Politecnico di Torino

Master's degree in Energy and Nuclear Engineering

Master's degree Thesis

A TIMES-like open-source model for the Italian energy system

Supervisor
Prof. Laura Savoldi
Co-supervisor
Daniele Lerede

Candidate
Matteo Nicoli

Academic Year 2020 - 21

“Mentre vivete la vostra vita terrena, cercate di fare qualche cosa di buono che possa rimanere dopo di voi. […] E ricordate che essere buoni è qualche cosa, ma che fare il bene è molto di più.”

Scouting for Boys, B.P.

Summary

Abstract

One of the main challenges that humanity as a whole is facing, even if still without dramatic success, is the fight against climate change, caused by the progressive increase in the average temperature of the planet that is occurring in the recent decades, and which is expected to continue throughout the next decades, if anything will be done. The scientific community is now unanimous in believing that the main cause of the increase in temperature is the increase in the average concentration of greenhouse gases in the atmosphere, in turn mainly caused by the consumption of fossil fuels to satisfy human activities.

In this context, the energy strategies that will be applied in the coming years play a crucial role. In recent years, institutions have increasingly engaged in taking initiatives aimed at fighting the causes of climate change, implementing energy transition strategies to adopt abate climate-altering emissions into the atmosphere.

Scientific research on energy system modeling plays a key role in supporting policy makers, providing accurate scenario analyses, and investigating possible solutions aimed at the development of an increasingly sustainable energy mix. The models currently used in that research field are typically commercial software that aim at satisfying given final energy service demands using different technologies, to be selected on the basis of an optimization criterion identified in the minimization of the total cost of the energy system. The use of such software involves some issues: being commercial software, they are not freely available but only by purchasing them, limiting the audience of users to those who have the necessary economic resources; moreover, access to the source code is allowed under conditions, and finally a third-party verification of the results, their reproducibility, thus scientific reliability and free access to input data and assumptions is not ensured.

In that framework, the purpose of this thesis activity is then to develop an open-source model for the Italian energy system. The model has been developed taking as a reference the TIMES-Italia model, an energy model developed by ENEA in the last years and belonging to the TIMES family.

The work first reviews and updates the TIMES-Italia database, subsequently implemented in an energy system modeling framework developed in Python language, the so-called Tools for Energy Modeling Optimization and Analysis, or Temoa, in brief. Finally, a comparison between the results obtained by the tools is performed, demonstrating that any discrepancies are within a tolerance threshold and that they can be considered equivalent for the purpose of scenario analysis.

At the end of the thesis activity, an open-source and validated model for the Italian energy system is available to the community of modelers, model for which it is possible to implement modifications and add-ons, also regarding the optimization criteria and the algorithm. In particular, possible future developments include the introduction of a sustainability criterion in the optimization paradigm, to overcome simple cost minimization and offer policy makers a detailed and useful analysis of the effect of any legislative interventions that are being taken into consideration.

List of acronyms

BOF	Basic oxygen furnace
CCS	Carbon capture and storage
CEH	Electricity-to-heat coefficient
CHP	Cogeneration of heat and power
CHPR	Heat-to-power ratio
CPLEX	IBM ILOG CPLEX Optimization Studio
EAF	Electric arc furnace
ETM	EUROfusion TIMES Model
EU	European Union
GAMS	General Algebraic Modeling System
GDP	Gross domestic product
GHG	Greenhouse gases
GLPK	GNU Linear Programming Kit
IEA	International Energy Agency
MF	Multi-family
NEOS	Network-Enabled Optimization System
O&M	Operation and maintenance
OSeMOSYS	Open Source Energy Modelling System
PNIEC	Piano Nazionale Integrato per l'Energia e il Clima
RES	Reference energy system
SF	Single-family
Temoa	Tools for Energy Modeling Optimization and Analysis
TIMES	The Integrated MARKAL-EFOM System
VEDA	VErsatile Data Analyst

List of symbols

A_{low}	Lower activity boundary
AF	Availability factor
CH_4	Methane
CO_2	Carbon dioxide
d	Driver
D	Final service demand
e	Elasticity
eff	Efficiency
err	Relative error
E	Energy consumption
EF	Emission factor
f	Share factor
N_2O	Nitrous oxide
R_f	Relaxation factor
SO_x	Sulfur oxide
P	Power
t	Time period

Subscripts

af	Aggregated fuel
avg	Average
bt	Building type
conv	Conversion
es	Energy service
eu	End use
f	Fuel
md	Machine drive
s	Steam
tech	Technology
tm	Transportation mode

Superscripts

f	Final
u	Useful

List of figures

List of tables

Chapter 1

Introduction

1.1. Climate change and Italian energy strategy

Climate change is probably the main challenge that humanity as a whole is facing, still without dramatic successes, and that it will undoubtedly have to face in the coming years and decades. The average increase in the planet's temperature has direct consequences on the climate, in particular by increasing the frequency and intensity of extreme weather phenomena (and related damage to exposed populations), melting of glaciers and rising average sea levels, endangering the delicate balance of ecosystems. The cause of the increase in average temperature is now unanimously recognized in the increase in the average concentration of greenhouse gases in the atmosphere, due to the exploitation of fossil fuels to satisfy human activities. In recent decades there has been an increase in annual emissions, driven by the growth of the world population, by economic development and by an increasingly widespread access to energy, even in developing countries. [1]

Figure 1 shows the relationship between different observed phenomena, namely: positive and increasing atmospheric temperature anomaly characterizing the last decades, with respect to the beginning of the 1850-1900 levels (Figure 1 (a)); positive and increasing sea level change (Figure 1 (b)); the increasing greenhouse gases average concentration in the atmosphere, shown for CO_2, CH_4 and N_2O (Figure 1 (c)); global anthropogenic CO_2 emissions (Figure 1 (d)).

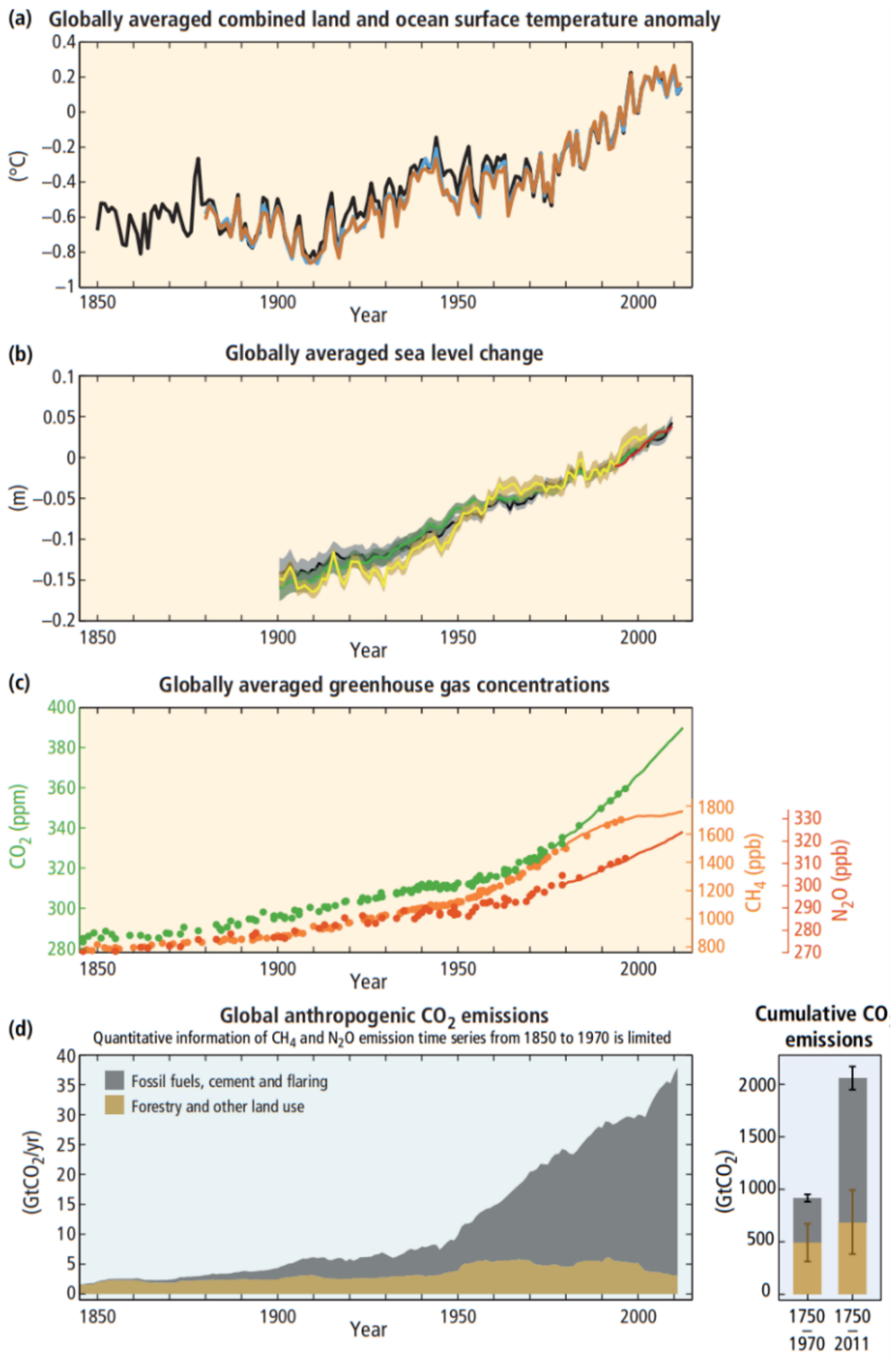


Figure 1. Relationship between the observed phenomena. (a) Annually and globally averaged temperature anomalies relative to the average over the period 1986 to 2005. (b) Annually and globally averaged sea level change relative to the average over the period 1986 to 2005. (c) Atmospheric concentration of carbon dioxide (CO_2), methane (CH_4) and nitrous oxide (N_2O). (d) Global anthropogenic CO_2 emissions [1].

Governments in recent years have devoted much attention to the issue of climate change, at least in proclamations, with the milestones of the Kyoto Protocol of 1998 [2] and the Paris Agreement of 2015 [3]. In this context, the Italian government has established qualitative objectives and quantitative targets to achieve the energy transition relying more and more on energy resources that do not involve greenhouse gas emissions.

First, the "Strategia Energetica Nazionale 2017" [4] aims to make the national energy system more competitive (reducing the energy price gap with respect to the rest of the Europe), sustainable (reaching the objectives of decarbonization established by the Paris Agreement) and secure (improving procurement security and infrastructures flexibility).

The quantitative targets established by the national energy strategy are:

a. Reducing final consumption to 108 Mtoe in 2030.
b. 28% of renewable energy resources on the total consumption in 2030 (with sector-specific targets).
c. Cessation of coal-fired electricity generation in 2025.
d. Emissions reduction of 39% in 2030 and 63% in 2050 with respect to 1990 levels.
e. Doubling of investments in research and development to 444 million euros in 2021.
f. Energy dependence reduction to 64% in 2030.

The energy strategy has been included in 2020 in the "Piano Nazionale Integrato per l'Energia e il Clima" (PNIEC) [5], aiming to reach to 30% of renewable energy sources on the total consumption in 2030 and 55% on consumption for electricity generation, with an estimation of 180 billion euros of additional investments in the period 2017-2030 with respect to the current policy scenario.

Having the objective of reducing the exploitation of fossil fuels (increasing the share of consumption of renewable energy sources for all energy sectors) requires the ability to evaluate the future energy policies taken into consideration, providing an assessment of their effectiveness in achieving the established objectives, also considering with an independent analysis the energy measures' efficiency in terms of achieved results per unit of public economic resources consumed.

1.2. Bottom-up energy system modeling

Energy system modeling and scenario analysis have a key role in the investigation of future energy systems, particularly concerning the expected evolution of the energy mix in the during large time scales. Studies developed on these themes should be more and more are increasingly becoming a reference for policy makers, in other to guarantee that energy policies are supported by robust, independent, and verifiable evaluations on their effectiveness achieving the desired effects. [6]

In this context, several tools are used to perform scenario analyzes, including the bottom-up energy models [7]. This kind of models are characterized by a detailed description of the energy technologies involved in the energy system, working with a high disaggregation level, and thus needing extensive database of empirical data to support the description of each component.

The main peculiarities of bottom-up models are [7]:

a. Exogenous assumption on the development of the economy.
b. Use of physical or engineering relationship between energy and energy utilization processes.
c. Demand driven by socio-economic variables, often derived by econometric models.
d. Usually, they do not account pricing effect on demand (which is very critical when demand for a fuel is highly elastic).

Bottom-up models can be classified according to several criteria [7]:

a. Time horizon: short-term and long-term approach. Short-term (or static) models usually analyze the energy system configuration in a target year, while long-term models inspect the energy system along the entire time horizon (usually longer) until the target year.
b. Optimization approach: perfect-foresight approach (with the formulation of a unique optimization problem analyzing all the time-periods simultaneously and having fully information on cost trends, consumption variation, decay of performance of certain technologies) or myopic approach (realized subdividing

the time horizon in a sequence of optimization problems, to simulate real decisions, taken without a complete information about the future).

c. Analyzed energy sectors: models of the entire energy system and models limited to some of the sectors included. The advantage of using a model which includes all the sectors is to have the possibility to study sector-coupling, to assess the benefits coming from interactions and synergies among different sectors, to maximize the efficiency of the whole system.
d. Geographical scale: models at the macroscale (single- or multi-regional), mesoscale, or microscale.
e. Time resolution: related to the number of time-slices in which the simulation year is subdivided. Time-slices are stylized temporal representations mainly used to describe the seasonal and daily differences of energy consumption and production. The time resolution of the models is becoming an important aspect since renewable energy sources are strongly variable in the time (being dependent on weather conditions).
f. Methodology: accounting models (simply working on the present configuration of the energy system), simulation models (allowing to test a certain configuration of the energy system and derive useful indicators related to the total cost, the GHG emissions etc.), optimization models (deriving the energy system configuration correspondent to the optimization of a certain objective function, typically the total cost of the system).

1.3. TIMES-family models

Models belonging to the TIMES framework [8] are bottom-up energy models, sharing the following common features:

a. Technology-explicit formulation: each technology in all energy sectors is explicitly described with technical and economic parameters.
b. Possibility to implement a multi-regional network (models at the macroscale).
c. Equilibrium in competitive markets with perfect foresight (optimization models).

In its current formulation, the model operates with a linear programming algorithm, and its objective function is the maximization of total economic surplus, that is equivalent to the minimization of the total cost of the energy system during the model time horizon [8].

The maximization of the total surplus is reached at the equilibrium point of the demand curve and the supply curve, as shown in Figure 2, for each energy service involved in the modelled energy system. Different technologies, in economic competition inside the same sector, are characterized by different investment and production-related (operation and maintenance, energy) costs. The demand curve determines the quantity of energy service that the market is willing to purchase when the purchase price increases; the supply curve for each one of them determines the quantity of energy service that is offered by the production system at the correspondent production cost. As the quantity produced increases, one or more resources are exhausted, and therefore the system must start using a different and more expensive technology or set of technologies in order to produce additional units of the required commodity, even if at higher unit cost. For each commodity, the demand-supply equilibrium is found at the intersection between the demand and supply curves, corresponding to a certain quantity-price equilibrium couple. When the equilibrium is reached, the total surplus is maximized, and it corresponds to the sum of the consumer's (the difference between what the consumer would be willing to pay and what he actually pays) and the producer's (the difference between the producer income and the production cost) surpluses [8].

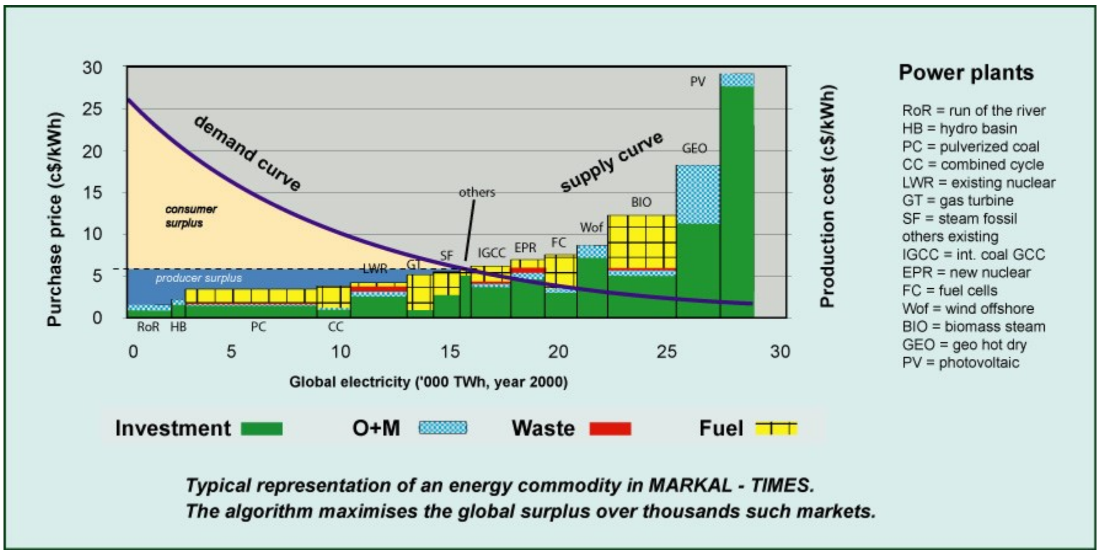


Figure 2. Market equilibrium mechanism representation for the electricity commodity [8].

1.4. The need for open science

A growing awareness is spreading in the scientific community about open science. Open science means having the possibility to freely diffuse data and results of scientific research, increasing responsiveness and spreading knowledge without economic limitations [9].

The importance of this issue is such that it falls within the priority policies of the European Commission [10], focusing on 8 ambitions of the EU's open science policy:

a. Open data: open data sharing by default for EU-funded scientific research.
b. European Open Science Cloud to store, share, process, and reuse research digital objects.
c. Development of alternative metric to evaluate the impact of research outcomes.
d. All peer-reviewed scientific publications should be freely accessible.
e. Research career evaluation systems should fully acknowledge open science activities.
f. Reproducibility of scientific results.
g. All scientists should have the necessary skills to apply open science practices.
h. Possibility for the general public to make significant contributions to the scientific research.

In the energy system modeling context, particularly, the data and results free sharing and the results reproducibility are crucial to guarantee that results presented in publications are solid and based on the scientific method, avoiding any external pressure on researchers, especially when it is necessary to evaluate possible legislative interventions, particularly subject to political evaluations [11].

The openness should regard both the data collected in research activities and the tools used to elaborate input data and derive results. From the perspective of software, open-source software allows the source code to be examined, modified, and integrated. The utilization of open-source tools is also fundamental to ensure transparency of the algorithms included in the models [7].

With respect to the energy system modeling, several open models have been developed in recent years, with different characteristics. Some focused on the optimization of the electricity system alone, while others refer to the overall energy system. Some developed on a municipal or local scale, others on a national or international scale. Focusing on models developed for national-scale energy system optimization, two main tools ca be considered, namely Open Source Energy Modelling System (OSeMOSYS [12]) and Tools for Energy Model Optimization and Analysis (Temoa [13]).

Table 1 compares OSeMOSYS and Temoa features with TIMES. Both the two cited tools allow to perform energy system optimization analyses at the macro-scale, with increasing database complexity with the complexity of the energy system. The two open models are also similar in terms of possibility to set future evolution of parameters (being possible to set any kind of trend: linear, exponential, etc.), programming language, formulation of the problem, possibility to modify and improve the code and current impossibility to perform multi-objective optimization (but could be implemented).

The two main differences between OSeMOSYS and Temoa regard the optimization software and, subsequently, the complexity of the energy system that can be optimized by the model. Indeed, using freely available software with OSeMOSYS, only relatively simple energy systems can be optimized with acceptable computational cost. On the contrary, with Temoa there is the possibility to use also solvers allowing to solve the optimization problem for large-size energy systems (anyway GLPK can be used for simpler study cases).

Table 1. Comparison of available tools for macro-scale energy system optimization.

Model / Feature	OSeMOSYS	TEMOA	TIMES
Features of the input data entering tool	Several steps are required for the definition of the time-scale, space-scale and technological characterizations, but allows a prompt visualization of the RES network.	Complexity increases with the complexity of the energy system, but the code formulation makes it straightforward.	Complexity increases with the complexity of the energy system (especially with the number of regions), due to the large number of Excel files to be managed.
Future evolution of parameters	The required values must be declared at each desired time-step, with the possibility of assigning different evolution trends between two points in time (linear, exponential, …).	The required values must be declared at each desired time-step, with the possibility of assigning different evolution trends between two points in time (linear, exponential, …).	The required values must be declared at each desired time-step, with the possibility of assigning different evolution trends between two points in time (linear, exponential, …).
Programming language(s)	GNU $^{\text{open-source}}$ / Python $^{\text{open-source}}$ / GAMS $^{\text{commercial}}$	Python $^{\text{open-source}}$	GAMS $^{\text{commercial}}$
Type of programming language	High-level	High-level	High-level
Optimization software (solver)	GLPK for GNU $^{\text{open-source}}$ / GLPK for Python $^{\text{open-source}}$ / GAMS $^{\text{commercial}}$	GLPK for Python $^{\text{open-source}}$ / CPLEX for Python $^{\text{commercial (but can be run on an external server)}}$ / Gurobi for Python $^{\text{commercial (but available with free academic license)}}$	CPLEX for GAMS $^{\text{commercial}}$
Features of the optimization software	Suitable for simple energy systems if using-open-source solvers.	Suitable for large-scale energy systems.	Suitable for large-scale energy systems.
Formulation of the problem	Scenario-based, long-term, multi-regional, minimum cost optimization linear problem considering competitive markets with perfect foresight.	Scenario-based, long-term, multi-regional, minimum cost optimization linear problem considering competitive markets with perfect foresight.	Scenario-based, long-term, multi-regional, minimum cost optimization linear problem considering competitive markets with perfect foresight.
Possibility to modify/improve the code	Possible.	Possible.	Impossible.
Possibility to perform stochastic optimization	Impossible at present state, but an extension can be formulated.	Possible with an already implemented Python module.	Possible, but time-consuming and complex due to the difficult data handling.
Possibility to perform multi-objective optimization	Impossible at present state, but could be implemented.	Impossible at present state, but could be implemented.	Impossible.

1.5. Aim of the thesis

Basing on the premises about open science made in Section 1.1, the aim of this thesis activity is to develop an open-source model on the Italian energy system. The selected reference model for the open-source implementation is TIMES-Italia [14], belonging to the TIMES family and developed by ENEA [15] during the last years. The features of TIMES-Italia and its database will be discussed in detailed in Section 0.

The motivations for having this purpose are strictly correlated with the European Commission's objectives in terms of open science. The common belief is that relying on an open-source energy model would allow to perform scenario analyses and evaluations of the possible future energy policies characterized by independence and reliability on the results, being them transparent, inspectable and reproducible.

Another consideration is relevant, concerning the openness of the tool use for energy system modeling. As it was already mentioned in Section 1.1 and Section 1.1, the currently used commercial software are based on a single optimization paradigm, such as the minimization of the total cost of the energy system. Of course, the cost is important, and it is true that energy system evolves aiming to satisfy the energy-related needs of the society possibly minimizing the efforts required to achieve this goal. Considering only the economic point of view, it appears quite obvious that the expenditures related to new investments for energy plants and operation and maintenance of existing ones should be as limited as possible.

However, especially in recent years, an increasing awareness has grown in the population and in institutions (governmental and non-governmental) regarding the objectives of environmental sustainability. In this regard, two important statements by the international community can be cited.

First, the 2015 Paris Agreement [3] aims to keep the average global temperature rise well below 2 °C above pre-industrial levels, with the aim of limiting it to 1.5 °C. In this perspective, it is intended to reach the peak of global greenhouse gas emissions as soon as possible, to then reduce them significantly and quickly and reach climate neutrality by the second half of the century. Secondly, the United Nations has set itself 17 Sustainable

Development Goals [16], with the aim of ending poverty, focusing on education, inequalities reduction and economic growth but, at the same time, fighting climate change and preserving natural ecosystems.

With those purposes, it seems realistic that the focus, concerning the next energy policies, will be not only on the expected expenditures, but firstly on their effectiveness in reaching those objectives, also accepting higher expenditures.

For that reason, it become crucial to have a simulation and optimization tool capable to account for several optimization criteria, related in particular to the sustainability of the energy system. To realize that it is needed to go beyond the optimization algorithms and paradigms currently present in commercial software, and it can be achieved only through open-source models, having the access to the source code and the possibility to investigate, modify and integrate it taking into account a sort of sustainability optimization, additional to the economic one.

Finally, this thesis work is structured as follows:

a. Review and update of the TIMES-Italia energy model (Section 0).
b. Open-source implementation of the model (Section 0).
c. Comparison of results obtained by the commercial and open-source tools (Section 0), to validate the new model.

Chapter 2

The TIMES-Italia model

In the context just introduced of energy system modeling, a model concerning the Italian energy system has been developed, within the TIMES framework, by ENEA during the last years: the so-called TIMES-Italia [14]. It is a single-region model and the relationship between Italy and other economically and energy trade-connected countries is defined via import-export parameters.

The time scale is subdivided in several time periods, namely the time-steps in which the model optimizes commodity quantities and prices. The first time-step, i.e., the base year, is 2006. In the base year, energy balances are analyzed to define the composition of the RES and initial service demands in each demand sector. After 2006, several time-steps are included in the current TIMES-Italia [14] up to the final year, 2050: 2007, 2008, 2010, 2012, 2014, 2016, 2018, 2020, 2022, 2025, 2030, 2040 and 2050. To each of those milestone years, driver and demand elasticity projections are associated, in order to derive future service demands. The TIMES-Italia time resolution consists of subdividing the simulation year in four seasons (spring, summer, fall and winter) each of which was assigned 1/4 of the total time of the year, and further subdividing each season in three times of the day (day, night and peak, correspondent to the hour of peak consumption), with different shares dependent on the correspondent season. In this way, the model time resolution is set to twelve time-slices per year, as shown in Table 2.

Table 2. Time-slice subdivision of the model year.

Seasons / Times of day	Spring	Summer	Fall	Winter
Day	$\frac{1}{4}*\frac{11}{24}=11.5\%$	$\frac{1}{4}*\frac{12}{24}=12.5\%$	$\frac{1}{4}*\frac{11}{24}=11.5\%$	$\frac{1}{4}*\frac{10}{24}=10.5\%$
Night	$\frac{1}{4}*\frac{12}{24}=12.5\%$	$\frac{1}{4}*\frac{11}{24}=11.5\%$	$\frac{1}{4}*\frac{12}{24}=12.5\%$	$\frac{1}{4}*\frac{13}{24}=13.5\%$
Peak	$\frac{1}{4}*\frac{1}{24}=1.0\%$	$\frac{1}{4}*\frac{1}{24}=1.0\%$	$\frac{1}{4}*\frac{1}{24}=1.0\%$	$\frac{1}{4}*\frac{1}{24}=1.0\%$

This is done to achieve a detailed characterization of energy production and consumption during time. Indeed, both seasonal and daily subdivisions of time periods allow to consider variations in energy production (e.g., availability of hydroelectric electricity is strongly dependent on the season, or photovoltaic production does not occur during night) as well as for consumptions (e.g., heating is only necessary during cold months and cooling during hot months, and lighting only during night). Also, one hour of peak-demand per day is considered, as the model does not only consider the balance between production and consumption in terms of energy, but also in terms of power; for this reason, it is important to consider peak-demand periods, in order not to underestimate the value of power instantaneously required to satisfy the demand.

2.1. Reference energy system

The reference energy system (RES) in energy models is a network representation of all the processes involved in the transformation from energy supply to final demands, consisting of a techno-economic characterization of both supply-side and demand-side energy sectors.

The TIMES-Italia [14] RES is composed by five technology sectors, distinguished in three demand-side sectors and two supply-side sectors. The demand-side sectors are buildings (further divided in agriculture, commercial and residential sector), transport and industry. The supply-side sectors are the power sector and the upstream sector. Each sector includes a set of several technologies, characterized by proper tecno-economic parameters, used to produce the intermediate and final commodities necessary to guarantee the production of the requested energy service demands.

The techno-economic description of the sectors is performed distinguishing the base year of the time horizon from its future years. The base year is a time period for which energy consumption, final service demands, and the efficiency of the installed technologies are known or, at least, can be estimated from energy balances or different statistics. The scope of the base year description is to represent the existing reference energy system, from the supply of energy to the service demands. Differently, future years are the time periods for which the optimization of the energy system must be performed by the model, depending on the assumptions made on socio-economic development and on available new technologies. To drive the progressive disposal of the base year technologies during the time, constraints on the residual capacity of these technologies are used, usually starting from the base year value and following a linear trend up to 0 after a certain time interval.

Figure 3 represents the generic structure of the TIMES-Italia [14] RES. In the upper section of the diagram (in orange), the upstream sector is represented, divided in fossil fuel technologies and renewable energy source technologies, producing upstream output commodities. Power sector is represented in yellow, and it consumes upstream commodities to produce electricity (distinguished in centralized and distributed) and heat, which are the input for the demand-side sectors (together with the upstream output commodities). In the bottom section of the diagrams, demand-side sectors are synthetically represented, listing the final service demand categories to be satisfied.

