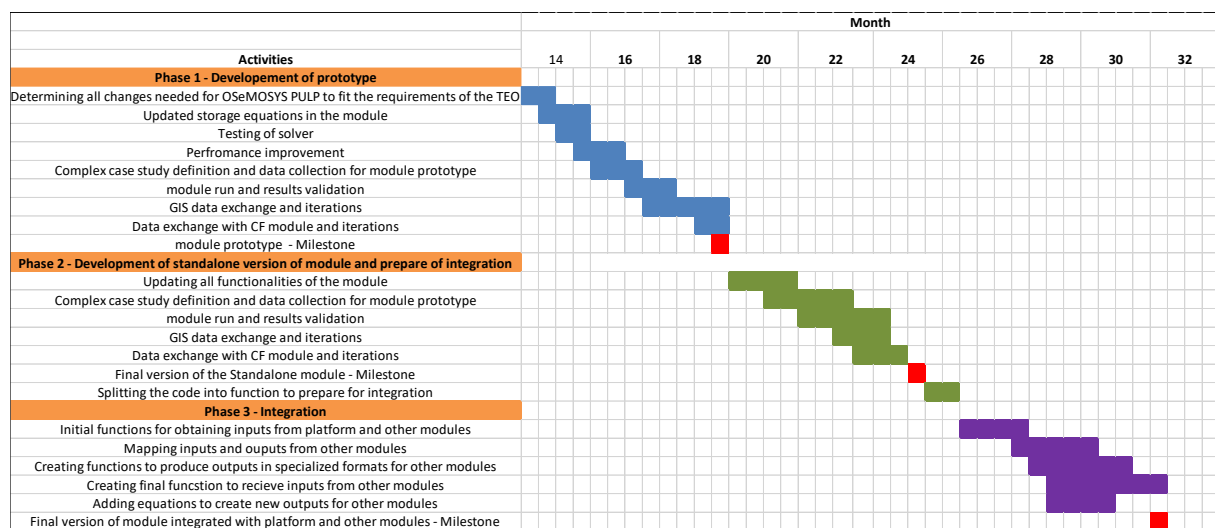


Figure 1: Module development timeline (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

The initial steps in the module development involved the review of the existing tool 'OSeMOSYS – PULP' to determine its capabilities to model a case of excess heat



recovery and meet the requirements of the TEO in the EMB3RS platform. Further, some additions were made to the tool in terms of updated storage equations represent the operation of thermal storage, adding equations to include losses from the heating network in the energy balance, improving the performance of the code by reducing the size of the solution matrix and testing the module with different solvers. These additions led to the prototype version of the module.

The prototype was first tested with a simple use case and the results were validated. In the second phase, all pre-existing functionalities of the model were updated to fit the requirements of the TEO module and validated. A complex real-life use case was used to test and validate the module code. Furthermore, this phase also included some iterations with the GIS and the CF modules. The methodology for the iteration and the data exchange was initially discussed in this phase. The TEO-GIS iteration was tested first using a simple test case and validated. Later, the complex test case was also simulated using the TEO and the GIS and the results were further validated. The final version of the standalone code was thus developed. The code was further split into several functions to make it more modular for the integration. The function for building and running the model was separated from the function for obtaining inputs and writing out the outputs. This version of the code can be accessed on GitHub [here](#).

Finally, in the third phase, the module and the functions were further developed and changed to facilitate the integration process. The standalone version of the module obtained inputs from an excel file. This had to be changed for integration. An initial function was developed to create a part of the inputs. The module was first tested with some inputs and default values for others. The function is further developed to obtain inputs from the platform, the CF module and the GIS module. Simultaneously, the data exchange between the TEO and the other modules was discussed and finalized. To facilitate the data exchange, additional equations were added to outputs for the business and the market modules. Furthermore, the inputs from the TEO to the GIS module have a specialized format that needs significant post-processing of the TEO results. A function was created for this purpose. Lastly, the TEO module was fully integrated with the platform and the other modules when all functions were tested.



2 System Manual

2.1 Purpose and scope

The Techno-Economic Optimization (TEO) module identifies the least-cost combinations of technologies for using and conveying excess Heat and Cold (HC) from defined sources to defined sinks. The user (representing the excess heat producer - i.e., source – or a demand point – i.e., sink) wants to evaluate the options of utilizing excess HC generated to meet the heating/cooling demand for one or more known/assumed sinks. The objective of the optimisation is to find the least-cost mix of technologies and match between sources and sinks that satisfies the demands under constraints dictated by regulation, availability of heat, load profiles, techno-economic characteristics of technologies, investment plans, etc. The mix of technologies may include the District Heating/Cooling Network (DH/CN), technologies to upgrade the Temperature level on the sink or the source side, thermal storage on the sink or the source side, as well as heating alternatives.

The main desired features of the tool for building the techno-economic optimisation module within the EMB3RS framework are:

- High temporal resolution – Daily to hourly
- Low simulation time
- High flexibility and ability to be modified
- Interaction and interoperation with other modules
- Open access

The techno-economic optimisation module should provide the following results:

- The technology mix (existing and newly installed yearly capacities in terms of energy flows throughout the supply-demand chain)
- Share of each technology in meeting the demand in any time step of the analysis (where the time resolution is defined by the user within certain limits) and throughout the analysis period
- Annual costs (investment, fuel, operation & maintenance, Levelized costs of heat (LCOH) etc.) associated with the technologies
- Emissions, emission savings and emission costs over the defined period.

2.2 Main Features of the TEO Module

- The TEO module optimizes the matching between the different sources and the sinks while taking into accounts various technical and economic constraints,



such as demand profiles, technology cost, efficiencies and losses while also considering thermal energy storage.

- An optimal mix of investments in technologies and optimal capacities in storage and district heating network can be determined. The operation of the technologies and the intra-annual heat supply are also optimized in the module.
- The module can also analyse the competition between centralized and decentralized solutions. The current input data includes competition between waste heat sources and decentralized solar thermal based heating solutions.
- The TEO module optimizes the matching between the different sources and the sinks while accounting for various technical and economic constraints, such as demand profiles, technology cost, efficiencies and losses while also considering thermal energy storage.
- The TEO module carries out a socio-economic type of optimisation, where the total system cost is minimised, irrespective of who bears it. It does not take a policy-maker, investor, or business perspective.
- The time domain, time resolution and technological options are flexible and chosen by the user. For example, an analysis can be carried out for a time domain of 5, 10 or 30 years. Similarly, the time resolution can be of few time steps in a year, up to 8760 hourly time steps. The types of technologies that can be modelled include heat exchangers, heat pumps, boilers etc. The module is a model generator, where none of the above is pre-defined.
- The module relies on two core types of objects: Technologies and Fuels. These are very flexibly defined so that many different processes and commodities can be represented in a model. A Technology is nothing but a process - I.e. a box – with inputs, outputs, a transfer function between them, and several associated techno-economic characteristics. A Fuel is any commodity entering or exiting a Technology. Therefore, with a Technology, the user may represent a heat exchanger or a heat pump and for Fuel, the user may represent electricity or the excess heat stream.

2.3 General module architecture

The TEO module is developed in Python. The module is organised into several functions as shown in Figure 2. The TEO module receives inputs from the CF module, the GIS module and the user. These inputs are then prepared in the format needed by the TEO module using the 'prepare_inputs' function. The inputs are then segregated into 'sets', 'parameters' and 'defaults' by separate functions. The segregation of the input data is essential for the data to suit the structure of the TEO module, which is described in section 3.2. Further, the 'buildmodel' function is used to build the linear program based optimisation model and solve it to determine the least cost matching of



sinks and sources. The function has several sub-functions that assist in data preparation and solving the model. Finally, the 'CreateResults' function is used to prepare the results in the required format for the user and the other modules. All the functions can be found on the GitHub repository 'EMB3RS-TEO-Module' [here](#).

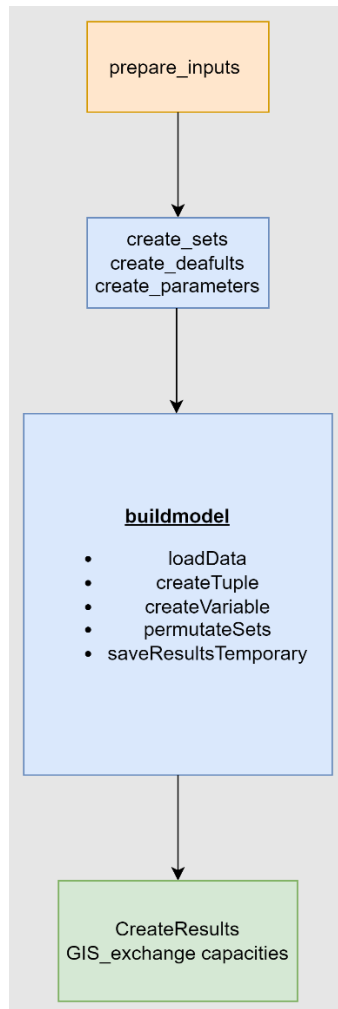


Figure 2: TEO Architecture (Author: Shravan Kumar, licensed under [CC-BY 4.0](#))

2.4 Module requirements

The TEO module is developed using Python 3.9.0 and based on the PULP optimization package. The packages are needed to run the TEO module are listed below

- **numpy**>=1.16.4
- **pandas**>=0.25.1
- **PuLP**>=1.6.8
- **python**>=3.7.7

The major dependencies and the requirements are described in the 'Requirements.txt' file on GitHub [here](#).

2.5 Detailed Module description

2.5.1 Structure of the TEO module

The code of the TEO module is based on the long-term energy-planning tool OSeMOSYS (Apache License 2.0) [1]. The TEO is built using the python version of OSeMOSYS written in the PULP package of python, which can be accessed at [here](#). The standalone version of the TEO module can be accessed from [here](#).

The code for the TEO is written in PULP, python [2]. The user needs to install python and then the python package PULP to run the TEO. The code is organised in three python files, 'TEO_Model', 'TEO_functions' and 'TEO_running_file'. The 'TEO_Model' file contains the code of the TEO module and all the equations of the optimization model. 'TEO_functions' contains certain pre and post-processing functions that are needed to run the module. 'TEO_running_file' is the executable file of the TEO. The user can specify the input file and desired format of outputs in the 'TEO_running_file'.

The TEO module has been formulated as a linear (mixed-integer) optimisation problem. The objective function is the minimisation of the net present costs of the energy system under analysis, over the time domain of the case. The costs include operational and capital costs. The optimisation is deterministic and assumes perfect foresight and perfect competition. In the TEO module, the user defines the list of existing and potential future technologies as well as the energy vectors flowing between them. Based on the level of temperature, for example, Heat Exchanger (HE), Heat Pump (HP), Waste Heat Recovery (WHR) Boiler and thermal energy storage. The model will then choose the least-cost mix of technologies needed to match the source and sink based on defined constraints of capacity, costs etc.

The model is structured into SETS, PARAMETERS and VARIABLES. The model contains equations written based on a linear/mixed-integer linear program. The SETS, PARAMETERS and VARIABLES are described below. The optimisation is dynamic, over several years. Each year is divided into several time steps. Both the years and the time steps can be defined by the user. The time-domain can span over decades and the time resolution can be up to hourly. For a large model i.e. a model with several sources and sinks and amounting to more than 50 technologies, optimization at an hourly resolution might take several hours to a day and might need a large memory space for example, up to 128 or 156 GB of RAM.

2.5.1.1 SETS

The 'sets' define the physical structure of a model, usually independent of the specific scenarios which will be run. They define the time domain and time split, the spatial coverage, the technologies and energy vectors to be considered, etc. For instance, when a variable is defined as a function of the set 'YEAR' it will be indicated as **variablename[y]** at it will be computed for every year listed in the set. The sets of the TEO are presented in Table 1 [3].



Table 1: SETS (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

Name	Description	Index	Obtained from
YEAR	It represents the time frame of the model, it contains all the years to be considered in the study.	y	User
TECHNOLOGY	It includes any element of the energy system that changes a commodity from one form to another, uses it or supplies it. All system components are set up as a 'technology' in OSeMOSYS. As the model is an abstraction, the modeller is free to interpret the role of technology at will, where relevant. It may for example represent a single real technology (such as a power plant) or can represent a heavily aggregated collection of technologies (such as the stock of several million light bulbs), or may even simply be a 'dummy technology', perhaps used for accounting purposes.	t	CF module
TIMESLICE	It represents the time split of each modelled year, therefore the time resolution of the model. Common to several energy systems modelling tools (incl. MESSAGE / MARKAL / TIMES), the annual demand is 'sliced' into representative fractions of the year. It is necessary to assess times of the year when demand is high separately from times when demand is low, for fuels that are expensive to store. To reduce the computation time, these 'slices' are often grouped. Thus, the annual demand may be split into aggregate seasons where demand levels are similar (such as 'summer, winter and intermediate'). Those seasons may be subdivided into aggregate 'day types' (such as workdays and weekends), and the day further subdivided (such as into day and night) depending on the level of demand.	l	User
FUEL	It includes any energy sector, energy service or proxies entering or exiting technologies. These can be aggregate groups, individual	f	CF module



	flows or artificially separated, depending on the requirements of the analysis.		
EMISSION	It includes any kind of emission potentially deriving from the operation of the defined technologies. Typical examples would include atmospheric emissions of greenhouse gasses, such as CO ₂ .	e	CF Module
MODE_OF_OPERATION	It defines the number of modes of operation that the technologies can have. If technology can have various input or output fuels and it can choose the mix (i.e. any linear combination) of these input or output fuels, each mix can be accounted as a separate mode of operation. For example, a CHP plant may produce heat in one mode of operation and electricity in another.	m	User
REGION	It sets the regions to be modelled, e.g. different countries. For each of them, the supply-demand balances for all the energy vectors are ensured, including trades with other regions. On some occasions, it might be computationally more convenient to model different countries within the same region and differentiate them simply by creating ad hoc fuels and technologies for each of them.	r	User
STORAGE	It includes storage facilities in the optimization model	s	User

2.5.1.2 PARAMETERS

The parameters are the user-defined numerical inputs to the model. While usually the structure of a model, therefore the sets, remains fixed across scenarios, it is common practice to change the values of some parameters when running different scenarios and/or sensitivity analyses. As will be clear in the following, each parameter is a function of the elements in one or more sets. For instance, **CapitalCost[r, t, and y]** indicates that the capital cost is a function of the region (r), the technology (t) and the year (y). A list and a brief description of the parameters declared in the master version of OSeMOSYS are given in Table 2[3].



Table 2: Parameters (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

Name	Description	Obtained from	Units	Data type
YearSplit[l,y]	Duration of a modelled time slice expressed as a fraction of the year. The sum of each entry over one modelled year should equal 1.	Calculated within the module based on the number of TimeSlices)	N/A	Float
DiscountRateTech[r,t]	A discount rate is the rate of return used to discount future cash flows back to their present value. Technology specific value for the discount rate, expressed in decimals (e.g. 0.05)	Knowledge Base	N/A	Float
DiscountRateSto[r,t]	A discount rate is the rate of return used to discount future cash flows back to their present value. Storage specific value for the discount rate, expressed in decimals (e.g. 0.05)	Knowledge Base	N/A	Float

DepreciationMethod[r]	Binary parameter defining the type of depreciation to be applied. It has value 1 for sinking fund depreciation, and value 2 for straight-line depreciation.	Knowledge Base	N/A	Float
SpecifiedAnnualDemand[r,f,y]	Total specified demand for the year, linked to a specific 'time of use' during the year.	CF module	kWh	Float



SpecifiedDemandProfile [r,f,l,y]	Annual fraction of energy service or commodity demand that is required in each time slice. For each year, all the defined SpecifiedDemandPr ofile input values should sum up to 1.	CF module	N/A	Array of float
CapacityToActivityUnit[r, t]	Conversion factor relating the energy that would be produced when one unit of capacity is fully used in one year.	User	N/A	Float

CapacityFactor[r,t,l,y]	Capacity available per each TimeSlice is expressed as a fraction of the total installed capacity, with values ranging from 0 to 1. It gives the possibility to account for forced outages.	CF module	N/A	Array of float
-------------------------	--	-----------	-----	----------------



AvailabilityFactor[r,t,y]	The maximum time technology can run in the whole year, as a fraction of the year ranging from 0 to 1. It gives the possibility to account for planned outages.	User	N/A	Float
OperationalLife[r,t]	The useful lifetime of technology is expressed in years.	Knowledge Base	Years	Float
ResidualCapacity[r,t,y]	Remained capacity available from before the modelling period.	User	kW	Float



InputActivityRatio[r,t,f,m,y]	Rate of use of a commodity by technology, as a ratio of the rate of activity.	CF module	N/A	Float
OutputActivityRatio[r,t,f,m,y]	Rate of commodity output from technology, as a ratio of the rate of activity.	CF module	N/A	Float
OuputModeofOpeartion[r,t,m,y]	Binary parameter indicating the mode of operation in which technology has an output activity ratio	Knowledge Base	N/A	Float
CapitalCost[r,t,y]	Capital investment cost of technology, per unit of capacity.	CF module	€/kW	Float

VariableCost[r,t,m,y]	Cost of technology for a given mode of operation (Variable O&M cost), per unit of activity.	CF module	€/kWh	Float
FixedCost[r,t,y]	Fixed O&M cost of technology, per unit of capacity.	CF module	€/kW	Float
TechnologyToStorage[r,t,s,m]	Binary parameter linking technology to the storage facility it charges. It has a value of 1 if the technology and the storage facility are linked, 0 otherwise.	User	N/A	Float



TechnologyFromStorage[r,t,s,m]	Binary parameter linking a storage facility to the technology it feeds. It has a value of 1 if the technology and the storage facility are linked, 0 otherwise.	User	N/A	Float
StorageLevelStart[r,s]	Level of storage at the beginning of the first modelled year, in units of activity.	User	kWh	Float
StorageMaxChargeRate[r,s]	Maximum charging rate for the storage, in units of activity per year.	User	kWh	Float



StorageMaxDischargeRate[r,s]	The maximum discharging rate for the storage in units of activity per year.	User /KB?	kWh	Float
MinStorageCharge[r,s,y]	It sets a lower bound to the amount of energy stored, as a fraction of the maximum, with a number ranging between 0 and 1. The storage facility cannot be emptied below this level.	User /KB?	N/A	Float
OperationalLifeStorage[r,s]	Useful lifetime of the storage facility.	User /KB?	Years	Float



CapitalCostStorage[r,s,y]	Investment costs of storage additions, defined per unit of storage capacity.	User/KB?	€/kW	Float
ResidualStorageCapacity[r,s,y]	Exogenously defined storage capacities.	User	kW	Float
StorageUvalue[r,s]	Heat transfer coefficient of the thermal energy storage tank.	User /KB?	kJ/kg K	Float
StorageFlowTemperature[r,s]	The temperature of water inflow into thermal energy storage	User	°C	Float



StorageReturnTemperature[r,s]	The return water temperature in the heating grid where the thermal energy storage is connected	User	°C	Float
StorageAmbientTemperature[r,s]	The ambient temperature of the locations where the thermal energy storage is located.	User	°C	Float



StorageL2D[r,s]	The binary parameter indicates the length to diameter ratio of the thermal energy storage tank. Value is 0 if the L2D is 2 and is 1 if the L2D is 4.	User /KB?	N/A	Float
StorageTagheating[r,s]	Binary parameter indicating whether the thermal energy storage is connected to the district heating network. 1 if it is connected and 0 if it is not.	User	N/A	Float



<p>TotalAnnualMaxCapacity[r,t,y]</p>	<p>Total maximum existing (residual plus cumulatively installed) capacity allowed for technology in a specified year.</p>	<p>CF module</p>	<p>kW</p>	<p>Float</p>
<p>TotalAnnualMinCapacity[r,t,y]</p>	<p>Total minimum existing (residual plus cumulatively installed) capacity allowed for technology in a specified year.</p>	<p>User</p>	<p>kW</p>	<p>Float</p>
<p>TotalAnnualMaxCapacityInvestment[r,t,y]</p>	<p>The maximum capacity of technology; expressed in power units.</p>	<p>User</p>	<p>kW</p>	<p>Float</p>

TotalAnnualMinCapacityInvestment[r,t,y]	The minimum capacity of technology; expressed in power units.	User	kW	Float
TotalTechnologyAnnualActivityUpperLimit[r,t,y]	The total maximum level of activity allowed for technology in one year.	User	kWh	Float
TotalTechnologyAnnualActivityLowerLimit[r,t,y]	The total minimum level of activity allowed for technology in one year.	User	kWh	Float

TotalTechnologyModelPeriodActivityUpperLimit[r,t]	The total maximum level of activity allowed for technology in the entire modelled period.	User	kWh	Float
TotalTechnologyModelPeriodActivityLowerLimit[r,t]	The total minimum level of activity allowed for technology in the entire modelled period.	User	kWh	Float
EmissionActivityRatio[r,t,e,m,y]	The emission factor of a technology per unit of activity, per mode of operation.	CF module	Kg / kWh	Float
EmissionsPenalty[r,e,y]	Penalty per unit of emission.	Knowledge Base	€/Kg	Float



AnnualEmissionLimit[r,e,y]	The annual upper limit for a specific emission generated in the whole modelled region.	User	Kg	Float
----------------------------	--	------	----	-------

2.5.1.3 VARIABLES

The variables are the outputs computed by the code. As much as the parameters, also the variables are functions of the elements in one or more sets. In Table 3, a list and a brief description of all the variables computed by the code of OSeMOSYS (in its full version) are given. As will be explained next in this manual, a shortened version of OSeMOSYS has been created, to improve the computational capability at the expense of the readability of the code. In such a version, only some of the variables here listed are computed. When reasonable, the domain of several variables has been constrained to be positive, to decrease the size of the solution space and therefore the computational effort. The list of variables is shown in Table 3 [3].



Table 3: Variables (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

Name	Description	Units	Datatype
RateOfDemand[r,l,f,y]>=0	Intermediate variable. It represents the energy that would be demanded in a one-time slice l if the latter lasted the whole year. It is a function of the parameters SpecifiedAnnualDemand and SpecifiedDemandProfile. Energy (per year)	kWh/year	Dictionary
Demand[r,l,f,y]>=0	Demand for one fuel in one time slice.	kWh	Dictionary
RateOfStorageCharge[r,s,ls,ld,lh,y]	Intermediate variable. It represents the commodity that would be charged to the storage facility s in a one-time slice l if the latter lasted the whole year. It is a function of the RateOfActivity and the parameter TechnologyToStorage. Energy (per year)	kWh/year	Dictionary
RateOfStorageDischarge[r,s,ls,ld,lh,y]	Intermediate variable. It represents the commodity that would be discharged from storage facilities in one timeslice if the latter lasted the whole year. It is a function of the RateOfActivity and the parameter TechnologyFromStorage.	kWh/year	Dictionary
NetChargeWithinYear[r,s,ls,ld,lh,y]	Net quantity of commodity charged to storage facility s in year y. It is a function of	kWh/year	Dictionary



	the <code>RateOfStorageCharge</code> and the <code>RateOfStorageDischarge</code> and it can be negative.		
<code>NetChargeWithinDay[r,s,ls,ld,lh,y]</code>	Net quantity of commodity charged to storage facility <code>s</code> in daytype <code>ld</code> . It is a function of the <code>RateOfStorageCharge</code> and the <code>RateOfStorageDischarge</code> and can be negative.	kWh	Dictionary
<code>StorageLevelYearStart[r,s,y]>=0</code>	Level of stored commodity in storage facility <code>s</code> in the first time step of year <code>y</code> .	kWh	Dictionary
<code>StorageLevelYearFinish[r,s,y]>=0</code>	Level of stored commodity in storage facility <code>s</code> in the last time step of year <code>y</code> .	kWh	Dictionary
<code>StorageLevelSeasonStart[r,s,ls,y]>=0</code>	Level of stored commodity in storage facility <code>s</code> in the first time step of season <code>ls</code> .	kWh	Dictionary
<code>StorageLevelDayTypeStart[r,s,ls,ld,y]>=0</code>	Level of stored commodity in storage facility <code>s</code> in the first time step of daytype <code>ld</code> .	kWh	Dictionary
<code>StorageLevelDayTypeFinish[r,s,ls,ld,y]>=0</code>	Level of stored commodity in storage facility <code>s</code> in the last time step of daytype <code>ld</code> .	kWh	Dictionary
<code>StorageLowerLimit[r,s,y]>=0</code>	Minimum allowed level of stored commodity in storage facility <code>s</code> , as a function of the storage capacity and the user-defined <code>MinStorageCharge</code> ratio.	kWh	Dictionary
<code>StorageUpperLimit[r,s,y]>=0</code>	Maximum allowed level of stored commodity in storage facility <code>s</code> . It corresponds to the total existing capacity of storage facility <code>s</code> (summing	kWh	Dictionary

	newly installed and pre-existing capacities).		
AccumulatedNewStorageCapacity[r,s,y] ≥ 0	The cumulative capacity of newly installed storage from the beginning of the time domain to year y.	kWh	Dictionary
NewStorageCapacity[r,s,y] ≥ 0	Capacity of newly installed storage in year y.	kWh	Dictionary
StorageLevelTimesliceStart[r,s,l,y]	Energy stored in storage in timeslice l.	kWh	Dictionary
StorageLosses[r,s,l,y]	Thermal energy losses from the storage in timeslice l.	kWh	Dictionary
CapitalInvestmentStorage[r,s,y] ≥ 0	Undiscounted investment in new capacity for storage facility s. Derived from the NewStorageCapacity and the parameter CapitalCostStorage.	€	Dictionary
DiscountedCapitalInvestmentStorage[r,s,y] ≥ 0	Investment in new capacity for storage facility 's' discounted through the parameter DiscountRate.	€	Dictionary
SalvageValueStorage[r,s,y] ≥ 0	Salvage value of storage facility s in year y, as a function of the parameters OperationalLifeStorage and DepreciationMethod.	€	Dictionary
DiscountedSalvageValueStorage[r,s,y] ≥ 0	Salvage value of storage facility 's', discounted through the parameter DiscountRate.	€	Dictionary
TotalDiscountedStorageCost[r,s,y] ≥ 0	Difference between the discounted capital investment in new storage	€	Dictionary



	facilities and the salvage value in year y.		
NumberOfNewTechnologyUnits[r,t,y] ≥ 0 , integer	Number of newly installed units of technology t in year y, as a function of the parameter CapacityOfOneTechnology Unit. No unit	N/A	Dictionary
NewCapacity[r,t,y] ≥ 0	Newly installed capacity of technology t in year y.	kW	Dictionary
AccumulatedNewCapacity[r,t,y] ≥ 0	Cumulative newly installed capacity of technology t from the beginning of the time domain to year y.	kW	Dictionary
TotalCapacityAnnual[r,t,y] ≥ 0	Total existing capacity of technology t in year y (sum of cumulative newly installed and pre-existing capacity).	kW	Dictionary
RateOfActivity[r,l,t,m,y] ≥ 0	Intermediate variable. It represents the activity of technology t in one mode of operation and in time slice l, if the latter lasted the whole year. Energy (per year)	kWh/year	Dictionary
RateOfTotalActivity[r,t,l,y] ≥ 0	Sum of the RateOfActivity of a technology over the modes of operation.	kWh/year	Dictionary
TotalTechnologyAnnualActivity[r,t,y] ≥ 0	Total annual activity of technology t.	kWh	Dictionary
TotalAnnualTechnologyActivityByMode[r,t,m,y] ≥ 0	The annual activity of technology t in mode of operation m.	kWh	Dictionary

TotalTechnologyModelPeriodActivity[r,t]	Sum of the TotalTechnologyAnnualActivity over the years of the modelled period.	kWh	Dictionary
RateOfProductionByTechnologyByMode[r,l,t,m,f,y] >=0	Intermediate variable. It represents the quantity of fuel f that technology t would produce in one mode of operation and in time slice l, if the latter lasted the whole year. It is a function of the variable RateOfActivity and the parameter OutputActivityRatio.	kWh/year	Dictionary
RateOfProductionByTechnology[r,l,t,f,y] >=0	Sum of the RateOfProductionByTechnologyByMode over the modes of operation.	kWh/year	Dictionary
ProductionByTechnology[r,l,t,f,y] >=0	Production of fuel f by technology t in time slice l.	kWh	Dictionary
ProductionByTechnologyAnnual[r,t,f,y] >=0	Annual production of fuel f by technology t.	kWh	Dictionary
RateOfProduction[r,l,f,y] >=0	Sum of the RateOfProductionByTechnology over all the technologies.	kWh/year	Dictionary
Production[r,l,f,y] >=0	Total production of fuel f in time slice l. It is the sum of the ProductionByTechnology over all technologies.	kWh	Dictionary
RateOfUseByTechnologyByMode[r,l,t,m,f,y] >=0	Intermediate variable. It represents the quantity of fuel f that technology t would use in one mode of operation and in time slice l, if the latter lasted the whole year. It is the function of the variable RateOfActivity and the	kWh/year	Dictionary

	parameter InputActivityRatio.		
RateOfUseByTechnology[r,l,t,f,y] >=0	Sum of the RateOfUseByTechnologyByMode over the modes of operation.	kWh/year	Dictionary
UseByTechnologyAnnual[r,t,f,y] >=0	Annual use of fuel f by technology t.	kWh	Dictionary
UseByTechnology[r,l,t,f,y] >=0	Use of fuel f by technology t in time slice l.	kWh	Dictionary
Use[r,l,f,y] >=0	Total use of fuel f in time slice l. It is the sum of the UseByTechnology over all technologies.	kWh	Dictionary
Trade[r,rr,l,f,y]	Quantity of fuel f traded between region r and rr in time slice l.	kWh	Dictionary
TradeAnnual[r,rr,f,y]	Annual quantity of fuel f traded between region r and rr. It is the sum of the variable Trade over all the time slices.	kWh	Dictionary
ProductionAnnual[r,f,y] >=0	Total annual production of fuel f. It is the sum of the variable Production over all technologies.	kWh	Dictionary
UseAnnual[r,f,y] >=0	Total annual use of fuel f. It is the sum of the variable Use over all technologies.	kWh	Dictionary
CapitalInvestment[r,t,y] >=0	Undiscounted investment in new capacity of technology t. It is a function of the NewCapacity and the parameter CapitalCost. Monetary units	€	Dictionary

DiscountedCapitalInvestment[r,t,y] >=0	Investment in new capacity of technology t, discounted through the parameter DiscountRate.	€	Dictionary
SalvageValue[r,t,y] >=0	Salvage value of technology t in year y, as a function of the parameters OperationalLife and DepreciationMethod.	€	Dictionary
DiscountedSalvageValue[r,t,y] >=0	Salvage value of technology t, discounted through the parameter DiscountRate.	€	Dictionary
OperatingCost[r,t,y] >=0	Undiscounted sum of the annual variable and fixed operating costs of technology t.	€	Dictionary
DiscountedOperatingCost[r,t,y] >=0	Annual OperatingCost of technology t, discounted through the parameter DiscountRate.	€	Dictionary
AnnualVariableOperatingCost[r,t,y] >=0	Annual variable operating cost of technology t. Derived from the TotalAnnualTechnologyActivityByMode and the parameter VariableCost.	€	Dictionary
AnnualFixedOperatingCost[r,t,y] >=0	Annual fixed operating cost of technology t. Derived from the TotalCapacityAnnual and the parameter FixedCost.	€	Dictionary
TotalDiscountedCostByTechnology[r,t,y] >=0	Difference between the sums of discounted operating cost/capital cost/emission penalties and the salvage value.	€	Dictionary
TotalDiscountedCost[r,y] >=0	Sum of the TotalDiscountedCostByTech	€	Dictionary

	nology over all the technologies.		
ModelPeriodCostByRegion[r] ≥ 0	Sum of the TotalDiscountedCost overall modelled years.	€	Dictionary
TotalCapacityInReserveMargin[r, y] ≥ 0	Total available capacity of the technologies required to provide reserve margin. It is derived from the TotalCapacityAnnual and the parameter ReserveMarginTagTechnology. Energy	kW	Dictionary
DemandNeedingReserveMargin[r, l, y] ≥ 0	Quantity of fuel produced that is assigned to a target of reserve margin. Derived from the RateOfProduction and the parameter ReserveMarginTagFuel.	kWh	Dictionary
TotalREProductionAnnual[r, y]	Annual production by all technologies tagged as renewable in the model. Derived from the ProductionByTechnologyAnnual and the parameter RETagTechnology.	kWh	Dictionary
RETotalProductionOfTargetFuelAnnual[r, y]	Annual production of fuels tagged as renewable in the model. Derived from the RateOfProduction and the parameter RETagFuel.	kWh	Dictionary
AnnualTechnologyEmissionByMode[r, t, e, m, y] ≥ 0	Annual emission of agent e by technology t in mode of operation m. Derived from the RateOfActivity and the parameter EmissionActivityRatio.	Kg	Dictionary

AnnualTechnologyEmission[r,t,e,y] >=0	Sum of the AnnualTechnologyEmission ByMode over the modes of operation.	Kg	Dictionary
AnnualTechnologyEmissionPenaltyByEmission[r,t,e,y] >=0	Undiscounted annual cost of emission e by technology t. It is a function of the AnnualTechnologyEmission and the parameter EmissionPenalty.	€	Dictionary
AnnualTechnologyEmissionsPenalty[r,t,y] >=0	Total undiscounted annual cost of all emissions generated by technology t. Sum of the AnnualTechnologyEmission PenaltyByEmission over all the emitted agents.	€	Dictionary
DiscountedTechnologyEmissionPenalty[r,t,y] >=0	Annual cost of emissions by technology t, discounted through the DiscountRate.	€	Dictionary
AnnualEmissions[r,e,y] >=0	Sum of the AnnualTechnologyEmission over all technologies.	Kg	Dictionary
ModelPeriodEmissions[r,e] >=0	Total system emissions of agent e in the model period, accounting for both the emissions by technologies and the user defined ModelPeriodExogenousEmission.	Kg	Dictionary

2.5.2 Inputs and Outputs

The module obtains inputs from the CF module, GIS module and the user. It provides outputs to be used by the market and business modules. Further, some results of the TEO module must also be used for the visualization. The TEO module needs several inputs from the CF and the user. Some of these inputs, namely, the ‘SETS’ will be used to structure the model while other inputs, ‘Parameters’ will be used to make the calculations within the model.

The user gives inputs through the platform. To improve the user-friendliness of the platform, a detailed description is provided for each user input. Furthermore, it will also



be conveyed to the user if the input is mandatory or not. If the user does not give a value for a no-mandatory input, it will be obtained from the default values stored in the knowledge base. This is further discussed in the next section. A preliminary list of labels for the user inputs is shown in Table 4.

Table 4: User inputs to TEO (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

Label	Description	Mandatory
Region	It sets the regions to be modelled, e.g. different countries, cities, counties etc. For the purpose of this analysis, it is enough to have one region name. For each of them, the supply-demand balances for all the energy vectors are ensured. On some occasions, it might be computationally more convenient to model different countries within the same region and differentiate them simply by creating ad hoc fuels and technologies for each of them.	TRUE
Emission	It includes any kind of emission potentially deriving from the operation of the defined technologies. Typical examples would include atmospheric emissions of greenhouse gasses, such as CO ₂ . The user must fill in 'co2' as a mandatory entry. Other entries are also allowed	TRUE
Time resolution	It represents the time steps of each modelled year, therefore the time resolution of the model.	TRUE
Time period	It represents the period of the model; it contains all the years to be considered in the analysis.	TRUE
Mode of operation	It defines the number of modes of operation that the technologies can have. If a technology can have various input or output fuels and it can choose the mix (i.e. any linear combination) of these input or output fuels, each mix can be accounted as a separate mode of operation. The user must input at least 1 mode of operation. There must be two modes of operation if storage is used in the model	TRUE

Storage	It includes storage facilities in the model.	FALSE
Availability factor	Maximum time a technology can run in the whole year, as a fraction of the year ranging from 0 to 1. It gives the possibility to account for planned outages.	FALSE
Technology discount rate	Technology specific value for the discount rate, expressed in decimals (e.g. 0.04).	FALSE
Capacity to Activity ratio	Conversion factor relating the energy that would be produced when one unit of capacity is fully used in one year.	FALSE
Residual capacity	Remained capacity available from before the modelling period.	FALSE
Maximum annual capacity addition	Maximum capacity of a technology that can be added in a year, expressed in power units.	FALSE
Minimum capacity	Total minimum existing (residual plus cumulatively installed) capacity allowed for a technology in a year.	FALSE
Minimum annual capacity addition	Minimum capacity of a technology that must be added in a year, expressed in power units.	FALSE
Minimum annual heat generation	Total minimum heat generation allowed for a technology in one year.	FALSE
Maximum annual heat generation	Total maximum heat generation allowed for a technology in one year.	FALSE
Minimum model period heat generation	Total minimum heat generation allowed for a technology in the entire modelled period.	FALSE
Maximum model period heat generation	Total maximum heat generation allowed for a technology in the entire modelled period.	FALSE
Storage capital cost	Investment costs of storage additions, defined per unit of storage capacity.	FALSE
Storage discount rate	Storage specific value for the discount rate, expressed in decimals	FALSE
Storage operational life	Useful lifetime of the storage facility.	FALSE

Storage maximum charge rate	Maximum charging rate for the storage	FALSE
Storage maximum discharge rate	Maximum discharging rate for the storage	FALSE
Storage length to diameter ratio	Binary parameter which indicates the length to diameter ratio of the thermal energy storage tank. Value is 0 if the L2D is 2 and is 1 if the L2D is 4.	FALSE
Storage heating tag	Binary parameter indicating whether the thermal energy storage is connected to the district heating network. 1 if it is connected and 0 is if is not.	FALSE
Storage cooling tag	Binary parameter indicating whether the thermal energy storage is connected to the district cooling network. 1 if it is connected and 0 is if is not.	FALSE
Storage hot water return temperature	The return water temperature in the heating grid where the thermal energy storage is connected.	FALSE
Storage hot water supply temperature	The temperature of water inflow into thermal energy storage.	FALSE
Average ambient temperature of the region	The average ambient temperature of the locations where the thermal energy storage is located.	FALSE
Residual storage capacity	Exogenously defined storage capacities at the start of the modeling period	FALSE
Maximum storage capacity	Maximum allowed capacity of each storage in a year	FALSE
Starting level of storage	Level of storage at the beginning of first modelled year	FALSE
Heat transfer co-efficient of the thermal storage	Heat transfer co-efficient of the thermal energy storage tank.	FALSE
Annual emission limit	Annual upper limit for a specific emission generated in the entire modelled region.	FALSE

Technology charging storage	Binary parameter linking a storage facility to the technology it feeds. It has value 1 if the technology and the storage facility are linked, 0 otherwise.	FALSE
Technology discharge getting from storage	Binary parameter linking a technology to the storage facility it charges. It has value 1 if the technology and the storage facility are linked, 0 otherwise.	FALSE

The inputs from the CF and the GIS modules along with a detailed description are shown in Table 5.

Table 5

Table 5: Inputs from other modules to TEO (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

Input	Module	Description
TECHNOLOGY	CF module	The set of all conversion technologies that have been deemed feasible by the CF: This will include the sources (heat generation technologies), heat conversion technologies (such as heat exchanger) and temperature boosting technologies such as heat pumps and boilers. This set will also contain the district heating network.
FUEL	CF module	It includes the set of all fuels that will be used by each technology in the 'TECHNOLOGY' set. Both input and output fuels of all technologies will be included
output	CF module	This is input for each technology in the 'TECHNOLOGY' set and identifies the

		'OutputActivityRatio' of the technology
turnkey	CF module	This is input for each technology in the 'TECHNOLOGY' set and identifies the 'CapitalCost' of the technology
om_fix	CF module	This is input for each technology in the 'TECHNOLOGY' set and identifies the 'FixedCost' of the technology
om_var	CF module	This is input for each technology in the 'TECHNOLOGY' set and identifies the 'VariableCost' of the technology
emissions_factor	CF module	This is input for each technology in the 'TECHNOLOGY' set and identifies the 'EmissionActivityRatio' of the technology
input	CF module	This is input for each technology in the 'TECHNOLOGY' set and identifies the 'InputActivityRatio' of the technology
specified_annual_demand_cf	CF module	This includes the annual demand in energy units for each stream in every sink.
specified_demand_profile_cf	CF module	This includes the hourly demand profile of each stream in every sink.
capacity_factor_cf	CF module	This includes the hourly heat generation profile for each stream in every source.
losses_in_kw	GIS module	The power losses in the heating network are determined by the GIS,



		These losses are then added to the sink demand in each hour using this parameter.
cost_in_kw	GIS module	This identifies the capital 'cost of the heat network per capacity unit' determined by the GIS module.

2.5.3 Contributions and requirements for the knowledge base

The TEO is linked to the knowledge base for the module to provide the default values that are needed to run the model. All the parameters in the TEO are necessary for the model to be run. However, the user might not have inputs for all the parameters for each technology. Thus, a default value is used if the user is unable to provide values for the parameters. The default values are stored in the knowledge base as shown in Table 6.

Table 6: Default values in the knowledge base (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

Parameter	Value
YearSplit	0
StorageTagCooling	0
StorageTagHeating	0
AccumulatedAnnualDemand	0
AnnualEmissionLimit	999999999
AnnualExogenousEmission	0
AvailabilityFactor	1
CapacityFactor	1
CapacityOfOneTechnologyUnit	0
CapacityToActivityUnit	8760
CapitalCost	0
CapitalCostStorage	0
ConversionId	0
ConversionLh	0
ConversionLs	0
DaysInDayType	7
DaySplit	0
DepreciationMethod	1
DiscountRateTech	0.05
DiscountRateSto	0.05
EmissionActivityRatio	0



EmissionsPenalty	0
FixedCost	0
InputActivityRatio	0
MinStorageCharge	0
ModelPeriodEmissionLimit	999999999
ModelPeriodExogenousEmission	0
OperationalLife	15
OperationalLifeStorage	99
OutputActivityRatio	0
OutputModeofoperation	1
REMinProductionTarget	0
ReserveMargin	1
ReserveMarginTagFuel	0
ReserveMarginTagTechnology	0
ResidualCapacity	0
ResidualStorageCapacity	0
RETagFuel	0
RETagTechnology	0
SpecifiedAnnualDemand	0
SpecifiedDemandProfile	0
StorageMaxCapacity	999999999
StorageMaxChargeRate	999999999
StorageMaxDischargeRate	999999999
StorageLevelStart	0
StorageUvalue	0.22
StorageFlowTemperature	80
StorageReturnTemperature	50
StorageAmbientTemperature	15
StorageL2D	0
TechnologyFromStorage	0
TechnologyToStorage	0
TechWithCapacityNeededToMeetPeakTS	0
TotalAnnualMaxCapacity	999999999
TotalAnnualMaxCapacityInvestment	999999999
TotalAnnualMinCapacity	0
TotalAnnualMinCapacityInvestment	0
TotalTechnologyAnnualActivityLowerLimit	0
TotalTechnologyAnnualActivityUpperLimit	999999999
TotalTechnologyModelPeriodActivityLowerLimit	0
TotalTechnologyModelPeriodActivityUpperLimit	999999999
TradeRoute	0
VariableCost	0
GIS_Losses	0



The default values are assigned in a manner such that they do not constrain the optimization model. For example, the ‘LowerLimit’ highlights the minimum allowed value in the model. Thus, the default value is assigned as zero. Similarly, the ‘UpperLimits’ are assigned to be very high values such as 99999999, so that they do not interfere with the solution.

2.5.4 Functioning of the TEO module

The TEO module is a techno-economic optimization model that determines the least-cost investment and operation of the system. The module is built based on the long-term energy system optimization tool OSeMOSYS. The model optimizes the capacities and the operation of the heat generation and conversion technologies while seeking to minimize the overall net present costs of the energy system over a time horizon relevant for investments (i.e. years or decades). The optimization algorithm is formulated as a linear program using the PULP package in python. The module uses the inputs specified in Table 1 and Table 2 to formulate a linear program based model.

The module considers the temporal availability of excess heat and the heat demand to determine the optimal capacities and the operation of the heat generation and conversion technologies to maintain the energy balance as shown in Equation 1.

$$Production(r, f, l, y) = Use(r, f, l, y) + Demand(r, f, l, y)$$

Equation 1: Energy balance

Furthermore, the module also uses constraints to ensure that the capacity adequacy is respected, i.e. the heat generation in each time step is less than the installed capacity of heat generation. Here, the Availability Factor represents the maximum time a technology can run in the whole year, as a fraction of the year ranging from 0 to 1. It gives the possibility to account for planned outages and the Capacity Factor represents capacity available per each TimeSlice expressed as a fraction of the total installed capacity, with values ranging from 0 to 1. It gives the possibility to account for forced outages.

$$Production(r, t, l, y) \leq Capacity(r, t, l, y) * CapacityFactor(r, t, l, y) * AvailabilityFactor(r, t, y)$$

Equation 2: Capacity Adequacy

The maximum capacity for each technology in each year ‘x’ can be constrained by using the parameter TotalAnnualMaxCapacity and TotalAnnualMinCapacity as shown in Equation 3 and Equation 4. These parameters are described in Table 2.

$$\sum_{y=1 \text{ to } x} InstalledCapacity(r, t, x) \leq TotalAnnualMaxCapacity(r, t, x)$$

Equation 3: Maximum allowed capacity in each year



$$\sum_{y=\text{startyear to } x} \text{InstalledCapacity}(r, t, x) \geq \text{TotalAnnualMinCapacity}(r, t, x)$$

Equation 4: Maximum allowed capacity in each year

The annual production from each technology is calculated as shown in Equation 5.

$$\text{AnnualProduction}(r, t, y) = \sum_l \text{Production}(r, l, t, y)$$

Equation 5: Annual Production

Furthermore, the production from each technology can be constrained in each year and for the model period as shown in Equation 6 to Equation 9.

$$\text{AnnualProduction}(r, t, y) \geq \text{AnnualActivityLowerLimit}(r, t, y)$$

Equation 6: Maximum allowed annual production

$$\text{AnnualProduction}(r, t, y) \leq \text{AnnualActivityUpperLimit}(r, t, y)$$

Equation 7: Minimum allowed annual production

$$\sum_y \text{AnnualProduction}(r, t, y) \leq \text{ModelPeriodActivityUpperLimit}(r, t, y)$$

Equation 8: Maximum allowed model period production

$$\sum_y \text{AnnualProduction}(r, t, y) \geq \text{ModelPeriodActivityLowerLimit}(r, t, y)$$

Equation 9: Minimum allowed model period production

The model considers storage as an intermediary between two technologies thus transferring energy between the two technologies and simultaneously storing energy. Consider two technologies T1 and T2 and a storage STO1. The storage equations in the TEO would be represented as shown in Equation 10. Here, 'Production' indicates the production of a fuel from the first technology 'T1' while 'Use' indicates the use of the fuel to the next technology 'T2'. Thus, using the storage the fuel is stored and dispatched from the storage when needed by the second technology.

$$\begin{aligned}
 & Production(r, T1, l, y) * TechnologyToStorage(r, T1, ST01, y) \\
 & = StorageLevel (ST01, l, y) - StorageLevel (ST01, l - 1, y) \\
 & - StorageLosses(ST01, l, y) + Use(r, T2, l, y) \\
 & * TechnologyFromStorage(r, T2, ST01, y)
 \end{aligned}$$

Equation 10: Storage operation

The capacity of the storage is also accounted for and limited as shown in Equation 11 and Equation 12.

$$StorageCapacity(r, s, y) \geq StorageLevel (s, l, y)$$

Equation 11: Storage Capacity

$$StorageMaxCapacity(r, s, y) \geq StorageCapacity(r, s, y)$$

Equation 12: Maximum allowed storage capacity

The model also accounts for the emissions, constrains the limits, and calculates the emission penalties as shown in Equation 13 to Equation 15.

$$AnnualTechnologyEmission(r, t, e, y) \leq StorageLevel (ST01, l, y)$$

Equation 13: Emissions accounting

$$\sum_y AnnualTechnologyEmission(r, t, e, y) \leq AnnualEmissionLimit(r, e, y)$$

Equation 14: Emission constraints

$$AnnualEmissionPenalty(r, t, e, y) \leq AnnualTechnologyEmission(r, t, e, y) * EmissionPenalty(r, e, y)$$

Equation 15: Annual emission penalty

The model minimizes the total costs of the energy system. Thus, it also calculated different parts of the total cost as shown in Equation 16 to Equation 20.

$$TotalCapitalCost = \sum_{t,y} Capitalinvestment(r, t, y) * InstalledCapacity(r, t, y) - Salvage Value (r, t, y)$$

Equation 16: Total capital cost



$$TotalFixedOperatingCost = \sum_{t,y} FixedCost(r, t, y) * InstalledCapacity(r, t, y)$$

$$TotalFixedOperatingCost = \sum_{t,y} FixedCost(r, t, y) * InstalledCapacity(r, t, y)$$

Equation 17: Fixed operating cost

$$TotalVariableOperatingCost = \sum_{t,y} VariableCost(r, t, y) * AnnualProduction(r, t, y)$$

Equation 18: Total variable operating cost

$$TotalEmissionPenalty = \sum_{t,e,y} AnnualEmissionPenalty(r, t, e, y)$$

Equation 19: Total emission penalty

$$Total\ operating\ Cost = TotalVariableOperatingCost + TotalFixedOperatingCost + TotalEmissionPenalty$$

Equation 20: Total fixed operating cost

$$Total\ System\ Cost = TotalCapitalCost + Total\ operating\ Cost$$

The main objective of the optimization model is called the objective function. Here, the objective is to minimize the total system cost. This is shown in Equation 21.

$$Objective = Mimimize(Total\ System\ Cost)$$

Equation 21: Objective function

All the equations in the code of the TEO module can be found in Appendix.