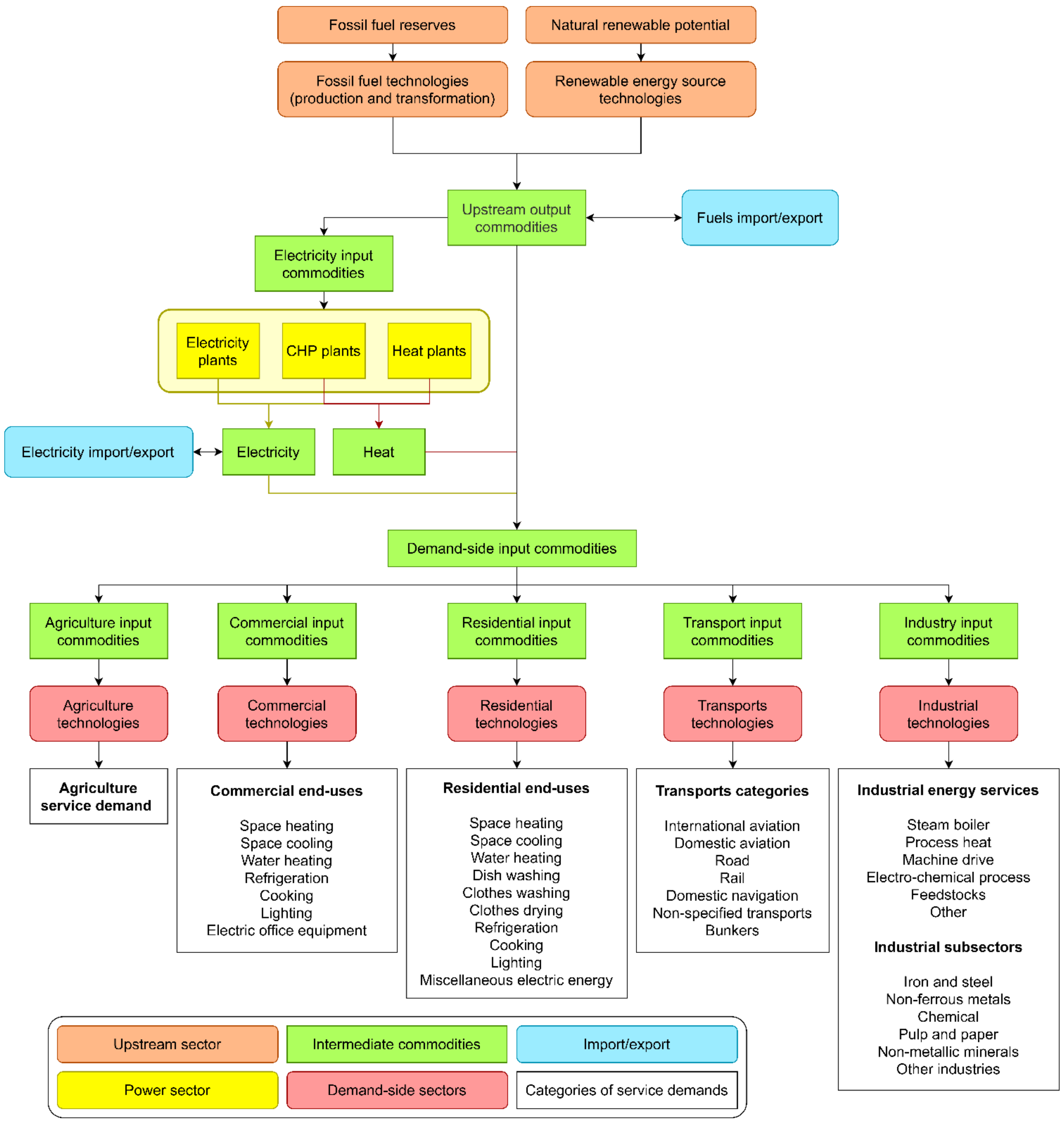


Figure 3. Generic structure of the TIMES-Italia reference energy system, and connections between the different sectors.

A relevant comment on the structure of the RES can be done, concerning the different role of base year modeling with respect to new technologies modeling.

New technologies included in the model database are represented by a detailed techno-economic characterization of all the energy-related processes that are expected to enter the market during the considered time horizon and are used to substitute the base year technologies, which progressively phase out. The main intent of base year is, instead, to quantify the final service demands, especially for those that are not derived by statistics but simply estimated knowing the energy consumption of the subsector and evaluating the efficiency of the base year technologies.

Being the service demands evaluation directly dependent on those assumptions and parameters characterization, a precise modeling of the base year is a crucial point to obtain reliable and arguable results. For that reason, in Section 1.1 the TIMES-Italia [14] base year is described in detail, to clarify how energy consumptions (known from IEA statistics) are converted into final service demands and based on which assumptions.

2.2. Demand-side sectors

2.2.1. Base year

The energy consumption at base year (2006 for TIMES-Italia [14]), is known from IEA World Energy Balances [17], that provide energy consumption by energy sector and split by fuels. For what concerns transport and industry sectors, IEA data are already split by subsectors. The unit of measure is PJ.

Two procedures can be followed to model the base year structure and quantify final service demands:

a. If the total service demand is known from external national statistics, the efficiency of base year technologies can be estimated (consistently with efficiency of installed technologies) as ratio between energy consumptions and demand production.
b. If the service demand is unknown and must be estimated, the estimation is performed assuming efficiency values for base year technologies (based on their tecno-economic characterization) and deriving the demand as product between energy consumption and efficiency.

In the TIMES framework, also the evaluation of sector-specific commodity-based emission factors is performed within the base year files. The emission factors are necessary to evaluate the atmospheric emissions of each technology and, consequently, of the entire energy system. Those factors are sector-specific and commodity-based in the sense that they are specified for each sector-specific commodity (fuel consumed by the technology) and the absolute values of the emissions is evaluated by the model on the basis of each commodity consumption. The sector-specific emission factors are indicated as "dynamic" (EF_D) in contrast with the "static" one ($EF_{S,i}$) which are emission factors for the upstream commodities required to produce the sector-specific commodities.

Upstream commodities $C_{i,j}$ [PJ] of the model are connected to the sector-specific commodities C_j [PJ] with appropriate share factors f_i [%], used to describe the mix correctly (as shown by Equation 1). Share factors are fixed for the base year, and constraints on them are imposed for the projection years, typically upper constraints (to set a maximum boundary to the percentage of some generic commodity).

$$\mathrm{f_i}\ [\%] = \frac{\mathrm{C_{i,j}}\ [\mathrm{PJ}]}{\mathrm{C_j}\ [\mathrm{PJ}]} \tag{1}$$

Since sector-specific commodities are produced from upstream ones, the evaluation of dynamic emission factor is simply performed multiplying each static factor by the share factor associated to the correspondent commodity, as shown in Equation 2. They are "dynamic" being dependent on the share of upstream commodities involved.

$$\mathrm{EF_D}\left[\frac{\mathrm{kt}}{\mathrm{PJ}}\right] = \sum_{\mathrm{i}} \mathrm{EF_{S,i}}\left[\frac{\mathrm{kt}}{\mathrm{PJ}}\right] * \mathrm{f_i}\ [\%] \tag{2}$$

The considered emission commodities are CO_2, CH_4, N_2O and SO_x (the latter for the industrial sector only). In the following sections, at the end of each section, the evaluation of sector-specific commodity-based emission factors is also reported. In Table 3 static emission factors for the most representative upstream commodities are reported, for the sake of completeness.

Table 3. Static emission factors.

Commodity	**Emission commodity [kt/PJ]**		
	CO_2	CH_4	N_2O
Hard coal	98.30	5.00	1.40
Brown coal	101.20	5.00	1.40
Crude oil	73.30	2.00	0.60
Gasoline	69.30	2.00	0.60
Gas oil	74.10	2.00	0.60
Natural gas	56.10	5.00	0.10
Liquified petroleum gas	63.10	5.00	0.10
Waste	85.85	300.00	4.00

2.2.1.1. Agriculture

The first subsector considered is agriculture, having a very simple data structure. Energy balances for the sector are reported, stating energy consumption for several fuels. Subsequently, they are aggregated into a reduced number of modelled fuels, as in Equation 3, where E_{af} is the energy consumption attributed to the aggregated fuel, and $E_{f,i}$ is the energy consumption of the ith fuel contributing to the aggregated one.

$$E_{af}\,[PJ] = \sum_{i} E_{f,i}\,[PJ] \tag{3}$$

The result is reported in Table 4, showing that most of the energy consumption for the agriculture sector is due to diesel consumption, as it might be expected.

Table 4. IEA statistics, aggregated fuels for the agriculture.

IEA Fuels	Aggregated Fuels	E_{af} [PJ]
Natural gas Gas works gas Coke oven gas	Natural Gas	6.22
Diesel Other non-specified	Distillate	104.45
Motor gasoline	Gasoline	0.66
LPG	LPG	3.08
Solid biomass Charcoal Gas biomass, Other liquid biofuels Municipal waste (non-renewable), municipal waste (renewable) Industrial waste	Biomass	8.20
Electricity	Electricity	19.82
Geothermal	Geothermal	5.35

Made that first aggregation the service demand is very simply assumed to be equal to the total energy consumption of the subsector, assuming unitary efficiency for the base year technology producing the service demand. The sum gives a service demand D_{AGR} of 147.79 PJ for agriculture, as shown in Equation 4.

$$D_{AGR}[PJ] = \sum_{j} E_{af,j}[PJ] = 147.79\ PJ \qquad (4)$$

Figure 4 summarizes data organization and interactions, from IEA statistics to the calculation of service demand.

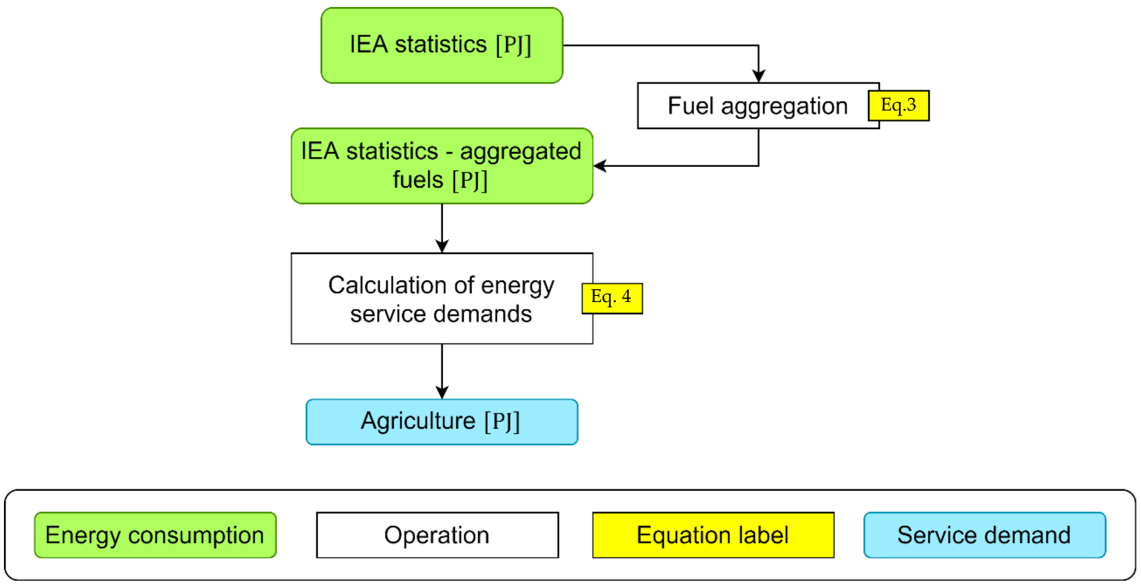


Figure 4. Flow-chart for the evaluation of service demand in agriculture. Associated to each step are specified the involved equations, as listed in this report.

Eventually, the computation of emission factors for agriculture sector leads to the values that follow in Table 5. It should be noted for instance that the highest value of CO_2 emission factors is related to coal, reasonably being the fuel with the highest carbon content. Also, a high CH_4 emission is associated to biomass.

Table 5. Emission factors for agriculture sector-specific fuels.

Commodity	**Emission factors [kt/PJ]**							
	Natural gas	Diesel	Gasoline	Heavy fuel oil	Kerosene	Coal	LPG	Biomass
CO_2	56.10	74.07	69.30	77.37	71.87	98.27	63.07	0.00
CH_4	5.00	5.00	20.00	5.00	1.00	50.00	5.00	300.00
N_2O	0.10	0.60	0.60	0.60	0.60	1.40	0.10	4.00

2.2.1.2. Commercial

In this section, the commercial sector is investigated. Energy service demands for the commercial sector are listed in Table 6.

Table 6. Commercial service demands.

Commercial service demands
Space heating
Space cooling
Water heating
Lighting
Cooking
Refrigeration
Electric office equipment

As already seen for agriculture sector, an aggregation of energy consumption by similar fuels is needed to reduce their number in a more easily treatable form. This operation is done accordingly to Equation 3, where E^{f}_{af} is the final energy consumption attributed to the aggregated fuel and $E_{f,i}$ is the energy consumption of the ith fuel that contributes to the aggregated one. Table 7 is the output of the aggregation. In this sector, the energy consumption is mainly subdivided among natural gas and electricity consumption.

Table 7. IEA statistics, aggregated fuels for the commercial sector.

IEA Fuels	**Aggregated Fuels**	$\mathbf{E^{f}_{af}}$ **[PJ]**
Natural gas Gas works gas Coke oven gas	Natural Gas	318.50
Diesel Motor gasoline Other non-specified	Distillate	19.29
LPG	LPG	0.09
Electricity	Electricity	281.06
Heat	Heat	6.30
Geothermal	Geothermal	1.78

The next step, as for residential sector, is the splitting of energy consumptions into the different end-uses of the sectors. To do this, fractional shares of end-use are assigned by assumption [14] (Table 8) and they are multiplied by the relative consumption of fuel to obtain absolute values of energy consumption for every fuel and every end-use, as Equation 5 shows. Table 9 reports the results of that operation and, focusing on natural gas and electricity (representing the most relevant consumption), showing that according to the assumed fractional share natural gas is mainly allocated for space heating, while electricity is more evenly divided among all end uses.

Table 8. Fractional end-uses share for the commercial end-uses.

Fuel	**Fractional end-uses shares f_{eu} [%]**						
	Space heating	Space cooling	Water heating	Lighting	Cooking	Refrigeration	Electric office equipment
Natural Gas	0.88	0.05	0.03		0.04		
Distillate	0.82	0.08	0.1				
LPG	0.67		0.25		0.08		
Electricity	0.08	0.13	0.05	0.30	0.01	0.07	0.38
Heat	0.7		0.3				
Geothermal	1						

Table 9. End-use energy consumptions for commercial services.

Fuel	**End-uses energy consumptions $E^{f}_{eu,\,af}$[PJ]**						
	Space heating	Space cooling	Water heating	Lighting	Cooking	Refrigeration	Electric office equipment
Natural Gas	280.28	15.92	9.55		12.74		
Distillate	15.82	1.54	1.93				
LPG	0.06		0.02		0.01		
Electricity	21.08	35.41	12.65	84.32	2.25	19.67	105.68
Heat	4.41		1.89				
Geothermal	1.78						

$$E^{f}_{eu,\,af}\,[PJ] = f_{eu}\,[\%] * E^{f}_{af}\,[PJ] \tag{5}$$

In Table 10 all the technologies modelled to satisfy each energy service are listed, with specifications about input commodity, share of input commodity consumption (i.e., the fraction of input commodity energy consumption that is due to the technologies, it is less than 100 if more than one technology use the same fuel), final energy, efficiency and useful energy. The disaggregation performed to consider the possibility that more than one technology is associated to the consumption of a single fuel in certain end uses is shown in Equation 6, where $E^{f}_{tech,eu,\,af}$ is the final energy consumption associated to each technology.

$$E^{f}_{tech,eu,\,af}[PJ] = f_{tech}[\%] * E^{f}_{eu,af}[PJ] \tag{6}$$

In Table 10, also the useful energy is derived for each technology, simply multiplying the final energy consumption $E^{f}_{tech,eu,\,af}$ by the technology efficiency eff, according to Equation 7.

$$E^{u}[PJ] = eff[\%] * E^{f}_{tech,eu,\,af}[PJ] \tag{7}$$

Table 10. Demand-side technologies by commercial energy services.

Energy service	**Technology**	**Input commodity share f_{tech}[%]**	**Input commodity**	**Final energy $E^f_{tech,eu,af}$[PJ]**	**Efficiency eff[%]**	**Useful energy E^u[PJ]**
Space heating	Natural gas boiler	99.99	Natural Gas	280.25	70	196.17
	Natural gas heat pump	0.01		0.03	190	0.05
	Distillate boiler	100	Distillate	15.82	70	11.07
	LPG boiler	100	LPG	0.06	60	0.04
	Resistance	25	Electricity	5.27	90	4.74
	Electricity heat pump	75		15.81	200	31.62
	District heating	100	Heat	4.41	90	3.97
	Geothermal heat pump	100	Geothermal	1.78	90	1.61
Space cooling	Natural gas absorption chiller	100	Natural Gas	15.92	120	19.11
	Distillate chiller	100	Distillate	1.54	84	1.30
	Centralized heat pump	54.4	Electricity	19.27	360	69.36
	Electricity heat pump	0. 3	Electricity	0.12	372	0.43
	Room heat pump	11.3	Electricity	3.99	360	14.37
	Electric chiller rooftop	34	Electricity	12.04	372	44.78
Water heating	Natural gas boiler	100	Natural Gas	9.55	65	6.21
	Distillate boiler	100	Distillate	1.93	65	1.25
	LPG boiler	100	LPG	0.02	60	0.01
	Electric heater	100	Electricity	12.65	91	11.51
	District heating	100	Heat	1.89	100	1.89
Refrigeration	Refrigerator	100	Electricity	19.67	100	19.67
Cooking	Natural gas cooker	100	Natural Gas	12.74	50[1]	6.37
	LPG cooker	100	LPG	0.01	50[1]	0.00
	Electricity cooker	100	Electricity	2.25	70[1]	1.57
Electric office equipment	Electric Equipment	100	Electricity	105.68	100	105.68
Lighting	Incandescent big	3	Electricity	2.53	117	2.96
	Halogen small (12 V)	1		0.84	160	1.35
	Halogen IRC (12 V)	1		0.84	209	1.76
	Fluorescent small	37.5		31.62	563	178.08
	Fluorescent large	37.5		31.62	698	220.74
	Fluorescent compact	9		7.59	593	44.98
	Mercury	10		8.43	320	26.98
	Sodium low pressure	1		0.84	800	6.75

[1] Efficiency of cooking appliances are differentiated for different cooking technologies. [18]

The results shown in Table 10 consist of an estimation of the useful energy of each commercial technology, based on its energy consumption and efficiency. Focusing in particular on the efficiencies of technologies, it can be underlined that the efficiencies of heat pumps (both for space heating and space cooling) are higher than 100%, representing the coefficient of performance (COP) of the technologies. The only heat pump having an efficiency lower than 100% is the geothermal heat pump for space heating, since in this case the final energy consumption is not the electricity required by the compressor, but the inlet heat at the heat pump evaporator. Furthermore, the efficiencies of cooking technologies have been updated ([18] both for existing and new technologies in the model) knowing that different types of cookers are characterized by different cooking efficiency and notably LPG or natural gas burners are less efficient than induction electric plates, for instance.

The service demands for commercial sector are assumed to be equal to the total useful energy associated to each energy service therefore, to derive the value of j^{th} service demand $D_{COM,j}[PJ]$, it is sufficient to sum the useful energy values $E_j^u[PJ]$ of all the technologies included in the j^{th} energy service, as explained in Equation 8. The results are listed in Table 11; being all the service demands quantified in useful energy terms, the unit of measure for all the commercial service demands is PJ.

$$D_{COM,j}[PJ] = \sum_j E_j^u[PJ] \tag{8}$$

Table 11. Service demands for commercial sector energy services.

End use	$\mathbf{D_{COM}}$
Space heating	249.27 PJ
Space cooling	149.35 PJ
Water heating	20.88 PJ
Lighting	483.59 PJ
Cooking	7.95 PJ
Refrigeration	19.67 PJ
Electric office equipment	105.68 PJ

Figure 5 is a summary of how data are organized and how they interact, from IEA statistics to service demand.

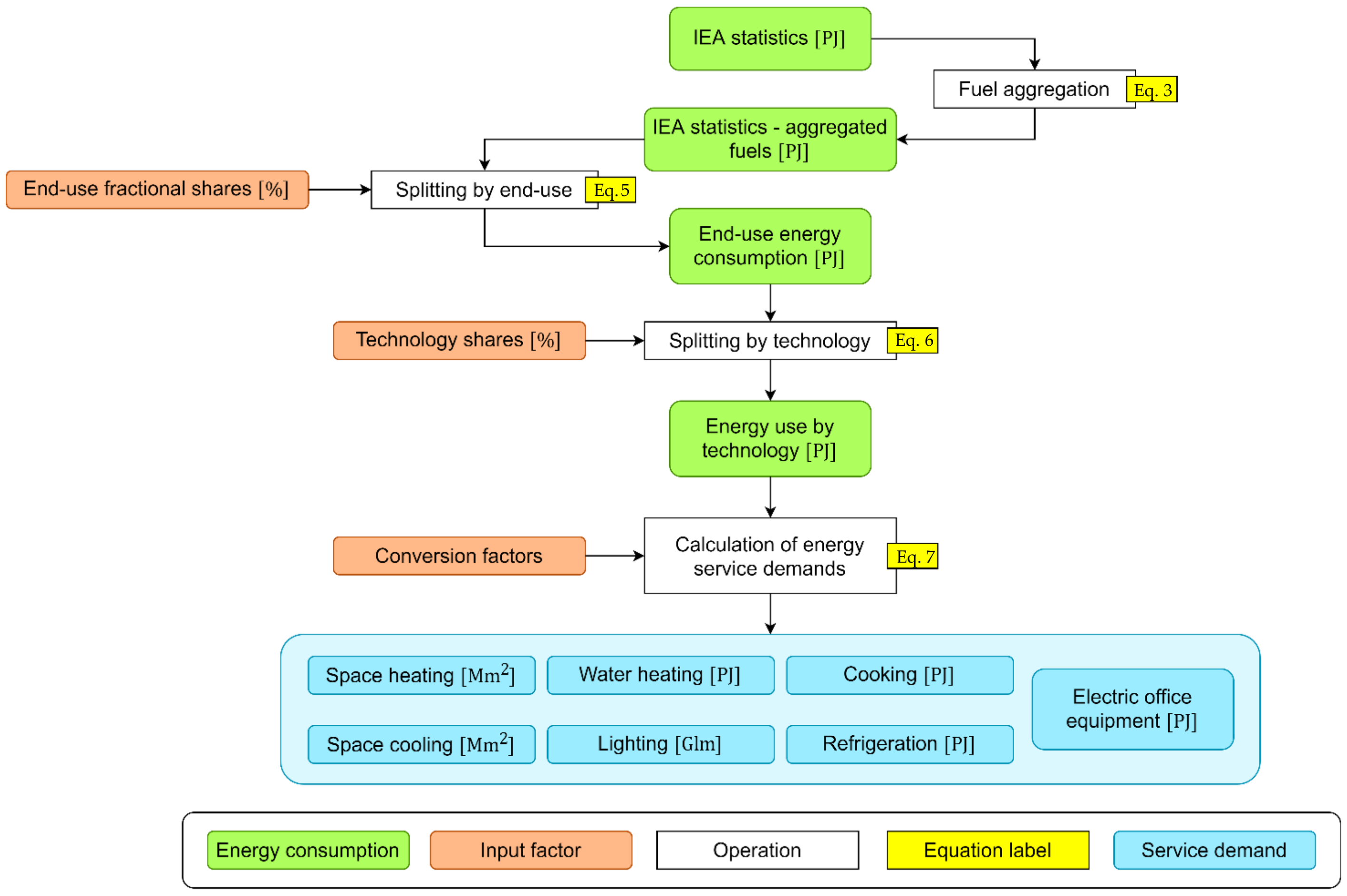


Figure 5. Flow-chart for the evaluation of service demand in the commercial sector. Associated to each step are specified the involved equations, as listed in this report.

Eventually, the computation of emission factors for commercial sector leads to the values that follow in Table 12, obtained according to Equation 2. It should be noted for instance that the highest value of CO_2 emission factors is related to coal, reasonably being the fuel with the highest carbon content. Also, a high CH_4 emission is associated to biomass.

Table 12. Emission factors for commercial sector-specific fuels.

Commodity	**Emission factors [kt/PJ]**						
	Natural gas	Diesel	Heavy fuel oil	Kerosene	Coal	LPG	Biomass
CO_2	56.10	73.78	77.37	71.87	98.27	63.07	0.00
CH_4	5.00	5.89	5.00	1.00	50.00	5.00	300.00
N_2O	0.10	0.60	0.60	0.60	1.40	0.10	4.00

2.2.1.3. Residential

Concerning residential sector, the included service demands are listed in Table 13.

Table 13. Residential service demands.

Residential service demands
Space heating
Space cooling
Water heating
Refrigeration
Clothes drying
Cooking
Clothes washing
Dishwashing
Miscellaneous electric energy
Lighting

Energy balances, stating the energy consumption for several fuels, are aggregated according to Equation 3. The output of this first aggregation is reported in Table 14, where the correspondence between IEA and aggregated TIMES-Italia [14] modelled fuels is shown, along with the resulting energy consumption. The results of the aggregation show that, similarly to what observed for the commercial sector, the most consumed commodities by residential are natural gas and electricity, with not negligible consumption of distillate fuels (diesel), LPG and biomass.

Table 14. IEA statistics, aggregated fuels for the residential sector.

IEA Fuels	Aggregated Fuel	Energy consumption E_{af}^{f} [PJ]
Natural gas Gas works gas Coke oven gas	Natural Gas	713.73
Diesel Motor gasoline Other non-specified oil products	Distillate	131.21
Fuel oil Crude oil	Heavy fuel oil	6.40
Other kerosene	Kerosene	0.77
Hard coal, Patent fuel, Anthracite, Other bituminous coal Peat, brown coal briquettes, Brown coal, Sub-bituminous coal, Lignite Coke oven coke	Coal	0.29
LPG	LPG	85.28
Solid biomass Charcoal Gas biomass, Other liquid biofuels Municipal waste (non-renewable), Municipal waste (renewable) Industrial waste	Biomass	67.88
Electricity	Electricity	243.53
Heat	Heat	27.63
Geothermal	Geothermal	1.78
Solar thermal	Solar thermal	1.47

The data are now ready to be split into end-uses of residential sector. This is done assigning by assumption fractional shares [14], as in Table 15, to each end-use for each fuel, and then using Equation 5 to obtain energy consumption of each fuel for all residential end-uses, for which results are shown in Table 16, showing that the subsector consuming the most relevant fraction of energy is space heating, while electricity is quite evenly divided among the different end-uses (similarly to the commercial sector).

Table 15. Fractional end-use shares for the residential sector.

Fuel	Fractional end-uses shares f_{eu} [%]									
	Space heating	Space cooling	Water heating	Refrigeration	Clothes drying	Cooking	Clothes washing	Dishwashing	Miscellaneous electric energy	Lighting
Natural Gas	0.895		0.075			0.03				
Distillate	0.94		0.06							
Heavy fuel oil	0.95		0.045			0.005				
Kerosene	1									
Coal	1									
LPG	0.78		0.07			0.15				
Biomass	0.97		0.03							
Electricity	0.01	0.06	0.115	0.18	0.005	0.045	0.095	0.065	0.281	0.144
Heat	0.7		0.3							
Geothermal	1									
Solar	0.15		0.85							

Table 16. End-use energy consumption for residential services.

Fuel	End-use energy consumption $E^{f}_{eu,\,af}$ [PJ]									
	Space heating	Space cooling	Water heating	Refrigeration	Clothes drying	Cooking	Clothes washing	Dishwashing	Miscellaneous electric energy	Lighting
Natural Gas	638.79		53.53			21.41				
Distillate	123.34		7.87							
Heavy fuel oil	6.08		0.29			0.03				
Kerosene	0.77									
Coal	0.29									
LPG	66.52		5.97			12.79				
Biomass	65.84		2.04							
Electricity	2.44	14.61	28.01	43.84	1.22	10.96	23.14	15.83	68.43	35.07
Heat	19.34		8.29							
Geothermal	1.78									
Solar	0.22		1.25							

Another step is required for data splitting, for space heating demand only. Indeed, that is different according to different types of buildings, and specifically:

a. SF-Old: single-family house (old).
b. SF-New: single-family house (new).
c. MF-Old: multi-family house (old).
d. MF-New: multi-family house (new).

The difference between "old" and "new buildings is in the efficiency terms (from final energy to useful energy), being higher for new buildings with respect to old buildings. Regarding the differentiation in "SF" buildings and "MF" buildings, this is performed considering different conversion factors from useful energy to service demands, that are equal to 1.94 Mm^2/PJ for SF buildings and to 2.66 Mm^2/PJ for MF buildings (these values are derived from [19] and [20]). Natural gas is allocated for the 94.3% to SF-Old buildings, while the remaining 4.2% is allocated to MF-Old buildings, the 1.4% to SF-New, and the 0.1% to MF-New buildings. All other fuels (distillate, heavy fuel oil, kerosene, coal, LPG, biomass, electricity, heat, geothermal and solar thermal energy) are allocated for the 94% to SF-Old buildings, for the 0.5% to MF-Old buildings, and for the remaining 0.01% to SF-New buildings.

Equation 9 explains how space heating energy consumption, split by building type, $E_{bt,eu,af}$ is calculated, with the proper share factors f_{bt} [%] for the allocation to the different building types.

$$E^{f}_{bt,eu,af}\,[PJ] = f_{bt}\,[\%] * E^{f}_{eu,af}\,[PJ] \qquad (9)$$

In Table 17 the technologies modelled to satisfy each energy service are listed, with specifications about input commodity. It is possible now to convert energy consumptions in final service demands. This is done with appropriate conversion factors [14], specific for every single end-use and differentiated for each technology. To do so, one last needed step is to consider that, for some end-uses and fuels, there is more than one technology used to meet the final service demand. Just as an example, electricity consumption for lightning can be considered: that can be satisfied via several technologies, each one with different efficiencies: fluorescent lamp, halogen lamp, incandescent lamp, LED, etc. For this reason, share factors f_{tech} should be applied to assign to every technology the corresponding amount of consumed energy, as from Equation 10, where $E^{f}_{tech,bt,eu,\,af}$ is the final energy consumption associated to each technology. Eventually, the useful energy E_u for each technology is derived (as already done in commercial sector), knowing the efficiency eff of each technology (Equation 11).

$$E^{f}_{tech,bt,eu,af}\,[PJ] = f_{tech}\,[\%] * E^{f}_{bt,eu,af}\,[PJ] \tag{10}$$

$$E^{u}[PJ] = eff[\%] * E^{f}_{tech,bt,eu,af}\,[PJ] \tag{11}$$

Table 17. Demand-side technologies by residential energy services.