2.5.5 Interaction with other modules

The TEO module is linked to all the modules in the EMB3RS platform. The module obtains inputs from the CF module, has a two-way input-output link with the GIS module and provides inputs for the market and business modules.

Figure 3 shows the inputs to the TEO and the outputs from the TEO to the modules.

2.5.6 Pre-Conditions

For the simulation of the TEO module on the EMB3RS platform, the following conditions:

- **The user must be logged into the platform**
- **The user must have inserted all mandatory input data for TEO**
- **The CF module must have run successfully and generated inputs for the TEO**
- **The GIS module must have run successfully and generated inputs for the TEO**



2.5.7 Input data error handling

The TEO will be capable of picking out errors in the user’s input data before running the module. A simple script is being added, which will look for conflicting inputs from the user which could lead to an infeasible model. For example, if the user inputs the minimum allowed heat generation from technology to be greater than the maximum allowed heat generation, then the TEO would send out a message asking the user to check the inputs for the two parameters. This script is currently being developed and will be integrated into the code as a function that will check the inputs before executing the model.

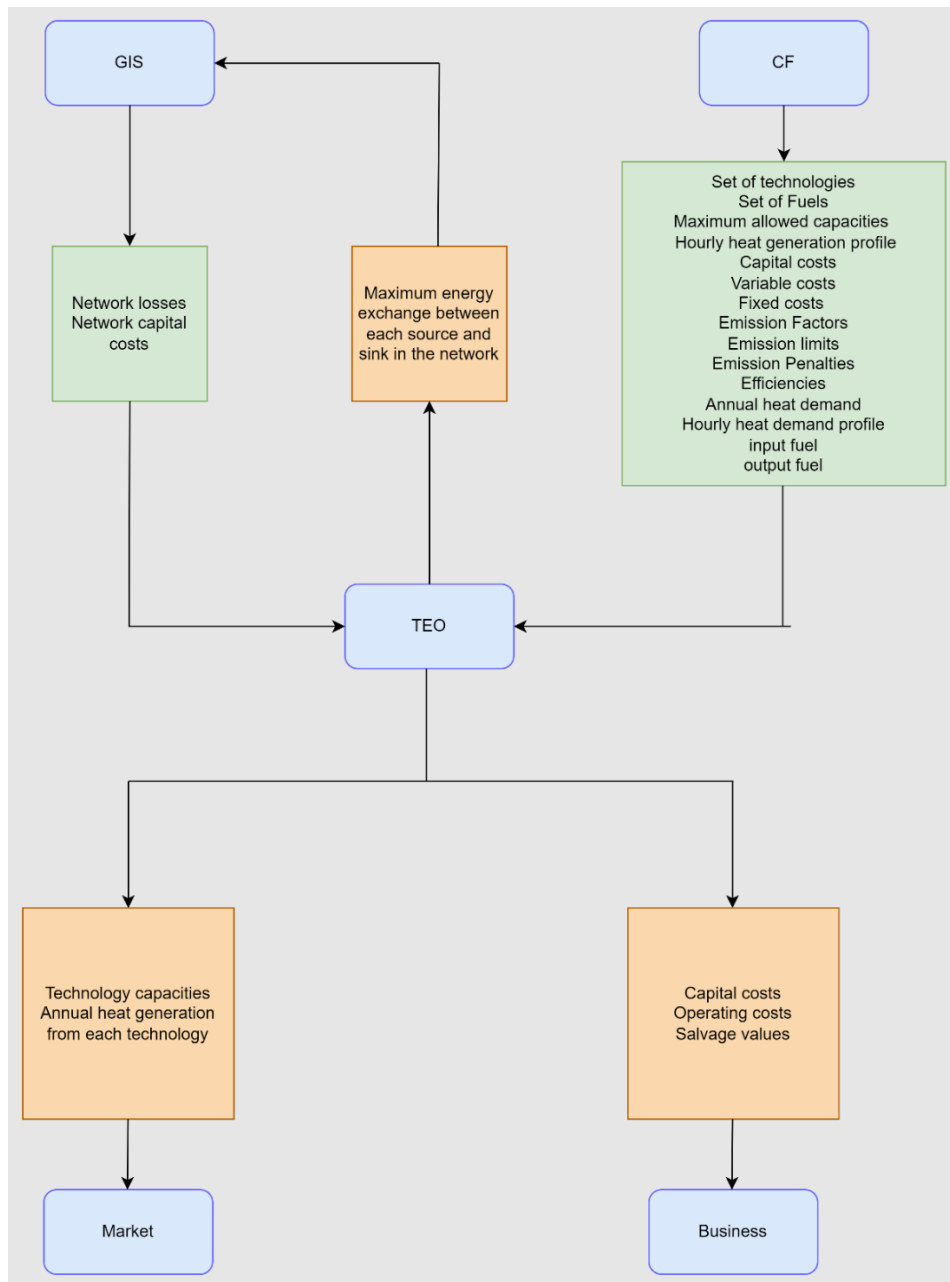


Figure 3: Data transfer between TEO and the other modules (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))



The CF accesses the emission factors and penalties from the regulatory framework. These are then passed to the TEO. TEO module is linked to the GIS and CF module to design the capacities of the heat generation technologies and the heating network. The outputs from TEO are given to the GIS module for the iterative process. Other than this, the output from the TEO module provides some inputs to the market and business module.

The iteration between the TEO, the CF and the GIS modules will take place in two steps. In the first step, the GIS module will provide values for DHC costs and average network losses considering all possible connections in the network. The losses in the network consist both of energy and exergy losses. The CF module calculates the maximum possible heat generation capacities at the sources based on the temperature availability. To compensate for the exergy losses, the technologies on the source side must produce heat at higher temperatures. The CF module will account for the exergy losses and calculate the maximum possible heat generation capacities for all the sources side technologies. In some cases, additional temperature-boosting technologies such as heat pumps would be needed to overcome the exergy losses. The CF will provide the corrected maximum capacities of the technologies to the TEO. TEO will then determine the least cost matching of sources and sinks considering the energy losses in the network. Since the loss values from the GIS are as power losses (in terms of kW), these losses are added to the sink demand in each hour. The TEO then determines the optimal matching of sources and sinks. Based on this, the exchange capacities between the sources and the sinks i.e. the maximum exchange between each source and sinks are calculated. The exchange capacities indicate the hourly heat flow from each source to each sink in the network. The maximum hourly heat exchange between each source and sink will be used by the GIS module to design the pipe capacities in the district heating network. The sources and the sinks in the network and the maximum hourly heat exchange between each source and sink will be fed back to the GIS. In Some cases, the TEO might discard certain sources due to the lack of profitability. This information is also passed on to the GIS module.

In the second step, the GIS will use the calculated maximum exchange capacities to determine the accurate losses and the investments costs of the DHC. These losses are once again fed into the CF to determine the corrected maximum capacities accounting for the exergy losses and forward this information to the TEO. These results are then fed into the TEO to obtain the accurate least-cost mix of technologies. A schematic of the iteration is shown in Figure 4.



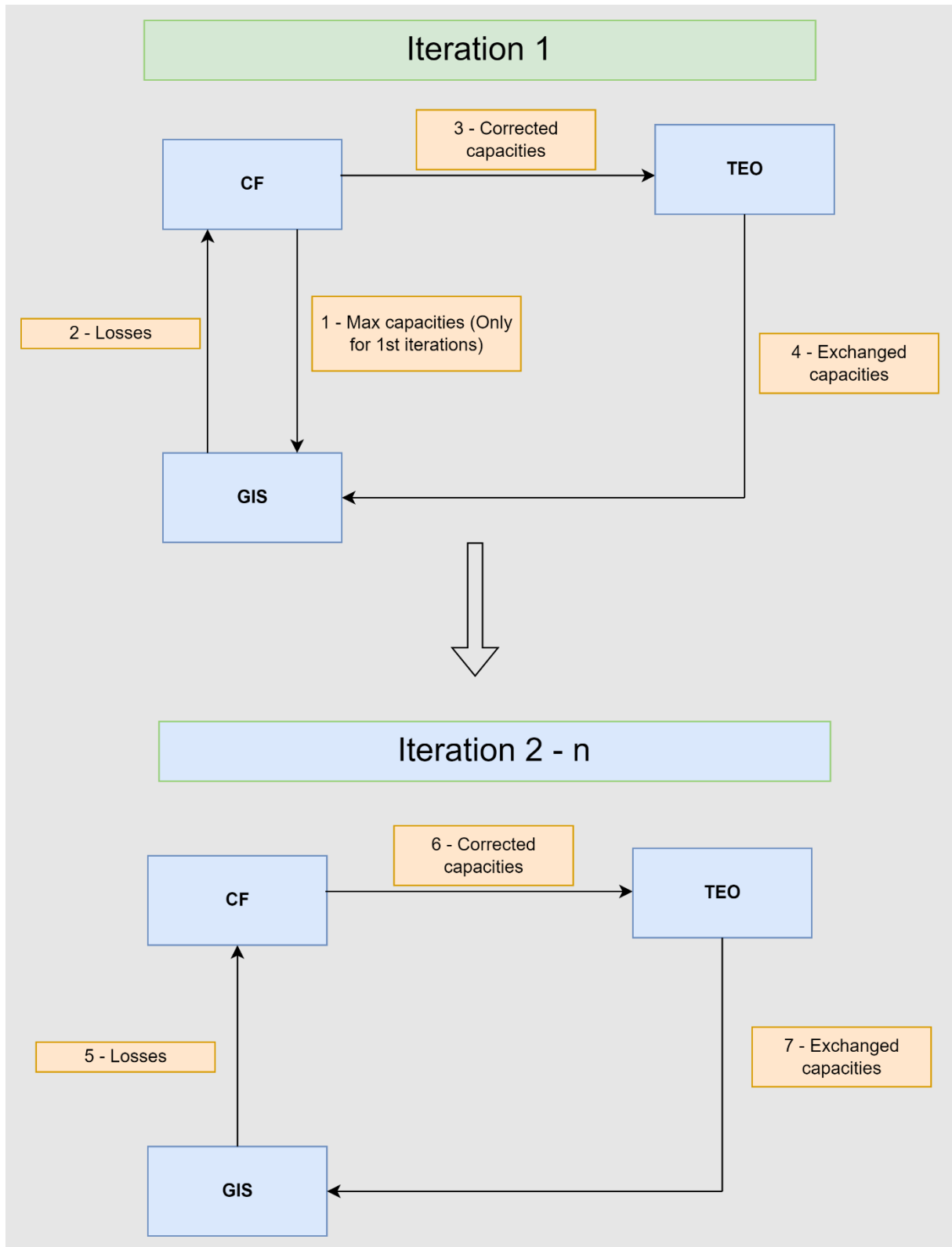


Figure 4: TEO-CF-GIS iterations (Author: Shравan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

In every iteration, the loss value from the GIS is monitored and used as a critical value for stopping the iteration. When the difference in the loss values in two consecutive iterations is below 0.01%, the iterations are stopped.

The market and business modules directly use the TEO results. The market module uses the installed capacities of the different technologies to calculate the dispatch from each technology. The business module uses the capital investment and operation and maintenance costs of the different technologies and storage, and the salvage values to analyse the financial feasibility of the project.

2.6 Reports

The simulation reports and the results produced by the TEO module are presented in this section.

2.6.1 Contribution to the main Simulation report

The main results of the TEO module will be included in its contribution to the main simulation report. The main results of the TEO are the Cost of the total system and the annual capacities and heat generation and consumption of the sources and the sinks respectively. Furthermore, the logs in the TEO can also be used to identify whether the optimal solution is reached or if there is an error in the module. The description of the main results and the formatting instruction are presented in Table 7.

Table 7: Main results from TEO (Author: Shравan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

Variable Name	Description	Formatting instructions
Cost	The overall cost of the system	None – This will be displayed as a number
AccumulatedNewCapacity	The installed capacity of all technologies on the sink and source side for each year in the analysis period.	This can be displayed as a graph over each year as shown in Figure 5.
ProductionByTechnologyAnnual	The yearly heat generation from each technology on the source and the sink sides	This can be displayed for each year as a graph over all technologies as shown in Figure 6.

Source capacities

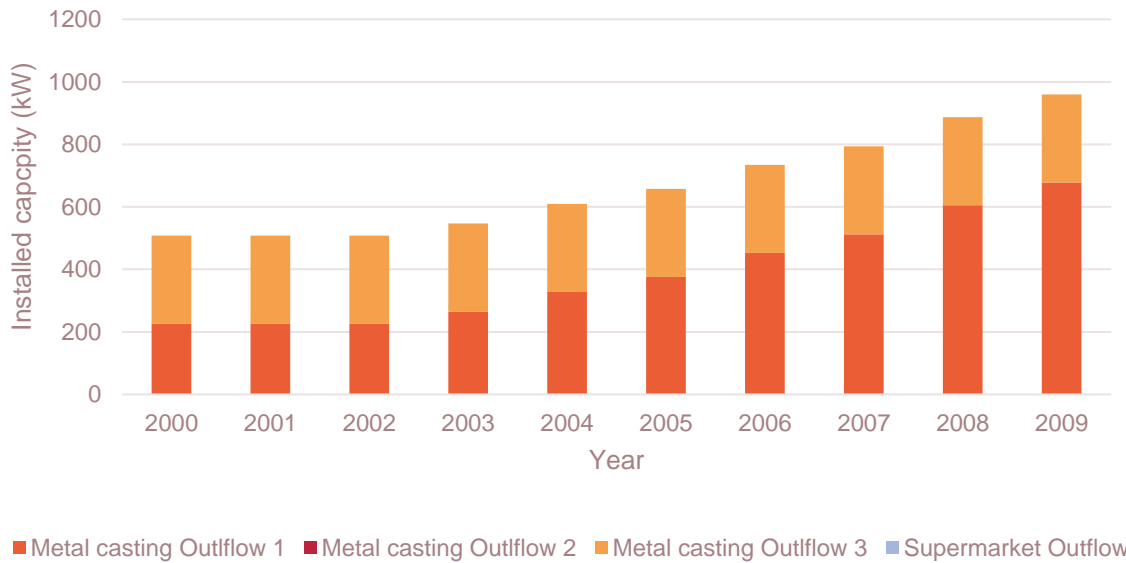


Figure 5: Example output for AccumulatedNewcapacity (Author: Shravan Kumar, licensed under [CC-BY 4.0](#))

Annual heat generation from Sources

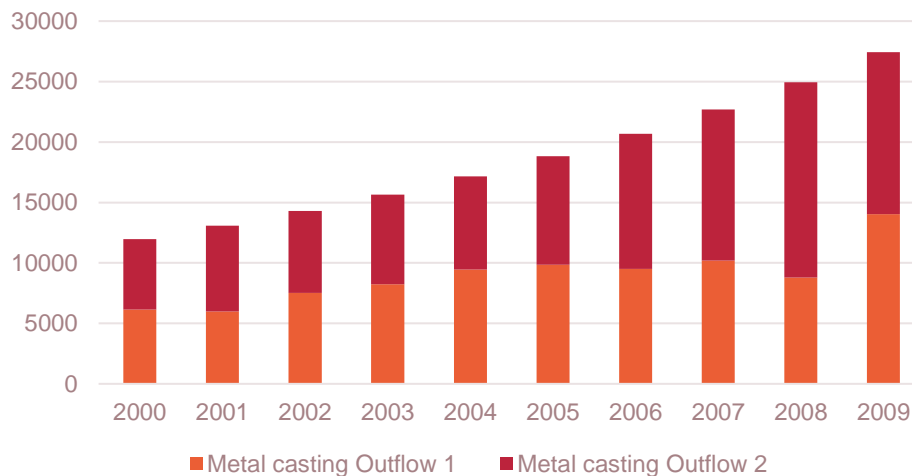


Figure 6: Example for ProductionByTechnologyAnnual (Author: Shravan Kumar, licensed under [CC-BY 4.0](#))

2.6.2 Detailed Module report and configuration

This section of reports will contain all the main results from the TEO. These are listed along with the description and the formatting instruction in

Table 8: Detailed results from the TEO (Author: Shravan Kumar, licensed under [CC-BY 4.0](#))

Variable Name	Description	Formatting instructions	Configuration by the user
AccumulatedNewStorageCapacity	The capacity of the	This can be presented as	The user can select the year



	analysed storage facilities in each year.	a graph for a selected year as shown in Figure 7.	to be used to be visualized
ProductionByTechnology	The intra annual heat generation from each technology	This can be presented as a graph for a selected year as shown in Figure 8	The user can select the year and the technologies to be visualized
StorageLevelTimesliceStart	This indicates the intra annual level of each storage. This can be useful to look at the charging and the discharging of the storage facilities	This can be presented as a graph for a selected year as shown in Figure 9	The user can select the year and the storage facilities to be visualized

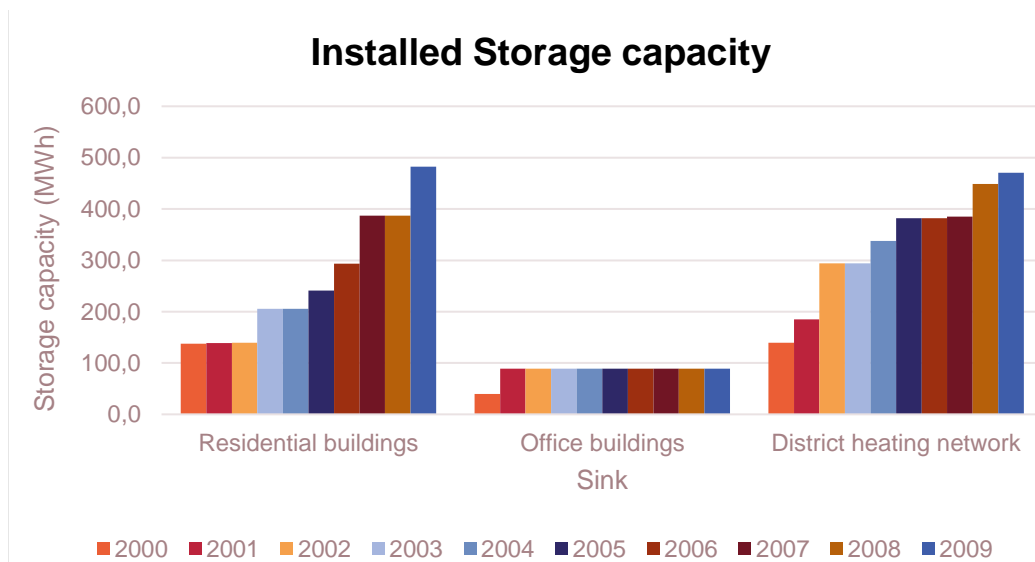


Figure 7: Accumulated New Storage capacity (Author: Shравan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))



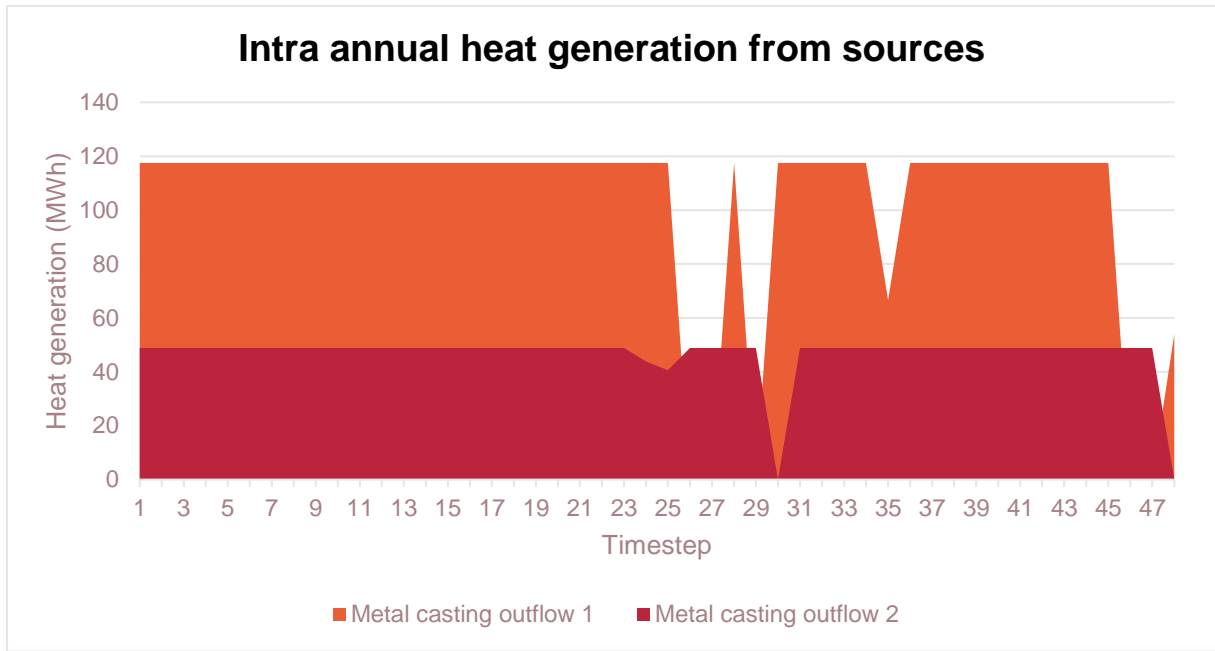


Figure 8: Production by technology (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

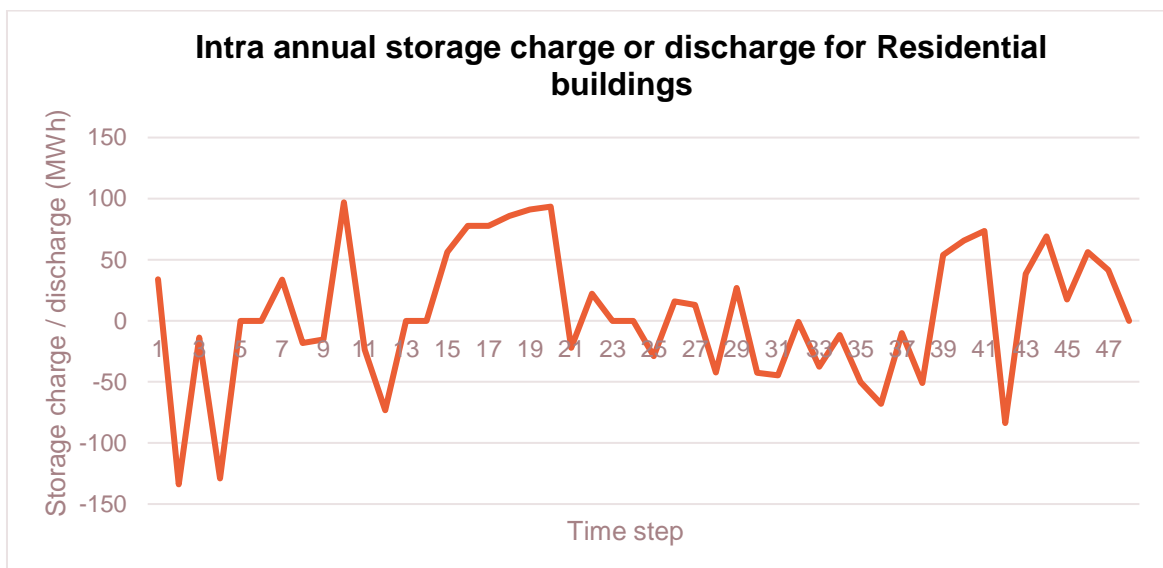


Figure 9: Storage level TimeSlice start (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

3 User Manual

3.1 Introduction to the TEO

The Techno-Economic Optimization (TEO) module identifies the least-cost combinations of technologies for using and conveying excess Heat and Cold (HC) from defined sources to defined sinks. The user (representing the excess heat producer - i.e., source – or a demand point – i.e., sink) wants to evaluate the options of utilizing excess HC generated to meet the heating/cooling demand for one or more known/assumed sinks. The objective of the optimisation is to find the least-cost mix of technologies and match between sources and sinks that satisfies the demands under



constraints dictated by regulation, availability of heat, load profiles, techno-economic characteristics of technologies, investment plans, etc. The mix of technologies may include the District Heating/Cooling Network (DH/CN), technologies to upgrade the Temperature level on the sink or the source side, thermal storage on the sink or the source side, as well as heating alternatives.

The main requirements of the tool for building the techno-economic optimisation module within the EMB3RS framework are:

- High temporal resolution – Daily to hourly
- Low simulation time
- High flexibility and ability to be modified
- Interaction and interoperation with other modules
- Open access

The techno-economic optimisation module should provide the following results:

- The technology mix (in terms of existing and newly installed yearly capacities in terms of energy flows throughout the supply-demand chain)
- Share of each technology in meeting the demand in any time step of the analysis (where the time resolution is defined by the user within certain limits) and throughout the analysis period
- Annual costs (investment, fuel, operation & maintenance, Levelized costs of heat (LCOH) etc.) associated with the technologies
- Emissions, emission savings and emission costs over the defined period.

3.2 Main Features of the TEO Module

- The TEO module optimizes the matching between the different sources and the sinks while taking into accounts various technical and economic constraints, such as demand profiles, technology cost, efficiencies and losses while also considering thermal energy storage.
- An optimal mix of investments in technologies and optimal capacities in storage and district heating network can be determined. The operation of the technologies and the intra-annual heat supply are also optimized in the module.
- The module can also analyze the competition between centralized and decentralized. The current input data includes competition between waste heat sources and decentralized solar thermal based heating solutions



- The TEO module optimizes the matching between the different sources and the sinks while taking into account various technical and economic constraints, such as demand profiles, technology cost, efficiencies and losses while also considering thermal energy storage.
- The TEO module carries out a socio-economic type of optimisation, where the total system cost is minimised, irrespective of who bears it. It does not take a policy-maker, investor, or business perspective.
- The time domain, time resolution and technological options are flexible and chosen by the user. For example, an analysis can be carried out for a time domain of 5, 10 or 30 years. Similarly, the time resolution can be of few time steps in a year, up to 8760 hourly time steps. The types of technologies that can be modelled include heat exchangers, heat pumps, boilers etc. The module is a model generator, where none of the above is pre-defined.
- The module relies on two core types of objects: Technologies and Fuels. These are very flexibly defined so that many different processes and commodities can be represented in a model. A Technology is nothing but a process - i.e. a box – with inputs, outputs, a transfer function between them, and several associated techno-economic characteristics. A Fuel is any commodity entering or exiting a Technology. Therefore, with a Technology, the user may represent a heat exchanger or a heat pump and for Fuel, the user may represent electricity or the excess heat stream.

3.3 User inputs

The user gives inputs through the platform. To improve the user-friendliness of the platform, a detailed description is provided for each user input. Furthermore, it will also be conveyed to the user if the input is mandatory or not. If the user does not give a value for a no-mandatory input, it will be obtained from the default values stored in the knowledge base.

3.3.1.1 SETS

The initial inputs to the TEO module consist of the global sets in the model. The ‘sets’ define the physical structure of a model, usually independent from the specific scenarios which will be run. They define the time domain and time resolution, the spatial coverage, the technologies, and energy vectors to be considered, etc. Some of these inputs are mandatory since they are essential for building the cost optimization. These are shown in Table 9. The description for each set is also provided in Table 9. A sample value to be entered for each SET is shown in the column ‘Value’. An intra-annual time resolution of 48 time slices is used in this case (but it can be lower or higher, up to hourly time steps!). This has been done to obtain quick results for the purpose of the workshop. A time period of one year has been used (but longer time



periods can be analysed, with results given per year). Two storage facilities have been implemented to demonstrate the two types of storage that can be modelled in the TEO. ‘CST’ identifies the centralised storage which is connected to the district heating network and ‘DST’ refers to a de-centralised storage on the source site.

Table 9: TEO Global SETS

Label	Description	Value
Region	It sets the regions to be modelled, e.g., different countries, cities, counties etc. For the purpose of this analysis, it is enough to have one region name. For each of them, the supply-demand balances for all the energy vectors are ensured. On some occasions, it might be computationally more convenient to model different countries within the same region and differentiate them simply by creating ad hoc fuels and technologies for each of them.	‘Greece’
Emission	It includes any kind of emission potentially deriving from the operation of the defined technologies. Typical examples would include atmospheric emissions of greenhouse gasses, such as CO ₂ . The user must fill in 'co2' as a mandatory entry. Other entries are also allowed	‘CO ₂ ’
Time resolution	It represents the time steps of each modelled year, therefore the time resolution of the model.	Weekly
Time period (Year)	It represents the period of the model; it contains all the years to be considered in the analysis.	2023
Mode of operation	It defines the number of modes of operation that the technologies can have. If a technology can have various input or output fuels and it can choose the mix (i.e., any linear combination) of these input or output fuels, each mix can be accounted as a separate mode of operation. The user must input at least 1 mode of operation. There must be two modes of operation if storage is used in the model	1

Taking the values from Table 9, insert the value for each set in the TEO inputs section of the ‘Create simulation’ window on the platform as shown in Figure 10 and Figure 11.



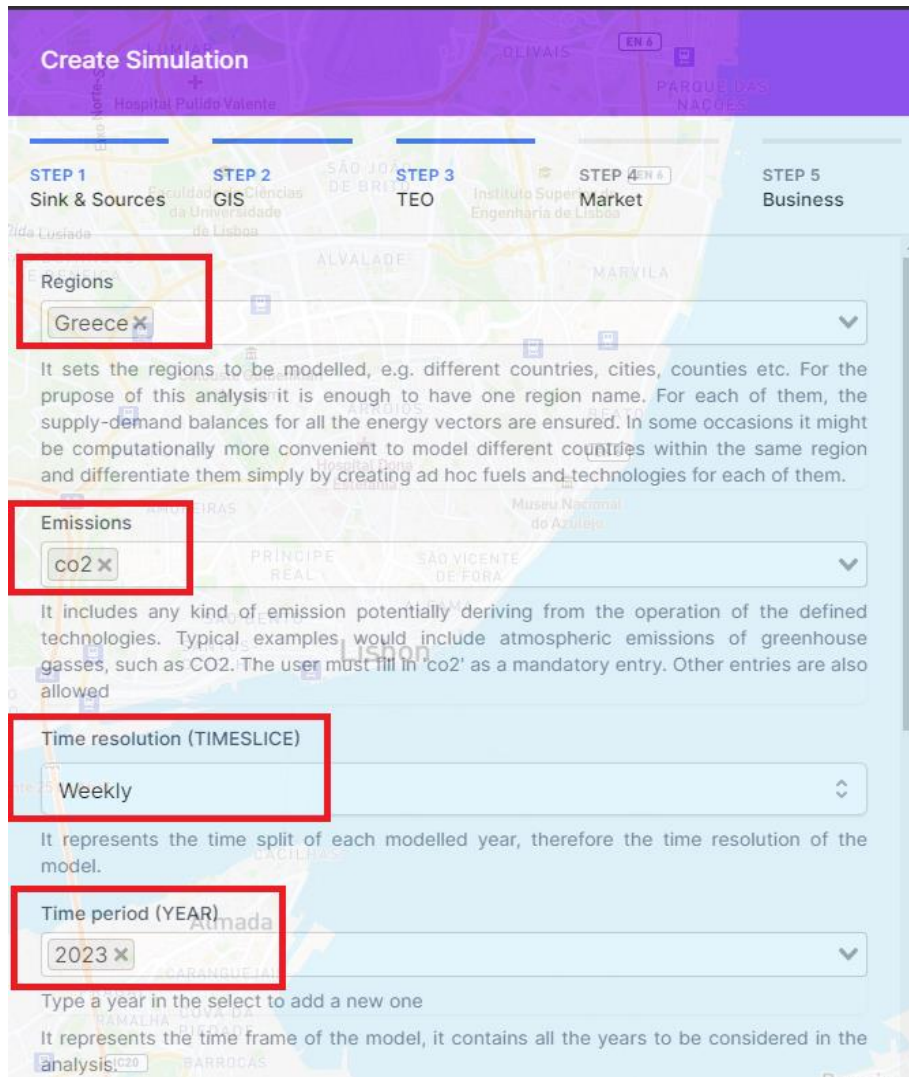


Figure 10: Inputs to the TEO -1

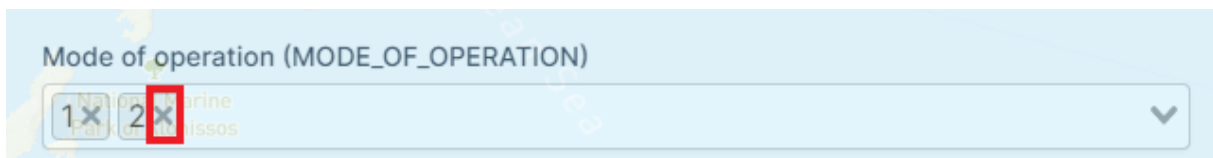


Figure 11: Inputs to the TEO -2

3.3.1.2 Step 6.2: Storage inputs

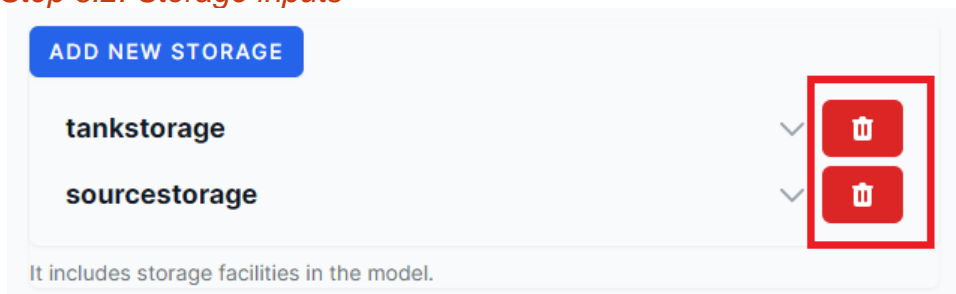


Figure 12: Storage inputs to the TEO

Table 10: Storage inputs

Label	Description	Value (Unit)	
		tankstorage	sourcestorage
Storage capital cost	Investment costs of storage additions, defined per unit of storage capacity.	600 €/kWh	300 €/kWh
Storage discount rate	Storage specific value for the discount rate, expressed in decimals	0.04	0.05
Storage operational life	Useful lifetime of the storage facility.	50	30
Storage maximum charge rate	Maximum charging rate for the storage	200 kWh	50 kWh
Storage maximum discharge rate	Maximum discharging rate for the storage	200 kWh	50 kWh
Storage length to diameter ratio	Binary parameter which indicates the length to diameter ratio of the thermal energy storage tank. Value is 0 if the L2D is 2 and is 1 if the L2D is 4.	1	0
Storage heating tag	Binary parameter indicating whether the thermal energy storage is connected to the district heating network. 1 if it is connected and 0 if is not.	1	1
Storage cooling tag	Binary parameter indicating whether the thermal energy storage is connected to the district cooling network. 1 if it is connected and 0 if is not.	0	0
Storage hot	The return water	60 °C	65 °C



water return temperature	temperature in the heating grid where the thermal energy storage is connected.		
Storage hot water supply temperature	The temperature of water inflow into thermal energy storage.	80 °C	85 °C
Average ambient temperature of the region	The average ambient temperature of the locations where the thermal energy storage is located.	15 °C	15 °C
Residual storage capacity	Exogenously defined storage capacities at the start of the modeling period	0 kWh	0 kWh
Maximum storage capacity	Maximum allowed capacity of each storage in a year	1000 kWh	100 kWh
Starting level of storage	Level of storage at the beginning of first modelled year	0 kWh	0 kWh
Heat transfer coefficient of the thermal storage	Heat transfer co-efficient of the thermal energy storage tank.	0.22 kW/KgK	0.19 kW/KgK

3.3.1.3 Step 6.3: Emission and budget constraints inputs

The TEO module also accounts for the emission from each technology in the system. In the SETS, we had earlier defined ‘CO2’ in the ‘EMISSION’ set (more emissions could be defined, but we do not consider them in this case, for simplicity). The emission factor (kgCO2/kWh) for each technology is provided to the TEO from the the default database. Using this, the emission from each technology is calculated in the TEO. As a constraint, it is possible to limit the emission from the excess heat recovery system. For the workshop, the emission limit is defined as shown in Table 11.

Table 11: TEO Emission Limit

Label	Description	Value (Unit)
Annual emission limit	Annual upper limit for a specific emission generated in the entire modelled region.	150000000000 Kg CO2

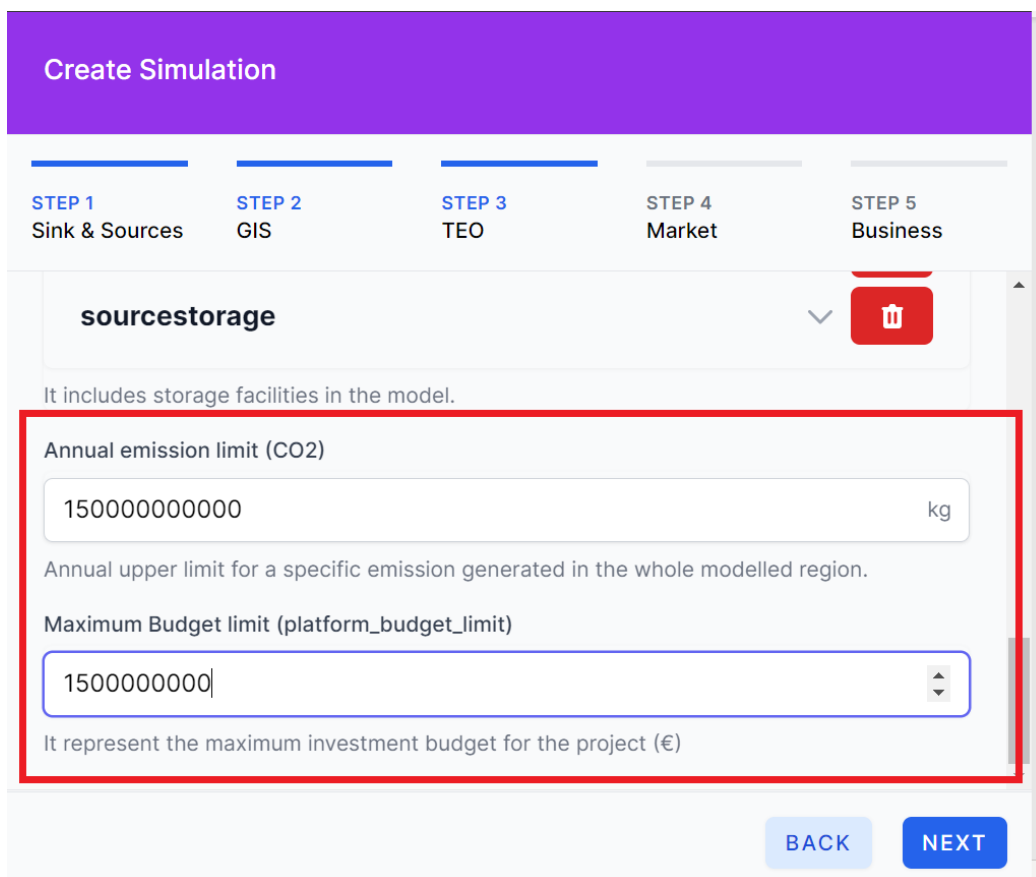


The TEO module also accounts for the maximum budget for the project. This can be done using the ‘Maximum Budget Limit’ constraint. The input for this exercise is as shown in the Table 12. The value is set as very high so that we do not consider any budget constraint.

Table 12: TEO Budget Limit

Label	Description	Value (Unit)
Maximum Budget limit	It represents the maximum investment budget for the project (€)	1500000000 €

The inputs can be entered as shown in Figure 13.



The screenshot shows a web interface titled "Create Simulation" with five steps: STEP 1 Sink & Sources, STEP 2 GIS, STEP 3 TEO, STEP 4 Market, and STEP 5 Business. The "sourcestorage" section is expanded, showing a description: "It includes storage facilities in the model." Below this, two input fields are highlighted with a red box: "Annual emission limit (CO2)" with a value of 150000000000 kg, and "Maximum Budget limit (platform_budget_limit)" with a value of 15000000000. The description for the budget limit is "It represent the maximum investment budget for the project (€)". At the bottom, there are "BACK" and "NEXT" buttons.

Figure 13: Emissions and budget constraints for the TEO module

3.4 Simulation

3.4.1 Actors

The term ‘Actors’ identifies all the separate entities that will be involved during a simulation of the TEO module. In the case of the TEO module, it will involve the user,



the knowledge base and all the modules that provide any inputs to the TEO as shown below:

- **Platform User**
- **Knowledge base**
- **Core functionalities (CF) module**
- **Techno-economic optimization (TEO) module**
- **Geographical Information System (GIS) module**

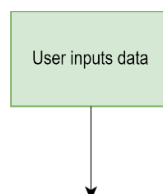
3.4.2 Pre-Conditions

For the simulation of the TEO module on the EMB3RS platform, the following conditions:

- **The user must be logged into the platform**
- **The user must have inserted all mandatory input data for TEO**
- **The CF module must have run successfully and generated inputs for the TEO**
- **The GIS module must have run successfully and generated inputs for the TEO**

3.4.3 Basic Flow for the user

The basic flow of the simulation is shown in Figure 14. The TEO module obtains inputs from the CF and the GIS module. The inputs are first prepared and then the function 'buildmodel' is executed. If the data present a feasible model, the model is solved and the optimal solution is found. If the data do not present a feasible solution, an error message appears, which explains to the user where the run was not successful. The results are post-processed and forwarded to the user and other modules.



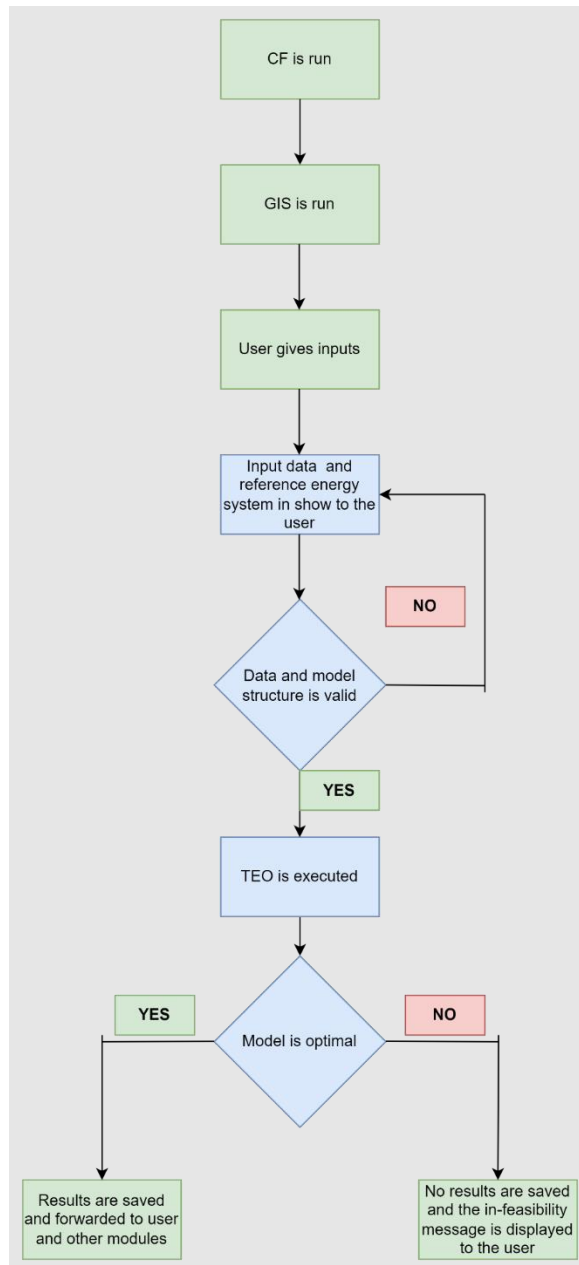


Figure 14: Basic flow for the user (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

For the simulation:

1. The user enters the platform and runs the CF and the GIS modules
2. The user provides the inputs for the sets, storages and technologies
 - a. **SETS:**
 - i. The user chooses the modelled region
 - ii. The user chooses the emissions to be included in the model
 - iii. The user chooses the time resolution of the model
 - iv. The user includes the period of the analysis
 - v. The user chooses the number of modes of operation
 - vi. The user enters the number of storage options

Example input for Time period (number of years to be analysed) and Time resolution (number of intra-annual time steps) is shown in Figure 15.

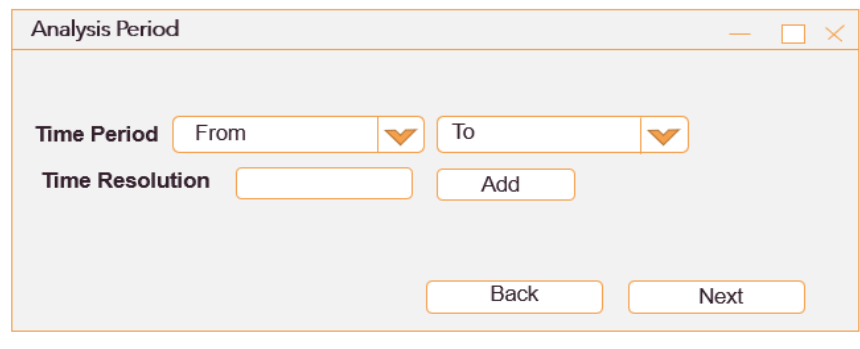


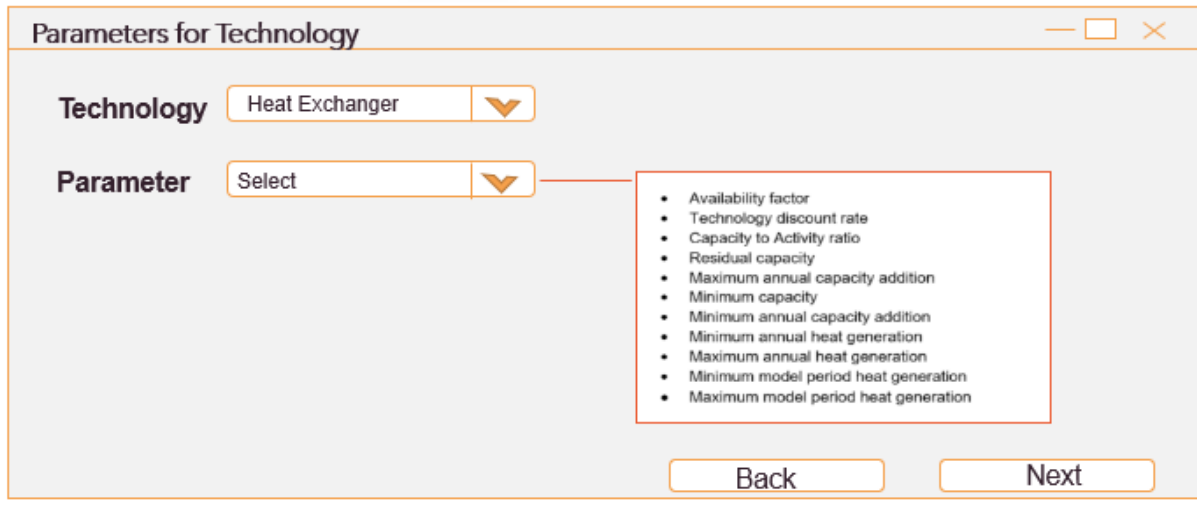
Figure 15: Mock-up example input of analysis period and Time resolution (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

b. Technology:

i. The user enters the following data for each technology

- 1. Availability factor (No units)**
- 2. Technology discount rate (No units)**
- 3. Capacity to Activity ratio (kWh/kW)**
- 4. Residual capacity (kW)**
- 5. Maximum annual capacity addition (kW)**
- 6. Minimum capacity (kW)**
- 7. Minimum annual capacity addition (kW)**
- 8. Minimum annual heat generation (kWh)**
- 9. Maximum annual heat generation (kWh)**
- 10. Minimum model period heat generation (kWh)**
- 11. Maximum model period heat generation (kWh)**

Example input for the user input parameters for each technology is shown in Figure 16. Here, the list of technologies is obtained from the CF module. The user can choose to enter a value for each parameter for each technology. If the user does not enter the values, the default values from the knowledge base (shown in Table 6) will be used. The second step of the input is shown in Figure 17.