Energy service	**Technology**	**Input commodity**	**Input commodity**	**Final energy $E^{f}_{tech,eu,af}$[PJ]**	**Efficiency eff[%]**	**Useful energy E^{u}[PJ]**
Space heating (SF-Old)	Natural gas boiler	99	Natural Gas	596.36	73	434.15
	Natural gas heat pump	1		6.02	110	6.63
	Distillate boiler	100	Distillate	115.93	73	84.40
	Heavy fuel oil boiler	100	Heavy fuel oil	5.72	73	4.16
	Kerosene boiler	100	Kerosene	0.73	6	0.05
	Coal boiler	100	Coal	0.27	55	0.15
	LPG boiler	100	LPG	62.53	68	42.68
	Biomass boiler	100	Biomass	61.89	25	15.47
	Resistance	84	Electricity	1.92	90	1.73
	Electricity heat pump	16		0.37	200	0.73
	District heating	100	Heat	18.18	90	16.36
	Geothermal heat pump	100	Geothermal	1.68	380	6.37
Space heating (MF-Old)	Natural gas boiler	99	Natural Gas	26.56	73	19.34
	Natural gas heat pump	1		0.27	110	0.30
	Distillate boiler	100	Distillate	6.17	73	4.49
	Heavy fuel oil boiler	100	Heavy fuel oil	0.30	73	0.22
	Kerosene boiler	100	Kerosene	0.04	06	0.00
	Coal boiler	100	Coal	0.01	55	0.01
	LPG boiler	100	LPG	3.33	68	2.27
	Biomass boiler	100	Biomass	3.29	25	0.82
	Resistance	84	Electricity	0.10	90	0.09
	Electricity heat pump	16		0.02	200	0.04
	District heating	100	Heat	0.97	90	0.87
	Geothermal heat pump	100	Geothermal	0.09	380	0.34
Space heating (SF-New)	Natural gas boiler	99	Natural Gas	8.85	76	6.77
	Natural gas heat pump	1		0.09	116	0.10
	Distillate boiler	100	Distillate	1.23	76	0.94
	Heavy fuel oil boiler	100	Heavy fuel oil	0.06	76	0.05
	Kerosene boiler	100	Kerosene	0.01	07	0.00
	Coal boiler	100	Coal	0.00	57	0.00
	LPG boiler	100	LPG	0.67	72	0.48
	Biomass boiler	100	Biomass	0.66	26	0.17
	Resistance	84	Electricity	0.02	95	0.02
	Electricity heat pump	16		0.00	210	0.01
	District heating	100	Heat	0.19	95	0.18
	Geothermal heat pump	100	Geothermal	0.02	399	0.07

Continued on page 47

Continued from page 46

Space heating (MF-New)	Natural gas boiler	99	Natural Gas	0.63	76	0.48
	Natural gas heat pump	1		0.01	116	0.01
Space cooling	Centralized heat pump	54	Electricity	7.85	240	18.85
	Electricity heat pump	4		0.61	280	1.71
	Room heat pump	42		6.15	240	14.76
Water heating	Natural gas boiler	100	Natural Gas	53.53	65	34.79
	Distillate boiler	100	Distillate	7.87	58	4.57
	Heavy fuel oil boiler	100	Heavy fuel oil	0.29	58	0.17
	LPG boiler	100	LPG	5.97	54	3.22
	Biomass boiler	100	Biomass	2.04	25	0.51
	Electric heater	100	Electricity	28.01	90	25.21
	District heating	100	Heat	8.29	85	7.05
	Solar heater	100	Solar	1.25	100	1.25
Refrigeration	Refrigerator	78	Electricity	34.19		
	Freezer	22		9.64		
Clothes washing	Clothes washer	100	Electricity	23.14		
Clothes drying	Electric clothes drier	100	Electricity	1.22		
Dishwashing	Dishwasher	100	Electricity	15.83		
Cooking	Natural gas cooker	100	Natural Gas	21.41	50	10.71
	LPG cooker	100	LPG	12.79	50	6.40
	Electricity cooker	100	Electricity	10.96	80	8.77
Miscellaneous electric energy	Electric Appliances	100	Electricity	68.43		
Lighting	Fluorescent large	7	Electricity	2.45		
	Fluorescent small	9		3.16		
	Halogen large (220 V)	1		0.35		
	Halogen small IRC (12 V)	1		0.35		
	Halogen small (12 V)	2		0.70		
	Incandescent medium	40		14.03		
	Incandescent small	40		14.03		

The results shown in Table 17 consist of an estimation of the useful energy of each residential technology, based on its energy consumption and efficiency. Focusing in particular on the efficiencies of technologies, it can be underlined that the efficiencies of heat pumps (both for space heating and space cooling) are higher than 100%, representing the coefficient of performance (COP) of the technologies. Furthermore, the efficiencies of cooking technologies have been updated [18] (both for existing and new technologies in the model) knowing that different types of cookers are characterized by different cooking efficiency and notably LPG or natural gas burners are less efficient than induction electric plates, for instance.

The residential service demands, differently from the commercial ones, are expressed also with non-energy units of measure. For this reason, a last step is required for their evaluations. Proper conversion factors [14] $\mathrm{eff_{conv}}$ are used to derive from the useful energy $\mathrm{E^u}$ of each technology the correspondent energy service demand $\mathrm{D_{RES}}$ (Equation 12). This is done, of course, only for the service demands that are not expressed in energy terms, for which instead the conversion factor is assumed to be unitary, and the service demand corresponds to the useful energy.

$$\mathrm{D_{RES}} = \mathrm{eff_{conv}} * \mathrm{E^u}[\mathrm{PJ}] \tag{12}$$

The conversion factors used, and the resulting service demands are shown in Table 9, with the total service demand amount for each residential end-use energy service. According to the reported conversion factors, it should be highlighted that multi-family buildings are more efficient in terms of energy required for space heating. That is reasonable and due to the lower surface exposed to the external environment per unit of heated volume characterizing multi-apartment buildings. Secondly, for water heating, refrigeration, cooking, and “miscellaneous electric energy” no conversion factors are reported, since the final service demand is in terms of useful energy and measured in PJ.

Table 18. Service demands for the residential sector.

End-use service	$\mathbf{eff_{conv}}$		$\mathbf{D_{RES}}$	
Space heating (SF-Old)	1.94	Mm^2/PJ	1188.71	Mm^2
Space heating (SF-New)	1.94	Mm^2/PJ	76.58	Mm^2
Space heating (MF-Old)	2.66	Mm^2/PJ	17.06	Mm^2
Space heating (MF-New)	2.66	Mm^2/PJ	1.31	Mm^2
Space cooling	14.73	Mm^2/PJ	520.00	Mm^2
Water heating			76.76	PJ
Refrigeration			9.01	Gl
Clothes drying	0.07	Glav/PJ	0.08	Glav
Cooking			25.87	PJ
Clothes washing	0.26	Glav/PJ	6.02	Glav
Dishwashing	0.18	Glav/PJ	2.77	Glav
Miscellaneous electric energy			68.43	PJ
Lighting	12.30 [2]	Glm/PJ	431.24	Glm

Figure 6 is a summary of how data are organized and how they interact in residential sectors, from IEA statistics to service demand.

[2] Different values for the conversion factor of lighting service demand are applied to each technology. The value reported in the table is an average value obtained dividing the lighting service demand by the total useful energy of the subsector.

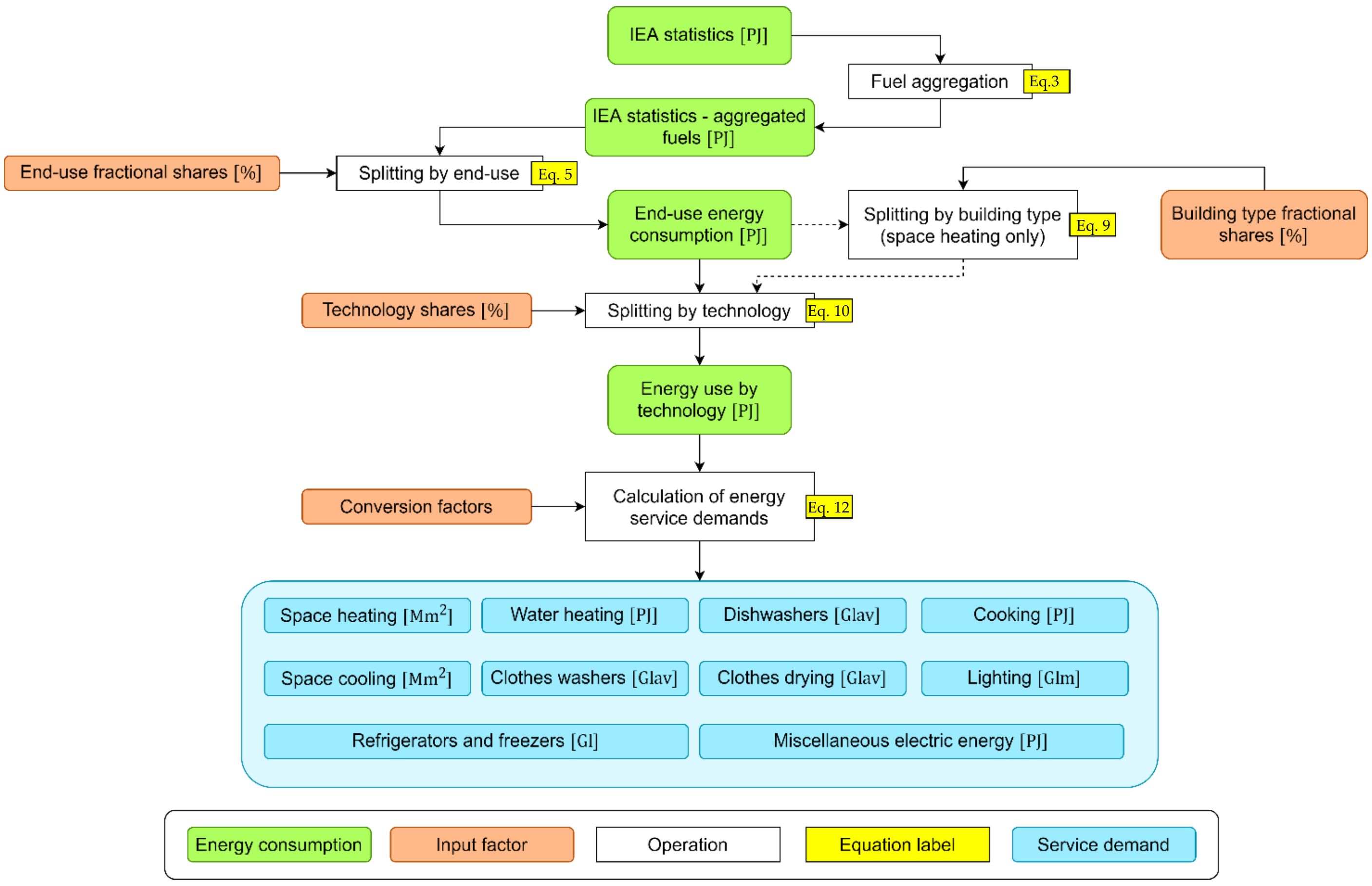


Figure 6. Flow-chart for the evaluation of service demand in the residential sector. Associated to each step are specified the involved equations, as listed in this report.

Eventually, the computation of emission factors for residential sector-specific fuels leads to the values reported in Table 19. It should be noted for instance that the highest value of CO_2 emission factors is related to coal, reasonably being the fuel with the highest carbon content. Also, a high CH_4 emission is associated to biomass.

Table 19. Emission factors for residential sector-specific fuels.

Commodity	**Emission factors [kt/PJ]**						
	Natural gas	Diesel	Heavy fuel oil	Kerosene	Coal	LPG	Biomass
CO_2	56.10	74.07	77.37	71.87	98.27	63.07	0.00
CH_4	5.00	5.00	5.00	1.00	50.00	5.00	300.00
N_2O	0.10	0.60	0.60	0.60	1.40	0.10	4.00

2.2.1.4. Transport

In the transport sector, several energy services are satisfied, according to the type of considered transport category. Transport categories considered in the TIMES-Italia [14] are reported in Table 20.

Table 20. Transport categories.

Transport categories
International aviation
Domestic aviation
Road
Rail
Domestic navigation
Non-specified transports
Bunkers

Again, the first operation involves the aggregation of similar fuels to reach a simpler classification of consumption (Equation 3). Results for the first step are reported in Table 21. In the table header, the IEA fuels (according to the classification adopted in [17]) are listed in the second row and they are associated to the corresponding aggregated fuel reported in the third row. For each aggregated fuel, the energy consumption spit by transport category is then reported in the table. The three largest values of consumption are diesel for road transport (1018.65 PJ), motor gasoline for road transport (555.76 PJ), and kerosene for international aviation (134.47 PJ).

Table 21. IEA statistics, aggregated fuels for transports sector.

Transport category	**IEA statistics by aggregated fuels E_{af} [PJ]**								
	Solid biomass, Charcoal, Gas biomass, Other liquid biofuels, Municipal waste (non-renewable), Municipal waste (renewable), Industrial waste	Natural gas	LPG	Motor gasoline	Aviation gasoline, Gasoline type jet fuel	Jet kerosene, Other kerosene	Diesel	Fuel oil, Other non-specified	Electricity
	Biodiesel	Natural gas	LPG	Motor Gasoline	Aviation Gasoline	Kerosene	Diesel	Heavy fuel oil	Electricity
International aviation					0.01	134.47			0.25
Domestic aviation					0.70	31.51			0.08
Road	6.73	17.31	45.40	555.76			1018.65		
Rail							4.77		15.75
Domestic navigation							10.05		0.12
Non-specified transports									20.59
Bunkers							25.96	117.76	

A higher level of detailed will be now performed concerning "Road" and "Rail" categories. Indeed, different transportation modes are present in those categories, with different techno-economic features. For this reason, a further splitting is done, as usual with share factors [14] (reported in Table 22) and the subsequent multiplication (Equation 13) to obtain split energy values for every transportation mode, collected in Table 23.

$$E_{tm,\,af}\,[PJ] = f_{tm}\,[\%] * E_{af}\,[PJ] \tag{13}$$

Table 22. Share factors for transportation modes.

Transport modes		Share factors for transportation modes f_{tm} [%]								
		Biodiesel	Natural Gas	LPG	Motor Gasoline	Aviation Gasoline	Kerosene	Diesel	Heavy Fuel Oil	Electricity
Road	Light Duty Vehicles Share	29.20	90.00	100.00	97.00			29.20		0.00
	Cars	100.00	100.00	100.00	87.75			100.00		100.00
	Two Wheels				12.25					
	Other Road Vehicles	70.80	10.00	0.00	3.00			70.80		100.00
	Bus	0.83	100.00					9.32		100.00
	Heavy Trucks	1.08						37.00		
	Medium Trucks	15.24						15.50		
	Commercial Trucks	82.85			100.00			38.18		
Rail	Freight							67.00		67.00
	Passengers							33.00		33.00

Table 23. Splitting by transportation mode.

Transportation modes		Splitting by transportation modes $E_{tm, af}$ [PJ]								
		Biodiesel	Natural Gas	LPG	Motor Gasoline	Aviation	Kerosene	Diesel	Heavy Fuel Oil	Electricity
Air	International aviation					0.01	134.47			
	Domestic aviation					0.70	31.51			
Road	Cars	1.97	15.58	45.40	473.04			297.45		
	Two wheels				66.05					
	Bus	0.04	1.73					67.23		
	Heavy trucks	0.05						266.85		
	Medium trucks	0.73						111.79		
	Commercial trucks	3.95			16.67			275.34		
Rail	Freight							3.20		10.56
	Passengers							1.57		5.20
Navigatio	Domestic navigation							10.05		
	Bunkers							25.96	117.76	

In Table 24 all the technologies modelled to satisfy each energy service are listed, with specifications about input commodity, share of input commodity consumption (i.e., the fraction of input commodity energy consumption that is due to the technologies, it is less than 100 if more than one technology use the same fuel) and final energy.

Table 24. Demand-side transports technologies by transportation modes.

Transportation mode		**Technology**	**Input commodity share f_{tech}[%]**	**Input commodity**	**Final energy $E_{tech,tm,af}$ [PJ]**
Air	International aviation	International aircraft	100	Kerosene	134.47
		International aircraft	100	Aviation Gasoline	0.01
	Domestic aviation	Domestic aircraft	100	Kerosene	31.51
		Domestic aircraft	100	Aviation Gasoline	0.70
Road	Cars	Gasoline car	100	Gasoline	473.04
		Diesel car	100	Diesel	297.45
		LPG car	100	LPG	45.40
		Natural gas car	100	Natural gas	15.58
		Biodiesel car	100	Biodiesel	1.97
	Three wheels	Other electric transports	100	Electricity	20.59
	Two wheels	Moped	32	Gasoline	21.14
		Motorcycle	68		44.91
	Bus	Diesel bus	100	Diesel	67.23
		Natural gas bus	100	Natural gas	1.73
		Biodiesel bus	100	Biodiesel	0.04
	Heavy trucks	Diesel heavy trucks	100	Diesel	266.85
		Biodiesel heavy trucks	100	Biodiesel	0.05
	Medium trucks	Diesel medium truck	100	Diesel	111.79
		Biodiesel medium truck	100	Biodiesel	0.73
	Commercial trucks	Gasoline commercial trucks	100	Gasoline	16.67
		Diesel commercial trucks	100	Diesel	275.34
		Biodiesel commercial trucks	100	Biodiesel	3.95
Rail	Freight	Freight diesel train	100	Diesel	3.20
		Freight electric train	100	Electricity	10.56
	Passengers	Passengers diesel train	100	Diesel	1.57
		Passengers electric train	100	Electricity	5.20
Navigation	Domestic navigation	Domestic navigation ship	100	Diesel	10.05
	Bunkers	International navigation ship	100	Diesel	25.96
		International navigation HFO ship	100	Heavy Fuel Oil	117.76

Appropriate conversion factors [14] are required to evaluate the transport service demands from final energy for each technology (derived according to Equation 14). The factors are unitary and dimensionless for service demands assumed equal to the relating energy consumption, while they represent an efficiency of conversion for the service demands quantified in $\mathrm{Bv} * \mathrm{km}$ (billion vehicles kilometer). Those factors are listed for each technology in Table 25, with the correspondent calculated service demand (according to Equation 15).

$$\mathrm{E}_{\mathrm{tech,tm,af}}[\mathrm{PJ}] = \mathrm{f}_{\mathrm{tech}}[\%] * \mathrm{E}_{\mathrm{tm,af}}[\mathrm{PJ}] \tag{14}$$

$$\mathrm{D}_{\mathrm{TRA}} = \mathrm{eff}_{\mathrm{conv}} * \mathrm{E}_{\mathrm{tech,tm,af}}[\mathrm{PJ}] \tag{15}$$

Table 25. Conversion factors and service demands for transport technologies.

Transportation mode		Technology	Conversion factor $\mathbf{eff_{conv}}$	Service demand $\mathbf{D_{TRA}}$
Air	International aviation	International aircraft	1.000	134.47 PJ
		International aircraft	1.000	0.01 PJ
	Domestic aviation	Domestic aircraft	1.000	31.51 PJ
		Domestic aircraft	1.000	0.70 PJ
Road	Cars	Gasoline car	0.299 Bvkm/PJ	141.35 Bvkm
		Diesel car	0.362 Bvkm/PJ	107.81 Bvkm
		LPG car	0.247 Bvkm/PJ	11.19 Bvkm
		Natural gas car	0.274 Bvkm/PJ	4.26 Bvkm
		Biodiesel car	0.442 Bvkm/PJ	0.87 Bvkm
	Three wheels	Other electric transports	1.000	21.03 PJ
	Two wheels	Moped	1.315 Bvkm/PJ	27.79 Bvkm
		Motorcycle	1.026 Bvkm/PJ	46.09 Bvkm
	Bus	Diesel bus	0.052 Bvkm/PJ	3.53 Bvkm
		Natural gas bus	0.038 Bvkm/PJ	0.07 Bvkm
		Biodiesel bus	0.056 Bvkm/PJ	0.00 Bvkm
	Heavy trucks	Diesel heavy trucks	0.045 Bvkm/PJ	11.97 Bvkm
		Biodiesel heavy trucks	0.027 Bvkm/PJ	0.00 Bvkm
	Medium trucks	Diesel medium truck	0.090 Bvkm/PJ	10.09 Bvkm
		Biodiesel medium truck	0.090 Bvkm/PJ	0.07 Bvkm
	Commercial trucks	Gasoline commercial trucks	0.241 Bvkm/PJ	4.02 Bvkm
		Diesel commercial trucks	0.276 Bvkm/PJ	76.10 Bvkm
		Biodiesel commercial trucks	0.276 Bvkm/PJ	1.09 Bvkm
Rail	Freight	Freight diesel train	1.000	3.20 PJ
		Freight electric train	1.000	10.56 PJ
	Passengers	Passengers diesel train	1.000	1.57 PJ
		Passengers electric train	1.000	5.20 PJ
Navigation	Domestic navigation	Domestic navigation ship	1.000	10.05 PJ
	Bunkers	International navigation ship	1.000	25.96 PJ
		International navigation HFO ship	1.000	117.76 PJ

The total service demand for each end-use is given by the sum of final services produce by all the technologies related to the same end-use. Results are reported in Table 26. It is clear for the results that, for instance, the higher energy consumption for aviation is related to international flights, cars represent the highest final demand in road category in terms of $Bv * km$ and about two thirds of rail energy consumption is due to freight transport.

Table 26. Service demands for transports sector.

End-use	$\mathbf{D_{serv}}$	
International aviation	134.48	PJ
Domestic aviation	32.21	PJ
Cars	344.24	$Bv * km$
Two wheels	55.67	$Bv * km$
Bus	6.29	$Bv * km$
Heavy trucks	29.80	$Bv * km$
Medium trucks	18.94	$Bv * km$
Commercial trucks	84.57	$Bv * km$
Freight	13.75	PJ
Passengers	6.77	PJ
Domestic navigation	10.05	PJ
Bunkers	143.72	PJ
Others	21.03	PJ

Figure 7 summarizes data organization and interactions, from IEA statistics to the calculation of service demand.

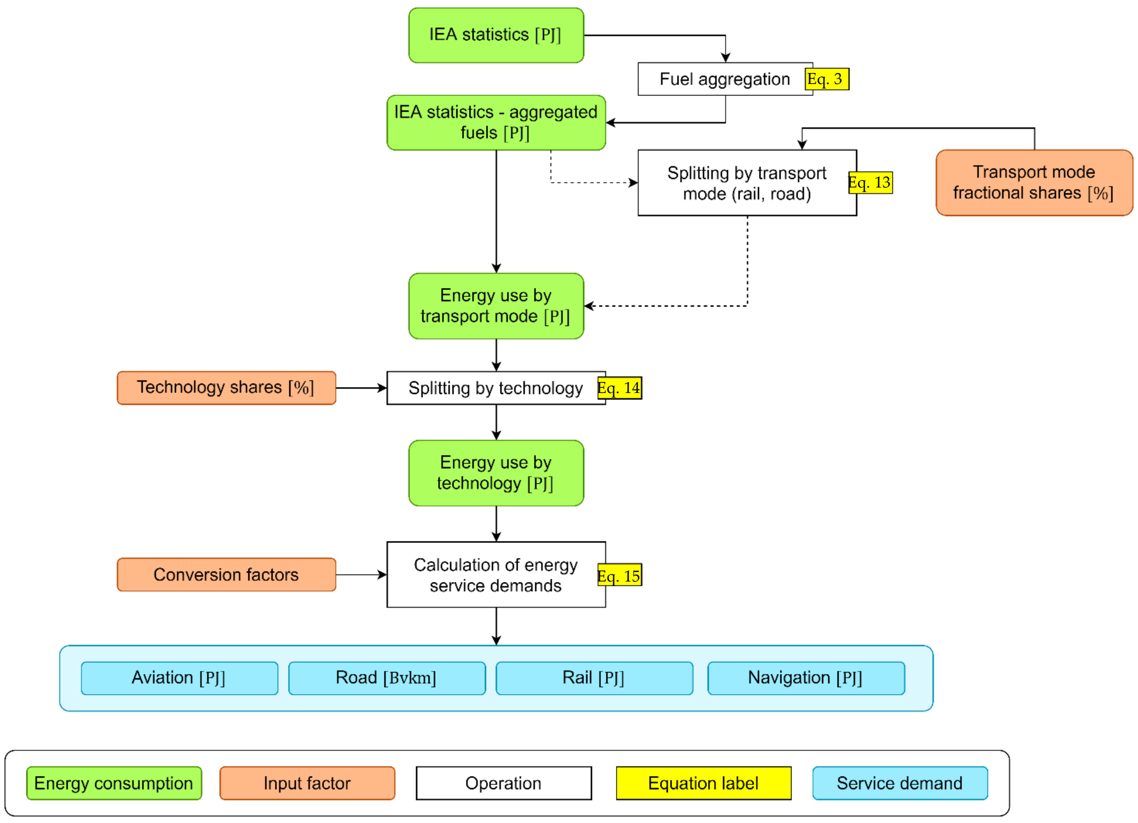


Figure 7. Flow-chart for the evaluation of service demand in the transport sector. Associated to each step are specified the involved equations, as listed in this report.

Eventually, the computation of emission factors from residential sector leads to the values that follow in Table 27. In this case, the highest CO_2 emission factor is associate to diesel and heavy fuel oil, even higher than gasoline one. It is known that, for example, gasoline cars emit more CO_2 per unit of kilometer traveled than diesel cars and it could seem to be contradictory with these results, but it should be noted that that is not due to the fuel chemical composition (associated to the computed emission factors) but to the higher efficiency of the diesel engine with respect to the gasoline one.

Table 27. Emission factors for transports sector-specific fuels.

Commodity	**Emission factors [kt/PJ]**								
	Natural gas	LPG	Gasoline	Aviation Gasoline	Kerosene	Diesel	Heavy fuel oil	Methanol	Ethanol
CO_2	56.10	63.07	69.30	69.30	71.50	74.07	74.07	0.00	0.00
CH_4	1.10	1.18	6.92	60.00	5.53	1.32	1.32	0.02	0.02
N_2O	1.00	9.00	6.60	6.86	6.10	3.36	3.36	0.00	0.00

2.2.1.5. Industry

Base year energy consumptions for the industrial sector are collected from IEA energy statistics, as for the other sectors. In order to assess industrial energy use, 10 subsectors are identified, as listed in Table 28.

Table 28. Industry subsectors.

Industry subsectors
Chemicals
Iron and steel
Non-ferrous metals
Non-metallic minerals
Pulp and paper
Other industries
Non-specified industry
Chemical feedstocks
Non-energy uses
Non-energy others

The first operation is to aggregate fuel consumptions to obtain data with a lower number of fuels for a simpler elaboration. This is done, as for other sectors, accordingly to Equation 3 and results are contained in Table 29. In the table header, the IEA fuels (according to the classification adopted in [17]) are listed in the second row and they are associated to the corresponding aggregated fuel reported in the third row.

Table 29. Energy consumption from IEA statistics by aggregated fuels of industrial energy-intensive subsectors.

Subsectors	**Energy consumption by aggregated fuels E_{af} [PJ]**												
	Electricity	Natural gas, Gas work gas	LPG	Hard coal, Patent fuel, Anthracite, Other bituminous coal, Peat, Brown coal briquettes, Brown coal, Sub-bituminous coal, Lignite	Coke oven coke	Gas coke, Blast furnace gas	Crude oil, Fuel oil, Other non-specified, Non crude oil	Diesel, Other kerosene, Motor gasoline, Aviation gasoline, Jet kerosene	Refinery gas, Ethane	Naphtha	Petroleum coke	Solid biomass, Charcoal, Gas biomass, Other liquid biofuels, Municipal waste (non-renewable), Municipal waste (renewable), Industrial waste	Heat
	Electricity	Natural Gas	LPG	Coal	Coke	Blast furnace gas	Heavy fuel oil	Oil	Ethane	Naphtha	Petroleum coke	Biomass	Heat
Iron and steel	78.09	78.98	1.10	13.80	69.34	0.18	3.48	0.38			0.13		0.24
Non-ferrous metals	20.71	16.67	0.87	0.05	0.46		2.00	0.34					0.02
Chemicals	67.83	109.03	1.20	0.08	0.17		15.84	11.63	6.19		0.32		43.45
Pulp and paper	39.17	36.78	0.41				8.08	0.94					30.92
Non-metallic minerals	53.80	146.99	6.62	24.97	0.17		17.16	2.17			106.21	8.30	4.74
Other industries	207.44	196.47	6.76				55.44	13.63				0.65	18.75
Non-specified industry	63.54	32.08	1.10		0.41		14.76	1.58				2.60	22.70
Chemical feedstocks		39.61	0.64				18.48	42.20	19.40	124.52			
Non-energy uses							153.44						
Non-energy others				7.03			14.84						

A higher level of detail is used for the representation of the industrial sector. Particularly, a breakout in different energy services is performed for the following subsectors: iron and steel, non-ferrous metals, chemicals, pulp and paper, non-metallic minerals, other industries. The energy services chosen for the breakout are steam, process heat, machine drive, electro-chemical processes, feedstocks, and others. Each of those energy services consumes fuels as inputs and produces its correspondent energy-service output industry specific-commodity. The energy service output is then the input for the subsector-specific technologies, that produce the required industrial products. For this reason, before applying the technology-specific efficiency, a splitting by energy service is necessary to differentiate energy consumption not only by fuels and industrial energy-intensive subsectors, but also by different energy services of the same subsector.

An exception to the mechanism just explained, for which fuels are used to produce the energy services (representing the inputs for the technologies that satisfy final industrial demand), is constituted by the process heat energy service. Indeed, no intermediate commodities are defined for that service, but fuels are connected directly and separately to the industrial technologies to produce final products. However, for pulp and paper and other industries sectors and intermediate commodity is used collecting the process heat energy service and consumed by the production technologies.

The breakout by energy services is done, as usual, adopting share factors [14] for the splitting of consumptions in every subsector by each single energy service. The share factors assumed for this operation are shown in Table 30. Th splitting is governed by Equation 16, where f_{es} [%] is the energy service share factor, and $C_{es, af}$ [PJ] is the resulting consumption for the correspondent energy service. The resulting values are reported in Table 31.

$$E_{es, af}\,[PJ] = f_{es}\,[\%] * E_{af}\,[PJ] \qquad (16)$$

Before applying the subsector-specific technological parameter, another step is necessary. Indeed, the “steam” and “machine drive” energy services are not described in a separate way for each subsector, but in two sheets dedicated to total industrial energy consumption for such services, so it is necessary to know the total energy consumption assigned to each energy service (not differentiated by subsector). Results are obtained from Equation 17 and shown in Table 32, where energy consumption is reported split by fuel, energy service and industrial subsector. The energy service labeled as “other” is used to collect the energy consumption non-associated to the five specified energy services (steam, process heat, machine drive, electro-chemical process and feedstocks).

$$E_{tot,es}[PJ] = \sum_{i} E_{es,af}\,[PJ] \tag{17}$$

Table 30. Fractional shares for industrial energy services.