Parameters for Technology

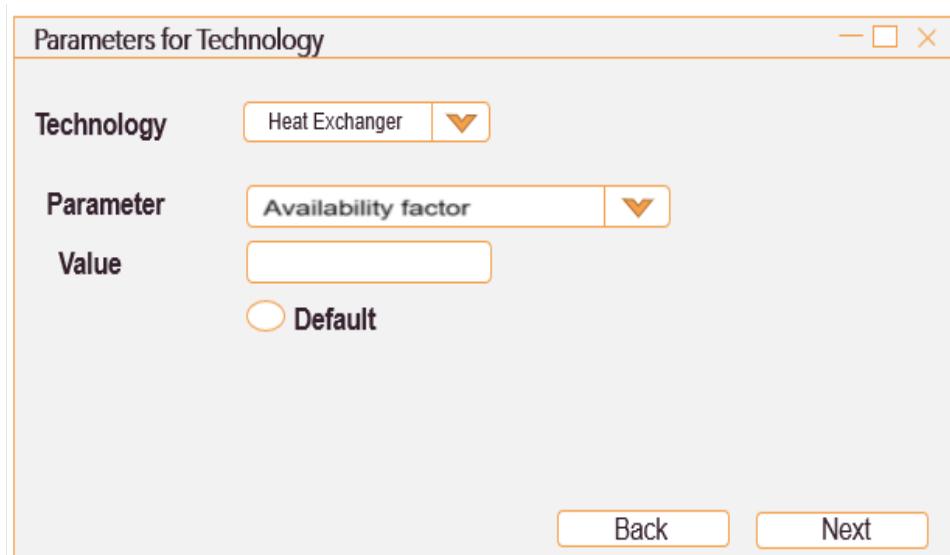
Technology: Heat Exchanger

Parameter: Select

- Availability factor
- Technology discount rate
- Capacity to Activity ratio
- Residual capacity
- Maximum annual capacity addition
- Minimum capacity
- Minimum annual capacity addition
- Minimum annual heat generation
- Maximum annual heat generation
- Minimum model period heat generation
- Maximum model period heat generation

Back Next

Figure 16: Mock-up user input for technology parameters (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))



Parameters for Technology

Technology: Heat Exchanger

Parameter: Availability factor

Value:

Default

Back Next

Figure 17: Mock-up for Step 2 of user input for technology parameters - entering values (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

c. Storage:

1. Storage capital cost (€/kWh)
2. Storage discount rate (No units)
3. Storage operational life (Years)
4. Storage maximum charge rate (kWh)
5. Storage maximum discharge rate (kWh)
6. Storage length to diameter ratio (No units)
7. Storage heating tag (No units)
8. Storage cooling tag (No units)
9. Storage hot water return temperature (°C)
10. Storage hot water supply temperature (°C)
11. Average ambient temperature of the region (°C)
12. Residual storage capacity (kWh)
13. Maximum storage capacity (kWh)

- 14. Starting level of storage (kWh)
- 15. Heat transfer co-efficient of the thermal storage (kJ /kg k)
- 16. Annual emission limit (Kg CO₂)
- 17. Technology charging storage (No units)
- 18. Technology getting discharge from storage (No units)

Example input for the user input parameters for each storage is shown in Figure 18. Here, the list of storages is obtained from previous user inputs. The user can choose to enter a value for each parameter for each storage. If the user does not enter the values, the default values from the knowledge base (shown in Table 6) will be used. The second step of the input is shown in Figure 19.

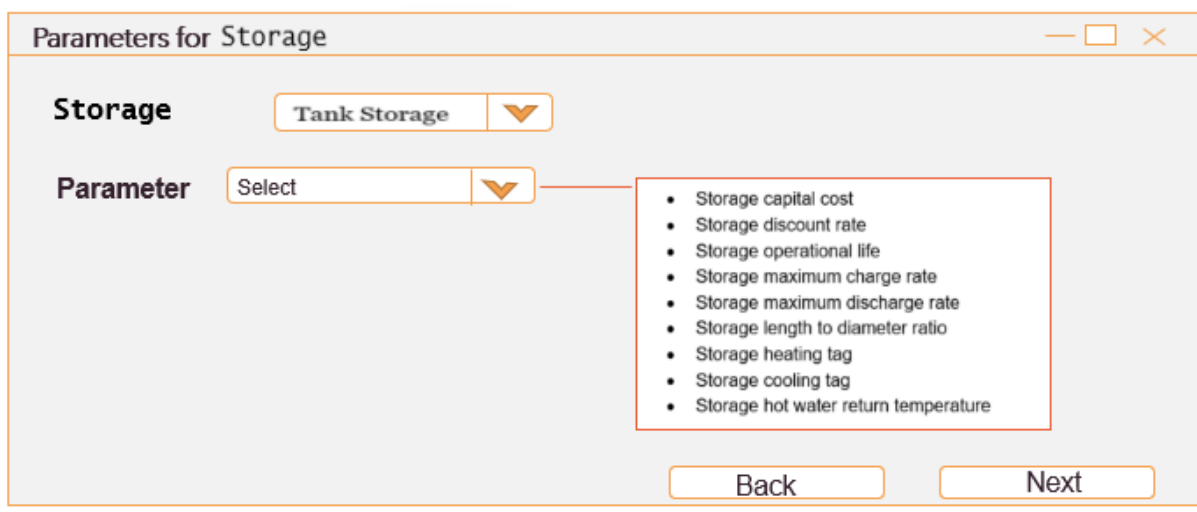


Figure 18: Example user input for storage parameters (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

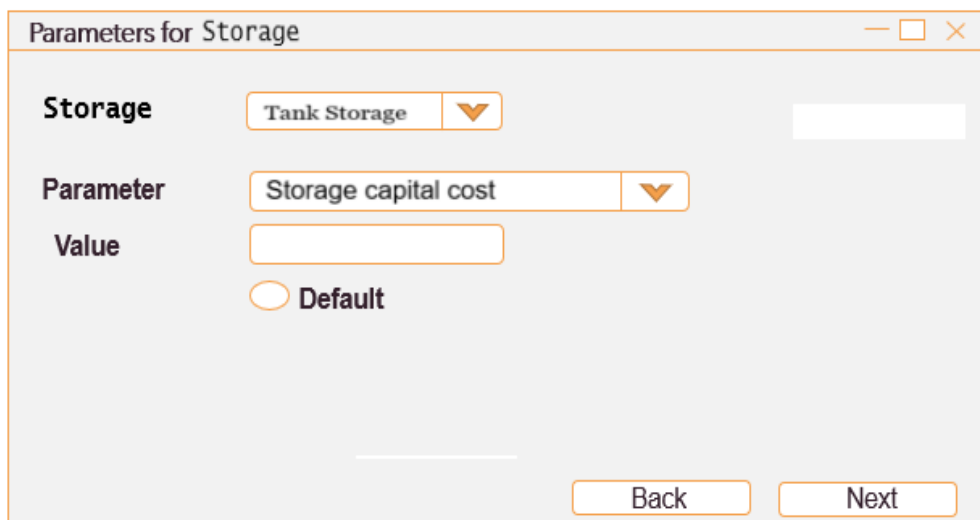


Figure 19: Step 2 of Example user input for storage parameters - entering values (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

3. Data and the Reference energy system are displayed to the user and they can validate and/or modify the data



When developing a model using an optimization, the energy system needs to be mapped to identify all the relevant technologies and fuels that will be involved in the analysis. The schematic representation of the system for such purposes is called Reference Energy System (RES). A reference energy system of a test case for the TEO is shown in Figure 20.

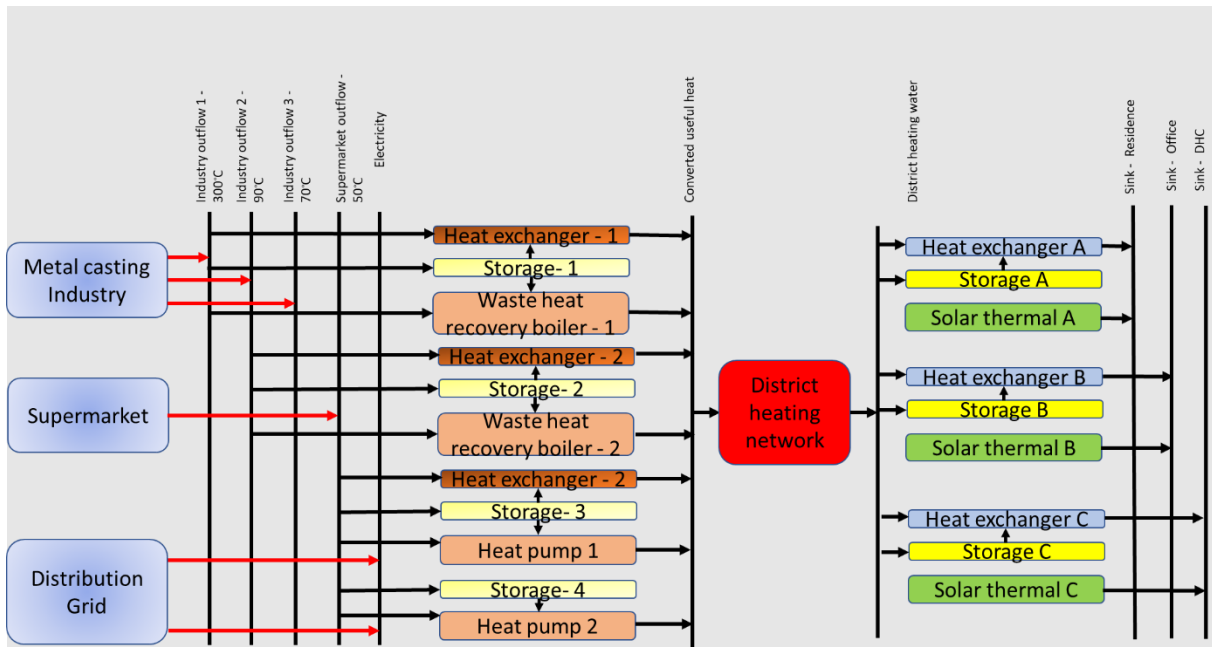


Figure 20: Sample Reference Energy System (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

The lines represent energy carriers (e.g., Flue gas, district heating water etc.) while the blocks represent heat conversion and generation transformation technologies (e.g., heat exchanger, heat pump, boiler etc.). The RES can be read from the left to the right. On the left-hand side, the excess heat resources are represented. Sources of excess heat are represented as technologies in the RES (i.e. boxes with outgoing lines representing the fuels they make available). Importantly, each chain must always start with technology. Moving from the left to right, the energy carriers are transformed by different technologies, each with a transfer function, to ultimately meet the final demand for energy or services, presented by the lines on the far right-hand side. This allows the user to visualize the analysed case from the recovery of excess heat to meeting the final demand for heat or cold.

4. Input data is saved automatically

5. The TEO Simulation is run

- a. If the simulation is feasible, the model is solved and the results are sent to the user and the other module.
- b. If the simulation is not feasible, a message saying that ‘Infeasible: No solution exists is displayed.
- c. Simulation is too large for the platform, a message saying that ‘Simulation: Out of memory, simulation is too large for the platform. Please reduce the time resolution.’

3.4.4 User-defined constraints

To increase the user-friendliness of the platform, ‘User-defined’ constraints can be introduced, through which the user can set some limits for the decision variables in the TEO module. Four preliminary user-defined constraints are suggested for the platform. The constraints, their description, the initial plan for executing the constraints in the platform and the status of execution are presented in Table 13.

Table 13: User-defined constraints (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

Constraint	Description	Plan for execution	Status of execution
Maximum storage capacity	The user can set the maximum possible capacity for each storage in the TEO	‘Maximum storage capacity is user input for the TEO. Using this parameter, this constraint can be added.	Completed
Minimum Share of excess heat	The user can set the minimum share of excess heat in the heating system as a percentage value. For example, if the user inputs a value of 25%, then at least 25% of the heat demand will be met using excess heat.	‘Minimum annual heat generation’ for each technology is a user input to the TEO. However, it is not the same value as the minimum share of waste heat. Hence, some calculations will be needed to determine the minimum annual heat generation from the minimum share of waste heat. It is proposed to have these calculations before in the platform before the inputs are sent to the TEO.	In progress
Maximum Budget for the project	This will let the user set a maximum capital investment cost for the technologies.	There is a possibility o limit the overall investment cost of the project in the TEO. This will simply constrain the capital investment for the technologies. Since the TEO module determines the least-cost solution by optimising the costs, placing a constraint on the cost component in the model will lead to sub-optimal results. Furthermore, the TEO module optimises the overall cost of the system and hence does not consider the disaggregation of costs for each actor. Thus, the constraint is	Completed



		specified for the overall project. Furthermore, if the specified budget constraint is too low, then the case becomes infeasible.	
Maximum emission limit	The user can set the maximum possible emission from the analysed system	'Annual emission limit' is user input for the TEO. Using this parameter, this constraint can be added.	Completed
Carbon tax	The user can set the carbon tax or emission penalty from the analysed system	'Emission penalty' is an input for the TEO. Using this parameter, this constraint can be added. It must be clarified whether this constraint overlaps with the inputs from the regulatory framework.	In progress

3.5 Running a Test Case using the standalone version of the TEO

The standalone version of the TEO indicates the version of the tool that can be accessed and run outside the EMB3RS platform which can be accessed [here](#). It has the same functionalities as the version integrated with the platform. However, all the inputs for this version are obtained from the user, since it is not linked with any other module. The standalone version uses an excel sheet for inputs to the module and to save the results from the module.

3.5.1 Description of the test case

This section introduces the user to the basic components of any application of TEO and describes the steps for the creation of a model. To this end, a sample case study is used and examples from it are shown throughout the section.

The sample case study for TEO represents a case of industrial excess heat recovery and use. The simple use case consists of two excess heat producers - Supermarket and Metal casting Industry - and three sink points - District Heating and Cooling grid (DHC), office buildings and residential buildings. Generic demand profiles are used for all the sinks. The model is simulated over 10 years and at an intra-annual time resolution of 48 TimeSlices. Note that a low time resolution is used for the test case so that the user can obtain the results quickly (in minutes). The TEO can model at higher resolution, but the simulation times also become longer. The boundaries of the system represented in the case study are shown in the schematic representation shown in

Figure 21.



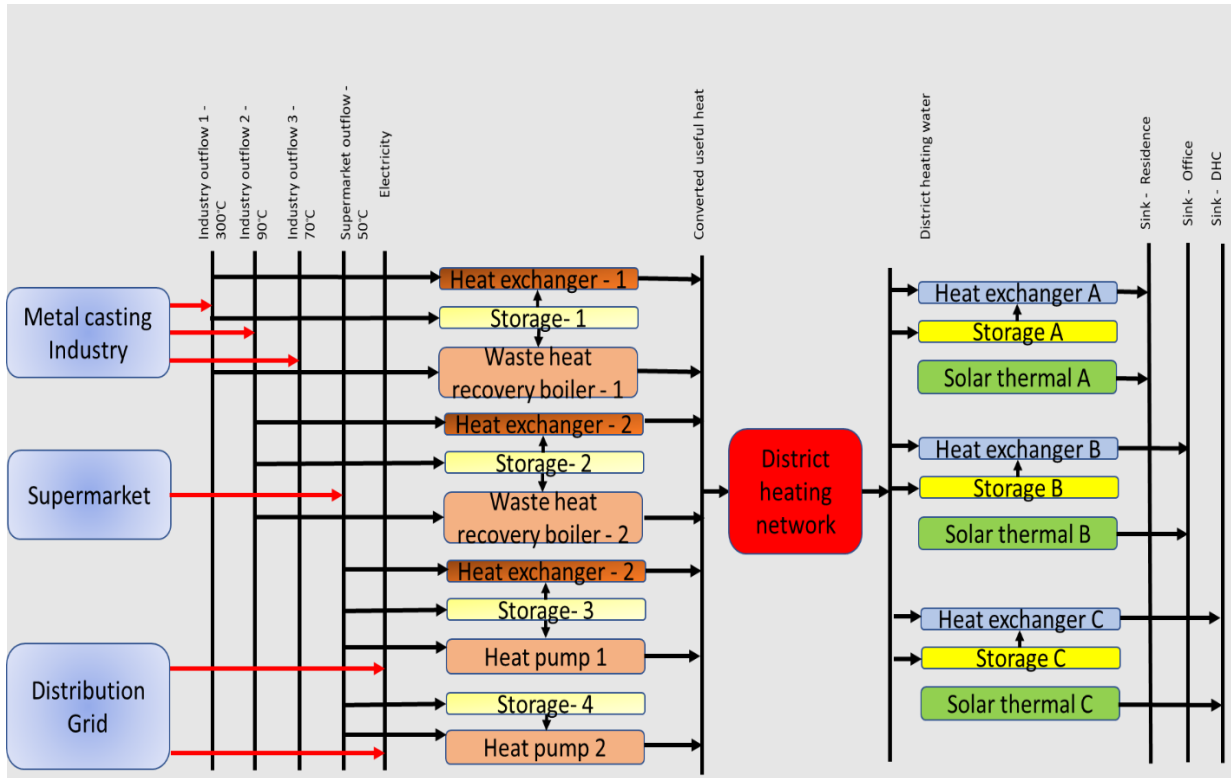


Figure 21: Reference energy system (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

In the Reference Energy System (RES), the rectangles represent the technologies, the arrows represent the flow of energy and the vertical lines represent fuels. The RES is read from the left to the right. The primary energy supply side is on the left and the final energy demand side is on the right. The source nodes, the metal casting industry and the supermarket are modelled as technologies, whereas the sink nodes, being residential buildings, office buildings and DHC are modelled as fuels. The distribution grid has also been added as technology as some components can either require or produce electricity. For the test case, the heat pumps in the system will require electricity to operate. The first set of vertical lines represents fuels at the primary level. These primary fuels consist of electricity, waste heat from the outflows of the metal casting industry at three different temperatures, and waste heat from the supermarket. The waste heat from each source is supplied to a set of technologies. Here, we assume a Heat Exchanger (HE), Waste Heat Recovery Boiler (WHRB) for the first waste heat outflow from the industry. Similarly, the second outflow makes use of a HE and a WHRB. The third outflow is provided to a Heat Pump (HP) and a HE. The supermarket waste heat is provided with an HP.

The outflow temperatures of the metal casting industry are to be at 300°C, 90°C and 70°C Celsius for its three outflows whereas the waste heat from the supermarket is at 50°C. On the sinks' side, the DHC demand temperature is the same as the average supply temperature of the DHN, which is at 90°C. The demand temperature profile of the office buildings is the same as that of the residential buildings at 90°C. The sources and the sinks are equipped with storage. This implies that the generation and supply

technologies be connected to storage technology, and also, the demand technologies are connected to storage systems.

Only one storage can be chosen for a set of technologies. Here, each technology is connected to a storage option. Based on the technology that is selected from the optimization process, the energy will be stored from the technology in the corresponding storage.

The technologies are assessed for feasibility and selected in the process. The converted useful heat at the suitable temperature (here, the network temperature) is stored and supplied to the secondary fuel level. The secondary level fuel is the converted useful heat. The converted useful heat is then supplied to the District Heating Network, which is modelled as a technology. The network is similar to the distribution grid being modelled as a technology, and they both account for losses. The heat from the network is supplied to all demand points by first being transformed into a tertiary level fuel of district heating water. To further assess the feasibility of the demand side system, solar technologies have been added. Solar thermal technologies have been added to all sinks.

3.5.2 Data and instructions to run the model

This section refers to the standalone version of the TEO module (not integrated with the platform or other modules). This can be accessed [here](#). The input file for the prototype is 'Input_file_TEO.xlsx', which can be accessed at the GitHub repository [here](#). To run the TEO, the code files named 'TEO_Model', 'TEO_functions', 'TEO_running_file', and the input file must be downloaded from the GitHub repository [here](#) and saved in a specific manner. The main folder called 'TEO' must be created and the code will be downloaded into this folder. Within this main folder, two subfolders named 'Input_data' and 'Output_data' must be created. The input file must be saved into the 'Input_data' folder. A representation of how the files must be organised in the folder is shown below.

- TEO (Main folder)
 - Input_data
 - Input_file_TEO.xlsx
 - Output_data
 - TEO_Model
 - TEO_functions
 - TEO_running_file

Once the TEO simulation is completed, the results file named 'TEO_Results.xlsx' will be saved in the 'Output_data' sub-folder.



Once the files are downloaded and the folder structure is established, the model can be run using the TEO_running_file. The name of the input directory and the input file must be checked in the TEO_running_file. Since the name of the input file is to be checked and altered, it is advisable to open the TEO_running_file in a python IDE or other open-source python notebook interface such as 'Jupyter lab' or 'Visual studio code'. Both these are freeware and can be downloaded. The TEO module can output results in two formats, excel and CSV. The preference for the output format can also be set in the TEO_running_file by specifying a 'True' or 'False' next to the output formats in the TEO_running_file.

3.5.3 Note on the solvers

Two solvers, GLPK (GNU linear programming kit) and CBC (Coin-or branch and cut) are inbuilt into the module. To use other solvers, they should be downloaded and installed. Instruction for this can be found [here](#). After the installation of the solver, the solver path needs to be added as an environment variable and then should be called into python using solver commands. Step by step instructions for adding the environment variables can be found [here](#). The user can analyse the data based on the results saved in the output file. The user can also use other solvers such as CPLEX and Gurobi to run the TEO. The solver name and path must be specified in the TEO_running_file as shown in Figure 22.



```

import os
import datetime as dt
import logging
import numpy as np
import pandas as pd
import pulp
import itertools

#Importing all required functions
from TEO_functions import *
from TEO_Model import buildmodel

logging.basicConfig(level=logging.DEBUG)
logging.info(f"\t\t{dt.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\t\t05eMDSYS-PuLP-HP started.")

# Input data
inputFile = "PAPER_4B_6.xlsx" # Update with actual filename
inputDir = ".\Input_Data\\"
modelName = inputFile.split('.')[0]
sheetSets = "SETS"
sheetParams = "PARAMETERS"
sheetParamsDefault = "PARAMETERS_DEFAULT"
sheetMcs = "MCS"
sheetMcsNum = "MCS_num"
outputDir = ".\Output_Data\\"

# -----
# ---SETUP - DATA SOURCES and MONTE CARLO SIMULATION
# -----

# Output data
save_as_csv = True # True: Output data will be saved as CSV file; False: No saving. Note: Rapid process.
save_as_excel = True # True: Output data will be saved as Excel file; False: No saving. Note: Takes a lot of time.

# -----
# LOAD DATA
# -----

inputPath = os.path.join(inputDir, inputFile)
sets_df, df, defaults_df, mcs_df, n = loadData(
    inputPath, sheetSets, sheetParams, sheetParamsDefault, sheetMcs, sheetMcsNum)
parameters_mcs = mcs_df['PARAM'].unique() # list of parameters to be included in monte carlo simulation

logging.info(f"\t\t{dt.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\t\t"
            f"Data is loaded.")

# -----
# BUILD AND SOLVE MODEL
# -----
res_df1 = buildmodel(sets_df, df, defaults_df, mcs_df, n, modelName)

# -----
# SAVE ALL RESULTS
# -----

# Logging.info(f"\t\t{dt.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\t\t"
#             f"Analysis is finished. Please wait until the results are saved!")

# CSV
if save_as_csv is True:
    outputFileCSV = f"{modelName}_results.csv"
    saveResultsToCSV(res_df1, outputDir, outputFileCSV)

# Excel
if save_as_excel is True:
    outputFileExcel = f"{modelName}_results.xlsx"
    saveResultsToExcel(res_df1, outputDir, outputFileExcel)

logging.info(f"\t\t{dt.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}\t\t"
            f"All results are saved now.")

```

Figure 22: Updating input file in the code (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

3.5.4 Results from the Test Case

If run correctly, the Test Case will provide the results shown in the following. Note that these results are only a selection of the key outputs, condensing some of the important insights. Outputs for all the ‘variables’ listed in the previous sub-sections are calculated. They can also be extracted and visualised.

- The technology mix (in terms of existing and newly installed yearly capacities in terms of energy flows throughout the supply-demand chain)



- The installed capacities at the different sources are shown in
- Figure 23. The model only uses two sink outflows from the Metal casting industry. The excess heat from the supermarket is not used due to the low temperatures and thus needs expensive investments.

Source capacities

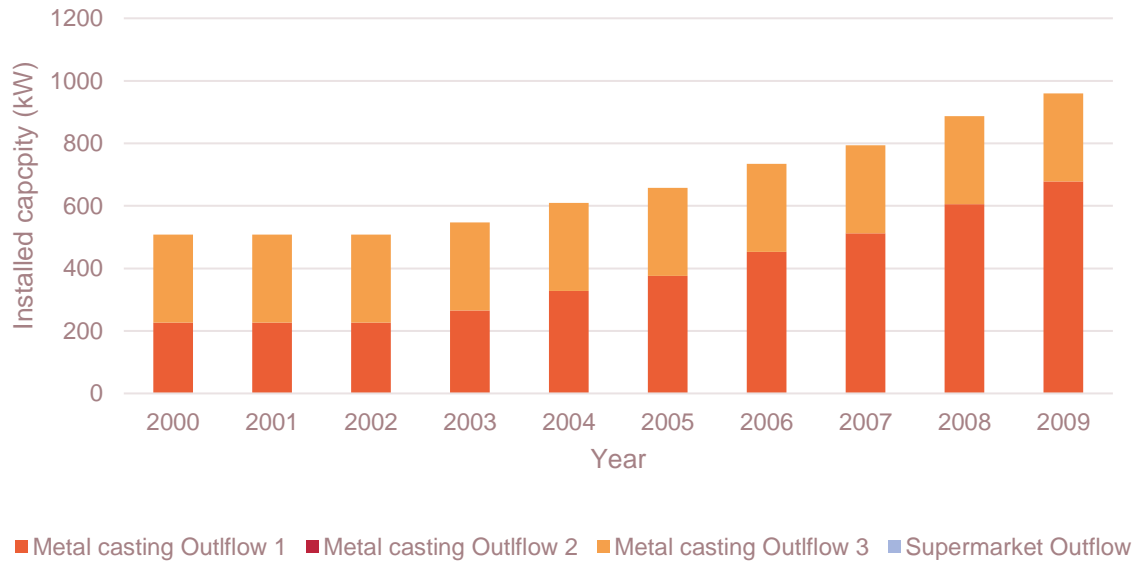


Figure 23: Installed capacity at the sources (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

- The sink capacities are based on the sink demand. As the demand grows over the years, the capacity is also increased.