Energy-intensive subsector	Energy services	Fractional shares for industrial energy services f_{es} [%]											
		Electricity	Natural gas	LPG	Coal	Coke	Blast furnace gas	Heavy fuel oil	Oil	Ethane	Petroleum coke	Biomass	Heat
Chemicals	Steam		45		100		100	30					100
	Process Heat		20					15	7				
	Machine Drive	65							3				
	Electro-Chemical Process	25											
	Feedstocks												
	Other	10	35	100	0	100	0	55	90	100	100	100	0
Iron and steel	Steam		15	15				15	15				100
	Process Heat	40	85	85	100		100	85	85				
	Machine Drive	47											
	Electro-Chemical Process												
	Feedstocks					100				100	100		
	Other	13	0	0	0	0	0	0	0	0	0	100	0

Continued on page 65

Continued from page 64

Non-ferrous metals	Steam		10		25		100	25	25			100	100
	Process Heat		85	100	75	70		75	75				
	Machine Drive	10											
	Electro-Chemical Process	85											
	Feedstocks					10							
	Other	5	5	0	0	20	0	0	0	100	100	0	0
Non-metallic minerals	Steam		10				100	20	35				100
	Process Heat	20	85	90	100			80	47		100	100	
	Machine Drive	70							8				
	Electro-Chemical Process												
	Feedstocks												
	Other	10	5	10	0	100	0	0	10	100	0	0	0
Pulp and paper	Steam		93	85	100		100	95	90			100	100
	Process Heat	3	7					5	5				
	Machine Drive	94											
	Electro-Chemical Process												
	Feedstocks												
	Other	3	0	15	0	100	0	0	5	100	100	0	0
Other industries	Steam		10	10	35			30	30			100	100
	Process Heat		90	90	65		100	70	70				
	Machine Drive	85											
	Electro-Chemical Process												
	Feedstocks												
	Other	15	0	0	0	100	0	0	0	100	100	0	0

Table 31. Breakout by energy services for industrial sector.

Energy-intensive subsector	**Energy service**	**Breakout by energy services $E_{es, af}$ [PJ]**											
		Electricity	Natural Gas	LPG	Coal	Coke	Blast furnace gas	Heavy fuel oil	Oil	Ethane	Petroleum coke	Biomass	Heat
Chemicals	Steam		49.06		0.08		0.00	4.75					43.45
	Process Heat		21.81					2.38	0.81				
	Machine Drive	44.09							0.35				
	Electro-Chemical Process	16.96											
	Feedstocks												
	Other	6.78	38.16	1.20	0.00	0.17	0.00	8.71	10.46	6.19	0.32	0.00	0.00
Iron and steel	Steam		11.85	0.17				0.52	0.06				0.24
	Process Heat	31.46	67.14	0.94	13.80		0.18	2.96	0.33				
	Machine Drive	36.56											
	Electro-Chemical Process												
	Feedstocks					69.34				0.00	0.13		
	Other	10.08	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Continued on page 67

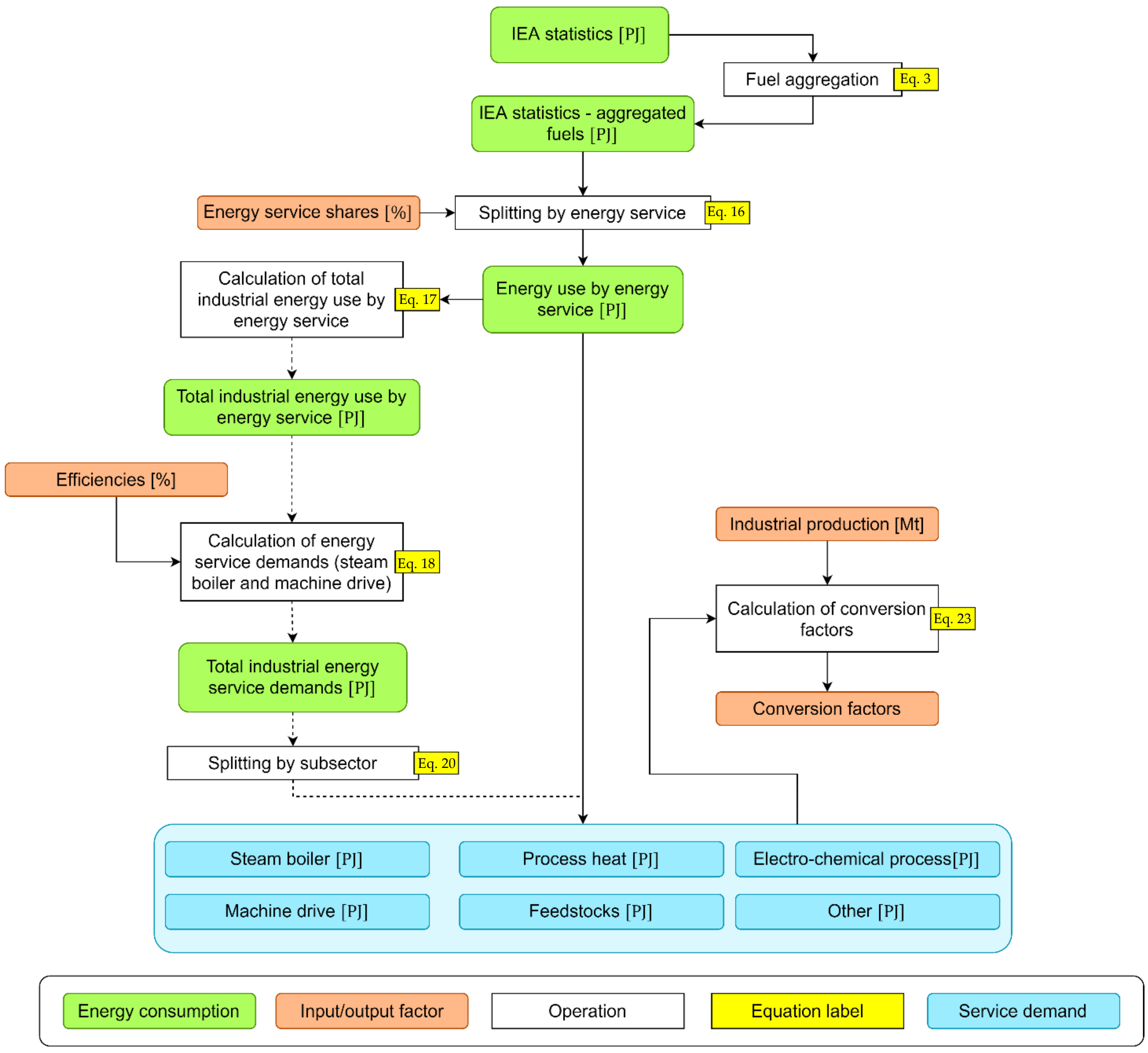
Continued from page 66

Non-ferrous metals	Steam		1.67		0.01		0.00	0.50	0.09			0.00	0.02
	Process Heat		14.17	0.87	0.04	0.32		1.50	0.26				
	Machine Drive	2.07											
	Electro-Chemical Process	17.61											
	Feedstocks					0.05							
	Other	1.04	0.83	0.00	0.00	0.09	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Non-metallic minerals	Steam		14.70				0.00	3.43	0.76				4.74
	Process Heat	10.76	124.94	5.96	24.97			13.73	1.02		106.21	8.30	
	Machine Drive	37.66							0.17				
	Electro-Chemical Process												
	Feedstocks												
	Other	5.38	7.35	0.66	0.00	0.17	0.00	0.00	0.22	0.00	0.00	0.00	0.00
Pulp and paper	Steam		34.20	0.35	0.00	0.00	0.00	7.68	0.84			0.00	30.92
	Process Heat	1.13	2.57					0.40	0.05				
	Machine Drive	36.91											
	Electro-Chemical Process												
	Feedstocks												
	Other	1.13	0.00	0.06	0.00	0.00	0.00	0.00	0.05	0.00	0.00	0.00	0.00
Other industries	Steam		19.65	0.68	0.00			16.63	4.09			0.65	18.75
	Process Heat		176.82	6.09	0.00		0.00	38.81	9.54				
	Machine Drive	176.33											
	Electro-Chemical Process												
	Feedstocks												
	Other	31.12	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 32. Total industrial energy consumption by energy services for chemicals, iron and steel, non-ferrous metals, non-metallic minerals, pulp and paper and other industries.

Energy services	**Total energy consumption by energy services $E_{tot,es}$ [PJ]**											
	Electricity	Natural Gas	LPG	Coal	Coke	Blast furnace gas	Heavy fuel oil	Oil	Ethane	Petroleum coke	Biomass	Heat
Steam		131.13	1.19	0.09			33.51	5.83			0.65	98.12
Process Heat	43.35	407.45	13.86	38.81	0.32	0.18	59.77	12.00		106.21	8.30	
Machine Drive	333.62							0.52				
Electro-Chemical Process	34.56											
Feedstocks					69.39					0.13		
Other	55.52	46.34	1.92		0.44		8.71	10.73	6.19	0.32		

In Figure 8, the main operations regarding the industrial sector are summarized.



IEA statistics [PJ]
Fuel aggregation
Eq. 3
IEA statistics - aggregated fuels [PJ]
Energy service shares [%]
Splitting by energy service
Eq. 16
Calculation of total industrial energy use by energy service
Eq. 17
Energy use by energy service [PJ]
Total industrial energy use by energy service [PJ]
Efficiencies [%]
Calculation of energy service demands (steam boiler and machine drive)
Eq. 18
Industrial production [Mt]
Calculation of conversion factors
Eq. 23
Conversion factors
Total industrial energy service demands [PJ]
Splitting by subsector
Eq. 20
Steam boiler [PJ]
Process heat [PJ]
Electro-chemical process[PJ]
Machine drive [PJ]
Feedstocks [PJ]
Other [PJ]
Energy consumption
Input/output factor
Operation
Equation label
Service demand

Figure 8. Flow-chart for the evaluation of energy service demand in the industrial sector. Associated to each step are specified the involved equations, as listed in this report.

For what concerns the industry sector-specific emission factors, they are evaluated as for other sectors, according to Equation 2. As anticipated, for the industrial sector also the emission of SO_X is considered, and the correspondent emission factors are calculated for LPG, coal, coke, heavy fuel oil, oil, ethane and petroleum coke. The results are reported in Table 33, where high values of CO_2 emission are associated to coal and petroleum coke, while it is zero for biomass. As already seen for commercial and residential sector, biomass is characterized by a high CH_4 emission factor.

Table 33. Emission factors for industry sector-specific fuels.

Commodity	Emission factors [kt/PJ]									
	Natural gas	LPG	Coal	Coke	Blast furnace gas	Heavy fuel oil	Oil	Ethane	Petroleum coke	Biomass
CO_2	56.10	63.10	98.30	94.60	0.00	77.40	72.00	54.10	100.50	0.00
CH_4	5.00	5.00	5.00	5.00	5.00	2.00	2.00	5.00	5.00	300.00
N_2O	0.10	0.10	1.40	1.40	1.40	0.60	0.60	0.10	1.40	4.00
SO_X		0.57	1.20	1.20		0.57	0.57	0.57	0.57	

As anticipated, the next steps of the conversion towards the evaluation of service demands are slightly different for "steam" and "machine drive", and again different computations are done for the different industrial energy-intensive subsectors. For that reason, it is useful to examine separately for each subsector which are the required operations.

2.2.1.5.1. Steam

From Table 32 it is possible to read the total final energy consumed by fuel for steam production $E_{s,fe}$. From those values, the values of useful energy $E_{s,ue}$ are derived, simply multiplying by the correspondent boiler efficiency $eff_{tech,s}$ (Equation 18). The results of this first step are reported in Table 34. The results are then added together (Equation 19) to obtain the total service demand for the "steam" energy service D_s which amounts to 234.83 PJ. The total demand is split (Equation 20) assigning a fraction of it to the i^{th} subsector $D_{s,i}$, according to the consumption share $f_{s,cons}$, in terms of final energy consumed. The derived service demands for each subsector are shown in Table 35, highlighting that the major steam consumption are related to "chemicals" and "pulp and paper" subsectors.

$$E_{s,ue}\,[PJ] = eff_{tech,s}\,[\%] * E_{s,fe}\,[PJ] \qquad (18)$$

$$\mathrm{D_s\ [PJ]} = \sum \mathrm{E_{s,ue}\ [PJ]} \tag{19}$$

$$\mathrm{D_{s,i}\ [PJ]} = \mathrm{f_{s,cons}\ [\%]} * \mathrm{D_s\ [PJ]} \tag{20}$$

Table 34. Final energy, efficiencies, and useful energy for "steam" energy-service.

Fuel	$\mathbf{E_{s,fe}}$[**PJ**]	$\mathbf{eff_{tech,s}}$[%]	$\mathbf{E_{s,ue}}$[**PJ**]
Natural gas	131.13	82	107.52
LPG	1.19	82	0.98
Coal	0.09	82	0.08
Heavy fuel oil	33.51	82	27.48
Oil	5.83	82	4.78
Biomass	0.65	82	0.53
Heat	98.12	100	98.12

Table 35. Splitting by different subsectors of "steam" service demand.

Subsector	$\mathbf{D_{s,i}}$ [**PJ**]
Chemicals	86.18
Iron and steel	11.36
Non-ferrous materials	2.02
Non-metallic minerals	20.92
Pulp and paper	65.51
Other industries	53.51

2.2.1.5.2. Machine drive

For what concerns the "machine drive" service, the operations to be performed are very similar to what just seen for steam production. From Table 32 it is possible to read the total final energy consumed from "machine drive" technologies differentiated by fuel $E_{md,fe}$. Starting from those values, useful energy $E_{md,ue}$ are derived, simply multiplying by the correspondent efficiency $eff_{tech,md}[\%]$ (Equation 18). This first step is reported in Table 36. The results are then added together (Equation 19) to obtain the total service demand for "machine drive" D_{md}, that amounts to 287.09 PJ. The total demand is split (Equation 20) assigning a fraction of it to the i^{th} subsector $D_{md,i}$, according to the share of consumption $f_{md,cons}$ in terms of final energy consumed. The derived service demands for every subsector are shown in Table 37, highlighting that the major steam consumption are related to "other industries" and "chemicals" subsectors.

Table 36. Final energy, efficiencies, and useful energy for "machine drive".

Fuel	$\mathbf{E_{md,fe}[PJ]}$	$\mathbf{eff_{tech,md}[\%]}$	$\mathbf{E_{md,ue}[PJ]}$
Electricity	333.62	86%	286.91
Oil	0.52	35%	0.18

Table 37. Splitting by different subsectors of "machine drive" service demand.

Subsector	$\mathbf{D_{md,i}}$ **[PJ]**
Chemicals	38.18
Iron and steel	31.41
Non-ferrous materials	1.78
Non-metallic minerals	32.51
Pulp and paper	31.71
Other industries	151.50

2.2.1.5.3. Chemicals

In TIMES-Italia [14], the chemical sector is very simply described, with only one fictitious production technology consuming all the fuels related to the sector and producing the chemical service demand (assumed to be equal to the total energy consumption of the sector, in other words with unitary efficiency for the production technology).

A more detailed techno-economic modeling of the sector was desired, in order to have a precise distinction between the several chemical products produced and the related technologies producing them, characterized properly and independently one each other. To achieve this goal, a new calibration of the base year for the chemical sector has been performed, taking as a reference for the techno-economic technologies characterization the EUROfusion TIMES Model (ETM) [21], an international energy system model developed by EUROfusion consortium [22].

Five main chemical products are considered for the calibration: olefins, aromatics (BTX), ammonia, methanol, and chlorine. A sixth commodity is produced by a dedicated base year technology, namely "other chemicals", collecting the residual energy consumption between the resulting from energy balances and the base year technologies consumption.

First of all, the amount of chemical products produced at base year has to be quantified.

Table 38. Chemical production statistics (from [23] and [24]).

Chemical product	**Production**			**2006 production**
	[Mt]	Year	Reference	$\mathbf{C_{prod}}$ **[Mt]**
Olefins	2.98	2017	[23]	2.94
Aromatics (BTX)	0.81	2017	[23]	0.80
Ammonia	0.60	2010	[24]	0.64
Methanol	0.05	2017	[23]	0.05
Chlorine	0.20	2010	[24]	0.21

Once the production C_{prod} of chemical products is known, assuming appropriate conversion factors it is possible to estimate the energy consumption required to produce each product. The calibration has been performed assuming the same conversion factors eff_{conv} of ETM [21], evaluating consequently the energy services E_{es} (according to Equation 22) and assigning to the sixth chemical subsector (other chemical) the residual energy consumption.

Table 39 summarizes the main results of the calibration. It should be noted that the higher total energy consumption within "chemicals" id due to olefins production (220.68 PJ). Also, the efficiencies of olefins and aromatics production are very similar (75.18 PJ/Mt and 73.21 PJ/Mt), being the aromatics byproducts of the olefins production.

$$E_{es}\,[PJ] = eff_{conv}\,[PJ/Mt] * C_{prod}[Mt] \qquad (21)$$

Table 39. Calibration and techno-economic characterization of chemical sector base year.

Chemical product	**Energy service**	**Fuel**	**Share factors for energy services $\mathbf{f_{tech}}$ [%]**	**Energy service $\mathbf{E_{es}}$ [PJ]**	**Conversion factors $\mathbf{eff_{conv}}$ [PJ/Mt]**
Olefins	Electro-chemical		12.7	2.15	0.73
	Process heat	Natural Gas	27.2	16.29	5.55
		LPG	87.8	1.05	0.36
		Heavy fuel oil	57.5	6.37	2.17
		Oil	46.3	5.22	1.78
		Ethane	48.9	3.02	1.03
	Feedstocks		63.2	154.61	52.67
	Steam		37.0	31.86	10.85
	Machine drive		0.3	0.11	0.04
	Total			220.68	75.18
Aromatics (BTX)	Electro-chemical		2.7	0.46	0.57
	Process heat	Natural Gas	4.3	2.60	3.26
		LPG	18.7	0.22	0.28
		Heavy fuel oil	12.3	1.36	1.70
		Oil	9.9	1.11	1.39
		Ethane	20.8	1.29	1.62
	Feedstocks		17.2	42.16	52.85
	Steam		10.6	9.11	11.42
	Machine drive		0.3	0.10	0.12
	Total			58.41	73.21
Ammonia	Electro-chemical		3.4	0.58	0.91
	Process heat	Natural Gas	33.0	19.81	31.04
	Feedstocks		3.5	8.65	13.56
	Steam		14.2	12.25	19.19
	Machine drive		1.5	0.56	0.88
	Total			41.86	65.57
Methanol	Electro-chemical		0.2	0.03	0.69
	Process heat	Natural Gas	1.4	0.81	16.39
	Feedstocks		0.3	0.76	15.38
	Steam		0.9	0.73	14.92
	Machine drive		0.2	0.07	1.47
	Total			2.41	48.86
Chlorine	Electro-chemical		13.4	2.26	10.64
	Steam		0.5	0.41	1.93
	Machine drive		0.2	0.06	0.28
	Total			2.73	12.85
Other chemicals	Electro-chemical		67.6	11.47	0.71
	Feedstock		15.8	38.67	2.41
	Steam		36.9	31.82	1.98
	Steam		97.6	37.28	2.32
	Machine drive		100.0	41.30	2.57
	Total			160.53	10.00

For each energy service, service-specific technologies are used to convert fuel consumption into energy service demands (that are the inputs for technologies that produce final industrial products). In Table 40 all those technologies are listed – organized by energy service – with their input commodity and final energy consumption.

Table 40. Demand-side chemical technologies by energy services.

Energy service	**Technology description**	**Input commodity**	**Final energy [PJ]**
Electro-chemical	Chemical – Electro-chemical – Electricity	Electricity	16.96
Feedstock	Chemical – Feedstock – Natural gas	Natural gas	39.61
	Chemical – Feedstock – LPG	LPG	0.64
	Chemical – Feedstock – Kerosene	Kerosene	8.69
	Chemical – Feedstock – Heavy fuel oil	Heavy fuel oil	17.32
	Chemical – Feedstock – Distillate	Distillate	27.26
	Chemical – Feedstock – Oil	Oil	1.16
	Chemical – Feedstock – Gasoline	Gasoline	6.25
	Chemical – Feedstock – Naphtha	Naphtha	124.52
	Chemical – Feedstock – Refinery gas	Refinery gas	19.40
Other	Chemical – Others – Heavy fuel oil	Heavy fuel oil	3.36
	Chemical – Others – Oil	Oil	4.95
	Chemical – Others – Natural gas	Natural gas	20.45
	Chemical – Others – Coke	Coke	0.17
	Chemical – Others – Ethane	Ethane	1.87
	Chemical – Others – Electricity	Electricity	10.17
	Chemical – Others – Petroleum coke	Petroleum coke	0.32

2.2.1.5.4. Iron and steel

The "iron and steel" subsectors produces steel as end-use demand commodity. Steel production values are known from industrial production statistics. Two technologies are present at the base year: basic oxygen furnace (BOF), producing 11.82 Mt, and electric arc furnace (EAF), producing 19.80 Mt. Final energy consumptions $E_{es,af}$ are reported in Table 31 and they are used to derive (Equation 22) the energy service demands E_{serv} [PJ], assuming appropriate share factors f_{tech} to split by technology; the final step involves the calculation of conversion factors between energy service demands and industrial products (Equation 23).

The main results are reported in Table 41, where values are differentiated by technology, energy service demand and consumed fuel (the specification of fuel is necessary for the process heat service demand, since a dedicated technology is not defined for process heat in iron and steel, but the fuel consumption are reported directly from statistics, separately).Concerning the share factors for service demands, it should be noted that some of the listed fuels (for the process heat service) are present only for one of the two technologies. In these cases, the factor is unitary for the other technology, and the sum of correspondent factors for BOF and EAF is equal to 100%. An exception in the previous reasoning is the splitting of natural gas energy consumption, that is fixed equal to 6.56 PJ for the EAF, and consequently its consumption for BOF is the difference between the total consumption of natural gas for "process heat" in "iron and steel" (from Table 31) and this value.

$$E_{es}\,[\mathrm{PJ}] = f_{tech}\,[\%] * E_{es,af}\,[\mathrm{PJ}] \tag{22}$$

$$eff_{conv}\,[\mathrm{Mt/PJ}] = P\,[\mathrm{Mt}]/E_{es}\,[\mathrm{PJ}] \tag{23}$$

Table 41. Calibration and techno-economic characterization of "iron and steel" sector base year.

Technology	Energy service	Fuel	Share factors for energy service demands f_{tech} [%]	Energy service E_{es} [PJ]	Conversion factors eff_{conv} [Mt/PJ]
Basic oxygen furnace	Steam		46	5.18	0.44
	Process heat	Natural Gas		60.57	5.12
		Coal	50	6.90	0.58
		Blast Furnace Gas	100	0.18	0.02
		Heavy Fuel Oil	100	2.96	0.25
		LPG	100	0.94	0.08
	Machine drive		30	9.50	0.80
	Feedstocks		90	62.83	5.31
	Others		37	3.77	0.32
	Total			152.83	0.08
Electric arc furnace	Steam		54	6.18	0.31
	Process heat	Natural Gas		6.56	0.33
		Coal	50	6.90	0.35
		Electricity	100	31.46	1.59
		Oil	100	0.33	0.02
	Machine drive		70	21.91	1.11
	Feedstocks		10	6.63	0.33
	Others		63	6.31	0.32
	Total			86.28	0.23

For each energy service, service-specific technologies are used to convert fuel consumption into energy service demands (that are the inputs for technologies that produce final industrial products). In Table 42 all those technologies are listed, along with their input commodity and final energy consumption. Obviously, the total production of "feedstocks" and "other" energy services equals their consumption by production technologies listed in Table 41.

Table 42. Demand-side iron and steel technologies by energy services.

Energy service	Technology description	Input commodity	Final energy [PJ]
Feedstocks	Iron and steel – Feedstocks – Coke	Coke	69.34
	Iron and steel – Feedstocks – Petroleum coke	Petroleum coke	0.13
Other	Iron and steel – Others – Electricity	Electricity	10.08

2.2.1.5.5. Non-ferrous metals

The "non-ferrous metals" subsector produces a single end-use demand commodity ("non-ferrous metals production", measured in Mt) representing the production of aluminum, copper, zinc and "other non-ferrous metals". Table 43 lists the numeric parameters involved in the techno-economic characterization of "non-ferrous metals" industrial subsector [21]. Conversion factors $\mathrm{eff_{conv}}$ are used to describe the efficiency of existing technologies to produce non-ferrous metals (aluminum, copper, zinc and "other non-ferrous metals"), while share factors $\mathrm{f_{tech}}$ are used to match the energy services required by each technology with the energy consumptions derived by IEA statistics.

Table 43. Calibration and techno-economic characterization of "non-ferrous metals" sector base year.

Non-ferrous metal	Energy service	Fuel	Share factors for energy services $\mathbf{f_{tech}}$ [%]	Energy service $\mathbf{E_{es}}$ [PJ]	Conversion factors $\mathbf{eff_{conv}}$ [PJ/Mt]
Aluminum	Electro-chemical		10.69	1.06	0.51
	Process heat	Natural Gas	56.92	8.54	4.07
		Heavy fuel oil	0.31	0.31	0.15
	Total			9.91	4.72
Copper	Electro-chemical		47.3	4.70	11.90
	Process heat	Natural Gas	17.3	2.59	6.55
		Coal	100.0	0.04	0.10
		Coke	100.0	0.46	1.17
		Heavy fuel oil	47.0	0.71	1.79
	Steam		71.7	1.45	3.67
	Total			9.95	25.18
Zinc	Electro-chemical		42.0	4.17	11.06
	Process heat	Natural Gas	6.0	0.89	2.37
		Heavy fuel oil	32.5	0.49	1.29
	Steam		28.4	0.57	1.52
	Machine drive		28.6	0.51	1.35
	Total			6.63	17.59
Other non-ferrous metals	Machine drive		71.4	1.27	0.90
	Other		100.0	12.82	9.10
	Total			14.09	10.0

For each energy service, service-specific technologies are used to convert fuel consumption into energy service demands (that are the inputs for technologies producing final industrial products). In Table 44 all those technologies are listed – organized by energy service – with their input commodity and final energy consumption. Obviously, the total production of "electro-chemical" and "other" energy services equals their consumption by production technologies listed in Table 43.

Table 44. Demand-side non-ferrous metals technologies by energy services.

Energy service	**Technology description**	**Input commodity**	**Final energy [PJ]**
Electro-chemical	Non-ferrous metals – Electro-chemical – Electricity	Electricity	9.93
Other	Non-ferrous metals – Others – Electricity	Electricity	8.71
	Non-ferrous metals – Others – Oil	Oil	0.26
	Non-ferrous metals – Others – Natural gas	Natural gas	2.98
	Non-ferrous metals – Others – LPG	LPG	0.87

2.2.1.5.6. Non-metallic minerals

The "non-metallic minerals" subsector is based on dedicated statistics for what concerns splitting of energy consumption in the five technologies that are part of it producing: cement, lime, glass, and bricks. For this reason, only a summary of the results is here reported, in Table 45, where technologies, total production, energy consumed by energy service and consequent conversion factors (the specification of fuel is necessary for the process heat service demand, since a dedicated technology is not defined for process heat in iron and steel, but the fuel consumption are reported directly from statistics, separately) are shown.

Table 45. Production, service demands and conversion factors for "non-metallic minerals".

Technology	Production [Mt]	Energy service	Fuel	Energy service E_{es} [PJ]	Conversion factors eff_{conv} [PJ/Mt]
Wet cement kilns	13.65	Steam		0.34	0.03
		Process heat	Natural Gas	0.80	0.06
			Coal	6.67	0.49
			Heavy Fuel Oil	3.50	0.26
			Petroleum coke	43.11	3.16
			Electricity	1.56	0.11
			Oil	0.76	0.06
			LPG	0.50	0.04
		Machine drive		5.54	0.41
		Others		0.40	0.03
Dry cement kilns	34.23	Steam		0.13	0.00
		Process heat	Natural Gas	0.20	0.01
			Coal	1.50	0.04
			Heavy Fuel Oil	0.30	0.01
			Petroleum coke	63.09	1.84
			Electricity	8.30	0.24
			Oil	3.01	0.09
			LPG	0.10	0.00
		Machine drive		10.69	0.31
		Others		0.77	0.02

Continued on page 82

Continued from page 81

Lime	6.13	Steam		4.07	0.78
		Process heat	Natural Gas	6.70	1.29
			Coal	10.70	2.06
			Heavy Fuel Oil	2.10	0.40
			Electricity	0.40	0.08
			LPG	5.37	1.03
		Machine drive		1.73	0.33
		Others		4.73	0.91
Glass	6.54	Steam		3.88	1.07
		Process heat	Natural Gas	30.37	8.34
			Heavy Fuel Oil	1.91	0.53
			Electricity	0.82	0.22
		Machine drive		3.18	0.87
		Others		4.84	1.33
Bricks	20.60	Steam		12.50	2.06
		Process heat	Natural Gas	86.87	14.29
			Coal	6.10	1.00
			Heavy Fuel Oil	5.92	0.97
			Electricity	0.27	0.04
			Oil	2.92	0.48
		Machine drive		11.37	1.87
		Others		3.05	0.50

For each energy service, service-specific technologies are used to convert fuel consumption into energy service demands (that are the inputs for technologies that produce final industrial products). In Table 46 all those technologies are listed – organized by energy service – with their input commodity and final energy consumption. The only intermediate energy service produced for the non-metallic minerals subsector is "other".

Table 46. Demand-side non-metallic minerals technologies by energy services.

Energy service	Technology description	Input commodity	Final energy [PJ]
Other	Other – Non-metallic minerals – Oil	Oil	0.22
	Other – Non-metallic minerals – Natural gas	Natural gas	7.35
	Other – Non-metallic minerals – Coke	Coke	0.17
	Other – Non-metallic minerals – Electricity	Electricity	5.38
	Other – Non-metallic minerals – LPG	LPG	0.66

2.2.1.5.7. Pulp and paper

The "pulp and paper" subsector consists in three different technologies for pulp production, and one technology representing the "paper mill", for production of paper (the end-use demand commodity) from pulp, which is then just an intermediate commodity used to generate the final production of paper of this subsector. The list of technologies and their production capacity at the base year are shown in Table 47. For this subsector, the splitting of energy service amounts to different technologies is done according to factor shares f_{tech} [%] that are taken from statistics specific to the paper production [14], as Table 48 shows.

Table 47. Base year technologies and production demands for "pulp and paper".

Technology	**Final product**	**Production [Mt]**
Chemical pulp	Pulp	0.15
Mechanical pulp	Pulp	0.35
Recycled pulp	Pulp	5.58
Paper mill	Paper	10.01

As previously operated for "iron and steel", E_{es} [PJ] and eff_{conv} [Mt/PJ] are derived according to Equation 22 and Equation 23.

Table 48. Share factors for service demands, service demands and conversion factors for "pulp and paper".

Technology	Energy service	Share factors for service demands f_{tech} [%]	Energy service E_{es} [PJ]	Conversion factors eff_{conv} [PJ/Mt]
Chemical pulp	Steam	2.41	1.58	10.22
	Process heat	5.87	0.24	1.58
	Machine drive	0.97	0.31	2.00
Mechanical pulp	Steam	-1.36 [3]	-0.89 [3]	-2.56 [3]
	Machine drive	7.22	2.29	6.59
Recycled pulp	Steam	1.74	1.14	0.20
	Process heat	6.62	0.28	0.05
	Machine drive	20.38	6.46	1.16
	Other	23.94	0.30	0.05
Paper mill	Steam	97.20	63.68	6.36
	Process heat	87.50	3.64	0.36
	Machine drive	71.42	22.65	2.26
	Other	76.06	0.94	0.09

For each energy service, service-specific technologies are used to convert fuel consumption into energy service demands (that are the inputs for technologies that produce final industrial products). In Table 49 those technologies are listed (organized by energy services) with the final energy consumption.

Table 49. Demand-side pulp and paper technologies by energy services.

Energy service	Technology description	Input commodity	Final energy [PJ]
Process heat	Pulp and paper – Process heat – Heavy fuel oil	Heavy fuel oil	0.40
	Pulp and paper – Process heat – Oil	Oil	0.05
	Pulp and paper – Process heat – Natural gas	Natural gas	2.57
	Pulp and paper – Process heat – Electricity	Electricity	1.13
Other	Pulp and paper – Others – Oil	Oil	0.05
	Pulp and paper – Others – Electricity	Electricity	1.13
	Pulp and paper – Others – LPG	LPG	0.06

[3] Consumption of steam for mechanical pulp is negative since steam is produced by the technology.

2.2.1.5.8. Other industries

The "other industries" subsector collects the industrial energy consumptions of all other non-specified minor industrial subsectors (the end-use demand commodity is named "other industries" and measured in PJ). All the values of service demands are simply assumed to be equal to the sum of energy consumption of all the fuels for the correspondent industrial energy services, written in Table 31. The resulting values of service demands for "other industries" subsectors are shown in Table 50.

Table 50. Service demands and conversion factors for "other industries".

Energy service	**Service demands D_{serv} [PJ]**
Steam	53.51
Process heat	231.26
Machine drive	151.50
Other	31.12
TOTAL	467.38

2.2.2. New technologies

In this section, an overview on the new technologies included in the current version of TIMES-Italia [14] is presented. In TIMES models, technologies characterized in "new technologies" templates are available to be installed after the base year and over the considered time horizon. The choice of which technologies are installed and introduced in the energy mix, is made from the algorithm on the basis of the economic optimization and under a set of technical constraints.

To ensure that the base year technologies are progressively disposed of, usually other constraints are used, in particular the installed capacity is often constrained as maximum value to the 100% of the base year value at the beginning of the time horizon, and to 0 after a certain period of time (often in 2020 or 2030), with the other values of maximum capacity obtained by interpolation. To produce the requested amounts of final service demands, additional capacity of new technologies is progressively installed from the new technology dataset.

The main parameters characterizing the new technologies included in the model are:

a. Efficiency (ratio between total output and total input commodities, possibly it is varying increasing over time).
b. Investment cost.
c. Fixed O&M cost.
d. Variable O&M cost.
e. Availability factor.
f. First year of availability.
g. Lifetime.

For what concerns, for instance, agriculture sector, non-road transports and "other industries" industrial subsector, there are no new technologies available for the installation, since the evolution of the energy mix in the time is simply obtained assuming increasing efficiency for the corresponding base year technologies (as shown in Table 51).

Table 51. Efficiency improvement for minor demand-side subsectors.

Sector/subsector	**Efficiency [-]**					
	2006	2010	2015	2020	2030	2050
Agriculture	1.00	1.00		1.15		1.25
Domestic aviation	1.00		1.05			1.25
International aviation	1.00		1.05			1.25
Domestic navigation	1.00	1.02				1.25
Bunkers	1.00	1.04				1.50
Other industries	1.00	1.05			1.11	1.18

For the other sectors, a synthetic overview of the new technologies are provided in Table 52 (buildings) and Table 54 (transport). Concerning the industry, the new technologies has been totally reviewed with respect to the original version of TIMES-Italia [14], according to the modeling performed in ETM [21]. Provides a very synthetic overview of the number of new technologies available for each industrial product.

Table 52. Overview of new technologies for buildings [14].

Sector	End-use	Number of new technologies
Residential	Refrigeration	11
	Water heating	11
	Clothes washing	5
	Clothes drying	3
	Dish washing	5
	Cooking	5
	Lighting	12
	Space heating	77 (21 for SF-Old buildings, 21 for MF-Old buildings, 19 for SF-New buildings and 12 for MF-New buildings)
	Space cooling	12
	Miscellaneous electric equipment	1
Commercial	Refrigeration	3
	Water heating	11
	Lighting	9
	Space heating	20
	Space cooling	18
	Electric office equipment	3

Table 53. Overview of new technologies for industry [21].

Subsector	Industrial product	Number of new technologies
Chemicals	Highly volatile compounds (HVC: olefins and aromatics)	8
	Ammonia	6
	Methanol	6
	Chlorine	3
Iron and steel	Iron and steel	14
Non-ferrous metals	Aluminum	6
	Copper	1
	Zinc	1
Non-metallic minerals	Cement	7
	Lime	1
	Glass	2
	Bricks	1
Pulp and paper	Pulp	5
	Paper	1

Table 54. Overview of new technologies for transport [21].

Transportation mode	New technologies' categories
Cars (small, medium, large)	Gasoline cars (fueled by a mixture of gasoline and ethanol) Diesel cars LPG cars Natural gas cars Hybrid cars Plug-in hybrid cars Electric cars Hydrogen (hydrogen internal combustion engine, liquid hydrogen, fuel cell, liquid hydrogen hybrid) cars (stored in a different spreadsheet, dedicated to hydrogen technologies).
Three Wheels	Gasoline three-wheelers Diesel three-wheelers
Two Wheels	Gasoline mopeds Gasoline motorcycles
Buses	Gasoline buses Diesel buses Natural gas buses LPG buses Electric buses Fuel cell buses (stored in a different spreadsheet, dedicated to hydrogen technologies)
Heavy Trucks	Standard diesel trucks Advanced diesel trucks Improved diesel trucks Natural gas trucks Fuel cell trucks (stored in a different spreadsheet, dedicated to hydrogen technologies)
Medium Trucks (standard, advanced, improved)	Gasoline commercial trucks Diesel commercial trucks LPG commercial trucks Natural gas commercial trucks
Commercial Trucks (standard, advanced, improved)	Gasoline commercial trucks Diesel commercial trucks LPG commercial trucks Natural gas commercial trucks

2.3. Supply-side sectors: base year and new technologies

The power sector and the upstream sector belong to the supply-side of the energy system, converting primary energy sources into energy vectors that are the inputs for the demand-side sectors.

2.3.1. Power sector

For what concerns the electric sector, the produced commodities are electricity (centralized or distributed) and heat, while the production systems are differentiated in three main categories: plants producing only electricity, combined heat and power (CHP) plants, plants producing only heat. For each technology, several characteristic parameters are collected:

a. Input commodities (fossil fuels, biofuels or renewable energy sources).
b. Input fuel shares (for multi-fuel technologies, to assign a fraction of the input energy to the relative fuels)
c. Output commodities (centralized electricity, distributed electricity or heat).
d. Efficiency and its expected evolution over projection years.
e. Heat-to-power ratio (CHPR), only for CHP plants.
f. Base year installed capacity.
g. Residuals of base year installed power at different projection years.
h. Fixed operation and maintenance cost.
i. Variable operation and maintenance cost.
j. Availability factor (ratio between the maximum equivalent hours of the plant and 8760 h, equivalent to one year; in other words, it is related to the maximum energy producible in one year, assuming that the plant operates constantly at the nominal power, or it does not work), that for renewable energy sources is not unique for the entire year but differentiated by time-slices).
k. Lower boundaries (for different projection years) to the energy production from the base year already installed plants.

Lower activity boundaries $A_{low}[PJ]$ are calculated according to Equation 24, starting from the installed capacity $P\,[GW]$ at the base year and the availability factor AF of each i^{th} technology. The factor R_f is different for different technologies and is used to express the fraction of the base year potential energy production that is imposed as boundary for the projection year at which the boundary is applied (for example R_f is usually equal to 0.5 or 0.7 for boundaries referring to 2007, one year later the base year).

$$A_{low}[PJ] = P\,[GW] * AF * 31.536\,\left[\frac{PJ}{GW}\right] * R_f \qquad (24)$$

In Table 55, the parameters concerning the base year of electricity production technology are reported. In terms of installed power at the base year, the technologies most used in the base year (higher than 5 GW) are, in order as follows: natural gas combined cycle, reservoir hydroelectric plant, natural gas cogenerative combined cycle, pumping hydroelectric plant, oil condensation steam cycle and coal condensation steam cycle. Focusing on renewable resources electric plants, it should be observed that in 2006 Italy relied almost exclusively on hydroelectricity (with about 21.38 GW of installed power, including flowing hydroelectric), with quite negligible installed power of wind, geothermal, solar (only 0.02 GW, before the 2005-2013 state subsidies for photovoltaics [25]) and biogas plants.

Table 55. Base year relative descriptive parameters of electricity production technologies.

Technology description	Input commodity	Output commodity	Efficiency	CHPR	Installed power [GW]	Fixed operation and maintenance cost [M€/GW]	Variable operation and maintenance cost [M€/PJ]	Availability factor
Coal condensation steam cycle	Coal	Centralized Electricity	0.34		5.09	32.32	0.58	0.65
Multi-fuel coal and oil plants	Coal Oil	Centralized Electricity	0.27		2.64	30.42	0.34	0.50
Natural gas and derived gas combined cycle	Derived Gas	Centralized Electricity	0.36		0.79	37.89	0.49	0.38
Natural gas combined cycle	Natural Gas	Centralized Electricity	0.50		16.45	13.50	0.41	0.55
Multi-fuel oil and gas plants	Natural Gas Oil	Centralized Electricity	0.39		4.31	17.86	0.35	0.16
Steam cycle with natural gas repowered gas turbines	Natural Gas	Centralized Electricity	0.42		3.64	15.40	0.33	0.35
Natural gas turbines	Natural Gas	Centralized Electricity	0.42		1.99	26.24	0.50	0.03
Natural gas thermoelectric plant	Natural Gas	Centralized Electricity	0.35		1.63	17.98	0.85	0.11
Diesel turbine	Oil	Centralized Electricity	0.27		0.75	22.19	0.48	0.03
Oil condensation steam cycle	Oil	Centralized Electricity	0.36		7.74	42.61	0.45	0.30
Biogas plant	Biogas	Distributed Electricity	0.35		0.27	12.50	0.36	0.51
Biomass centralized plant	Biomass	Centralized Electricity	0.23		0.23	12.50	0.36	0.70
Biomass distributed plant	Biomass	Distributed Electricity	0.23		0.26	12.50	0.36	0.70
Geothermal plant	Geothermal	Centralized Electricity	0.10		0.79	94.03	3.48	0.80
Solar plant	Solar	Distributed Electricity	1.00		0.02	30.80	13.89	0.24
Wind plant	Wind	Centralized Electricity	1.00		2.12	34.00	0.00	0.16

Continued on page 93

Continued from page 92

Flowing water hydroelectric plant	Hydroelectric	Centralized Electricity	1.00		2.69	33.65	0.08	0.37
Flowing water hydroelectric plant<10MW	Hydroelectric	Distributed Electricity	1.00		2.05	33.65	0.08	0.37
Reservoir hydroelectric plant	Hydroelectric	Centralized Electricity	1.00		9.55	13.29	0.08	0.26
Pumping hydroelectric plant	Centralized Electricity	Centralized Electricity	1.00		7.09	20.76	0.08	0.11
Natural gas and derived gas cogenerative combined cycle	Natural Gas	Centralized Electricity Heat	0.52	0.64	1.40	22.42	0.64	0.69
Heavy HCs gasification cogenerative combined cycle	Heavy HCs	Centralized Electricity Heat	0.50	0.62	1.83	29.14	0.45	0.84
Natural gas cogenerative combined cycle	Natural Gas	Centralized Electricity Heat	0.51	0.39	9.67	20.11	0.79	0.65
Centralized natural gas cogenerative gas turbine	Natural Gas	Centralized Electricity Heat	0.37	0.92	1.48	32.85	0.57	0.61
Distributed natural gas cogenerative gas turbine	Natural Gas	Distributed Electricity Heat	0.37	0.92	1.48	32.85	0.57	0.60
Natural gas condensation cogenerative steam cycle	Natural Gas	Centralized Electricity Heat	0.37	1.00	0.79	34.27	0.48	0.55
Centralized oil condensation cogenerative steam cycle	Oil	Centralized Electricity Heat	0.35	0.59	0.37	23.71	0.49	0.70
Distributed oil condensation cogenerative steam cycle	Oil	Distributed Electricity Heat	0.35	0.59	1.03	43.60	0.47	0.60
Municipal waste plant	Municipal Waste	Centralized Electricity Heat	0.22	1.25	0.58	220.50	0.83	0.44
Biogas cogenerative plant	Biogas	Centralized Electricity Heat	0.30	1.23	0.07	220.50	0.83	0.45
Coal gasification cogenerative combined cycle	Coal	Centralized Electricity Heat	0.37	0.27	0.91	220.50	0.83	0.65
Biomass cogenerative plant	Biomass	Centralized Electricity Heat	0.23	1.13	0.17	220.50	0.83	0.61

As for the demand-side sectors, also for the power sector specific emission factors are evaluated, according to Equation 2. Results are reported in Table 56.

Table 56. Emission factors for electricity sector-specific fuels.

Commodity	**Emission factors [kt/PJ]**						
	Natural gas	Coal	Oil	Solid biomass residual	Solid biomass virgin	Municipal waste	Biogas
CO_2	56.05	101.16	79.55	0.00	0.00	85.85	0.00
CH_4	0.13	1.06	5.15	30.00	30.00	0.02	300.00
N_2O	0.54	1.48	0.62	4.00	4.00	4.00	4.00

The TIMES-Italia database includes several new technologies, as Table 57 shows.

Table 57. New technologies for electricity and heat production.

Plant category	**Energy source**	**Number of new technologies**
Electricity	Natural gas	3
	Coal	1
	Oil	1
	Wind	6 (out of which 2 offshore plants)
	Hydroelectric	2
	Geothermal	2
	Solar PV	12
	Biogas	2
	CSP	4
Heat	Natural gas	1
	Oil	1
	Bioliquid	1
	Coal	1
	Geothermal	2
CHP	Natural gas	4
	Municipal waste	1

New power plants are characterized by the following parameters:

a. Starting year for the availability.
b. Plant lifetime.
c. Economic lifetime.
d. Investment cost.
e. Fixed O&M cost.
f. Variable O&M cost.
g. Discount rate.
h. Efficiency.
i. Capacity to activity factor.
j. Availability factor (differentiated according to the time of day and season for variable renewable energy sources).
k. Peak reserve margin.

CHP technologies are also characterized by the following parameters:

a. Heat-to-power ratio/maximum CHPR.
b. Electricity-to-heat coefficient CEH.

Micro-CHP plants are specifically dedicated to the production of residential, commercial, and industrial heat/electricity. The database includes 12 technologies for micro-CHP plants (4 for residential sector, 4 for commercial sector and 4 for industry). In [20], micro-CHP plants receive a more detailed characterization, which has been further updated in the latest ETM update, thus that will be used for the TIMES-Italia, too.

2.3.2. Upstream sector

The upstream sector is part (together with the power sector) of the supply-side of the TIMES-Italia reference energy system. The main purpose of this sector is the extraction is the production of fuels usable by other sectors to satisfy energy consumptions: it encompasses the steps from raw fuels materials extraction to the upstream transformation into usable fuels.

Technologies and commodities are listed in several spreadsheets, according to their role in the conversion chain of this sector. They contain information about the following steps of the supply chain:

a. Extraction of row fossil fuels (definition of extraction variable O&M costs and imposition of lower and upper boundaries, to constrain the base year extraction consistently with statistics).
b. Fuel primary production (list of different production technologies with the expected lifetime, fixed O&M cost and lower and upper boundaries, based on the base year statistics).
c. Fuel secondary transformation (list of different transformation, such as refineries, technologies with the expected lifetime, fixed O&M cost, and lower and upper boundaries, based on the base year statistics).
d. Natural potentials of renewable energy sources (definition of extraction variable O&M costs and imposition of lower and upper boundaries, to constrain the base year utilization consistently with statistics).

The main parameters describing the different technologies of the upstream sector are listed in Table 58.

Table 58. Descriptive parameters of upstream sector technologies.

Technology category	**Parameters**
Extraction of row fossil fuels (heavy oil, natural gas, hard coal)	Extraction cost
	Lower and upper boundaries to the extraction (based on known fossil reserves)
Primary production of fuels	Input commodities
	Output commodities
	Lifetime
	Fixed operation and maintenance cost
	Variable operation and maintenance cost
	Lower and upper boundaries to the production (based on 2000 statistics)
Secondary transformation of fuels	Input commodities
	Output commodities
	Lifetime
	Fixed operation and maintenance cost
	Variable operation and maintenance cost
	Lower and upper boundaries to the production (based on 2000 statistics)
Natural potentials of renewable energy sources	Extraction cost
	Lower and upper boundaries to the exploitation

Concerning the connections between technologies of different steps of the supply chain, they are clarified in Figure 9.

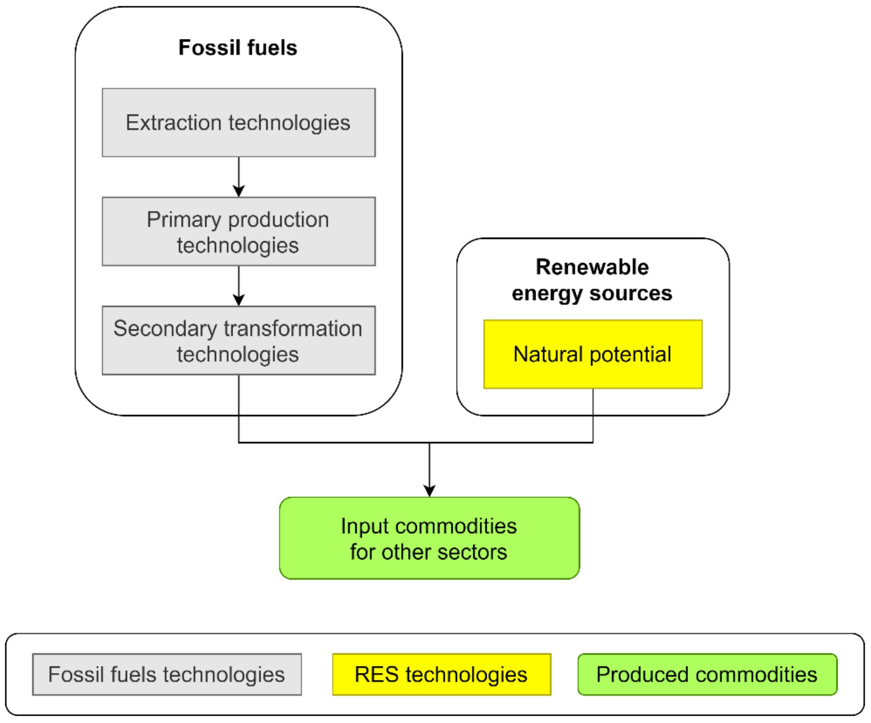

Fossil fuels
Extraction technologies
Primary production technologies
Secondary transformation technologies
Renewable energy sources
Natural potential
Input commodities for other sectors
Fossil fuels technologies
RES technologies
Produced commodities

Figure 9. Connection between different technologies of upstream sector.

Also, for the upstream sector, specific emission factors are evaluated for several fuels according to Equation 2. Results are reported in Table 59.

Table 59. Emission factors for upstream sector-specific fuels.

Commodity	**Emission factors [kt/PJ]**					
	Natural gas	Coal	Crude oil	Refined petroleum products (liquid)	Refined petroleum products (gas)	Biofuels
CO_2	50.50	102.59	73.33	84.99	56.23	85.85
CH_4	0.13	1.00	3.00	3.73	1.00	30.00
N_2O	0.54	1.40	0.60	0.86	0.10	4.00

2.4. Fuel import

Since the TIMES-Italia [14] only includes a single region (as already mentioned in Section 0), commercial exchanges with other regions are modelled with cost parameters associated with imported commodities. Such costs are imposed for coal, solid biomass, biofuels, oil products, natural gas, and electricity, and can vary with time, with different values assigned to several time steps, and namely 2006, 2008, 2010, 2013, 2015, 2020, 2030, 2040 and 2050. To perform different scenario analyzes, three set of importation prices are included, with high, medium, and low prices. Table 60 reports the medium prices for commodity importation.

Table 60. Fuels and electricity medium importation prices.

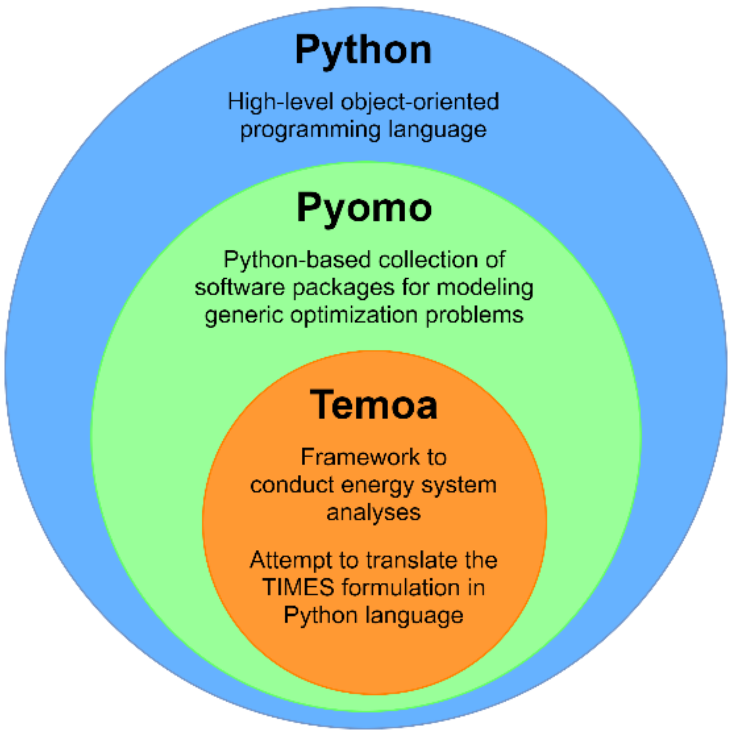
Fuel category	Fuel	Importation price [M€/PJ]								
		2006	2008	2010	2013	2015	2020	2030	2040	2050
Biomass and coal	Solid biomass	3.0	4.4	4.5	4.6	4.7	5.0	5.4	5.9	6.4
	Biodiesel	34.0	38.2	34.6	53.8	37.8	43.8	37.5	41.2	45.4
	Coal	27.2	30.5	27.7	43.0	30.2	35.0	30.0	33.0	36.3
	Coke	34.0	38.2	34.6	53.8	37.8	43.8	37.5	41.2	45.4
Oil products	Asphalt	50.0	56.1	50.9	79.2	55.6	64.4	55.1	60.7	66.7
	Aviation gasoline	17.0	19.1	17.3	26.9	18.9	21.9	18.7	20.6	22.7
	Crude oil feedstock	50.0	56.1	50.9	79.2	55.6	64.4	55.1	60.7	66.7
	Diesel	17.0	19.1	17.3	26.9	18.9	21.9	18.7	20.6	22.7
	Gasoline	17.0	19.1	17.3	26.9	18.9	21.9	18.7	20.6	22.7
	Heavy fuel oil	12.7	14.3	13.0	20.2	14.2	16.4	14.1	15.5	17.0
	Jet kerosene	17.0	19.1	17.3	26.9	18.9	21.9	18.7	20.6	22.7
	Kerosene	17.0	19.1	17.3	26.9	18.9	21.9	18.7	20.6	22.7
	Liquified natural gas	10.2	11.4	10.4	16.1	11.3	13.1	11.2	12.4	13.6
	Liquified petroleum gas	17.0	19.1	17.3	26.9	18.9	21.9	18.7	20.6	22.7
	Lubricant	50.0	56.1	50.9	79.2	55.6	64.4	55.1	60.7	66.7
	Naphtha	10.2	11.4	10.4	16.1	11.3	13.1	11.2	12.4	13.6
	Oil	8.5	9.5	8.7	13.5	9.5	10.9	9.4	10.3	11.3
	Oil additive	50.0	56.1	50.9	79.2	55.6	64.4	55.1	60.7	66.7
	Other non-specified oil products	22.7	25.5	23.2	36.0	25.3	29.3	25.1	27.6	30.3
	Petroleum coke	2.3	3.4	3.1	4.8	3.3	3.9	3.3	3.6	4.0
	Wax	50.0	56.1	50.9	79.2	55.6	64.4	55.1	60.7	66.7
	White spirit	50.0	56.1	50.9	79.2	55.6	64.4	55.1	60.7	66.7
Natural gas	Natural gas	5.0	6.5	6.0	8.2	6.0	6.3	6.9	7.6	8.3
	Liquified natural gas	5.0	6.5	6.0	8.2	6.0	6.3	6.9	7.6	8.3
Electricity	Electricity	5.6	7.8	7.8	7.2	6.8	6.9	7.3	7.6	7.8

Chapter 3

Development of the new Temoa-Italia model

3.1. The Temoa framework

Tools for Energy Model Optimization and Analysis (Temoa [13] [26]) is an open-source modeling framework for conducting energy system analyzes. Temoa is formulated as a linear programming problem and is implemented in Python using Pyomo, a Python-based open-source software package (Figure 10). Temoa is intended to address two critical deficiencies: the impossibility to verify results by third parties based on published models and the difficulty of performing a rigorous analysis of the uncertainty with models.



Python
High-level object-oriented programming language
Pyomo
Python-based collection of software packages for modeling generic optimization problems
Temoa
Framework to conduct energy system analyses
Attempt to translate the TIMES formulation in Python language

Figure 10. Temoa framework developed within Python-based Pyomo collection.

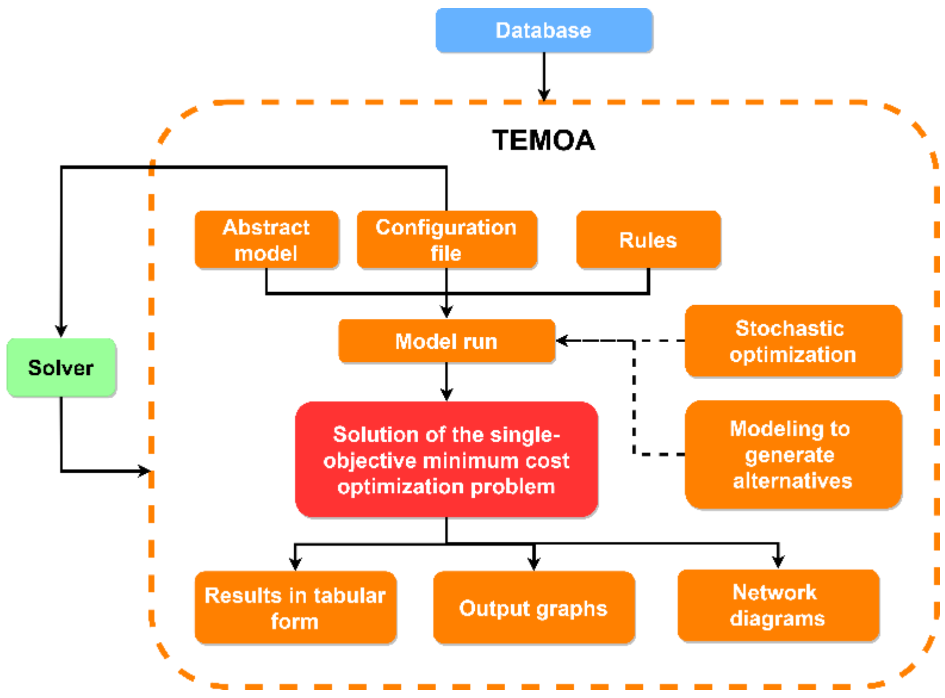


Figure 11. Main components constituents the Temoa structure.

Figure 11 shows schematically the structure of Temoa framework, highlighting the main blocks involved in the optimization. The Pyomo framework is based on five classes: sets, parameters, variables, objective, and constraint, necessary in every optimization problem. It supports a wide range of problem types: linear programming, non-linear programming, mixed-integer linear programming, etc. Temoa is the Python correspondent of the TIMES model generator [8] (thus implemented as a linear programming problem), in which different model instances with different energy system structures can be represented.

As Figure 11 shows, the Temoa model also includes the possibility to perform stochastic optimization [27], taking into account future uncertainties in the model, and modeling to generate alternative [28], exploring the near-optimal decision space to investigate alternative future energy system configurations.

Temoa has been selected as reference framework for the open-source implementation of the TIMES-Italia. A literature review has been performed to identify the most suitable framework. Notably, an alternative tool has also been considered (as already mentioned in Section 1.1), before opting for Temoa: OSeMOSYS [29]. Temoa has been chosen for its features:

a. Competitive energy markets with perfect foresight (as for TIMES).
b. High-level programming language.
c. Both commercial and open-source solvers can be used for the optimization.
d. Suitable for large-scale energy systems.
e. Possibility to modify and integrate the code (also to perform multi-objective optimization).
f. Possibility to perform stochastic optimization.

Concerning the solvers performing the solution of the optimization problem, any software including a Python interface can be used with Temoa. The reference solver for the Temoa framework is GNU Linear Programming Kit (GLPK [30]), being an open-source software package for both linear programming and mixed integer programming. Other options are, for instance, to solve the optimization problem with CPLEX [31], that is a commercial software but freely available to run on an external server (NEOS [32] [33] [34]), or with Gurobi [35], again commercial but freely available with academic licenses.

Being the framework already existing and freely available, the main activity required to develop the open version of TIMES-Italia, is the translation of technological dataset from TIMES formulation (Excel files containing parameters in the TIMES format) to the Temoa one (namely, a ".sql" database), relying on parameters included in the Temoa model to translate the TIMES-Italia reference energy system.

Already included elements for the description of RES in the Temoa database are listed in Table 61.

Table 61. Elements included in the Temoa database.