Sink capacities

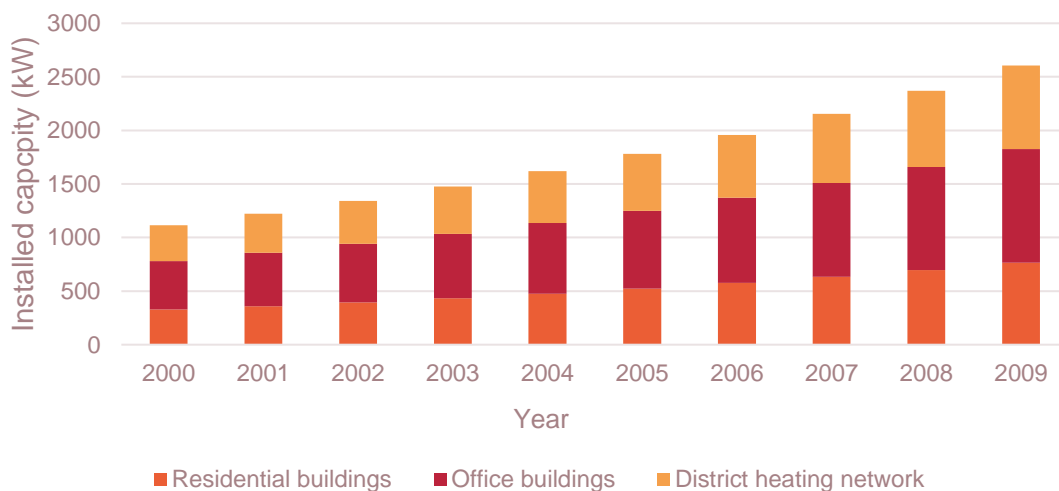


Figure 24: Installed capacity at the sinks (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

- Share of each technology in meeting the demand in any time step of the analysis (where the time resolution is defined by the user within certain limits) and throughout the analysis period
- The installed capacities of storage are shown in
- Figure 25. The model only installs storage at the sink site since it is easier to control installed capacities in the DHN if the storage is located after the DHN. The storage capacities are also based on the sink demand. As the demand grows over the years, the storage capacity is also increased.

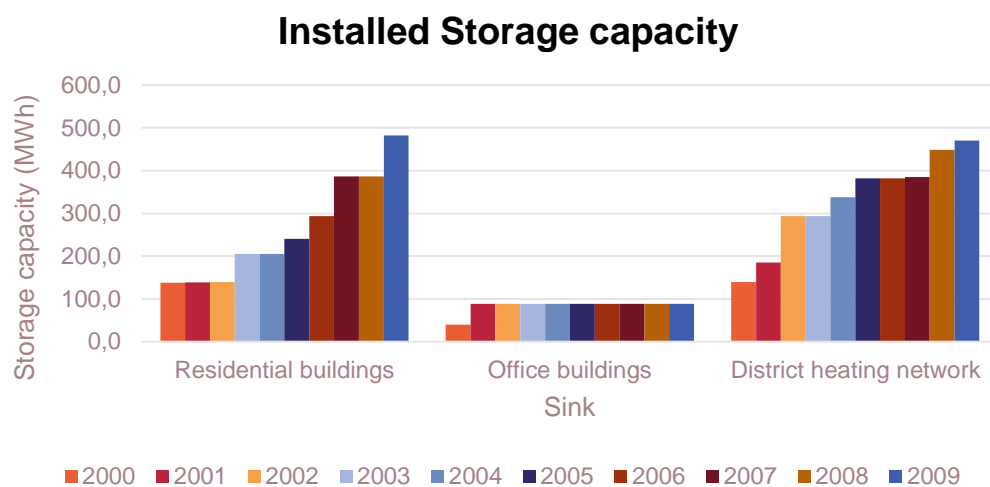


Figure 25: Installed capacity of storage (Author: Shравan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

- The Intra annual heat generation from sources is shown in
- Figure 26. The heat generated from the sources does not follow the demand profile due to the storage in the system. We see that the heat generation is constant in most time steps and there are drastic variations in a few TimeSlices. This is because of the charging and discharging from the storage, which is seen next.

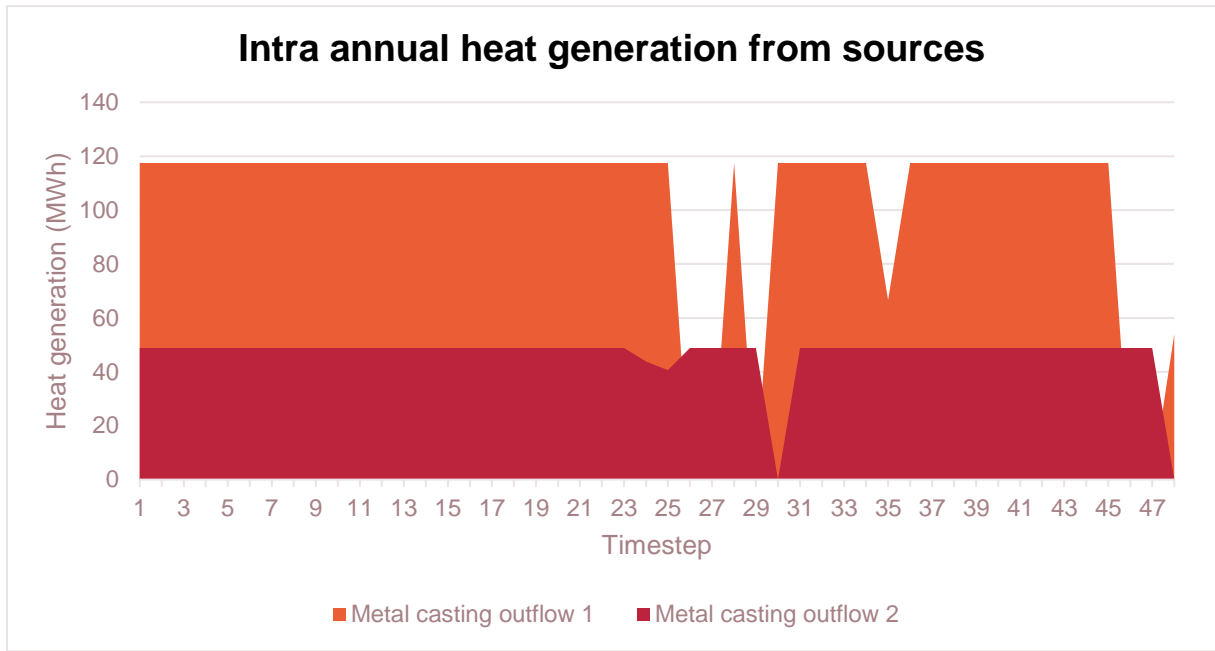


Figure 26: Intra annual heat generation from sources (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

The Intra annual storage charge or discharge for Residential buildings is shown in Figure 27. The storage is continuously cycled according to the sink demand profiles.

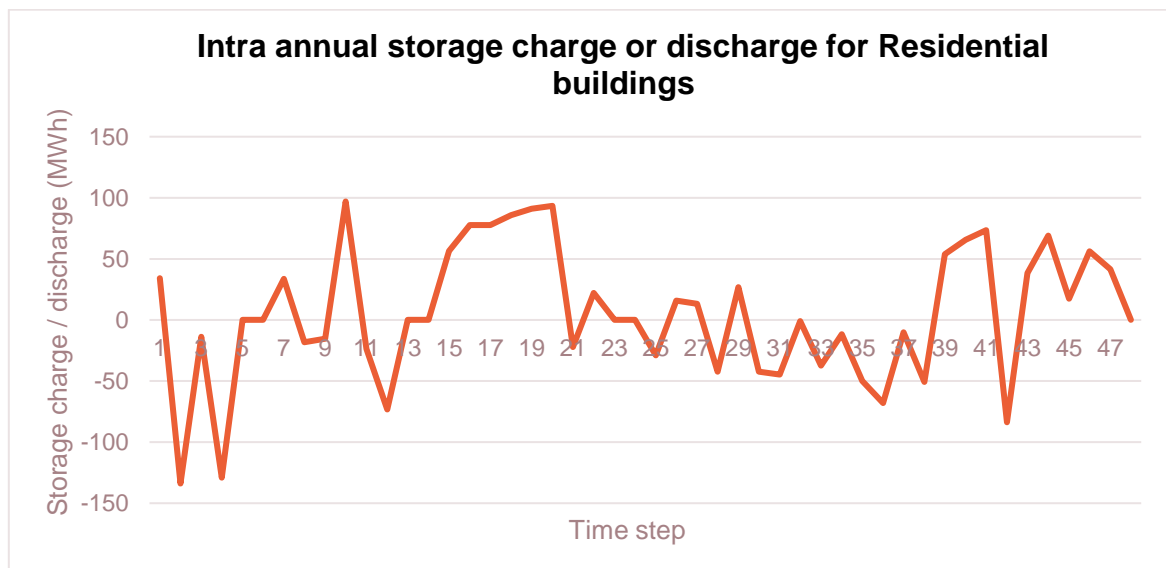


Figure 27: Intra annual storage charge or discharge for Residential buildings (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

- Annual costs (investment, fuel, operation & maintenance) associated with the technologies. The investment costs for the sinks are shown in
- Figure 28.

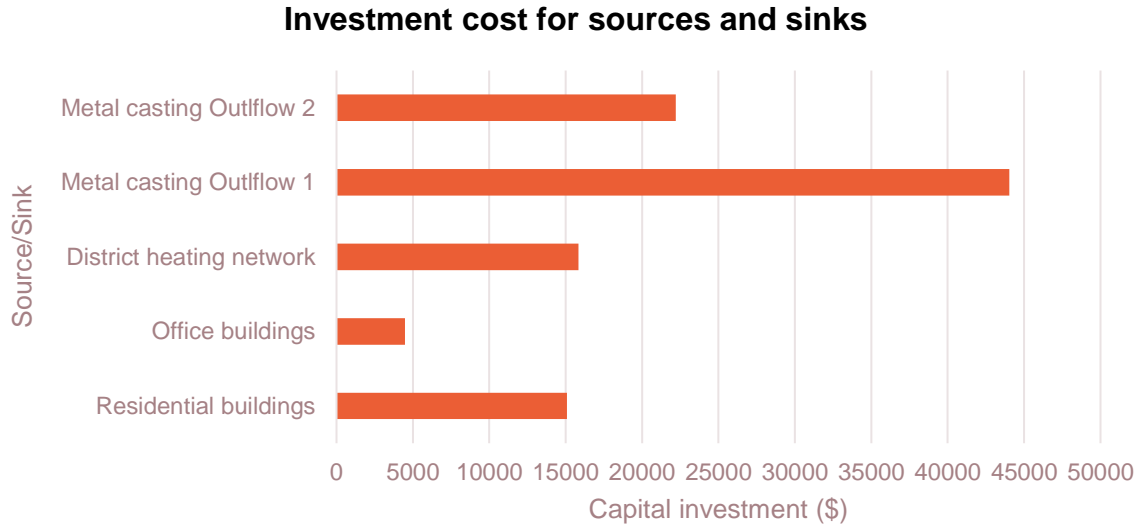


Figure 28: Investment cost for sources and sinks (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

- The operation and maintenance costs of the sources and the sinks are shown in
- Figure 29.

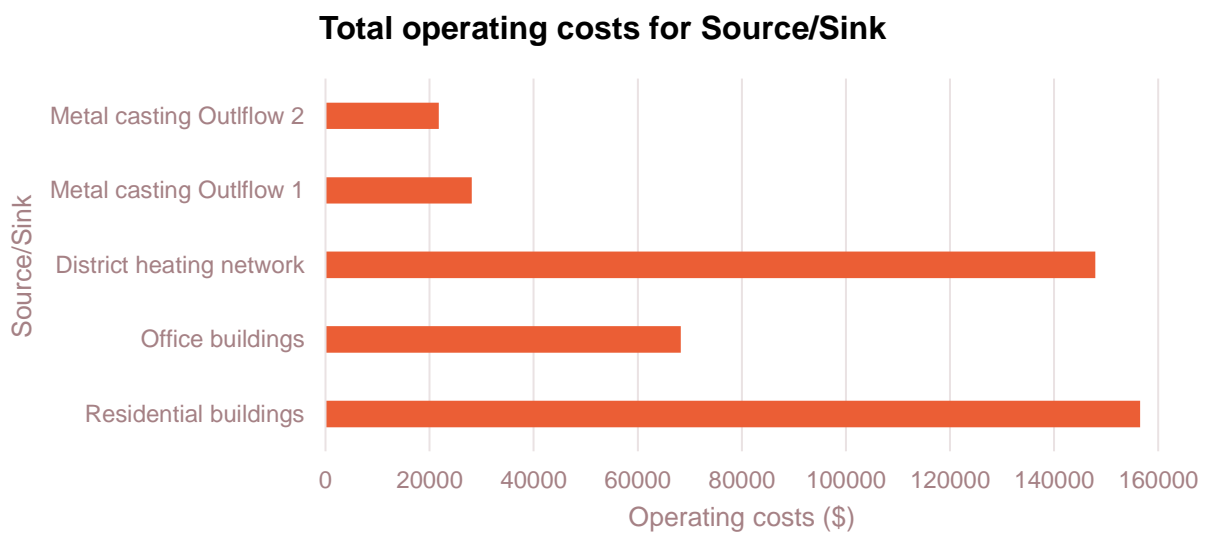


Figure 29: Total operating costs for Source/Sink (Author: Shravan Kumar, licensed under [CC-BY 4.0](https://creativecommons.org/licenses/by/4.0/))

4 References

- [1] M. Howells *et al.*, OSeMOSYS: The Open Source Energy Modeling System. An introduction to its ethos, structure and development., *Energy Policy*, vol. 39, no. 10, pp. 5850–5870, 2011, DOI: 10.1016/j.enpol.2011.06.033.
- [2] D. Dreier and M. Howells, “OSeMOSYS-PuLP: A Stochastic Modeling Framework for Long-Term Energy Systems Modeling,” *Energies*, vol. 12, no. 7, p. 1382, Apr. 2019, DOI: 10.3390/en12071382.
- [3] “Introduction to OSeMOSYS — OSeMOSYS 0.0.1 documentation.” <https://osemosys.readthedocs.io/en/latest/manual/Introduction.html> (accessed Feb. 04, 2022).



5 Appendix 1

Equations of the TEO code

```

for rfly in REGION_FUEL_TIMESLICE_YEAR:

    # EQ_SpecifiedDemand

    model += RateOfDemand.get(ci(rfly)) ==
SpecifiedAnnualDemand.get(ci([*rfly[0:2], rfly[3]]), dflt.get('SpecifiedAnnualDemand'))
* SpecifiedDemandProfile.get(ci(rfly), dflt.get('SpecifiedDemandProfile')) /
YearSplit.get(ci(rfly[2:4])), ""

    # ===== Capacity Adequacy A =====

for rfty in REGION_TIMESLICE_TECHNOLOGY_YEAR:

    # CAa3_TotalActivityOfEachTechnology

    model += RateOfTotalActivity.get(ci(rfty)) ==
pulp.lpSum([(RateOfActivity.get(ci([*rfty[0:2], m, *rfty[2:4]])) *
OutputModeofoperation.get(ci([rfty[0], m, *rfty[2:4]]),
dflt.get('OutputModeofoperation')) for m in MODE_OF_OPERATION]), ""

    # CAa4_Constraint_Capacity

    model += RateOfTotalActivity.get(ci(rfty)) <=
TotalCapacityAnnual.get(ci([rfty[0], *rfty[2:4]])) * CapacityFactor.get(ci(rfty),
dflt.get('CapacityFactor')) * CapacityToActivityUnit.get(ci([rfty[0], rfty[2]]),
dflt.get('CapacityToActivityUnit')), ""

for rty in REGION_TECHNOLOGY_YEAR:

    # CAa1_TotalNewCapacity

    model += AccumulatedNewCapacity.get(ci(rty)) ==
pulp.lpSum([NewCapacity.get(ci([*rty[0:2], yy])) for yy in YEAR if (float(int(rty[2]) -
int(yy)) < float(OperationalLife.get(ci(rty[0:2]), dflt.get('OperationalLife')))) and
(int(rty[2]) - int(yy) >= 0)]), ""

    # CAa2_TotalAnnualCapacity

```



```

        model += TotalCapacityAnnual.get(ci(rty)) ==
AccumulatedNewCapacity.get(ci(rty)) + ResidualCapacity.get(ci(rty),
dflt.get('ResidualCapacity')), ""

        if CapacityOfOneTechnologyUnit.get(ci(rty),
dflt.get('CapacityOfOneTechnologyUnit')) != 0:

            # CAa5_TotalNewCapacity

            model += NewCapacity.get(ci(rty)) ==
CapacityOfOneTechnologyUnit.get(ci(rty), dflt.get('CapacityOfOneTechnologyUnit')) *
NumberOfNewTechnologyUnits.get(ci(rty)), ""

        # ===== Capacity Adequacy B =====

        # CAb1_PlannedMaintenance

        model += pulp.lpSum([RateOfTotalActivity.get(ci(rlty)) * YearSplit.get(ci([l,
rlty[3]])) for l in TIMESLICE]) <= pulp.lpSum([TotalCapacityAnnual.get(ci([rlty[0],
*rlty[2:4]])) * CapacityFactor.get(ci(rlty), dflt.get('CapacityFactor')) * YearSplit.get(ci([l,
rlty[3]])) for l in TIMESLICE]) * AvailabilityFactor.get(ci([rlty[0], *rlty[2:4]]),
dflt.get('AvailabilityFactor')) * CapacityToActivityUnit.get(ci([rlty[0], rlty[2]]),
dflt.get('CapacityToActivityUnit')), ""

        # ===== Energy Balance A =====

        for rflmty in
REGION_FUEL_TIMESLICE_MODE_OF_OPERATION_TECHNOLOGY_YEAR:

            # EBa1_RateOfFuelProduction1

            if OutputActivityRatio.get(ci([*rflmty[0:2], *rflmty[3:6]]),
dflt.get('OutputActivityRatio')) != 0:

                model += RateOfProductionByTechnologyByMode.get(ci(rflmty)) ==
RateOfActivity.get(ci([rflmty[0], *rflmty[2:6]])) * OutputActivityRatio.get(ci([*rflmty[0:2],
*rflmty[3:6]]), dflt.get('OutputActivityRatio')), ""

```



```

else:

    model += RateOfProductionByTechnologyByMode.get(ci(rflmty)) == 0, ""

    # EBa4_RateOfFuelUse1t

    if InputActivityRatio.get(ci([*rflmty[0:2], *rflmty[3:6]]),
dflt.get('InputActivityRatio')) != 0:

        model += RateOfUseByTechnologyByMode.get(ci(rflmty)) ==
RateOfActivity.get(ci([rflmty[0], *rflmty[2:6]])) * (1 /
InputActivityRatio.get(ci([*rflmty[0:2], *rflmty[3:6]]), dflt.get('InputActivityRatio'))), ""

for rflty in REGION_FUEL_TIMESLICE_TECHNOLOGY_YEAR:

    # EBa2_RateOfFuelProduction2

    model += RateOfProductionByTechnology.get(ci(rflty)) ==
pulp.lpSum([RateOfProductionByTechnologyByMode.get(ci([*rflty[0:3], m, *rflty[3:5]]))
for m in MODE_OF_OPERATION if OutputActivityRatio.get(ci([*rflty[0:2], m,
*rflty[3:5]]), dflt.get('OutputActivityRatio')) != 0]), ""

    # EBa5_RateOfFuelUse2

    model += RateOfUseByTechnology.get(ci(rflty)) ==
pulp.lpSum([RateOfUseByTechnologyByMode.get(ci([*rflty[0:3], m, *rflty[3:5]])) for m
in MODE_OF_OPERATION if InputActivityRatio.get(ci([*rflty[0:2], m, *rflty[3:5]]),
dflt.get('InputActivityRatio')) != 0]), ""

for rfly in REGION_FUEL_TIMESLICE_YEAR:

    # EBa3_RateOfFuelProduction3

    model += RateOfProduction.get(ci(rfly)) ==
pulp.lpSum([RateOfProductionByTechnology.get(ci([*rfly[0:3], t, rfly[3]])) for t in
TECHNOLOGY]), ""

    # EBa6_RateOfFuelUse3

    # model += RateOfUse.get(ci(rfly)) ==
pulp.lpSum([RateOfUseByTechnology.get(ci([*rfly[0:3], t, rfly[3]])) for t in
TECHNOLOGY]), ""

    # EBa7_EnergyBalanceEachTS1

```

```

        model += Production.get(ci(rfly)) == RateOfProduction.get(ci(rfly)) *
YearSplit.get(ci(rfly[2:4])), ""

        # EBa8_EnergyBalanceEachTS2

        # model += Use.get(ci(rfly)) == RateOfUse.get(ci(rfly)) *
YearSplit.get(ci(rfly[2:4])), ""

        model += Use.get(ci(rfly)) ==
pulp.lpSum([RateOfUseByTechnology.get(ci([*rfly[0:3], t, rfly[3]])) for t in
TECHNOLOGY] * YearSplit.get(ci(rfly[2:4])), ""

        # EBa9_EnergyBalanceEachTS3

        model += Demand.get(ci(rfly)) == RateOfDemand.get(ci(rfly)) *
YearSplit.get(ci(rfly[2:4])), ""

        # EBa11_EnergyBalanceEachTS5

        model += Production.get(ci(rfly)) >= Demand.get(ci(rfly)) + Use.get(ci(rfly)) +
(GIS_Losses.get(ci([*rfly[0:2]]), dflt.get('GIS_Losses')) * (8760 /
int(max(TIMESLICE)))) + pulp.lpSum([Trade.get(ci([rfly[0], rr, *rfly[1:4]])) *
TradeRoute.get(ci([rfly[0], rr, rfly[1], rfly[3]]), dflt.get('TradeRoute')) for rr in REGION2]),
""

for rr2fly in REGION_REGION2_FUEL_TIMESLICE_YEAR:

        # EBa10_EnergyBalanceEachTS4

        model += Trade.get(ci(rr2fly)) == -Trade.get(ci([rr2fly[1], rr2fly[0], *rr2fly[2:5]])),
""

# ===== Energy Balance B =====

for rfy in REGION_FUEL_YEAR:

        # EBb1_EnergyBalanceEachYear1

```

```

        model += ProductionAnnual.get(ci(rfy)) ==
pulp.lpSum([Production.get(ci([*rfy[0:2], l, rfy[2]])) for l in TIMESLICE]), ""

        # Ebb2_EnergyBalanceEachYear2

        # model += UseAnnual.get(ci(rfy)) == pulp.lpSum([Use.get(ci([rfy[0], l,
*rfy[1:3]])) for l in TIMESLICE]), ""

# for rr2fy in REGION_REGION2_FUEL_YEAR:

# # Ebb3_EnergyBalanceEachYear3

# model += TradeAnnual.get(ci(rr2fy)) == pulp.lpSum([Trade.get(ci([*rr2fy[0:2],
l, *rr2fy[2:4]])) for l in TIMESLICE]), ""

#

# for rfy in REGION_FUEL_YEAR:

# Ebb4_EnergyBalanceEachYear4

# model += ProductionAnnual.get(ci(rfy)) >= UseAnnual.get(ci(rfy)) +
pulp.lpSum([TradeAnnual.get(ci([rfy[0], rr, *rfy[1:3]])) * TradeRoute.get(ci([rfy[0], rr,
*rfy[1:3]]), dflt.get('TradeRoute')) for rr in REGION2]) +
AccumulatedAnnualDemand.get(ci(rfy), dflt.get('AccumulatedAnnualDemand')), ""

#model += ProductionAnnual.get(ci(rfy)) >= pulp.lpSum([Use.get(ci([rfy[0], l,
*rfy[1:3]])) for l in TIMESLICE])+ pulp.lpSum([pulp.lpSum([Trade.get(ci([rfy[0], rr, l,
*rfy[1:3]])) for l in TIMESLICE]) * TradeRoute.get(ci([rfy[0], rr, *rfy[1:3]]),
dflt.get('TradeRoute')) for rr in REGION2]) + AccumulatedAnnualDemand.get(ci(rfy),
dflt.get('AccumulatedAnnualDemand')), ""

# ===== Accounting Technology Production/Use =====

for rfty in REGION_FUEL_TIMESLICE_TECHNOLOGY_YEAR:

# Acc1_FuelProductionByTechnology

        model += ProductionByTechnology.get(ci(rfty)) ==
pulp.lpSum([RateOfProductionByTechnologyByMode.get(ci([*rfty[0:3], m, *rfty[3:5]]))