Group	Element
Labels used for internal database processing	commodity_labels technology_labels time_periods_labels
Sets used within Temoa	commodities technologies time_periods time_season time_of_day
Parameters used to define processes within Temoa	GlobalDiscountRate Demand DemandSpecificDistribution Efficiency ExistingCapacity CapacityFactor CapacityFactorProcess Capacity2Activity CostFixed CostInvest CostVariable EmissionActivity LifetimeLoanTech LifetimeProcess LifetimeTech
Parameters used to define constraints within Temoa	GrowthRateSeed GrowthRateMax MinCapacity MaxCapacity MinActivity MaxActivity RampUp RampDown TechOutputSplit TechInputSplit

Temoa formulation presents some differences and limitations with respect to the TIMES structure of parameters. These differences concern definition of parameters and their elaboration, particularly:

a. Emission factors computation.
b. Service demands projection.
c. Data interpolation and extrapolation.
d. Results postprocessing.

3.2. Add-ons to the Temoa framework

To perform the required add-ons to the Temoa framework, two dedicated Python scripts have been developed to process the ".sqlite" database: one for the preprocessing (Appendix A) and the other for the postprocessing (Appendix D). Some tables have been added to the database to insert the required input data for the preprocessing. In the following sections, the functions implemented with the scripts and the required additional tables of the database are presented.

3.2.1. Emission factors computation

Emission factors are implemented in Temoa through a dedicated parameter called "Emission Activity", that is an emission factor expressed per unit of output commodity produced by the technology (its activity). As it has been already explained in Section 8, in TIMES the emission factors are evaluated per unit of sector-specific commodity consumed by the computed energy mix. For that reason, it necessary to derive from commodity-based emission factors implemented in TIMES, the technology-based emission factors to implement in Temoa.

To do that, a new table has been created within the Temoa ".sql" database ("CommodityEmissionFactor" table, Appendix B), containing the commodity-based emission factors included in TIMES templates. In the table the emission commodity ($\mathrm{CO_2}$, $\mathrm{CH_4}$, $\mathrm{N_2O}$ or $\mathrm{SO_x}$) must be specified, with the sector-specific commodity to which the emission is associated and the commodity-based emission factor numerical value $\mathrm{EF_{comm}}$. To obtain the technology-based emission factor $\mathrm{EF_{tech}}$, the commodity-based factor must be divided by the efficiency of the technology $\mathrm{eff_{tech}}$ having as input commodity the correspondent commodity, as shown in Equation 25.

$$\mathrm{EF_{tech}}\left[\frac{\mathrm{kt}}{\mathrm{PJ_{out}}}\right] = \frac{1}{\mathrm{eff_{tech}}} * \mathrm{EF_{in}}\left[\frac{\mathrm{kt}}{\mathrm{PJ_{in}}}\right] \tag{25}$$

The resulting technology-based emission factor represents the specific emission of the selected emission commodity per unit of output commodity produced by the technology. The results must be inserted in the "EmissionActivity" table of the Temoa database.

3.2.2. Service demand projections

Concerning the service demand projection along the considered time horizon (in the TIMES-Italia case up to 2050), this is operated in TIMES taking as inputs the base year demand level $\mathrm{D_{2006}}$, a selected driver d for the service demand and appropriate elasticity values e (Equation 26, where $\mathrm{D_t}$= "service demand at year t", $\mathrm{D_{t-1}}$= "service demand at year t-1", $\mathrm{d_t}$= "allocated driver at year t", $\mathrm{d_{t-1}}$= "allocated driver at year t - 1", $\mathrm{e_t}$= "associated elasticity at year t"). The elasticities are required to connect the driver trend to the allocated demand trend. Indeed, it is not required the demand trend to be, strictly speaking, proportional to the correspondent driver trend, but only dependent on it. This dependency is exactly driven by the elasticity values.

However, Temoa directly takes as input the service demand specified for each time-step of the considered time-period. Because of this, an automated operation is required to evaluate the service demands absolute values providing drivers and elasticities.

$$\mathrm{D_t} = \mathrm{D_{t-1}} \cdot \left(1 + \left(\frac{\mathrm{d_t}}{\mathrm{d_{t-1}}} - 1\right) \cdot \mathrm{e_t}\right) \tag{26}$$

With this objective, the addition of three further tables to the database is required to provide the required input data. Namely drivers and elasticities must be provided for each final service demand, and a table associating the service demands to the correspondent drivers and elasticities is required. The structure of those tables is reported in Appendix C. Table "Driver" should express the time evolution of the selected drivers for the demand projection, while table "Allocation" associates each demand commodity to the correspondent driver and table "Elasticity" provide elasticity values for each demand-driver couple.

In the preprocessing script (Appendix A), data from those three tables are elaborated to compute the absolute values for each final service demand and to fill with the resulting values the table "Demand" of the database.

3.2.3. Data interpolation and extrapolation

Another relevant difference between the two frameworks is that TIMES automatically interpolates and extrapolates parameters for milestone years for which they are not explicitly specified by the modelers (according to different interpolation and extrapolation rules). Differently, Temoa requires the specification of all parameters for all milestone years included in the database.

The preprocessing script has been developed to perform the interpolation and the extrapolation forward for all the parameters for which Temoa allows to specify different values for different time-periods. Notably, the involved parameters in this operation are:

a. Lifetime
b. Efficiency
c. TechInputSplit (used to imposed minimum consumption percentage of an input commodity for technologies with more than one input commodity)
d. TechOutputSplit (used to imposed minimum production percentage of an output commodity for technologies with more than one output commodity)
e. Emission factors
f. Investment cost
g. Fixed O&M cost
h. Variable O&M cost
i. Minimum capacity constraint

j. Minimum activity constraint
k. Maximum capacity constraint
l. Maximum activity constraint
m. Availability factor
n. Capacity factor
o. Capacity credit

For what concerns interpolation, the script performs a linear interpolation for each parameters presenting more than one value specified along the entire time horizon (Equation 27, where x_n is the generic parameter to be interpolated at i^{th} time-period t_i). Therefore, for the time interval included between the time periods correspondent to the first specified value and the last one, the interpolation curve is piecewise linear.

$$\mathrm{x_i = x_1 + \frac{t_i - t_1}{t_2 - t_1} * (x_2 - x_1)} \tag{27}$$

For parameters that are specified only for one time-period, the same value is repeated for all the future time periods starting from it. Similarly, an extrapolation repeating the last available value for all the future time periods is performed for those parameters presenting a piecewise linear interpolation that does not end at the last year of the time horizon (in the TIMES-Italia case, 2050).

Figure 12 report three examples concerning the possible trends obtained by the parameters' interpolation and extrapolation. In particular, in Figure 12 (a) a piecewise linear efficiency trend is shown, while in Figure 12 (b) a constant efficiency is represented, extrapolated starting from the first availability year of the technology (in the represented case, from 2020).

An exception to the piecewise linear interpolation of parameters for which more than one value is provided in the database is shown in Figure 12 (c). This is the case of a technology lifetime variable during the time. In this case, to maintain integer values of the parameter, it is kept equal to the last available for time periods included between two different values.

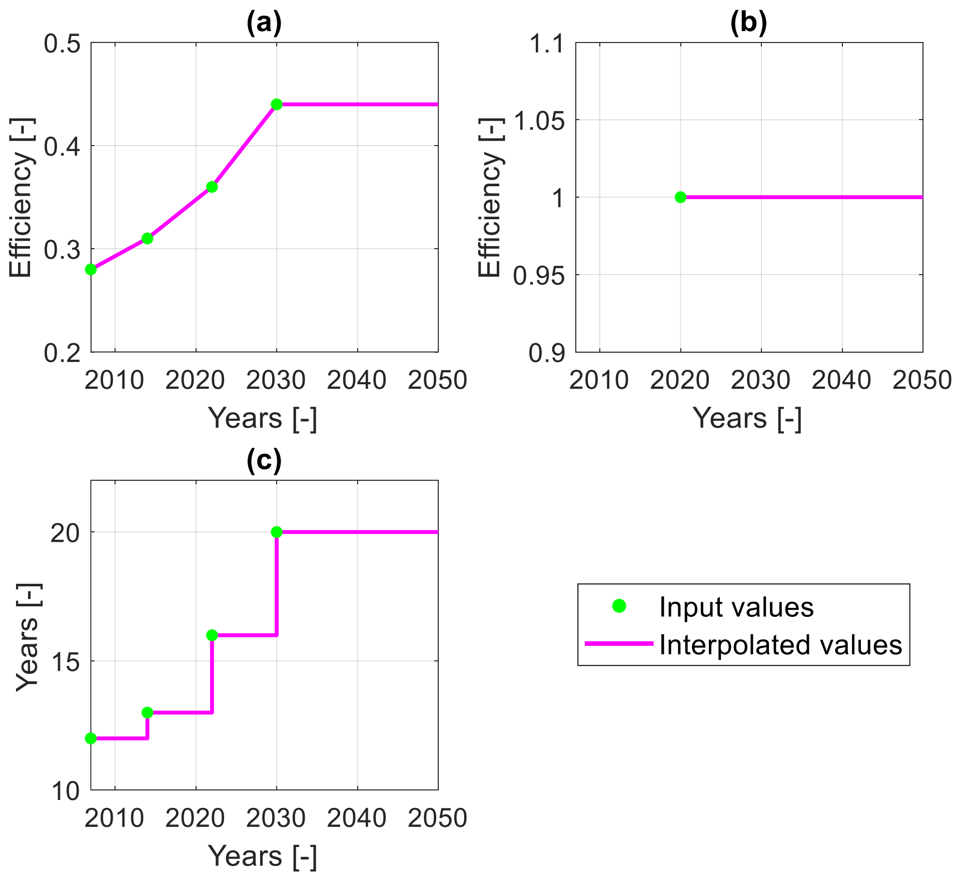


Figure 12. Example of data interpolation and extrapolation for (a) piecewise lineare trend, (b) constant trend and (c) piecewise constant trend (technology lifetime variable in the time).

3.2.4. Results postprocessing

The last required function to be implemented in Temoa framework concerns the reading of the results. In TIMES, results are collected in MDB databases and a commercial software (namely, VEDA-BE [36]) is employed to select data and read the results having the possibility to select among several technology and commodity set those to visualize.

Temoa output data are collected in dedicated tables within the same ".sqlite" database used to provide input data. The two most relevant tables to analyze the evolution of the energy mix are named "Output_VFlow_In" and "Output_VFlow_Out" containing, respectively, the consumption and production of each commodity by technology and time slice. The differentiation by time slices of results is made specifying the time period (the milestone year), the season of the year (spring, summer, fall or winter) and the time of the day (day, night or peak). It is useful to perform analyzes with an high degree of detail and especially for technologies characterized not only with annual parameters, but also with variable parameters during the year (for example, some kind of renewable energy sources that are strongly dependent on the season of the year or the hour of the day); however, an annual resolution to view the results is sufficient to assess the evolution of energy mix along the time.

Made those premises, the purpose of the postprocessing developed Python script is to extract data from the ".sqlite" output database and visualize year by year the annual total consumption and production for selected technologies and commodities.

The script allows to:

a. List the technologies for which to view the results.
b. List the commodities to include in the exported data (avoiding including intermediate commodities in the results that would lead to double counting).
c. Chose to view results split both by technology and commodity, only by technology, only by commodity or to view only the total consumption/production for the selected technologies and commodities.

The output of the script is an Excel file containing a sheet with consumption values and another sheet reporting production values for each optimization year.

Chapter 4

Comparison between TIMES-Italia and Temoa-Italia

In this section, the results obtained from TIMES-Italia and Temoa-Italia optimization are compared, to ensure that the two tools lead to the same results starting from the same input data. This kind of validation should prove the reliability of Temoa framework to use it as the reference open-source tool for future integrations and modifications of the optimization paradigms, starting from a model reproducing the same results obtained with well-established software such as the TIMES models.

4.1. Scenario

The considered scenario for the comparison (which is, of course, the same considered both in TIMES-Italia and Temoa-Italia) has the following features:

a. Medium level cost for import of primary resources.
b. No CO_2 emission limits.
c. No carbon tax applied.
d. No carbon capture and storage (CCS) technologies.

Gurobi [35] has been used as optimization solver, being a commercial software with a Python interface freely available with academic license (al already mentioned in Section 3.1). Gurobi is suitable to solve locally the Temoa-Italia, that having a large database cannot be solved by GLPK with a sufficiently low computational cost (short execution times) and overcomes the maximum size limit for output files (16 MB) solving it with CPLEX on NEOS server.

4.2. Drivers

Concerning the drivers for demands projection, the correspondent drivers and elasticities to the moderate growth scenario have been selected.

Table 62 reports the associated driver to each final service demand. Service demands projection is performed according to Equation 26.

A dedicated comment should be done concerning the residential space heating demands. Indeed, as it has been already reported in Section 1.1.1.1, the residential space heating service is split in four final service demands in the TIMES-Italia model, corresponding to two different building types (single-family and multi-family) each of them is further split in "old" and "new" buildings. "Old" buildings represent the existing buildings, while new constructions are labeled as "New". According to that, the final service demands of "SF-Old" and "MF-Old" residential space heating are assumed to be constant for the entire time horizon, while "SF-New" and "MF-New" are projected with dedicated exogenous drivers, so that the sum between "SF-Old" and "SF-New" buildings reproduce the expected single-family buildings demand for the future and that the sum between "MF-Old" and "MF-New" the expected multi-family one. The residential space heating subsector as a whole evolves with a 2006-normalized trend that is quite similar to the population driver, as could reasonably be expected.

Table 62. Allocation and numeric values of moderate drivers for service demand projection [14].

Sector	Service demands	Driver	Years [-]													
			2006	2007	2008	2010	2012	2014	2016	2018	2020	2022	2025	2030	2040	2050
Agriculture	Agriculture	Agriculture value added	1.00	1.00	0.88	0.88	0.86	0.86	0.87	0.89	0.90	0.91	0.94	0.97	1.05	1.14
Commercial	Space heating Space cooling Water heating Lighting Cooking Refrigeration Electric office equipment	Commercial value added	1.00	1.02	1.01	0.99	0.99	0.98	1.01	1.04	1.07	1.10	1.14	1.22	1.37	1.54
Residential	Space heating Space cooling Water heating Refrigeration Clothes drying Cooking Clothes washing Dishwashing Miscellaneous electric energy Lighting	Population	1.00	1.01	1.01	1.02	1.02	1.05	1.04	1.04	1.07	1.08	1.09	1.11	1.13	1.16

Continued on page 113

Continued from page 112

Transport	International aviation Domestic aviation Road Rail Domestic navigation Non-specified transports Bunkers	GDP	1.00	1.01	1.01	0.97	0.95	0.93	0.95	0.97	1.00	1.02	1.06	1.12	1.25	1.39
Industry	Chemicals	Chemical production	1.00	1.00	0.94	0.93	0.90	0.92	0.97	1.02	1.03	1.06	1.10	1.17	1.32	1.48
	Iron and steel	Iron and steel production	1.00	1.04	0.84	0.80	0.75	0.73	0.75	0.78	0.76	0.77	0.78	0.79	0.80	0.82
	Non-ferrous metals	Non-ferrous metals production	1.00	0.99	0.81	0.81	0.70	0.72	0.76	0.78	0.79	0.80	0.81	0.83	0.87	0.91
	Non-metallic minerals	Non-metallic minerals production	1.00	1.02	0.86	0.78	0.69	0.69	0.73	0.73	0.74	0.78	0.82	0.89	1.02	1.14
	Pulp and paper	Pulp and paper production	1.00	1.03	0.92	0.92	0.88	0.91	0.94	0.92	0.95	0.97	1.00	1.05	1.17	1.29
	Other industries	Other industries production	1.00	1.03	0.90	0.86	0.80	0.79	0.82	0.85	0.85	0.87	0.89	0.94	1.00	1.06
	Non-specified industry Chemical feedstocks Non-energy uses Non-energy others	GDP	1.00	1.01	1.01	0.97	0.95	0.93	0.95	0.97	1.00	1.02	1.06	1.12	1.25	1.39

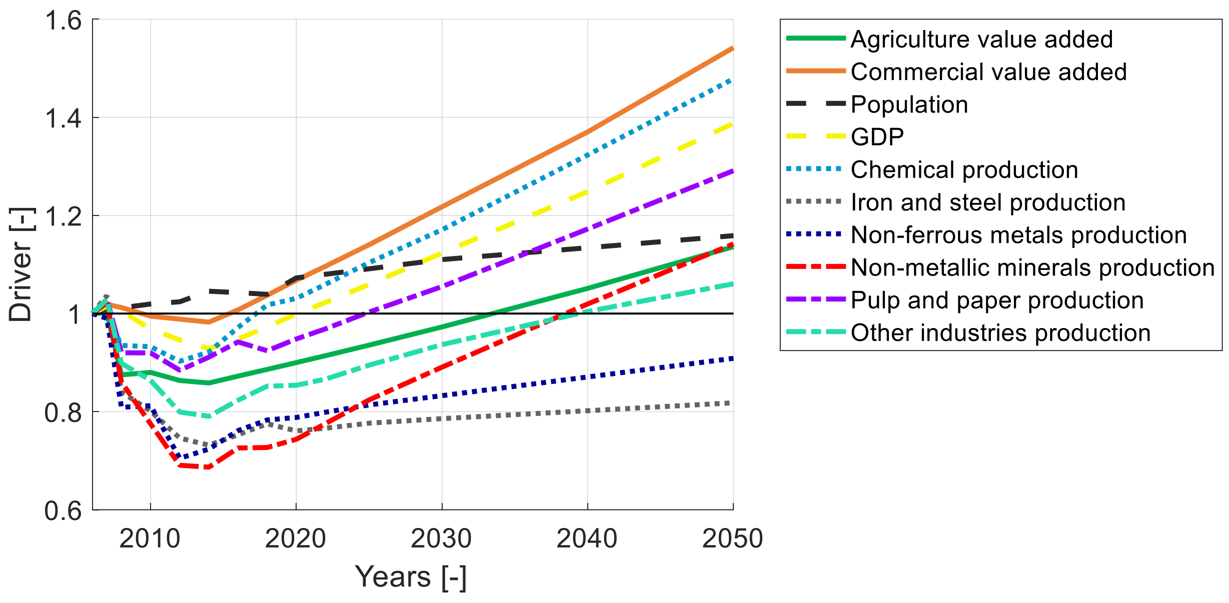


Figure 13. Driver 2006-normalized trend.

Figure 13 shows the trend for all the considered drivers in the model, normalized with respect to the value for the year 2006. Concerning the period 2006-2018, the adopted drivers are intended to follow historical data. In particular, the drastic drop of industrial production (more or less pronounced for all the industrial sectors) and GDP consequent to the 2008 financial crisis, and the period of economic stagnation for the immediately following years, should be noted, highlighting the updating and the reliability of such socio-economic projections. For the future period of the time horizon, drivers are evaluated according to the PNIEC 2020 forecasts [5]. Being the PNIEC forecasts developed on the basis of historical data up to 2019, a relevant comment is that they do not still take into account the effects of the COVID-19 pandemic of 2020.

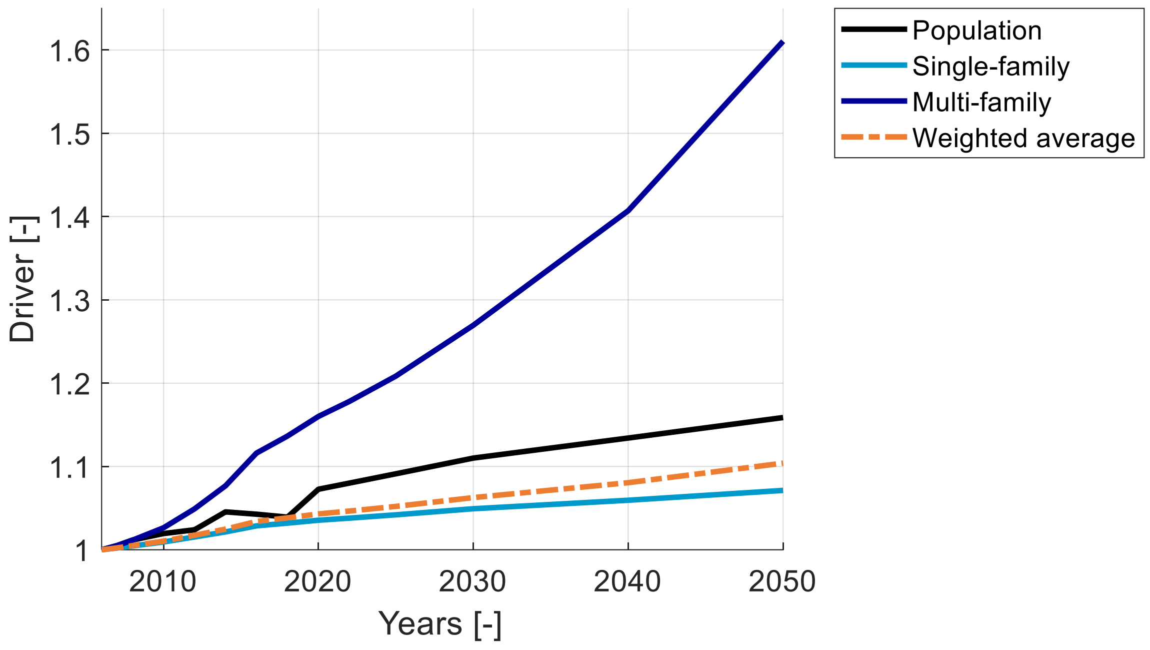


Figure 14. Driver 2006-normalized trend for population, "single-family" and "multi-family" buildings residential space heating and weighted average for total residential space heating demand.

Figure 14 shows in detail the trend of final serviced demand for "single-family" and "multi-family" residential space heating (both "Old" and "New" buildings). The blue curve is the derived 2006-normalized trend for the total residential space heating final demand, obtained from the two curves ("single-family" and "multi-family") weighting on the correspondent service demands, compared to the expected trend for population. An interesting comment is that an higher growth is expected for "multi-family" buildings with respect to "single-family", coherently with the urbanization trends of the population, which therefore is moving more and more into city buildings, usually consisting of several family units per building.

4.3. Constraints

Energy system models usually consider for the optimization a large dataset of constraints, in terms of minimum or maximum exploitation of natural resources, development of specific technologies, or input/output minimum or maximum shares of specific commodities for specific technologies or groups of technologies. In the context of this thesis activity, only a fraction of the constraints included into the TIMES-Italia dataset have been implemented in the Temoa-Italia.

The first constraint category that has been implemented in Temoa-Italia, is related to the base year technologies constraints. Those constraints are usually both minimum and maximum on the capacity or on the activity of those technologies. Minimum constraints are used to ensure that, even if hypothetically the new technologies dataset contained much cheaper or much more efficient new technologies with respect to the base year processes, those last are not immediately substituted by new technologies, but they are progressively disposed of during the time. The maximum constraints, on the other hand, are used to impose the progressive decrease in the use of the technologies of the base year, up to zero in a certain year, to be progressively replaced by the new processes.

Going into the details only for constraints related to industrial technologies, for the sake of conciseness, maximum constraints are usually imposed on the technologies' activity, with an upper boundary equal to 100% of the base year activity in 2007 (the first year categorized as "future" year, for which the optimization is performed in Temoa) and a lower boundary equal to 90% of the base year activity in 2007.

Some industrial subsectors present an exception concerning the lower boundary. Indeed, it is possible that the selected driver for the projection of certain service demands leads to compute a service demand value less than the lower boundary imposed to the base year technology producing the commodity associated to that service demand. This is likely for TIMES-Italia, having as base year 2006, just before the 2008 financial crisis, associated to relevant drop of industrial production (as it has been already stated in Section

1.1). If that problem occurs (and it occurs, specifically, for “Iron and steel” and “Non-ferrous metals” subsectors) it would entail wrong results in the TIMES framework (simply forcing the model to produce more demand than the required) while execution errors would stop the optimization without achieving results in the Temoa framework. For that reason, for those subsectors presenting this issue, lower activity boundaries must be lower that the “standard” values (90% of the base year activity imposed in 2007).

Figure 15 shows, for the “Non-metallic minerals” industrial subsector the constraints imposed to base year technologies activity.

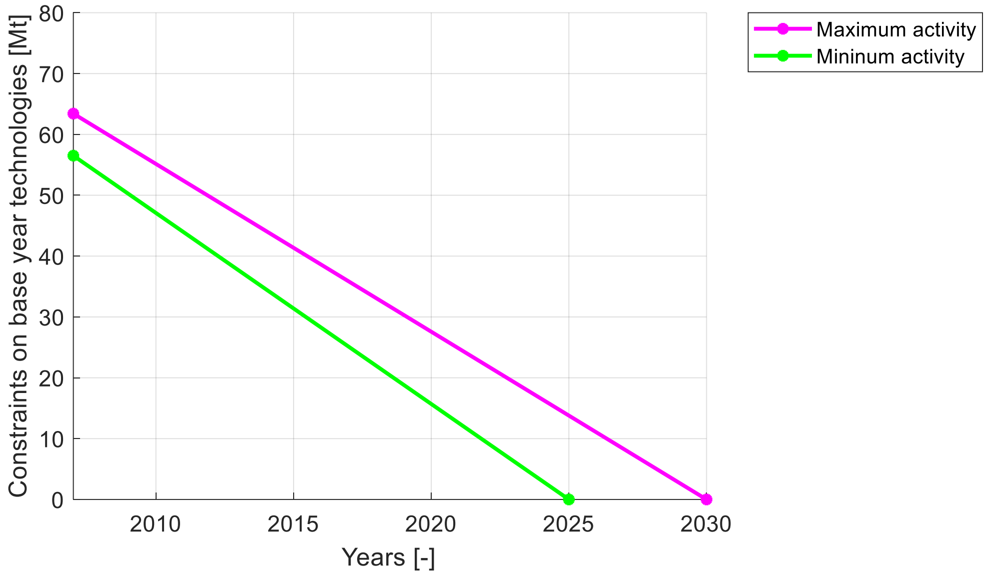


Figure 15. Comparison of minimum and maximum constraints imposed to "Non-metallic minerals" base year technologies in TIMES-Italia and Temoa-Italia.

The last constraints category, implemented both in TIMES and Temoa framework, concerns the fuel import. Indeed, upper boundaries are applied to the amounts of fuels imported from abroad, set equal to the 2006 value in 2007 and obtained multiplying the base year importations by assumed multiplying factors for 2050. Table 63 lists the numerical upper boundary values for each imported fuel.

Table 63. Imported fuels from abroad in 2006 and upper boundaries imposed in 2007 and 2050.

Fuel category	**Fuel**	**2006 importation [PJ]**	**Upper boundaries [PJ]**	
			2007	2050
Biomass and coal	Solid biomass	39.68	39.68	138.89
	Biodiesel	8.10	8.10	17.21
	Coal	681.30	681.30	1635.10
	Coke	20.73	20.73	41.47
Oil products	Crude oil feedstock	265.18	265.18	530.37
	Diesel	68.24	68.24	136.49
	Gasoline	8.49	8.49	16.98
	Heavy fuel oil	161.16	161.16	209.51
	Jet kerosene	5.10	5.10	10.21
	Kerosene	21.16	21.16	42.31
	Liquified petroleum gas	75.62	75.62	151.25
	Naphtha	79.42	79.42	103.25
	Oil	3642.77	3642.77	4735.60
	Oil additive	6.20	6.20	12.41
	Other non-specified oil products	33.16	33.16	66.32
	Petroleum coke	107.78	107.78	215.55
Natural gas	Natural gas	2533.01	2533.01	5066.03
	Liquified natural gas	121.00	121.00	605.00

4.4. Benchmark results

In this section, the comparison between results obtained with TIMES-Italia and those obtained with Temoa-Italia is performed. The aim is to verify that, with the same input data, the two alternative tools (TIMES and Temoa) provide the same results, within a certain tolerance.

Generally, a relative error will be calculated between the Temoa results and the TIMES ones, taking as a reference the TIMES values. The error err will be evaluated according to Equation 28 (where x is a generic result for which the error should be evaluated), and the tolerance interval for the errors is chosen equal to $\pm 1\%$. Errors included in the tolerance interval, will be ignored, and considered as approximation errors in the definition of parameters within the database. Any errors that exceed the tolerance interval below or above will be highlighted and appropriately commented.

$$\mathrm{err}[\%] = \frac{\mathrm{x}_{\mathrm{Temoa}} - \mathrm{x}_{\mathrm{TIMES}}}{\mathrm{x}_{\mathrm{TIMES}}} \tag{28}$$

For the sake of brevity, only results from "Non-metallic minerals" industrial subsectors will be analyzed in detailed.

4.4.1. Non-metallic minerals

The results review for “Non-metallic minerals” subsector is reported in this section. First of all, only to have a first qualitative comparison of the results for the subsector, Figure 16 shows the time trend of the total demand of non-metallic minerals and the associated total energy input required to produce it in the time. At first glance, the two demand curves seem to be superimposed, while little differences are visible between the two energy curves (notably in 2007, 2030 and 2050).

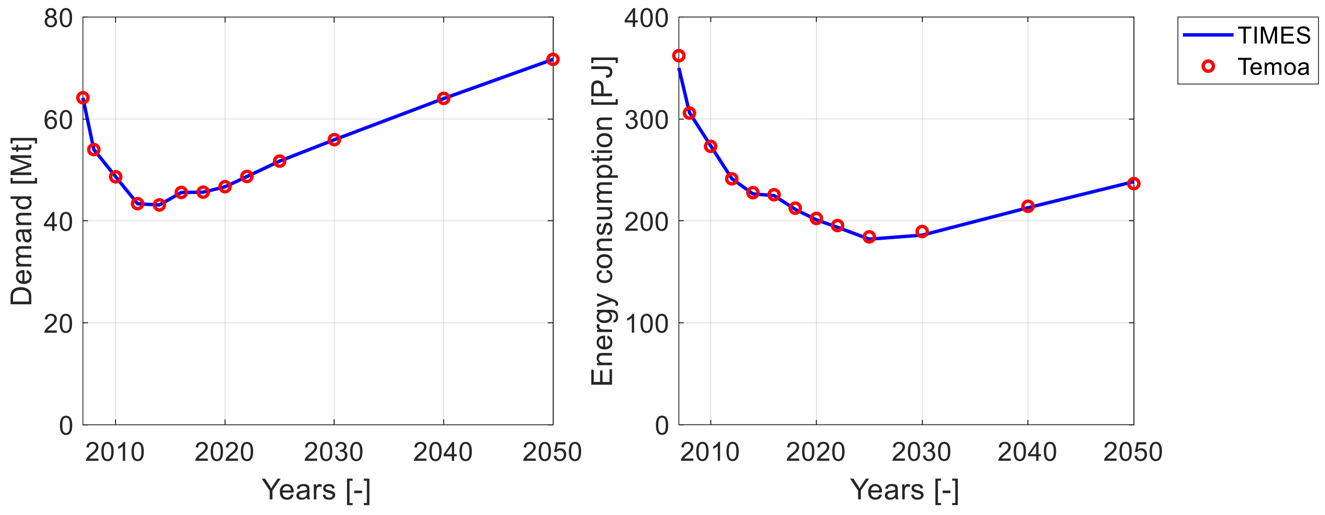


Figure 16. Comparison of "non-metallic minerals" total demand and total energy consumption, evaluated with TIMES-Italia and Temoa-Italia.

Obviously, a deeper analysis is required to ensure that the two tools are achieving the same results (within a certain tolerance), going more in the details with respect to observe only qualitatively those curves. With this objective, more detailed data will be presented below for which relative error values will be calculated between the two models. It will be analyzed, in particular:

a. The breakout of non-metallic minerals demand into the different non-metallic products of which it is composed.
b. The technology mix manufacturing that products.
c. The energy mix of the entire subsector in terms of consumed fuels.

Table 64. Comparison of "Non-metallic minerals" production evaluated with TIMES-Italia and Temoa-Italia and relative errors.

Non-metallic product	**Parameter**	**Year [-]**												
		2007	2008	2010	2012	2014	2016	2018	2020	2022	2025	2030	2040	2050
Total non-metallic minerals	TIMES-Italia production [Mt]	64.15	54.01	48.67	43.37	43.13	45.58	45.64	46.70	48.70	51.74	55.93	64.02	71.70
	Temoa-Italia production [Mt]	64.17	54.00	48.66	43.39	43.14	45.59	45.65	46.72	48.73	51.74	55.95	64.05	71.71
	Relative error [%]	0.04	-0.02	-0.01	0.05	0.03	0.02	0.02	0.05	0.06	0.00	0.04	0.05	0.01
Bricks	TIMES-Italia production [Mt]	6.21	5.23	4.71	4.20	4.18	4.41	4.42	4.52	4.72	5.01	5.42	6.20	6.94
	Temoa-Italia production [Mt]	6.24	5.23	4.71	4.20	4.18	4.44	4.44	4.55	4.74	5.03	5.44	6.23	6.98
	Relative error [%]	0.52	-0.05	-0.05	0.02	0.00	0.50	0.50	0.53	0.54	0.48	0.52	0.53	0.49
Cement	TIMES-Italia production [Mt]	48.91	41.19	37.11	33.07	32.89	34.76	34.80	35.61	37.14	39.46	42.65	48.82	54.68
	Temoa-Italia production [Mt]	48.90	41.18	37.10	33.08	32.89	34.74	34.79	35.60	37.13	39.43	42.63	48.81	54.64
	Relative error [%]	-0.03	-0.03	-0.02	0.05	0.03	-0.05	-0.05	-0.02	-0.01	-0.07	-0.04	-0.02	-0.06
Glass	TIMES-Italia production [Mt]	3.72	3.13	2.82	2.51	2.50	2.64	2.65	2.71	2.82	3.00	3.24	3.71	4.16
	Temoa-Italia production [Mt]	3.72	3.13	2.82	2.52	2.50	2.64	2.65	2.71	2.83	3.00	3.25	3.71	4.16
	Relative error [%]	0.09	0.03	0.04	0.10	0.08	0.07	0.07	0.10	0.11	0.05	0.09	0.10	0.06
Lime	TIMES-Italia production [Mt]	5.30	4.46	4.02	3.58	3.56	3.77	3.77	3.86	4.03	4.28	4.62	5.29	5.93
	Temoa-Italia production [Mt]	5.31	4.47	4.02	3.59	3.57	3.77	3.78	3.86	4.03	4.28	4.63	5.30	5.93
	Relative error [%]	0.09	0.03	0.04	0.11	0.08	0.07	0.07	0.10	0.11	0.05	0.09	0.10	0.06

Table 65.Comparison of TIMES-Italia and Temoa-Italia results of "Non-metallic minerals" production split by production technologies, with relative errors for each technology and average errors for each non-metallic product.

Non-metallic minerals technology	**Parameter**	**Year [-]**												
		2007	2008	2010	2012	2014	2016	2018	2020	2022	2025	2030	2040	2050
Bricks (existing)	TIMES	5.47	5.17	4.56	3.95	3.34	2.74	2.13	1.52	0.91				
	Temoa	5.48	5.17	4.56	3.95	3.34	2.73	2.13	1.52	0.91				
	Relative error [%]	0.15	0.04	0.00	-0.05	-0.12	-0.22	0.09	0.00	0.00				
	Production share[%]	88.10	98.82	96.77	94.11	80.08	61.99	48.15	33.62	19.34				
Bricks (new)	TIMES	0.74	0.06	0.15	0.25	0.83	1.68	2.29	3.00	3.80	5.01	5.42	6.20	6.94
	Temoa	0.76	0.06	0.15	0.25	0.84	1.71	2.31	3.03	3.83	5.03	5.44	6.23	6.98
	Relative error [%]	2.79	-7.56	-1.41	1.19	0.48	1.68	0.88	0.80	0.67	0.48	0.52	0.53	0.49
	Production share [%]	11.90	1.18	3.23	5.89	19.92	38.01	51.85	66.38	80.66	100.00	100.00	100.00	100.00
Bricks technologies average error [%]		0.46	0.13	0.05	0.12	0.19	0.77	0.50	0.53	0.54	0.48	0.52	0.53	0.49
Dry cement kilns (existing)	TIMES	34.23	29.10	25.84	23.16	23.82	20.84	17.86	14.88	11.91	7.44			
	Temoa	34.20	29.10	26.00	23.21	23.80	20.80	17.80	14.90	11.90	7.43			
	Relative error [%]	-0.10	0.00	0.61	0.18	-0.06	-0.18	-0.34	0.10	-0.06	-0.16			
	Production share [%]	69.99	70.65	69.63	70.05	72.42	59.95	51.32	41.80	32.06	18.86			
Wet cement kilns (existing)	TIMES	13.65	11.60	10.23	8.87	8.04	8.31	7.12	5.93	4.75	2.97			
	Temoa	13.70	11.60	10.20	8.88	8.10	8.34	7.15	5.96	4.77	2.98			
	Relative error [%]	0.40	0.01	-0.34	0.12	0.75	0.41	0.43	0.46	0.50	0.46			
	Production share [%]	27.90	28.16	27.58	26.82	24.44	23.90	20.46	16.66	12.78	7.52			
Dry cement kilns (new)	TIMES	1.03	0.49	1.03	1.03	1.03	5.61	9.82	14.79	20.48	29.05	42.65	48.82	54.68
	Temoa	1.00	0.48	0.90	1.00	1.00	5.60	9.84	14.74	20.46	29.02	42.63	48.81	54.64
	Relative error [%]	-3.56	-2.53	-12.99	-3.56	-3.56	-0.26	0.13	-0.34	-0.11	-0.11	-0.04	-0.02	-0.06
	Production share [%]	2.11	1.18	2.79	3.13	3.15	16.15	28.22	41.54	55.16	73.62	100.00	100.00	100.00
Cement technologies average error [%]		0.26	0.03	0.88	0.27	0.34	0.25	0.30	0.26	0.14	0.14	0.04	0.02	0.06

Continued on page 123

Continued from page 122

Glass (existing)	TIMES	3.64	3.13	2.82	2.51	2.50	2.22	1.90	1.58	1.27	0.79			
	Temoa	3.64	3.13	2.82	2.52	2.50	2.22	1.90	1.58	1.27	0.79			
	Relative error [%]	0.00	0.03	0.04	0.10	0.08	0.20	0.05	-0.16	0.31	-0.04			
	Production share [%]	97.89	100.00	100.00	100.00	100.00	83.85	71.78	58.46	44.84	26.38			
Glass (new)	TIMES	0.08	0.00	0.00	0.00	0.00	0.43	0.75	1.12	1.56	2.21	3.24	3.71	4.16
	Temoa	0.08	0.00	0.00	0.00	0.00	0.42	0.75	1.13	1.56	2.21	3.25	3.71	4.16
	Relative error [%]	4.09	NaN[4]	NaN[4]	NaN[4]	NaN[4]	-0.61	0.12	0.47	-0.06	0.08	0.09	0.10	0.06
	Production share [%]	2.11	0.00	0.00	0.00	0.00	16.15	28.22	41.54	55.16	73.62	100.00	100.00	100.00
Glass technologies average error [%]		0.09	0.03	0.04	0.10	0.08	0.26	0.07	0.29	0.17	0.07	0.09	0.10	0.06
Lime (existing)	TIMES	5.19	4.41	3.91	3.47	3.45	3.16	2.71	2.26	1.81	1.13			
	Temoa	5.19	4.41	3.91	3.47	3.45	3.16	2.71	2.26	1.81	1.13			
	Relative error [%]	0.00	-0.03	-0.08	-0.03	-0.05	0.03	0.08	0.15	0.26	0.15			
	Production share [%]	97.89	98.82	97.21	96.87	96.85	83.85	71.78	58.46	44.84	26.38			
Lime (new)	TIMES	0.11	0.05	0.11	0.11	0.11	0.61	1.06	1.60	2.22	3.15	4.62	5.29	5.93
	Temoa	0.12	0.06	0.12	0.12	0.12	0.61	1.07	1.60	2.22	3.15	4.63	5.30	5.93
	Relative error [%]	4.22	5.64	4.22	4.22	4.22	0.29	0.05	0.03	-0.02	0.01	0.09	0.10	0.06
	Production share [%]	2.11	1.18	2.79	3.13	3.15	16.15	28.22	41.54	55.16	73.62	100.00	100.00	100.00
Lime technologies average error [%]		0.09	0.10	0.20	0.16	0.18	0.07	0.07	0.10	0.13	0.05	0.09	0.10	0.06

[4] It is not possible to derive the relative error, since the production is equal to 0.

Table 66. Breakout by fuel of energy consumption for "Non-metallic minerals" subsector and relative error evaluation between TIMES-Italia and Temoa-Italia results.

Fuel	Parameter	Year [-]												
		2007	2008	2010	2012	2014	2016	2018	2020	2022	2025	2030	2040	2050
Total	TIMES-Italia consumption [PJ]	350.64	306.22	273.75	241.19	227.34	225.29	211.78	201.43	194.02	182.30	186.00	212.90	238.46
	Temoa-Italia consumption [PJ]	362.06	305.77	273.16	241.29	227.66	225.64	212.47	202.39	195.40	184.32	189.56	214.34	236.55
	Relative error [%]	3.26	-0.15	-0.22	0.04	0.14	0.15	0.33	0.48	0.71	1.11	1.91	0.67	-0.80
	Fuel share [%]	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00	100.00
Machine drive	TIMES-Italia consumption [PJ]	31.37	27.67	24.52	21.58	20.29	17.98	15.01	12.03	9.06	4.60			
	Temoa-Italia consumption [PJ]	32.53	27.68	24.85	21.60	20.31	17.99	15.01	12.05	9.07	4.60			
	Relative error [%]	3.70	0.06	1.33	0.11	0.09	0.00	0.02	0.15	0.16	0.05			
	Fuel share [%]	8.99	9.05	9.10	8.95	8.92	7.97	7.07	5.96	4.64	2.50			
Steam	TIMES-Italia consumption [PJ]	15.69	14.21	12.60	11.04	10.00	8.57	6.99	5.41	3.83	1.46			
	Temoa-Italia consumption [PJ]	16.68	14.22	12.60	11.18	10.15	8.70	7.00	5.41	3.83	1.46			
	Relative error [%]	6.34	0.01	-0.01	1.33	1.45	1.45	0.02	-0.07	0.07	-0.21			
	Fuel share [%]	4.61	4.65	4.61	4.63	4.46	3.86	3.29	2.67	1.96	0.79			
Biomass	TIMES-Italia consumption [PJ]	8.56	7.18	6.53	5.87	6.03	6.44	6.74	7.21	7.83	8.74	10.00	11.45	12.82
	Temoa-Italia consumption [PJ]	8.53	7.16	6.65	5.87	6.02	6.42	6.73	7.22	7.87	8.82	10.17	11.50	12.69
	Relative error [%]	-0.39	-0.28	1.96	-0.02	-0.23	-0.20	-0.09	0.25	0.46	0.87	1.67	0.51	-0.99
	Fuel share [%]	2.36	2.34	2.44	2.43	2.64	2.85	3.17	3.57	4.03	4.78	5.36	5.37	5.37
Coal	TIMES-Italia consumption [PJ]	26.81	22.38	21.20	18.89	18.26	27.56	34.83	43.69	54.01	69.21	93.25	106.74	119.56
	Temoa-Italia consumption [PJ]	27.49	22.42	19.53	18.97	17.96	27.44	34.95	43.94	54.63	70.39	95.96	108.70	119.80
	Relative error [%]	2.53	0.20	-7.89	0.43	-1.64	-0.46	0.33	0.58	1.14	1.71	2.90	1.84	0.20
	Fuel share [%]	7.59	7.33	7.15	7.86	7.89	12.16	16.45	21.71	27.96	38.19	50.62	50.72	50.64
Coke	TIMES-Italia consumption [PJ]	0.16	0.16	0.14	0.12	0.12	0.10	0.09	0.07	0.06	0.04			
	Temoa-Italia consumption [PJ]	0.16	0.38	0.43	0.41	0.40	0.08	0.06	0.04	0.03	0.00			
	Relative error [%]	-4.76	141.20	210.94	246.17	238.96	-23.43	-30.47	-40.46	-55.27	-100.00			
	Fuel share [%]	0.04	0.12	0.16	0.17	0.17	0.03	0.03	0.02	0.01	0.00			

Continued on page 125

Continued from page 124

Electricity	TIMES-Italia consumption [PJ]	14.32	12.39	11.16	9.83	9.12	10.85	11.84	13.21	14.95	17.48	22.80	26.09	29.22
	Temoa-Italia consumption [PJ]	14.07	12.08	10.70	9.58	9.89	12.09	13.21	14.53	16.19	18.63	21.20	23.95	26.45
	Relative error [%]	-1.71	-2.48	-4.07	-2.55	8.54	11.43	11.56	9.94	8.33	6.60	-6.99	-8.22	-9.48
	Fuel share [%]	3.89	3.95	3.92	3.97	4.35	5.36	6.22	7.18	8.29	10.11	11.19	11.17	11.18
Heavy fuel oil	TIMES-Italia consumption [PJ]	13.59	11.90	10.80	9.50	8.68	9.81	10.31	11.09	12.15	13.69	17.29	19.79	22.17
	Temoa-Italia consumption [PJ]	14.38	11.87	10.33	9.48	8.68	9.90	10.50	11.37	12.57	14.32	18.29	20.75	22.83
	Relative error [%]	5.82	-0.23	-4.40	-0.12	0.00	0.90	1.92	2.47	3.45	4.56	5.78	4.87	3.00
	Fuel share [%]	3.97	3.88	3.78	3.93	3.81	4.39	4.94	5.62	6.43	7.77	9.65	9.68	9.65
LPG	TIMES-Italia consumption [PJ]	6.60	5.66	5.01	4.43	4.31	3.94	3.35	2.77	2.18	1.38			
	Temoa-Italia consumption [PJ]	6.56	5.63	4.98	4.41	4.29	3.93	3.34	2.76	2.18	1.30			
	Relative error [%]	-0.70	-0.59	-0.60	-0.58	-0.51	-0.39	-0.31	-0.18	-0.02	-5.74			
	Fuel share [%]	1.81	1.84	1.82	1.83	1.89	1.74	1.57	1.36	1.11	0.70			
Natural gas	TIMES-Italia consumption [PJ]	125.54	113.34	100.88	88.40	80.53	71.67	61.38	51.45	41.84	27.28	20.77	23.78	26.63
	Temoa-Italia consumption [PJ]	132.94	113.05	100.39	88.18	79.85	70.71	60.40	50.33	40.72	26.11	21.61	24.27	26.91
	Relative error [%]	5.89	-0.26	-0.49	-0.25	-0.85	-1.33	-1.60	-2.17	-2.67	-4.27	4.04	2.07	1.05
	Fuel share [%]	36.72	36.97	36.75	36.55	35.08	31.34	28.43	24.87	20.84	14.17	11.40	11.32	11.38
Oil products	TIMES-Italia consumption [PJ]	1.78	1.06	0.94	0.81	0.72	3.71	5.83	8.32	11.16	15.35	21.89	25.05	28.06
	Temoa-Italia consumption [PJ]	2.44	1.04	0.92	0.80	0.71	3.73	5.90	8.46	11.31	15.61	22.33	25.15	27.86
	Relative error [%]	37.09	-1.69	-1.93	-1.58	-1.13	0.40	1.20	1.70	1.37	1.71	2.02	0.41	-0.70
	Fuel share [%]	0.67	0.34	0.34	0.33	0.31	1.65	2.78	4.18	5.79	8.47	11.78	11.74	11.78
Petroleum coke	TIMES-Italia consumption [PJ]	106.21	90.28	79.96	70.72	69.28	64.65	55.41	46.18	36.94	23.09			
	Temoa-Italia consumption [PJ]	106.27	90.24	81.77	70.79	69.41	64.66	55.37	46.27	36.99	23.10			
	Relative error [%]	0.06	-0.04	2.26	0.11	0.18	0.01	-0.07	0.20	0.12	0.04			
	Fuel share [%]	29.35	29.51	29.94	29.34	30.49	28.66	26.06	22.86	18.93	12.53			

Table 64 reports the absolute values of the produced non-metallic minerals both evaluated with TIMES-Italia and Temoa-Italia, and the relative error between them. It is relevant that for each product and year, the error is within the tolerance interval to guarantee ($\pm 1\%$). Table 65 lists the annual contribution of each non-metallic minerals technology to the production of the correspondent non-metallic product while Table 66 shows the detail of fuels consumed by "Non-metallic minerals" subsector, with the absolute values of energy consumption evaluated both with TIMES and Temoa, the relative error between the two models and fuel share (evaluated on the basis of Temoa values) for each optimization year. Both for Table 65 and Table 66, focusing on the relative errors, a large variance should be highlighted in the numeric values evaluated for each technology and fuel consumption. This is mainly due to the fact that for some technologies and fuels, characterized by a low share with respect to the total production or energy consumption, also errors due to parameters approximation (normally negligible) become relevant. This means that a way to give different importance to the relative errors is needed, also considering the production or fuel consumption share on the total.

This is achieved deriving an average relative error (weighted on the correspondent share, according to Equation 29) between Temoa and TIMES results for each optimization year. It is possible to draw an error curve, with the average relative error evolution in the time, for the entire time horizon.

$$\mathrm{err_{avg}}[\%] = \sum_{\mathrm{i}} \left|\mathrm{err^i}[\%]\right| * \mathrm{f^i}[\%] \tag{29}$$

Concerning the production by technologies, Figure 17 shows the average relative error evaluated for each non-metallic product, as already reported in Table 65. From the shape of the curves, it should be highlighted that the errors associated to the technologies' development are below the tolerance value along the entire time horizon. In the first years, higher error values can be noted, probably due to the fact that the base year (2006) is treated in a different way by the two tools. Indeed, in TIMES it is included in the optimization and in the results, while Temoa only evaluates results for the time periods labeled as "future" (in the Temoa-Italia case starting from 2007).

This difference could have an impact on the first years' results accuracy, being the production in those year largely due to the base year technologies.

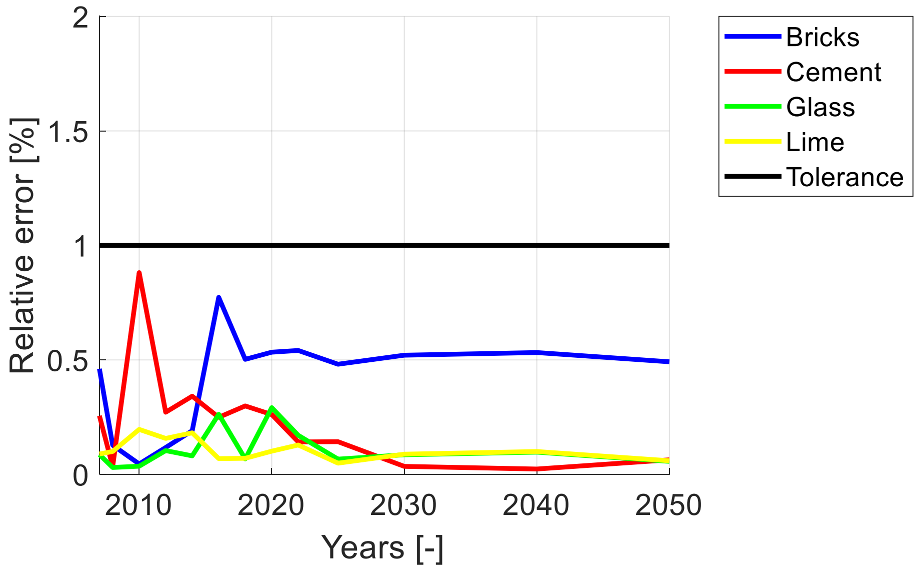


Figure 17. Average relative error (weighted on production shares) for "Non-metallic minerals" subsector technologies. Black line is used to represent the +1% tolerance value.

Figure 18 shows the computed relative error trend for fuel consumption (derived by data in Table 66, according to Equation 29).

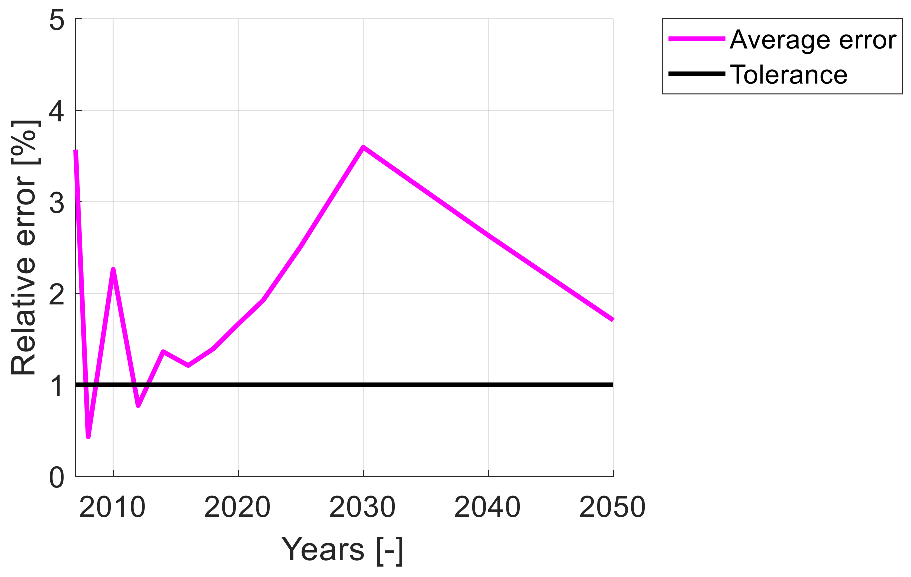


Figure 18. Average relative error (weighted on fuel shares) for "Non-metallic minerals" subsector fuel consumption. Black line is used to represent the +1% tolerance value.

In this case, the average error overcome the 1% tolerance almost for the entire time horizon (with the exceptions of 2008 and 2012). The first comment that can be done, is that error is higher in correspondence of the first simulation years (notably for 2007) and for 2030. Two reasons can explain this behavior. Concerning the first years, as already said before commenting Figure 17, this is probably due to the difference modeling of the base year between the two tools. For 2030, it should be remembered that this is the year at which the disposal of base-year technologies is forced (maximum activity imposed equal to 0). Probably the 2030, completing the shift from base year technologies to the new ones, is a critical year from the point of view of results accuracy. Anyway, it is of course true that, comparing results for the single fuel consumption, errors become higher with respect to simply analyze the total non-metallic minerals production and technologies' utilization, not only for 2007 and 2030 but generally. Energy consumption for single fuel is derived by the model taking into account several parameters: technologies' efficiency, constraints on fuel input share, constraints on commodity output share etc. For that reason, it was predictable that the total accuracy would be lower, at this degree of detail for the analyses. A reassuring comment consists in noting that, in any case, the average error remains on relatively low values (in any case less than 4%).

Energy consumption results evaluated by the two tools, split by fuel, are also presented in Figure 19 and Figure 20. These figures allow to appreciate the qualitative similarity of the results, in addition to the numerical evaluation of errors already provided.

It is important to remember again that these results are not significant from the point of view of the energy mix evolution, both for the period for which historical data of real fuel consumption are available (2006-2018) and for the future period for which the model provides forecasts (2020-2050). This is due to the fact that some important constraints (present in TIMES-Italia) are still not implemented in Temoa-Italia, as already said in Section 1.1 (and of course for that reason relative files have been also deselected for the TIMES optimization). For this reason, for example, the important increasing of the coal consumption shown in the figures is not realistic. In other words, the current results are useful only to prove that models provide the same results, independently on their significance for energy forecast.

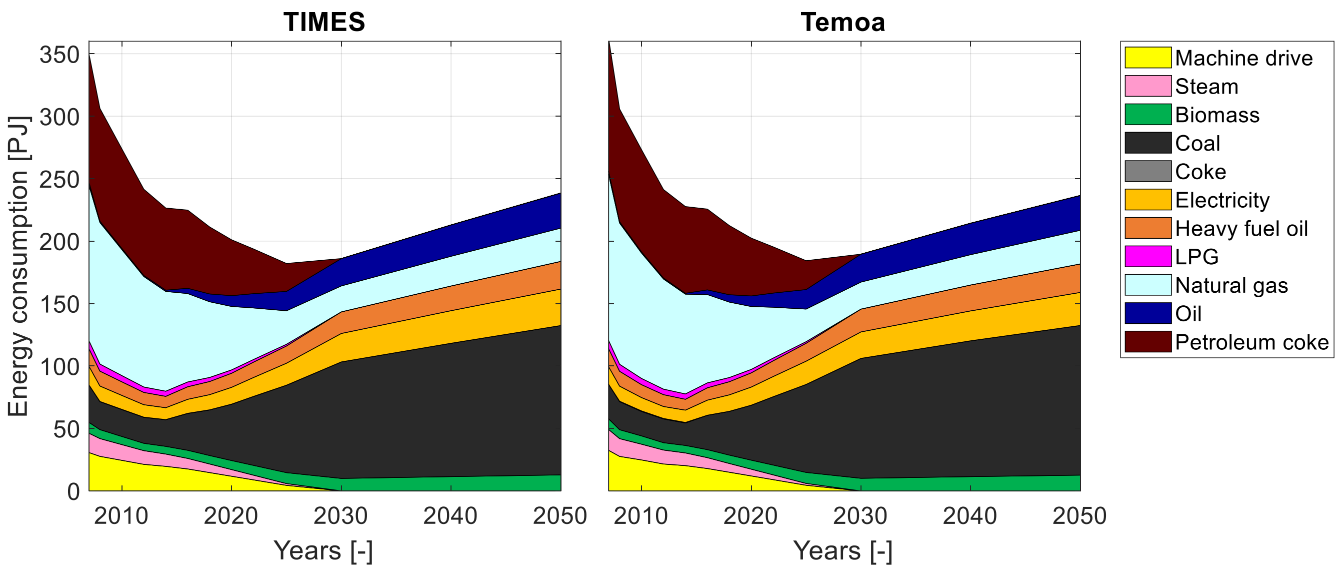


Figure 19. Energy consumption split by fuel for "non-metallic minerals" industrial subsector, obtained from TIMES-Italia and Temoa-Italia.

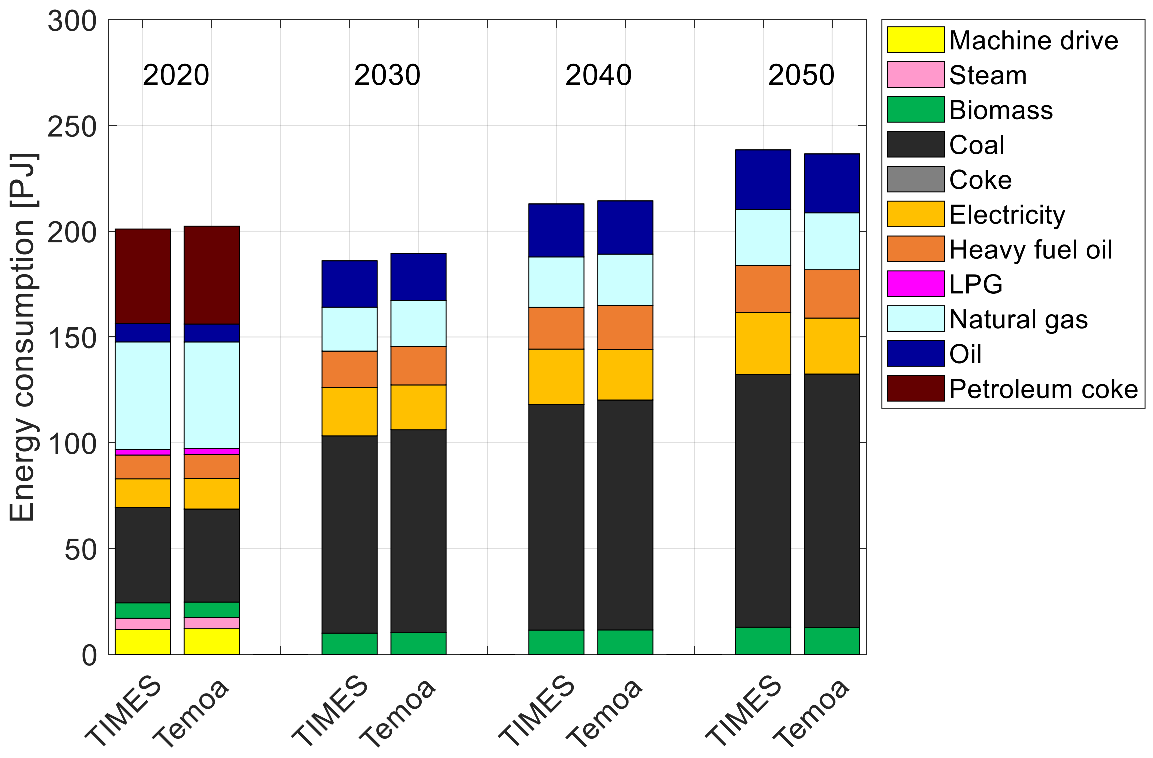


Figure 20. Comparison of TIMES-Italia and Temoa-Italia results split by fuel, detail for 2020, 2030, 2040 and 2050 results.

Chapter 5

Conclusions and perspective

Temoa-Italia constitutes a valid open-source alternative as an optimization model of the Italian energy system. Certainly, the model needs to be completed with the missing sets of constraints, which allow not only to obtain the same results as TIMES-Italia, but also that these results make sense and are consistent, both with the real historical evolution of the energy mix and the expectations on its future evolution.