```



```

for m in MODE_OF_OPERATION if OutputActivityRatio.get(ci(['*rflty[0:2]', m,
*rflty[3:5]]), dflt.get('OutputActivityRatio')) != 0) * YearSplit.get(ci(['rflty[2]', rflty[4]])), ""

    # Acc2_FuelUseByTechnology

    model += UseByTechnology.get(ci(rflty)) ==
RateOfUseByTechnology.get(ci(rflty)) * YearSplit.get(ci(['rflty[2]', rflty[4]])), ""

    for rfty in REGION_TIMESLICE_TECHNOLOGY_YEAR:

        # Acc1_FuelProductionByTechnology

        model += ProductionFromTechnology.get(ci(rfty)) ==
pulp.lpSum([ProductionByTechnology.get(ci(['*rfty[0:1]', f, *rfty[1:4]]) for f in FUEL)] , ""

    for rnty in REGION_MODE_OF_OPERATION_TECHNOLOGY_YEAR:

        # Acc3_AverageAnnualRateOfActivity

        model += TotalAnnualTechnologyActivityByMode.get(ci(rnty)) ==
pulp.lpSum([RateOfActivity.get(ci(['rnty[0]', l, *rnty[1:4]]) * YearSplit.get(ci(['l', rnty[3]]))
for l in TIMESLICE]), ""

    for r in REGION:

        # Acc4_ModelPeriodCostByRegion

        model += ModelPeriodCostByRegion.get(r) ==
pulp.lpSum([TotalDiscountedCost.get(ci(['r', y])) for y in YEAR]), ""

#

# # ===== Storage Equations =====

#           for           rldhlssy           in
REGION_DAYTYPE_DAILYTIMEBRACKET_SEASON_STORAGE_YEAR:

#     # S1_RateOfStorageCharge

#           model += RateOfStorageCharge.get(ci(rldhlssy)) ==
pulp.lpSum([RateOfActivity.get(ci(['rldhlssy[0]',           *lmt,           rldhlssy[5]]))           *

```



```

TechnologyToStorage.get(ci([rldhlssy[0],
dflt.get('TechnologyToStorage')) * Conversionls.get(ci([lmt[0],
dflt.get('Conversionls')) * Conversionld.get(ci([lmt[0],
dflt.get('Conversionld')) * Conversionlh.get(ci([lmt[0],
dflt.get('Conversionlh')) for lmt in
TIMESLICE_MODE_OF_OPERATION_TECHNOLOGY if
TechnologyToStorage.get(ci([rldhlssy[0], lmt[1], rldhlssy[4], lmt[2]]),
dflt.get('TechnologyToStorage')) > 0)), ""

# # S2_RateOfStorageDischarge

# model += RateOfStorageDischarge.get(ci(rldhlssy)) ==
pulp.lpSum([RateOfActivity.get(ci([rldhlssy[0], *lmt, rldhlssy[5]])) *
TechnologyFromStorage.get(ci([rldhlssy[0], *lmt[1:3], rldhlssy[3]]),
dflt.get('TechnologyFromStorage')) * Conversionls.get(ci([lmt[0], rldhlssy[3]]),
dflt.get('Conversionls')) * Conversionld.get(ci([lmt[0], rldhlssy[1]]),
dflt.get('Conversionld')) * Conversionlh.get(ci([lmt[0], rldhlssy[2]]),
dflt.get('Conversionlh')) for lmt in
TIMESLICE_MODE_OF_OPERATION_TECHNOLOGY if
TechnologyFromStorage.get(ci([rldhlssy[0], lmt[1], rldhlssy[4], lmt[2]]),
dflt.get('TechnologyFromStorage')) > 0)), ""

# # S3_NetChargeWithinYear

# model += NetChargeWithinYear.get(ci(rldhlssy)) ==
pulp.lpSum([(RateOfStorageCharge.get(ci(rldhlssy)) -
RateOfStorageDischarge.get(ci(rldhlssy))) * YearSplit.get(ci([l, rldhlssy[5]])) *
Conversionls.get(ci([l, rldhlssy[3]]), dflt.get('Conversionls')) * Conversionld.get(ci([l,
rldhlssy[1]]), dflt.get('Conversionld')) * Conversionlh.get(ci([l, rldhlssy[2]]),
dflt.get('Conversionlh')) for l in TIMESLICE if (Conversionls.get(ci([l, rldhlssy[3]]),
dflt.get('Conversionls')) > 0) and (Conversionld.get(ci([l, rldhlssy[1]]),
dflt.get('Conversionld')) > 0) and (Conversionlh.get(ci([l, rldhlssy[2]]),
dflt.get('Conversionlh')) > 0)]), ""

# # S4_NetChargeWithinDay

# model += NetChargeWithinDay.get(ci(rldhlssy)) ==
(RateOfStorageCharge.get(ci(rldhlssy)) - RateOfStorageDischarge.get(ci(rldhlssy)))
* DaySplit.get(ci([rldhlssy[2], rldhlssy[5]]), dflt.get('DaySplit')), ""

# for rsy in REGION_STORAGE_YEAR:

# # S5_and_S6_StorageLevelYearStart

# if int(rsy[2]) == int(min(YEAR)):

```



```

#          model += StorageLevelYearStart.get(ci(rsy)) ==
StorageLevelStart.get(ci(rsy[0:2]), dflt.get('StorageLevelStart')), ""

#   else:

#          model += StorageLevelYearStart.get(ci(rsy)) ==
StorageLevelYearStart.get(ci([*rsy[0:2],          str(int(rsy[2])-1]))          +
pulp.lpSum([NetChargeWithinYear.get(ci([*rsy[0:2], *ldhls, str(int(rsy[2])-1])))) for ldhls
in DAYTYPE_DAILYTIMEBRACKET_SEASON]), ""

#   # S7_and_S8_StorageLevelYearFinish

#   if int(rsy[2]) < int(max(YEAR)):

#          model += StorageLevelYearFinish.get(ci(rsy)) ==
StorageLevelYearStart.get(ci([*rsy[0:2], str(int(rsy[2])-1)])), ""

#   else:

#          model += StorageLevelYearFinish.get(ci(rsy)) ==
StorageLevelYearStart.get(ci(rsy))          +
pulp.lpSum([NetChargeWithinYear.get(ci([*rsy[0:2], *ldhls, rsy[2]]) for ldhls in
DAYTYPE_DAILYTIMEBRACKET_SEASON]), ""

#   for rlssy in REGION_SEASON_STORAGE_YEAR:

#   # S9_and_S10_StorageLevelSeasonStart

#   if int(rlssy[1]) == int(min(SEASON)):

#          model += StorageLevelSeasonStart.get(ci(rlssy)) ==
StorageLevelYearStart.get(ci([rlssy[0], *rlssy[2:4]])), ""

#   else:

#          model += StorageLevelSeasonStart.get(ci(rlssy)) ==
StorageLevelSeasonStart.get(ci([rlssy[0],          str(int(rlssy[1])-1),          *rlssy[2:4]]))          +
pulp.lpSum([NetChargeWithinYear.get(ci([rlssy[0],          str(int(rlssy[1])-1),          *ldh,
*rlssy[2:4]])) for ldh in DAYTYPE_DAILYTIMEBRACKET]), ""

#   for rldlssy in REGION_DAYTYPE_SEASON_STORAGE_YEAR:

#   # S11_and_S12_StorageLevelDayTypeStart

```



```

#     if int(rldlssy[1]) == int(min(DAYTYPE)):

#         model += StorageLevelDayTypeStart.get(ci(rldlssy)) ==
StorageLevelSeasonStart.get(ci([rldlssy[0], *rldlssy[2:5]])), ""

#     else:

#         model += StorageLevelDayTypeStart.get(ci(rldlssy)) ==
StorageLevelDayTypeStart.get(ci([rldlssy[0], str(int(rldlssy[1])-1), *rldlssy[2:5]])) +
pulp.lpSum([NetChargeWithinDay.get(ci([rldlssy[0], str(int(rldlssy[1])-1), lh,
rldlssy[2:5]])) * DaysInDayType.get(ci([rldlssy[2], str(int(rldlssy[1])-1), rldlssy[4]]),
dflt.get('DaysInDayType')) for lh in DAILYTIMEBRACKET]), ""

#     # S13_and_S14_and_S15_StorageLevelDayTypeFinish

#         if (int(rldlssy[1]) == int(max(DAYTYPE))) and (int(rldlssy[2]) ==
int(max(SEASON))):

#             model += StorageLevelDayTypeFinish.get(ci(rldlssy)) ==
StorageLevelYearFinish.get(ci([rldlssy[0], *rldlssy[3:5]])), ""

#         elif int(rldlssy[1]) == int(max(DAYTYPE)):

#             model += StorageLevelDayTypeFinish.get(ci(rldlssy)) ==
StorageLevelSeasonStart.get(ci([rldlssy[0], str(int(rldlssy[2])+1), *rldlssy[3:5]])), ""

#         else:

#             model += StorageLevelDayTypeFinish.get(ci(rldlssy)) ==
StorageLevelDayTypeFinish.get(ci([rldlssy[0], rldlssy[2], str(int(rldlssy[1])+1),
*rldlssy[3:5]])) - pulp.lpSum([NetChargeWithinDay.get(ci([rldlssy[0], str(int(rldlssy[1])-
1), lh, rldlssy[2:5]])) * DaysInDayType.get(ci([rldlssy[2], str(int(rldlssy[1])-1), rldlssy[4]]),
dflt.get('DaysInDayType')) for lh in DAILYTIMEBRACKET]), ""

#             ===== Storage Constraints =====

#                 for rldlhssy in
REGION_DAYTYPE_DAILYTIMEBRACKET_SEASON_STORAGE_YEAR:

#                     #
SC1_LowerLimit_BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInFirstWe
ekConstraint

#             model += (StorageLevelDayTypeStart.get(ci([*rldlhssy[0:2], *rldlhssy[3:6]]))
+ pulp.lpSum([NetChargeWithinDay.get(ci([*rldlhssy[0:2], lh, *rldlhssy[3:6]]) for lh in

```



```

in DAILYTIMEBRACKET if int(rldlhssy[2])-int(lh) > 0)) -
StorageLowerLimit.get(ci([rldlhssy[0], *rldlhssy[4:6]]) >= 0, ""

# #
SC1_UpperLimit_BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInFirstWeekConstraint

# model += (StorageLevelDayTypeStart.get(ci([*rldlhssy[0:2], *rldlhssy[3:6]]))
+ pulp.lpSum([NetChargeWithinDay.get(ci([*rldlhssy[0:2], lh, *rldlhssy[3:6]]) for lh in
DAILYTIMEBRACKET if int(rldlhssy[2])-int(lh) > 0)) -
StorageUpperLimit.get(ci([rldlhssy[0], *rldlhssy[4:6]]) <= 0, ""

# #
SC2_LowerLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInFirstWeekConstraint

# if int(rldlhssy[1]) > int(min(DAYTYPE)):

# model += (StorageLevelDayTypeStart.get(ci([*rldlhssy[0:2], *rldlhssy[3:6]]))
- pulp.lpSum([NetChargeWithinDay.get(ci([*rldlhssy[0:2], lh, str(int(rldlhssy[3])-1),
*rldlhssy[4:6]]) for lh in DAILYTIMEBRACKET if int(rldlhssy[2])-int(lh) < 0)) -
StorageLowerLimit.get(ci([rldlhssy[0], *rldlhssy[4:6]]) >= 0, ""

# #
SC2_LowerLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInFirstWeekConstraint

# if int(rldlhssy[1]) > int(min(DAYTYPE)):

# model += (StorageLevelDayTypeStart.get(ci([*rldlhssy[0:2], *rldlhssy[3:6]]))
- pulp.lpSum([NetChargeWithinDay.get(ci([*rldlhssy[0:2], lh, str(int(rldlhssy[3])-1),
*rldlhssy[4:6]]) for lh in DAILYTIMEBRACKET if int(rldlhssy[2]) - int(lh) < 0)) -
StorageUpperLimit.get(ci([rldlhssy[0], *rldlhssy[4:6]]) <= 0, ""

# #
SC3_LowerLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInLastWeekConstraint

# model += (StorageLevelDayTypeFinish.get(ci([*rldlhssy[0:2], *rldlhssy[3:6]]))
- pulp.lpSum([NetChargeWithinDay.get(ci([*rldlhssy[0:2], lh, *rldlhssy[3:6]]) for lh in
DAILYTIMEBRACKET if int(rldlhssy[2]) - int(lh) < 0)) -
StorageLowerLimit.get(ci([rldlhssy[0], *rldlhssy[4:6]]) >= 0, ""

# #
SC3_UpperLimit_EndOfDailyTimeBracketOfLastInstanceOfDayTypeInLastWeekConstraint

```



```

#      model += (StorageLevelDayTypeFinish.get(ci([*rldlhssy[0:2], *rldlhssy[3:6]]))
- pulp.lpSum([NetChargeWithinDay.get(ci([*rldlhssy[0:2], lhlh, *rldlhssy[3:6]])) for lhlh
in DAILYTIMEBRACKET if int(rldlhssy[2]) - int(lhlh) < 0)) -
StorageUpperLimit.get(ci([rldlhssy[0], *rldlhssy[4:6]])) <= 0, ""

#                                                                 #
SC4_LowerLimit_BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInLastWe
ekConstraint

#      if int(rldlhssy[1]) > int(min(DAYTYPE)):

#          model += (StorageLevelDayTypeFinish.get(ci([rldlhssy[0],
str(int(rldlhssy[1])-1),
*rldlhssy[3:6]]))
+
pulp.lpSum([NetChargeWithinDay.get(ci([*rldlhssy[0:2], lhlh, *rldlhssy[3:6]])) for lhlh
in DAILYTIMEBRACKET if int(rldlhssy[2]) - int(lhlh) > 0)) -
StorageLowerLimit.get(ci([rldlhssy[0], *rldlhssy[4:6]])) >= 0, ""

#                                                                 #
SC4_UpperLimit_BeginningOfDailyTimeBracketOfFirstInstanceOfDayTypeInLastWe
ekConstraint

#      if int(rldlhssy[1]) > int(min(DAYTYPE)):

#          model += (StorageLevelDayTypeFinish.get(ci([rldlhssy[0],
str(int(rldlhssy[1])-1),
*rldlhssy[3:6]]))
+
pulp.lpSum([NetChargeWithinDay.get(ci([*rldlhssy[0:2], lhlh, *rldlhssy[3:6]])) for lhlh
in DAILYTIMEBRACKET if int(rldlhssy[2]) - int(lhlh) > 0)) -
StorageUpperLimit.get(ci([rldlhssy[0], *rldlhssy[4:6]])) <= 0, ""

#      # SC5_MaxChargeConstraint

#          model += RateOfStorageCharge.get(ci(rldlhssy)) <=
StorageMaxChargeRate.get(ci(rldlhssy[4:6]), dflt.get('StorageMaxChargeRate')), ""

#      # SC6_MaxDischargeConstraint

#          model += RateOfStorageDischarge.get(ci(rldlhssy)) <=
StorageMaxDischargeRate.get(ci(rldlhssy[4:6]),
dflt.get('StorageMaxDischargeRate')), ""

```

===== Storage equations for Thermal storage - =====



for rsy in REGION_STORAGE_YEAR:

if(StorageL2D.get(ci(rsy[0:2]), dflt.get('StorageL2D')) == 0):

```

    model += StorageSurfaceArea.get(ci(rsy)) == 0.0744 *
    AccumulatedNewStorageCapacity.get(ci(rsy)) * Storagetagheating.get(ci(rsy[0:2]),
    dflt.get('Storagetagheating')) + 0.0361 * AccumulatedNewStorageCapacity.get(ci(rsy))
    * Storagetagcooling.get(ci(rsy[0:2]), dflt.get('Storagetagcooling')), ""

```

elif(StorageL2D.get(ci(rsy[0:2]), dflt.get('StorageL2D')) == 1):

```

    model += StorageSurfaceArea.get(ci(rsy)) == 0.1339 *
    AccumulatedNewStorageCapacity.get(ci(rsy)) * Storagetagheating.get(ci(rsy[0:2]),
    dflt.get('Storagetagheating')) + 0.065 * AccumulatedNewStorageCapacity.get(ci(rsy)) *
    Storagetagcooling.get(ci(rsy[0:2]), dflt.get('Storagetagcooling')), ""

```

#SL1_Storage_losses_thermal_storage

if(StorageL2D.get(ci(rsy[0:2]), dflt.get('StorageL2D')) == 0):

```

    # model += StorageSurfaceArea.get(ci(rsy)) == 0.0361 *
    AccumulatedNewStorageCapacity.get(ci(rsy)) * Storagetagcooling.get(ci(rsy[0:2]),
    dflt.get('Storagetagcooling')), "", ""

```

elif(StorageL2D.get(ci(rsy[0:2]), dflt.get('StorageL2D')) == 1):

```

    # model += StorageSurfaceArea.get(ci(rsy)) == 0.065 *
    AccumulatedNewStorageCapacity.get(ci(rsy)) * Storagetagcooling.get(ci(rsy[0:2]),
    dflt.get('Storagetagcooling')), "", ""

```

for rsly in REGION_STORAGE_TIMESLICE_YEAR:

#SL1_Storage_losses

#if (StorageL2D.get(ci([*rsly[0:2], rsly[3]]), dflt.get('StorageL2D')) == 0):

```

    #model += StorageLosses.get(ci([*rsly[0:2], str(int(rsly[2])-1)])) == 1.5374 *
    (8.76 / int(max(TIMESLICE))) * 0.0036 * (StorageUvalue.get(ci([*rsly[0:2], rsly[3]]),
    dflt.get('StorageUvalue'))) * (((StorageFlowTemperature.get(ci([*rsly[0:2], rsly[3]]),
    dflt.get('StorageFlowTemperature'))) - (StorageReturnTemperature.get(ci([*rsly[0:2],
    rsly[3]]), dflt.get('StorageReturnTemperature'))))
    StorageLevelTimesliceStart.get(ci(rsly))) +

```



```

(((StorageReturnTemperature.get(ci[*rsly[0:2], rsly[3]],
dflt.get('StorageReturnTemperature'))
-
(StorageAmbientTemperature.get(ci[*rsly[0:2], rsly[3]],
dflt.get('StorageAmbientTemperature')))) * StorageUpperLimit.get(ci[*rsly[0:2],
rsly[3]]))) , ""

```

```

model += StorageLossesheating.get(ci(rsly)) ==
(StorageSurfaceArea.get(ci[*rsly[0:2], rsly[3]]) * 0.0036 * (8760 /
int(max(TIMESLICE))) * (StorageUvalue.get(ci[*rsly[0:2], rsly[3]],
dflt.get('StorageUvalue')) * (((StorageFlowTemperature.get(ci[*rsly[0:2], rsly[3]],
dflt.get('StorageFlowTemperature')) + (StorageReturnTemperature.get(ci[*rsly[0:2],
rsly[3]], dflt.get('StorageReturnTemperature')))) / 2) -
(StorageAmbientTemperature.get(ci[*rsly[0:2], rsly[3]],
dflt.get('StorageAmbientTemperature')))) / 1000 * Storagetagheating.get(ci(rsly[0:2],
dflt.get('Storagetagheating')))) , ""

```

```

model += StorageLossescooling.get(ci(rsly)) ==
(StorageSurfaceArea.get(ci[*rsly[0:2], rsly[3]]) * 0.0036 * (8760 /
int(max(TIMESLICE))) * (StorageUvalue.get(ci[*rsly[0:2], rsly[3]],
dflt.get('StorageUvalue')) * ((StorageAmbientTemperature.get(ci[*rsly[0:2], rsly[3]],
dflt.get('StorageAmbientTemperature'))
-
(((StorageFlowTemperature.get(ci[*rsly[0:2], rsly[3]],
dflt.get('StorageFlowTemperature')) + (StorageReturnTemperature.get(ci[*rsly[0:2],
rsly[3]], dflt.get('StorageReturnTemperature')))) / 2)) / 1000 *
Storagetagcooling.get(ci(rsly[0:2], dflt.get('Storagetagcooling')))) , ""

```

```

model += StorageLosses.get(ci(rsly)) == StorageLossesheating.get(ci(rsly)) +
StorageLossescooling.get(ci(rsly)), ""

```

```

for rsy in REGION_STORAGE_YEAR:

```

```

    #S5_and_S6_StorageLevelYearStart

```

```

    if int(rsy[2]) == int(min(YEAR)):

```




```

        model          +=          StorageLevelYearStart.get(ci(rsy))          ==
StorageLevelStart.get(ci(rsy[0:2]), dflt.get('StorageLevelStart')), ""

```

else:

```

        model          +=          StorageLevelYearStart.get(ci(rsy))          ==
StorageLevelYearStart.get(ci([*rsy[0:2],          str(int(rsy[2])-1]))          +
pulp.lpSum([((RateOfStorageCharge.get(ci([*rsy[0:2],          l,          str(int(rsy[2])-1)))          -
RateOfStorageDischarge.get(ci([*rsy[0:2],          l,          str(int(rsy[2])-1))))          * YearSplit.get(ci([l,
str(int(rsy[2])-1)])) for l in TIMESLICE]), ""

```

for rsly in REGION_STORAGE_TIMESLICE_YEAR:

S1_RateOfStorageCharge

```

        model          +=          RateOfStorageCharge.get(ci(rsly))          ==
pulp.lpSum([RateOfActivity.get(ci([rsly[0],          rsly[2],          *mt,          rsly[3]])          *
TechnologyToStorage.get(ci([*rsly[0:2], *mt]), dflt.get('TechnologyToStorage')) for mt
in          MODE_OF_OPERATION_TECHNOLOGY          if
TechnologyToStorage.get(ci([*rsly[0:2], *mt]), dflt.get('TechnologyToStorage')) > 0]),
""

```

S2_RateOfStorageDischarge

```

        model          +=          RateOfStorageDischarge.get(ci(rsly))          ==
pulp.lpSum([RateOfActivity.get(ci([rsly[0],          rsly[2],          *mt,          rsly[3]])          *
TechnologyFromStorage.get(ci([*rsly[0:2], *mt]), dflt.get('TechnologyFromStorage'))
for          mt          in          MODE_OF_OPERATION_TECHNOLOGY          if
TechnologyFromStorage.get(ci([*rsly[0:2], *mt]), dflt.get('TechnologyFromStorage')) >
0]), ""

```

for rsly in REGION_STORAGE_TIMESLICE_YEAR:

#S1_and_S2_StorageLevelTimesliceStart

if int(rsly[2]) == int(min(TIMESLICE)):

```

        model          +=          StorageLevelTimesliceStart.get(ci(rsly))          ==
StorageLevelYearStart.get(ci([*rsly[0:2], rsly[3]])), ""

```

else:

```

        model          +=          StorageLevelTimesliceStart.get(ci(rsly))          ==
StorageLevelTimesliceStart.get(ci([*rsly[0:2],          str(int(rsly[2])-1),          rsly[3]])          -

```



```
StorageLosses.get(ci([*rsly[0:2], str(int(rsly[2])-1), rsly[3]])) +
((RateOfStorageCharge.get(ci([*rsly[0:2], str(int(rsly[2])-1), rsly[3]])) -
RateOfStorageDischarge.get(ci([*rsly[0:2], str(int(rsly[2])-1), rsly[3]]))) *
YearSplit.get(ci([str(int(rsly[2])-1), rsly[3]]))), ""
```

for rs in REGION_STORAGE:

#SC8_StorageRefilling

```
model += 0 == pulp.lpSum([RateOfActivity.get(ci([rs[0], *lnty])) *
TechnologyToStorage.get(ci([*rs[0:2], *lnty[1:3]]), dflt.get('TechnologyToStorage')) *
YearSplit.get(ci([lnty[0], lnty[3]])) for lnty in
TIMESLICE_MODE_OF_OPERATION_TECHNOLOGY_YEAR if
TechnologyToStorage.get(ci([*rs[0:2], *lnty[1:3]]), dflt.get('TechnologyToStorage')) >
0]) - pulp.lpSum([RateOfActivity.get(ci([rs[0], *lnty])) *
TechnologyFromStorage.get(ci([*rs[0:2], *lnty[1:3]]),
dflt.get('TechnologyFromStorage')) * YearSplit.get(ci([lnty[0], lnty[3]])) for lnty in
TIMESLICE_MODE_OF_OPERATION_TECHNOLOGY_YEAR if
TechnologyFromStorage.get(ci([*rs[0:2], *lnty[1:3]]),
dflt.get('TechnologyFromStorage')) > 0]), ""
```

#===== Storage Constraints =====

for rsy in REGION_STORAGE_YEAR:

SI3_TotalNewStorage

```
model += AccumulatedNewStorageCapacity.get(ci(rsy)) ==
pulp.lpSum([NewStorageCapacity.get(ci([*rsy[0:2], yy])) for yy in YEAR if
(float(int(rsy[2]) - int(yy)) < float(OperationalLifeStorage.get(ci(rsy[0:2]),
dflt.get('OperationalLifeStorage')))) and (int(rsy[2])-int(yy) >= 0)]), ""
```

SI1_StorageUpperLimit

```
model += StorageUpperLimit.get(ci(rsy)) ==
(AccumulatedNewStorageCapacity.get(ci(rsy)) +
ResidualStorageCapacity.get(ci(rsy), dflt.get('ResidualStorageCapacity'))), ""
```



```

# SI1_StorageMaxCapacity

model += StorageUpperLimit.get(ci(rsly)) <=
StorageMaxCapacity.get(ci(rsly[0:2]), dflt.get('StorageMaxCapacity')), ""

for rsly in REGION_STORAGE_TIMESLICE_YEAR:

#SC1_LowerLimit

model += StorageLevelTimesliceStart.get(ci(rsly)) >=
MinStorageCharge.get(ci(['rsly[0:2]', rsly[3]]), dflt.get('MinStorageCharge')) *
StorageUpperLimit.get(ci(['rsly[0:2]', rsly[3]])), ""

#SC2_Upper_Limit

model += StorageLevelTimesliceStart.get(ci(rsly)) <=
StorageUpperLimit.get(ci(['rsly[0:2]', rsly[3]])), ""

#SC3_Charging_Upper_Limit

#model += StorageMaxChargeRate.get(ci(['rsly[0:2]', rsly[3]]),
dflt.get('StorageMaxChargeRate')) >= StorageLevelTimesliceStart.get(ci(rsly)) -
StorageLevelTimesliceStart.get(ci(['rsly[0:2]', str(int(rsly[2])-1), rsly[3]])), ""

#SC4_Charging_Lower_Limit

#model += StorageMaxDischargeRate.get(ci(['rsly[0:2]', rsly[3]]),
dflt.get('StorageMaxDischargeRate')) >= StorageLevelTimesliceStart.get(ci(['rsly[0:2]',
str(int(rsly[2])-1), rsly[3]])) - StorageLevelTimesliceStart.get(ci(rsly)), ""

# ===== Storage Investments =====

# SI4_UndiscountedCapitalInvestmentStorage

for rsly in REGION_STORAGE_YEAR:

```

```

        model += CapitalInvestmentStorage.get(ci(rsy)) ==
CapitalCostStorage.get(ci(rsy), dflt.get('CapitalCostStorage')) *
NewStorageCapacity.get(ci(rsy)), ""

# S15_DiscountingCapitalInvestmentStorage

        model += DiscountedCapitalInvestmentStorage.get(ci(rsy)) ==
CapitalInvestmentStorage.get(ci(rsy)) * (1/ ((1+DiscountRateSto.get(ci(rsy[0:2]),
dflt.get('DiscountRateSto')))**(int(rsy[2]) - int(min(YEAR))))), ""

# S16_SalvageValueStorageAtEndOfPeriod1

        if float(int(rsy[2]) + OperationalLifeStorage.get(ci(rsy[0:2]),
dflt.get('OperationalLifeStorage')) - 1 <= float(max(YEAR))):

            model += SalvageValueStorage.get(ci(rsy)) == 0, ""

# S17_SalvageValueStorageAtEndOfPeriod2

        if ((DepreciationMethod.get(rsy[0], dflt.get('DepreciationMethod')) == 1) and
(float(int(rsy[2])+OperationalLifeStorage.get(ci(rsy[0:2]),
dflt.get('OperationalLifeStorage'))-1) > float(max(YEAR))) and
(DiscountRateSto.get(ci(rsy[0:2]), dflt.get('DiscountRateSto')) == 0)) or
((DepreciationMethod.get(rsy[0], dflt.get('DepreciationMethod')) == 2) and
(float(int(rsy[2])+OperationalLifeStorage.get(ci(rsy[0:2]),
dflt.get('OperationalLifeStorage'))-1) > float(max(YEAR))))):

            model += SalvageValueStorage.get(ci(rsy)) ==
CapitalInvestmentStorage.get(ci(rsy)) * (1-(int(max(YEAR))-
int(rsy[2])+1))/OperationalLifeStorage.get(ci(rsy[0:2]),
dflt.get('OperationalLifeStorage')), ""

# S18_SalvageValueStorageAtEndOfPeriod3

        if (DepreciationMethod.get(rsy[0], dflt.get('DepreciationMethod')) == 1) and
(float(int(rsy[2])+OperationalLifeStorage.get(ci(rsy[0:2]),
dflt.get('OperationalLifeStorage'))-1) > float(max(YEAR))) and
(DiscountRateSto.get(ci(rsy[0:2]), dflt.get('DiscountRateSto')) > 0):

            model += SalvageValueStorage.get(ci(rsy)) ==
CapitalInvestmentStorage.get(ci(rsy)) * (1-(((1+DiscountRateSto.get(ci(rsy[0:2]),
dflt.get('DiscountRateSto')))**(int(max(YEAR)) -
int(rsy[2])+1)-
1)/((1+DiscountRateSto.get(ci(rsy[0:2]),
dflt.get('DiscountRateSto')))**OperationalLifeStorage.get(ci(rsy[0:2]),
dflt.get('OperationalLifeStorage'))-1))), ""

# S19_SalvageValueStorageDiscountedToStartYear

```



```

model += DiscountedSalvageValueStorage.get(ci(rsy)) ==
SalvageValueStorage.get(ci(rsy)) * (1 / ((1 + DiscountRateSto.get(ci(rsy[0:2])),
dflt.get('DiscountRateSto')))**(int(max(YEAR))-int(min(YEAR))+1))), ""

```

```
# SI10_TotalDiscountedCostByStorage
```

```

model += TotalDiscountedStorageCost.get(ci(rsy)) ==
DiscountedCapitalInvestmentStorage.get(ci(rsy))-
DiscountedSalvageValueStorage.get(ci(rsy)), ""

```

```
# ===== Capital Costs =====
```

```
for rty in REGION_TECHNOLOGY_YEAR:
```

```
# CC1_UndiscountedCapitalInvestment
```

```

model += CapitalInvestment.get(ci(rty)) == CapitalCost.get(ci(rty),
dflt.get('CapitalCost')) * NewCapacity.get(ci(rty)), ""

```

```
# CC2_DiscountingCapitalInvestment
```

```

model += DiscountedCapitalInvestment.get(ci(rty)) ==
CapitalInvestment.get(ci(rty)) * (1/((1 + DiscountRateTech.get(ci(rty[0:2])),
dflt.get('DiscountRateTech')))**(int(rty[2]) - int(min(YEAR))))), ""

```

```
# ===== Business module Discounted Costs for storage and Technology =====
```

```
# ===== Storage =====
```

```
for rs in REGION_STORAGE:
```

```
# # SI9.1_SalvageValuebyStorage
```

```

model += DiscountedSalvageValueByStorage.get(ci(rs)) ==
pulp.lpSum([DiscountedSalvageValueStorage.get(ci([*rs, y])] for y in YEAR)), ""

```

```
# # SI5.1_DiscountingCapitalInvestmentbyStorageBusinessModule
```



```

model += DiscountedCapitalInvestmentByStorage.get(ci(rs)) ==
pulp.lpSum([DiscountedCapitalInvestmentStorage.get(ci([*rs, y])) for y in YEAR]), ""

```

```

for rty in REGION_TECHNOLOGY_YEAR:

```

```

# ===== Salvage Value =====

```

```

# SV1_SalvageValueAtEndOfPeriod1

```

```

if (DepreciationMethod.get(rty[0], dflt.get('DepreciationMethod')) == 1) and
(float(int(rty[2]) + OperationalLife.get(ci(rty[0:2]), dflt.get('OperationalLife'))) - 1 >
float(max(YEAR))) and (DiscountRateTech.get(ci(rty[0:2]),
dflt.get('DiscountRateTech')) > 0):

```

```

model += SalvageValue.get(ci(rty)) == CapitalCost.get(ci(rty),
dflt.get('CapitalCost')) * NewCapacity.get(ci(rty)) * (1 - (((1 +
DiscountRateTech.get(ci(rty[0:2]), dflt.get('DiscountRateTech')) ** (int(max(YEAR)) -
int(rty[2]) + 1) - 1) / ((1 + DiscountRateTech.get(ci(rty[0:2]),
dflt.get('DiscountRateTech')) ** OperationalLife.get(ci(rty[0:2]),
dflt.get('OperationalLife')) - 1))), ""

```

```

# SV2_SalvageValueAtEndOfPeriod2

```

```

if ((DepreciationMethod.get(rty[0], dflt.get('DepreciationMethod')) == 1) and
(float(int(rty[2]) + OperationalLife.get(ci(rty[0:2]), dflt.get('OperationalLife'))) - 1 >
float(max(YEAR))) and (DiscountRateTech.get(ci(rty[0:2]),
dflt.get('DiscountRateTech')) == 0) or ((DepreciationMethod.get(rty[0],
dflt.get('DepreciationMethod')) == 2) and (float(int(rty[2]) +
OperationalLife.get(ci(rty[0:2]), dflt.get('OperationalLife'))) - 1 > float(max(YEAR))))):

```

```

model += SalvageValue.get(ci(rty)) == CapitalCost.get(ci(rty),
dflt.get('CapitalCost')) * NewCapacity.get(ci(rty)) * (1 - (int(max(YEAR)) - int(rty[2]) + 1)
/ OperationalLife.get(ci(rty[0:2]), dflt.get('OperationalLife'))), ""

```

```

# SV3_SalvageValueAtEndOfPeriod3)

```

```

if float(int(rty[2]) + OperationalLife.get(ci(rty[0:2]), dflt.get('OperationalLife')) - 1)
<= float(max(YEAR)):

```

```

model += SalvageValue.get(ci(rty)) == 0, ""

```

```

# SV4_SalvageValueDiscountedToStartYear

```

```

model += DiscountedSalvageValue.get(ci(rty)) == SalvageValue.get(ci(rty)) *
(1 / ((1 + DiscountRateTech.get(ci(rty[0:2]), dflt.get('DiscountRateTech')) ** (1 +
int(max(YEAR)) - int(min(YEAR))))), ""

```

```

# ===== Operating Costs =====

```

```

# OC1_OperatingCostsVariable

```

```

model += AnnualVariableOperatingCost.get(ci(rty)) ==
pulp.lpSum([TotalAnnualTechnologyActivityByMode.get(ci([rty[0], m, *rty[1:3]])) *
VariableCost.get(ci([rty[0], m, *rty[1:3]]), dflt.get('VariableCost')) for m in
MODE_OF_OPERATION]), ""

```

```

# OC2_OperatingCostsFixedAnnual

```

```

model += AnnualFixedOperatingCost.get(ci(rty)) ==
TotalCapacityAnnual.get(ci(rty)) * FixedCost.get(ci(rty), dflt.get('FixedCost')), ""

```

```

# OC3_OperatingCostsTotalAnnual

```

```

model += OperatingCost.get(ci(rty)) == AnnualFixedOperatingCost.get(ci(rty))
+ AnnualVariableOperatingCost.get(ci(rty)), ""

```

```

# OC4_DiscountedOperatingCostsTotalAnnual

```

```

model += DiscountedOperatingCost.get(ci(rty)) == OperatingCost.get(ci(rty)) *
(1 / ((1 + DiscountRateTech.get(ci(rty[0:2]), dflt.get('DiscountRateTech')) ** (int(rty[2])
- int(min(YEAR)) + 0.5))), ""

```

```

# ===== Business module Discounted Costs for storage and Technology =====

```

```

# ===== Technology =====

```

```

for rt in REGION_TECHNOLOGY:

```

```

# # CC2.1_DiscountingCapitalInvestmentbytechnology

```

```

model += DiscountedCapitalInvestmentByTechnology.get(ci(rt)) ==
pulp.lpSum([DiscountedCapitalInvestment.get(ci([*rt, y])) for y in YEAR]), ""

```

```

# # OC4.1_DiscountedOperatingCostsbytechnologyBusinessModule

```



```

model += DiscountedOperatingCostByTechnology.get(ci(rt)) ==
pulp.lpSum([DiscountedOperatingCost.get(ci([*rt, y])) for y in YEAR], ""

```

```

# # SV4.1_DiscountedSalvageValuebytechnologyBusinessModule

```

```

model += DiscountedSalvageValueByTechnology.get(ci(rt)) ==
pulp.lpSum([DiscountedSalvageValue.get(ci([*rt, y])) for y in YEAR], ""

```

```

# ===== Total Discounted Costs =====

```

```

for ry in REGION_YEAR:

```

```

# TDC2_TotalDiscountedCost

```

```

model += TotalDiscountedCost.get(ci(ry)) ==
pulp.lpSum([TotalDiscountedCostByTechnology.get(ci([ry[0], t, ry[1]])) for t in
TECHNOLOGY]) + pulp.lpSum([TotalDiscountedStorageCost.get(ci([ry[0], s, ry[1]]))
for s in STORAGE]), ""

```

```

for rty in REGION_TECHNOLOGY_YEAR:

```

```

# TDC1_TotalDiscountedCostByTechnology

```

```

model += TotalDiscountedCostByTechnology.get(ci(rty)) ==
DiscountedOperatingCost.get(ci(rty)) + DiscountedCapitalInvestment.get(ci(rty)) +
DiscountedTechnologyEmissionsPenalty.get(ci(rty)) -
DiscountedSalvageValue.get(ci(rty)), ""

```

```

# ===== Total Capacity Constraints =====

```

```

# TCC1_TotalAnnualMaxCapacityConstraint

```

```

model += TotalCapacityAnnual.get(ci(rty)) <=
TotalAnnualMaxCapacity.get(ci(rty), dflt.get('TotalAnnualMaxCapacity')), ""

```

```

# TCC2_TotalAnnualMinCapacityConstraint

```

```

if TotalAnnualMinCapacity.get(ci(rty), dflt.get('TotalAnnualMinCapacity')) > 0:

```




```

model += TotalCapacityAnnual.get(ci(rty)) >=
TotalAnnualMinCapacity.get(ci(rty), dflt.get('TotalAnnualMaxCapacity')), ""

```

```
# ===== New Capacity Constraints =====
```

```
# NCC1_TotalAnnualMaxNewCapacityConstraint
```

```

model += NewCapacity.get(ci(rty)) <=
TotalAnnualMaxCapacityInvestment.get(ci(rty),
dflt.get('TotalAnnualMaxCapacityInvestment')), ""

```

```
# NCC2_TotalAnnualMinNewCapacityConstraint
```

```

if TotalAnnualMinCapacityInvestment.get(ci(rty),
dflt.get('TotalAnnualMinCapacityInvestment')) > 0:

```

```

model += NewCapacity.get(ci(rty)) >=
TotalAnnualMinCapacityInvestment.get(ci(rty),
dflt.get('TotalAnnualMinCapacityInvestment')), ""

```

```
# ===== Annual Activity Constraints =====
```

```
# AAC1_TotalAnnualTechnologyActivity
```

```

model += TotalTechnologyAnnualActivity.get(ci(rty)) ==
pulp.lpSum([RateOfTotalActivity.get(ci([rty[0], l, *rty[1:3]])) * YearSplit.get(ci([l, rty[2]]))
for l in TIMESLICE]), ""

```

```
# AAC2_TotalAnnualTechnologyActivityUpperLimit
```

```

model += TotalTechnologyAnnualActivity.get(ci(rty)) <=
TotalTechnologyAnnualActivityUpperLimit.get(ci(rty),
dflt.get('TotalTechnologyAnnualActivityUpperLimit')), ""

```

```
# AAC3_TotalAnnualTechnologyActivityLowerLimit
```

```

if TotalTechnologyAnnualActivityLowerLimit.get(ci(rty),
dflt.get('TotalTechnologyAnnualActivityLowerLimit')) > 0:

```



```

        model += TotalTechnologyAnnualActivity.get(ci(rty)) >=
TotalTechnologyAnnualActivityLowerLimit.get(ci(rty),
dflt.get('TotalTechnologyAnnualActivityLowerLimit')), ""

```

```
# ===== Total Activity Constraints =====
```

```
for rt in REGION_TECHNOLOGY:
```

```
    # TAC1_TotalModelHorizonTechnologyActivity
```

```

        model += TotalTechnologyModelPeriodActivity.get(ci(rt)) ==
pulp.lpSum([TotalTechnologyAnnualActivity.get(ci([*rt, y])) for y in YEAR]), ""

```

```
    # TAC2_TotalModelHorizonTechnologyActivityUpperLimit
```

```

        if TotalTechnologyModelPeriodActivityUpperLimit.get(ci(rt),
dflt.get('TotalTechnologyModelPeriodActivityUpperLimit')) > 0:

```

```

            model += TotalTechnologyModelPeriodActivity.get(ci(rt)) <=
TotalTechnologyModelPeriodActivityUpperLimit.get(ci(rt),
dflt.get('TotalTechnologyModelPeriodActivityUpperLimit')), ""

```

```
    # TAC3_TotalModelHorizenTechnologyActivityLowerLimit
```

```

        if TotalTechnologyModelPeriodActivityLowerLimit.get(ci(rt),
dflt.get('TotalTechnologyModelPeriodActivityLowerLimit')) > 0:

```

```

            model += TotalTechnologyModelPeriodActivity.get(ci(rt)) >=
TotalTechnologyModelPeriodActivityLowerLimit.get(ci(rt),
dflt.get('TotalTechnologyModelPeriodActivityLowerLimit')), ""

```

```
# ===== Reserve Margin Constraint =====
```

```
for ry in REGION_YEAR:
```

```
    # RM1_ReserveMargin_TechnologiesIncluded_In_Activity_Units
```

```

        model += TotalCapacityInReserveMargin.get(ci(ry)) ==
pulp.lpSum([TotalCapacityAnnual.get(ci([ry[0], t, ry[1]]))
ReserveMarginTagTechnology.get(ci([ry[0], t, ry[1]])),

```



```
dflt.get('ReserveMarginTagTechnology') * CapacityToActivityUnit.get(ci([ry[0], t]),
dflt.get('CapacityToActivityUnit')) for t in TECHNOLOGY), ""
```

```
for rly in REGION_TIMESLICE_YEAR:
```

```
    # RM2_ReserveMargin_FuelsIncluded
```

```
        model += DemandNeedingReserveMargin.get(ci(rly)) ==
pulp.lpSum([RateOfProduction.get(ci([rly[0], f, *rly[1:3]])) *
ReserveMarginTagFuel.get(ci([rly[0], f, rly[2]]), dflt.get('ReserveMarginTagFuel')) for f
in FUEL]), ""
```

```
    # RM3_ReserveMargin_Constraint
```

```
        model += DemandNeedingReserveMargin.get(ci(rly)) <=
TotalCapacityInReserveMargin.get(ci([rly[0], rly[2]])) * (1/ReserveMargin.get(ci([rly[0],
rly[2]]), dflt.get('ReserveMargin'))), ""
```

```
# ===== RE Production Target =====
```

```
for rfty in REGION_FUEL_TECHNOLOGY_YEAR:
```

```
    # RE1_FuelProductionByTechnologyAnnual
```

```
        model += ProductionByTechnologyAnnual.get(ci(rfty)) ==
pulp.lpSum([ProductionByTechnology.get(ci([rfty[0], l, *rfty[1:4]])) for l in TIMESLICE]),
""
```

```
for ry in REGION_YEAR:
```

```
    # RE2_TechIncluded
```

```
        # model += TotalREProductionAnnual.get(ci(ry)) ==
pulp.lpSum([ProductionByTechnologyAnnual.get(ci([ry[0], *ft, ry[1]])) *
RETagTechnology.get(ci([ry[0], ft[1], ry[1]]), dflt.get('RETagTechnology')) for ft in
FUEL_TECHNOLOGY]), ""
```

```
    # RE3_FuelIncluded
```



```

        model += RETotalProductionOfTargetFuelAnnual.get(ci(ry)) ==
pulp.lpSum([RateOfProduction.get(ci([ry[0], *fl, ry[1]])) * YearSplit.get(ci([fl[1], ry[1]])) *
RETagFuel.get(ci([ry[0], fl[0], ry[1]]), dflt.get('RETagFuel')) for fl in
FUEL_TIMESLICE]), ""

        # RE4_EnergyConstraint

        #        model += TotalREProductionAnnual.get(ci(ry)) >=
REMinProductionTarget.get(ci(ry), dflt.get('REMinProductionTarget')) *
RETtotalProductionOfTargetFuelAnnual.get(ci(ry)), ""

        # Combined: RE4_EnergyConstraint >= RE2_TechIncluded

        model += pulp.lpSum([ProductionByTechnologyAnnual.get(ci([ry[0], *ft, ry[1]]))
* RETagTechnology.get(ci([ry[0], ft[1], ry[1]]), dflt.get('RETagTechnology')) for ft in
FUEL_TECHNOLOGY]) >=
REMinProductionTarget.get(ci(ry),
dflt.get('REMinProductionTarget')) *
RETtotalProductionOfTargetFuelAnnual.get(ci(ry)), ""

        # for rfty in REGION_FUEL_TECHNOLOGY_YEAR:

        # # RE5_FuelUseByTechnologyAnnual

        #        model += UseByTechnologyAnnual.get(ci(rfty)) ==
pulp.lpSum([RateOfUseByTechnology.get(ci([*rfty[0:2], l, *rfty[2:4]]))
YearSplit.get(ci([l, rfty[3]]) for l in TIMESLICE)], ""

        # ===== Emissions Accounting =====

        for remty in
REGION_EMISSION_MODE_OF_OPERATION_TECHNOLOGY_YEAR:

        # E1_AnnualEmissionProductionByMode

        model += AnnualTechnologyEmissionByMode.get(ci(remty)) ==
EmissionActivityRatio.get(ci(remty), dflt.get('EmissionActivityRatio')) *
TotalAnnualTechnologyActivityByMode.get(ci([remty[0], *remty[2:5]])), ""

```



```

for rety in REGION_EMISSION_TECHNOLOGY_YEAR:

    # E2_AnnualEmissionProduction

    model += AnnualTechnologyEmission.get(ci(rety)) ==
    pulp.lpSum([AnnualTechnologyEmissionByMode.get(ci([*rety[0:2], m, *rety[2:4]])) for
    m in MODE_OF_OPERATION]), ""

    # E3_EmissionsPenaltyByTechAndEmission

    model += AnnualTechnologyEmissionPenaltyByEmission.get(ci(rety)) ==
    AnnualTechnologyEmission.get(ci(rety)) * EmissionsPenalty.get(ci([*rety[0:2],
    *rety[3]]), dflt.get('EmissionsPenalty')), ""

    print(EmissionsPenalty.get(ci([*rety[0:2],
    dflt.get('EmissionsPenalty'))))
    rety[3]),

```

```

for rty in REGION_TECHNOLOGY_YEAR:

    # E4_EmissionsPenaltyByTechnology

    model += AnnualTechnologyEmissionsPenalty.get(ci(rty)) ==
    pulp.lpSum([AnnualTechnologyEmissionPenaltyByEmission.get(ci([rty[0],
    *rty[1:3]])) for e in EMISSION]), ""

    # E5_DiscountedEmissionsPenaltyByTechnology

    model += DiscountedTechnologyEmissionsPenalty.get(ci(rty)) ==
    AnnualTechnologyEmissionsPenalty.get(ci(rty)) * (1 / ((1 +
    DiscountRateTech.get(ci(rty[0:2]), dflt.get('DiscountRateTech')) ** (int(rty[2]) -
    int(min(YEAR) + 0.5))), ""

```

```

for rey in REGION_EMISSION_YEAR:

    # E6_EmissionsAccounting1

    model += AnnualEmissions.get(ci(rey)) ==
    pulp.lpSum([AnnualTechnologyEmission.get(ci([*rey[0:2], t, rey[2]])) for t in
    TECHNOLOGY]), ""

    # E8_AnnualEmissionsLimit

```



```
model += AnnualEmissions.get(ci(rey)) <= AnnualEmissionLimit.get(ci(rey),
dflt.get('AnnualEmissionLimit')) - AnnualExogenousEmission.get(ci(rey),
dflt.get('AnnualExogenousEmission')), ""
```

```
for re in REGION_EMISSION:
```

```
    # E7_EmissionsAccounting2
```

```
    model += pulp.lpSum([AnnualEmissions.get(ci([*re, y])) for y in YEAR]) ==
ModelPeriodEmissions.get(ci(re)) - ModelPeriodExogenousEmission.get(ci(re),
dflt.get('ModelPeriodExogenousEmission')), ""
```

```
    # E9_ModelPeriodEmissionsLimit
```

```
    model += ModelPeriodEmissions.get(ci(re)) <=
ModelPeriodEmissionLimit.get(ci(re), dflt.get('ModelPeriodEmissionLimit')), ""
```