However, already at the current stage of development of the model, the objectives of the thesis have been achieved:

a. The TIMES-Italia model has been completely reviewed and updated, particularly with new technologies transport and industry modules, also performing updates on the calibration at the base year.
b. The new Temoa-Italia model database has been developed, translating in open-source format the same information contained in the TIMES-Italia dataset and developing integration to the Temoa framework, expanding its capabilities, and simplifying the data preprocessing. It is an important point having the perspective to perform further studies and works analyzes with Temoa.
c. The results obtained both with TIMES-Italia and Temoa-Italia have been compared in detail (as part of the thesis treatment, only for one industrial subsector) and it has been demonstrated that differences are contained in a certain acceptable tolerance interval or, where not, motivations for this have been provided. This has been done to validate the results reliability of Temoa with respect to a well-established commercial tool such as the TIMES model generator.

Having demonstrated that Temoa framework can be a valid alternative in the context of energy system modeling, it can be now used as a reference tool on which modifications or integrations on the optimization paradigms can be executed. This would be a more significant contribution to the transparency of scientific research in the context of scenario analysis, guaranteeing also third-part verification and the accessibility of data and tools exploited by modelers to a more and more large audience, without any commercial restriction.

Starting from the Temoa framework and, in this case, from Temoa-Italia, also future studies and developments are possible. First, a sustainability analysis could be integrated into the model; it is possible to do it, at least at the beginning, as a postprocessing of the results, in other to derive a set of indicators to evaluate the energy mix (evaluated on the classic economic optimization) from the sustainability point of view. However, as soon as those indicators will prove to be adequate and consistent with the purpose to assess the technologies environmental and social sustainability, they should be integrated in the tecno-economic characterization of the technologies included in the model database. This is fundamental to take into account the sustainability levels during the optimization process, for example with a multi-objective function, considering of course the total cost as optimization criterion, but also the sustainability level of the energy mix and any other indicators of interest.

Subsequently, it would certainly be desirable and necessary to apply the same concepts of open-source and extended optimization paradigm no longer only to a model limited to the Italian case, but to an international model.

Acknowledgements

Nell'ultima (spero) delle notti insonni che dedico a questa tesi, metto in ordine qualche appunto che ho preso nel corso delle scorse settimane, pensando a cosa scrivere in questa pagina di ringraziamenti. Approcciandomi a scrivere queste righe avverto un misto di banalità e imbarazzo, eppure sento il bisogno di rendere partecipe di questo mio piccolo traguardo chi ha condiviso con me anche solo un pezzetto del percorso di questi ultimi anni. Certamente lo farò in maniera non esaustiva.

Grazie, prima di tutto, a chi mi ha permesso di essere qui oggi. Alla mia famiglia, che mi ha sostenuto materialmente ed emotivamente in questi anni universitari. Ai miei genitori, che hanno accolto la scelta di allontanami da casa sei anni fa. Grazie papi, per avermi testimoniato il valore della famiglia e del lavoro, per essere per me un punto di riferimento. Grazie mamma, per i sacrifici che hai sempre fatto e continui a fare per tutti noi. Grazie Marco, perché penso che in molte cose sei un passo avanti. Grazie nonna Lucia per la tua allegria e grazie nonna Liliana per la tua presenza anche da lontano. Grazie nonno Giorgio e nonno Nino.

Grazie ai miei zii e cugini.

Grazie a chi, tra i primi "colleghi" che ho incontrato nelle grigie aule del Politecnico, c'è ancora adesso. Grazie a Gianpaolo, per la sua profondità, e grazie a Simone, per la sua presenza costante seppur a distanza e il suo coraggio.

Grazie a chi invece ho incontrato strada facendo, ai "colleghi energetici". In particolare, grazie a Chiara (per la tua capacità di ascolto e per Conca del Pra), a Silvio (per la tua scioltezza e rilassatezza), a Jonas e a Francesco (per essere un esempio di dedizione e tenacia).

Grazie a chi condivide con me le proprie vacanze, ai miei amici Marinesi da cui mi sono allontanato e progressivamente sento di continuare ad allontanarmi. Grazie per la spensieratezza che sapete donarmi quando necessario.

Grazie a chi, all'inizio del mio percorso da fuorisede, mi ha offerto l'opportunità di sentirmi parte di una comunità fin dall'inizio. Grazie al Torino 110, che ha saputo accogliermi e accompagnarmi verso la Partenza. Grazie a Marco, per essere stato un amico prima che un capo. Grazie ai miei compagni di clan, a coloro con cui ho vissuto la ricchezza della diversità. Grazie a Giulia Greco, per la nostra amicizia che spero di riuscire a riscoprire.

Grazie al Torino 24. E qui mi sarà concessa qualche parola in più. Grazie dell'accoglienza che mi è stata offerta fin da subito, "a scatola chiusa". Grazie alla Comunità Capi per essere, in ciascun suo componente, testimone viva e quotidiana di servizio, per la fatica e la gioia che condividiamo, per il senso di famiglia che respiro nel riunirci (nonostante le nostre tante imperfezioni).

Grazie a chi ha sopportato il mio modo di essere in staff in questi quattro anni. Grazie del dialogo intenso, del confronto e della mediazione. Grazie a chi c'è sempre stato e a chi ci sarà, grazie per le risate e per i pianti profondi. Grazie in particolare, alla mia ultima staff in Reparto. Grazie a Letizia. Grazie a MG per esserci sempre. Per quest'ultimo anno, grazie a Benni; per gli accordi e i disaccordi, per essere stata la mia memoria e per sostenere con forza le tue idee, quando pensi che sia necessario.

Grazie al Reparto Antingo e alla Comunità R/S Aquile Randagie, per essere stati ed essere ancora la più grande "sfida" della mia vita ed avermi davvero reso "un po' migliore di come mi avete trovato". Specialmente, grazie ad Anto R., Anto S., Elena, Elli, Ema, Fede, Rebecca, Vale e Vera, per essere la mia gioia e la mia debolezza.

Infine, grazie a Daniele, per la tua pazienza nell'accompagnarmi a scoprire tante cose per me nuove, per esserci sempre e per andare al di là dei ruoli e delle formalità. Grazie ad Antonio per aver vissuto in parallelo a me questo percorso di tesi e a tutto il gruppo di ricerca MAHTEP, che spero di avere l'opportunità di conoscere meglio anche in futuro.

Grazie a Laura Savoldi. Per aver creduto in me, anche quando non ne avrebbe avuto alcun motivo e per continuare a crederci anche oggi. Per offrirmi quotidianamente una testimonianza di servizio, dedizione al lavoro e ascolto del prossimo.

Grazie anche un po’ a me stesso, per le mie tante paure e debolezze, per lo stupore che nasce dai piccoli gesti di coraggio che scelgo di fare. Per le “piccole meraviglie” che ho vissuto e che fanno parte di me. Per tutto ciò che mi aspetta e che attendo di scoprire.

References

[1] IPCC, "Climate Change 2014 Synthesis Report Summary for Policymakers," February 2018. [Online]. Available: https://www.ipcc.ch/site/assets/uploads/2018/02/AR5_SYR_FINAL_SPM.pdf. [Accessed 29 June 2021].

[2] UNFCCC, "Kyoto Protocol to the United Nations Framework Convention on Climate Change," 1998. [Online]. Available: https://unfccc.int/resource/docs/convkp/kpeng.pdf. [Accessed 29 June 2021].

[3] European Commission, "Paris Agreement," [Online]. Available: https://ec.europa.eu/clima/policies/international/negotiations/paris_en. [Accessed 27 June 2021].

[4] Ministero dello Sviluppo Economico, Ministero della Transizione Ecologica, "Strategia Energetica Nazionale," 10 November 2017. [Online]. Available: https://www.minambiente.it/sites/default/files/archivio/allegati/testo-integrale-sen-2017.pdf. [Accessed 29 June 2021].

[5] Ministero dello Sviluppo Economico, Ministero della Transizione Ecologica, Ministero delle Infrastrutture e dei Trasporti, "PNIEC (Piano Nazionale Integratio per l'Energia e il Clima)," 17 January 2020. [Online]. Available: https://www.mise.gov.it/images/stories/documenti/PNIEC_finale_17012020.pdf. [Accessed 24 June 2021].

[6] IRENA, "Planning for the renewable future: Long-term modelling and tools to expand variable renewable power in emerging economies," January 2017. [Online]. Available: https://www.irena.org/publications/2017/Jan/Planning-for-the-renewable-future-Long-term-modelling-and-tools-to-expand-variable-renewable-power. [Accessed 28 June 2021].

[7] M. G. Prina, G. Manzolini, D. Mosera, B. Nastasic and W. Sparbera, "Classification and challenges of bottom-up energy system models - A

review," Renewable and Sustainable Energy Reviews, vol. 129, no. 109917, 2020.

[8] IEA-ETSAP, "TIMES," [Online]. Available: https://iea-etsap.org/index.php/etsap-tools/model-generators/times. [Accessed 16 June 2021].

[9] R. Vicente-Saez, R. Gustafsson and L. d. Brande, "The dawn of an open exploration era: Emergent principles and practices of open science and innovation of university research teams in a digital world," Technological Forecasting and Social Change, vol. 156, no. 120037, 2020.

[10] European Commission, "The EU's open science policy," [Online]. Available: https://ec.europa.eu/info/research-and-innovation/strategy/strategy-2020-2024/our-digital-future/open-science_en#the-eus-open-science-policy. [Accessed 26 June 2021].

[11] R. Morrison, "Energy system modeling: Public transparency, scientificreproducibility, and open development," Energy Strategy Reviews, vol. 20, pp. 49-63, 2018.

[12] OSeMOSYS community, "OSeMOSYS Open Source Energy Modelling System," [Online]. Available: http://www.osemosys.org/. [Accessed 30 June 2021].

[13] North Carolina State University, "Temoa Tools for Energy Model Optimization and Analysis," [Online]. Available: https://temoacloud.com/. [Accessed 22 June 2021].

[14] ENEA, "IL MODELLO ENERGETICO TIMES-Italia," 2011. [Online]. Available: https://biblioteca.bologna.enea.it/RT/2011/2011_9_ENEA.pdf. [Accessed 12 November 2020].

[15] ENEA, "ENEA," [Online]. Available: https://www.enea.it/. [Accessed 26 June 2021].

[16] United Nations, Department of Economic and Social Affairs, "Sustainable Development Goals," [Online]. Available: https://sdgs.un.org/goals. [Accessed 27 June 2021].

[17] IEA, "World Energy Balances," [Online]. Available: https://www.iea.org/data-and-statistics/data-product/world-energy-balances. [Accessed 16 June 2021].

[18] "IEA ETSAP - Cooking Appliances," [Online]. Available: https://iea-etsap.org/E-TechDS/PDF/R06_Cooking_FINAL_GSOK.pdf.

[19] GSE, "Valutazione del potenziale nazionale e regionale di applicazione della cogenerazione ad alto rendimento e del teleriscaldamento efficiente," 2016. [Online]. Available: https://ec.europa.eu/energy/sites/ener/files/documents/it_potenziale_car_tlr_nazionale_e_regionale_dic_2016.pdf. [Accessed 28 11 2020].

[20] RSE, "Analisi tecnico-economica di interventi di riqualificazione energetica del parco edilizio residenziale italiano," [Online]. Available: http://www.rse-web.it/documenti/documento/315557. [Accessed 28 11 2020].

[21] EUROfusion, "ETM Model," [Online]. Available: https://collaborators.euro-fusion.org/collaborators/socio-economics/economics/model/. [Accessed 21 June 2021].

[22] EUROfusion, "EUROfusion," [Online]. Available: https://www.euro-fusion.org/. [Accessed 21 June 2021].

[23] MISE, "Bollettino petrolifero - Anno 2017," [Online]. Available: https://dgsaie.mise.gov.it/bollettino-petrolifero?anno=2017. [Accessed 21 June 2021].

[24] JRC, "JRC-EU-TIMES model," [Online]. Available: https://ec.europa.eu/jrc/en/scientific-tool/jrc-eu-times-model-assessing-long-term-role-energy-technologies. [Accessed 21 June 2021].

[25] GSE, "Conto Energia," [Online]. Available: https://www.gse.it/servizi-per-te/fotovoltaico/conto-energia. [Accessed 04 July 2021].

[26] J. DeCarolis, K. Hunter and S. Sreepathi, "The TEMOA Project: Tools for Energy Model Optimization and Analysis," 23 June 2010. [Online]. Available: https://temoacloud.com/wp-

content/uploads/2019/12/DeCarolis_IEW2010_paper.pdf. [Accessed 30 June 2021].

[27] J. DeCarolis, K. Hunter and S. Sreepathi, "Multi-stage stochastic optimization of a simple energy system," 21 June 2012. [Online]. Available: https://temoacloud.com/wp-content/uploads/2019/12/DeCarolis_IEW2012_paper.pdf. [Accessed 30 June 2021].

[28] J. DeCarolis, S. Sreepathi, K. Hunter, B. Li and S. Kanungo, "Energy Scenario Exploration with Modeling to Generate Alternatives (MGA)," 2012. [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1912/1912.03788.pdf. [Accessed 30 June 2021].

[29] KTH Royal Institute of Technology, "2015 OSeMOSYS User Manual," 2015. [Online]. Available: http://www.osemosys.org/uploads/1/8/5/0/18504136/new-website_osemosys_manual_-_working_with_text_files_-_2015-11-05.pdf. [Accessed 24 September 2020].

[30] "GLPK (GNU Linear Programming Kit)," [Online]. Available: https://www.gnu.org/software/glpk/. [Accessed 22 June 2021].

[31] IBM, "IBM ILOG CPLEX Optimization Studio," [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio. [Accessed 22 June 2021].

[32] J. M. M. P. a. M. J. J. Czyzyk, "The NEOS Server," IEEE Journal on Computational Science and Engineering, vol. 5, no. 3, pp. 68-75, 1998.

[33] W. a. M. J. J. Gropp, "Optimization Environments and the NEOS Server," Approximation Theory and Optimization, Vols. M. D. Buhmann and A. Iserles, eds., Cambridge University Press, p. 167.182, 1997.

[34] E. Dolan, "The NEOS Server 4.0 Administrative Guide," 2001. [Online]. Available: https://www.mcs.anl.gov/papers/TM-250.pdf. [Accessed 22 June 2021].

[35] Gurobi Optimization, "Gurobi Optimization," [Online]. Available: https://www.gurobi.com/. [Accessed 22 June 2021].

[36] KanORS-EMR, "VEDA-BE," [Online]. Available: https://www.kanors-emr.org/veda-be. [Accessed 23 June 2021].

[37] D. Lerede, C. Bustreo, F. Gracceva, Y. Lechón and L. Savoldi, "Analysis of the Effects of Electrification of the Road Transport Sector on the Possible Penetration of Nuclear Fusion in the Long-Term European Energy Mix," Energies, vol. 13, 2020.

[38] D. Lerede, C. Bustreo, F. Gracceva, M. Saccone and L. Savoldi, "Techno-economic and environmental characterization of industrial technologies for transparent bottom-up energy modeling," Renewable and Sustainable Energy Reviews, vol. 140, 2021.

Appendix

A. Database preprocessing

```
import pandas as pd
import numpy as np
import sqlite3

database_name = "Industry.sqlite"
lifetime_default = 40

print_set = {'LifetimeProcess':         False,
             'Efficiency':              False,
             'TechInputSplit':          False,
             'TechOutputSplit':         False,
             'EmissionActivity':        False,
             'CostInvest':              False,
             'CostFixed':               False,
             'CostVariable':            False,
             'MinCapacity':             False,
             'MinActivity':             False,
             'MaxCapacity':             False,
             'MaxActivity':             False,
             'AvailabilityFactor':      False,
             'Demand':                  False,
             'CapacityFactorProcess':   False,
             'CapacityCredit':          False}

tosql_set = {'LifetimeProcess':         True,
             'Efficiency':              True,
             'TechInputSplit':          True,
             'TechOutputSplit':         True,
             'EmissionActivity':        True,
             'CostInvest':              True,
             'CostFixed':               True,
             'CostVariable':            True,
             'MinCapacity':             True,
             'MinActivity':             True,
             'MaxCapacity':             True,
             'MaxActivity':             True,
             'AvailabilityFactor':      True,
             'Demand':                  True,
             'CapacityFactorProcess':   True,
             'CapacityCredit':          True}



# LifetimeProcess

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
LifetimeProcess = pd.read_sql("select * from LifetimeProcess", conn)

regions = list()
tech = list()
vintage = list()
life_process = list()
life_process_notes = list()

tech_already_considered=list()
for i_tech in range(0, len(LifetimeProcess.tech)):
    tech_i = LifetimeProcess.tech[i_tech]

    flag_check = 0
    for check in range(0, len(tech_already_considered)):
        if tech_i == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(LifetimeProcess.tech)):
            if LifetimeProcess.tech[j_tech] == tech_i:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i)

        if flag == 0:  # No other values

            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= LifetimeProcess.vintage[i_tech] and
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:
                    regions.append(LifetimeProcess.regions[i_tech])
                    tech.append(LifetimeProcess.tech[i_tech])
                    vintage.append(int(time_periods.t_periods[i_year]))
                    life_process.append(int(LifetimeProcess.life_process[i_tech]))
                    life_process_notes.append(LifetimeProcess.life_process_notes[i_tech])

        else:
            for i_location in range(0, len(location)-1):
                year1 = LifetimeProcess.vintage[location[i_location]]
                year2 = LifetimeProcess.vintage[location[i_location+1]]
                life1 = LifetimeProcess.life_process[location[i_location]]
                life2 = LifetimeProcess.life_process[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        regions.append(LifetimeProcess.regions[i_tech])
                        tech.append(LifetimeProcess.tech[i_tech])
                        vintage.append(int(year))
                        life_process.append(life1) #int(life1 + (year-year1)/(year2-
year1)*(life2-life1))

life_process_notes.append(LifetimeProcess.life_process_notes[i_tech])

            year_last = LifetimeProcess.vintage[location[i_location+1]]
            cost = LifetimeProcess.life_process[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-1]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        regions.append(LifetimeProcess.regions[i_tech])
                        tech.append(LifetimeProcess.tech[i_tech])
                        vintage.append(int(year))

life_process.append(int(LifetimeProcess.life_process[location[i_location + 1]]))

life_process_notes.append(LifetimeProcess.life_process_notes[i_tech])
            else:
                regions.append(LifetimeProcess.regions[i_tech])
                vintage.append(int(year_last))
                tech.append(LifetimeProcess.tech[i_tech])
                life_process.append(int(LifetimeProcess.life_process[location[i_location +
1]]))
                life_process_notes.append(LifetimeProcess.life_process_notes[i_tech])

LifetimeProcess_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "tech": pd.Series(tech, dtype='str'),
        "vintage": pd.Series(vintage, dtype='int'),
        "life_process": pd.Series(life_process, dtype='int'),
        "life_process_notes": pd.Series(life_process_notes, dtype='str')
    }
)

if tosql_set['LifetimeProcess']:
    LifetimeProcess_DF.to_sql("LifetimeProcess", conn, index=False, if_exists='replace')

if print_set['LifetimeProcess']:
    pd.set_option('display.max_rows', len(LifetimeProcess_DF))
    pd.set_option('display.max_columns', len(LifetimeProcess_DF))
    print("\nLifetimeProcess DataFrame\n\n", LifetimeProcess_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = 1
print('[',print_i,'/',len(print_set),']     LifetimeProcess updated...')


# Efficiency

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
Efficiency = pd.read_sql("select * from Efficiency", conn)

regions = list()
input_comm = list()
tech = list()
vintage = list()
output_comm = list()
efficiency = list()
eff_notes = list()

tech_already_considered=list()
for i_tech in range(0, len(Efficiency.tech)):
    tech_i = Efficiency.tech[i_tech]

    flag_check = 0
    tech_i_check = Efficiency.input_comm[i_tech] + tech_i + Efficiency.output_comm[i_tech]
    for check in range(0, len(tech_already_considered)):
        if tech_i_check == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(Efficiency.tech)):
            tech_j_check = Efficiency.input_comm[j_tech] + Efficiency.tech[j_tech] +
Efficiency.output_comm[j_tech]
            if tech_j_check == tech_i_check:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i_check)

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= Efficiency.vintage[i_tech] and
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:
                    regions.append(Efficiency.regions[i_tech])
                    input_comm.append(Efficiency.input_comm[i_tech])
                    tech.append(Efficiency.tech[i_tech])
                    vintage.append(int(time_periods.t_periods[i_year]))
                    output_comm.append(Efficiency.output_comm[i_tech])

efficiency.append(float(np.format_float_scientific(Efficiency.efficiency[i_tech], 2)))
                    eff_notes.append(Efficiency.eff_notes[i_tech])

        else:
            for i_location in range(0, len(location)-1):
                year1 = Efficiency.vintage[location[i_location]]
                year2 = Efficiency.vintage[location[i_location+1]]
                eff1 = Efficiency.efficiency[location[i_location]]
                eff2 = Efficiency.efficiency[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        regions.append(Efficiency.regions[i_tech])
                        input_comm.append(Efficiency.input_comm[i_tech])
                        tech.append(Efficiency.tech[i_tech])
                        vintage.append(int(year))
                        output_comm.append(Efficiency.output_comm[i_tech])
                        efficiency.append(float(np.format_float_scientific(eff1 + (year-
year1)/(year2-year1)*(eff2-eff1), 2)))
                        eff_notes.append(Efficiency.eff_notes[i_tech])

            year_last = Efficiency.vintage[location[i_location+1]]
            eff = Efficiency.efficiency[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-1]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        regions.append(Efficiency.regions[i_tech])
                        input_comm.append(Efficiency.input_comm[i_tech])
                        tech.append(Efficiency.tech[i_tech])
                        vintage.append(int(year))
                        output_comm.append(Efficiency.output_comm[i_tech])

efficiency.append(float(np.format_float_scientific(Efficiency.efficiency[location[i_locati
on + 1]], 2)))
                        eff_notes.append(Efficiency.eff_notes[i_tech])
            else:
                regions.append(Efficiency.regions[i_tech])
                input_comm.append(Efficiency.input_comm[i_tech])
                tech.append(Efficiency.tech[i_tech])
                vintage.append(int(year_last))
                output_comm.append(Efficiency.output_comm[i_tech])

efficiency.append(float(np.format_float_scientific(Efficiency.efficiency[location[i_locati
on + 1]], 2)))
                eff_notes.append(Efficiency.eff_notes[i_tech])

Efficiency_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "input_comm": pd.Series(input_comm, dtype='str'),
        "tech": pd.Series(tech, dtype='str'),
        "vintage": pd.Series(vintage, dtype='int'),
        "output_comm": pd.Series(output_comm, dtype='str'),
        "efficiency": pd.Series(efficiency, dtype='float'),
        "eff_notes": pd.Series(eff_notes, dtype='str')
    }
)

if tosql_set['Efficiency']:
    Efficiency_DF.to_sql("Efficiency", conn, index=False, if_exists='replace')

if print_set['Efficiency']:
    pd.set_option('display.max_rows', len(Efficiency_DF))
    pd.set_option('display.max_columns', len(Efficiency_DF))
    print("\nEfficiency DataFrame\n\n", Efficiency_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     Efficiency updated...')
```

```
# TechInputSplit

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
TechInputSplit = pd.read_sql("select * from TechInputSplit", conn)

regions = list()
periods = list()
input_comm = list()
tech = list()
ti_split = list()
ti_split_notes = list()

tech_already_considered=list()
for i_tech in range(0, len(TechInputSplit.tech)):
    tech_i = TechInputSplit.tech[i_tech]

    flag_check = 0
    tech_i_check = TechInputSplit.input_comm[i_tech] + tech_i
    for check in range(0, len(tech_already_considered)):
        if tech_i_check == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(TechInputSplit.tech)):
            tech_j_check = TechInputSplit.input_comm[j_tech] + TechInputSplit.tech[j_tech]
            if tech_j_check == tech_i_check:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i_check)

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= TechInputSplit.periods[i_tech] and
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:
                    regions.append(TechInputSplit.regions[i_tech])
                    periods.append(int(time_periods.t_periods[i_year]))
                    input_comm.append(TechInputSplit.input_comm[i_tech])
                    tech.append(TechInputSplit.tech[i_tech])

ti_split.append(float(np.format_float_scientific(TechInputSplit.ti_split[i_tech], 2)))
                    ti_split_notes.append(TechInputSplit.ti_split_notes[i_tech])

        else:
            for i_location in range(0, len(location)-1):
                year1 = TechInputSplit.periods[location[i_location]]
                year2 = TechInputSplit.periods[location[i_location+1]]
                eff1 = TechInputSplit.ti_split[location[i_location]]
                eff2 = TechInputSplit.ti_split[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        regions.append(TechInputSplit.regions[i_tech])
                        periods.append(int(year))
                        input_comm.append(TechInputSplit.input_comm[i_tech])
                        tech.append(TechInputSplit.tech[i_tech])
                        ti_split.append(float(np.format_float_scientific(eff1 + (year-
year1)/(year2-year1)*(eff2-eff1), 2)))
                        ti_split_notes.append(TechInputSplit.ti_split_notes[i_tech])

            year_last = TechInputSplit.periods[location[i_location+1]]
            eff = TechInputSplit.ti_split[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-1]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        regions.append(TechInputSplit.regions[i_tech])
                        periods.append(int(year))
                        input_comm.append(TechInputSplit.input_comm[i_tech])
                        tech.append(TechInputSplit.tech[i_tech])

ti_split.append(float(np.format_float_scientific(TechInputSplit.ti_split[location[i_locati
on + 1]], 2)))
                        ti_split_notes.append(TechInputSplit.ti_split_notes[i_tech])
            else:
                regions.append(TechInputSplit.regions[i_tech])
                periods.append(int(year_last))
                input_comm.append(TechInputSplit.input_comm[i_tech])
                tech.append(TechInputSplit.tech[i_tech])

ti_split.append(float(np.format_float_scientific(TechInputSplit.ti_split[location[i_locati
on + 1]], 2)))
                ti_split_notes.append(TechInputSplit.ti_split_notes[i_tech])

TechInputSplit_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "periods": pd.Series(periods, dtype='int'),
        "input_comm": pd.Series(input_comm, dtype='str'),
        "tech": pd.Series(tech, dtype='str'),
        "ti_split": pd.Series(ti_split, dtype='float'),
        "ti_split_notes": pd.Series(ti_split_notes, dtype='str')
    }
)

if tosql_set['TechInputSplit']:
    TechInputSplit_DF.to_sql("TechInputSplit", conn, index=False, if_exists='replace')

if print_set['TechInputSplit']:
    pd.set_option('display.max_rows', len(TechInputSplit_DF))
    pd.set_option('display.max_columns', len(TechInputSplit_DF))
    print("\nTechInputSplit DataFrame\n\n", TechInputSplit_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     TechInputSplit updated...')



# TechOutputSplit

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
TechOutputSplit = pd.read_sql("select * from TechOutputSplit", conn)

regions = list()
periods = list()
output_comm = list()
tech = list()
to_split = list()
to_split_notes = list()

tech_already_considered=list()
for i_tech in range(0, len(TechOutputSplit.tech)):
    tech_i = TechOutputSplit.tech[i_tech]

    flag_check = 0
    tech_i_check = tech_i + TechOutputSplit.output_comm[i_tech]
    for check in range(0, len(tech_already_considered)):
        if tech_i_check == tech_already_considered[check]:
            flag_check = 1
    if flag_check == 0:
        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(TechOutputSplit.tech)):
            tech_j_check = TechOutputSplit.tech[j_tech] +
TechOutputSplit.output_comm[j_tech]
            if tech_j_check == tech_i_check:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i_check)

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= TechOutputSplit.periods[i_tech] and
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:
                    regions.append(TechOutputSplit.regions[i_tech])
                    periods.append(int(time_periods.t_periods[i_year]))
                    tech.append(TechOutputSplit.tech[i_tech])
                    output_comm.append(TechOutputSplit.output_comm[i_tech])

to_split.append(float(np.format_float_scientific(TechOutputSplit.to_split[i_tech], 2)))
                    to_split_notes.append(TechOutputSplit.to_split_notes[i_tech])

        else:
            for i_location in range(0, len(location)-1):
                year1 = TechOutputSplit.periods[location[i_location]]
                year2 = TechOutputSplit.periods[location[i_location+1]]
                eff1 = TechOutputSplit.to_split[location[i_location]]
                eff2 = TechOutputSplit.to_split[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        regions.append(TechOutputSplit.regions[i_tech])
                        periods.append(int(year))
                        tech.append(TechOutputSplit.tech[i_tech])
                        output_comm.append(TechOutputSplit.output_comm[i_tech])
                        to_split.append(float(np.format_float_scientific(eff1 + (year-
year1)/(year2-year1)*(eff2-eff1), 2)))
                        to_split_notes.append(TechOutputSplit.to_split_notes[i_tech])

            year_last = TechOutputSplit.periods[location[i_location+1]]
            eff = TechOutputSplit.to_split[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-1]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        regions.append(TechOutputSplit.regions[i_tech])
                        periods.append(int(year))
                        tech.append(TechOutputSplit.tech[i_tech])
                        output_comm.append(TechOutputSplit.output_comm[i_tech])

to_split.append(float(np.format_float_scientific(TechOutputSplit.to_split[location[i_locat
ion + 1]], 2)))
                        to_split_notes.append(TechOutputSplit.to_split_notes[i_tech])
            else:
                regions.append(TechOutputSplit.regions[i_tech])
                periods.append(int(year_last))
                tech.append(TechOutputSplit.tech[i_tech])
                output_comm.append(TechOutputSplit.output_comm[i_tech])

to_split.append(float(np.format_float_scientific(TechOutputSplit.to_split[location[i_locat
ion + 1]], 2)))
                to_split_notes.append(TechOutputSplit.to_split_notes[i_tech])

TechOutputSplit_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "periods": pd.Series(periods, dtype='int'),
        "tech": pd.Series(tech, dtype='str'),
        "output_comm": pd.Series(output_comm, dtype='str'),
        "to_split": pd.Series(to_split, dtype='float'),
        "to_split_notes": pd.Series(to_split_notes, dtype='str')
    }
)

if tosql_set['TechOutputSplit']:
    TechOutputSplit_DF.to_sql("TechOutputSplit", conn, index=False, if_exists='replace')

if print_set['TechOutputSplit']:
    pd.set_option('display.max_rows', len(TechOutputSplit_DF))
    pd.set_option('display.max_columns', len(TechOutputSplit_DF))
    print("\nTechOutputSplit DataFrame\n\n", TechOutputSplit_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     TechOutputSplit updated...')



# EmissionActivity

conn = sqlite3.connect(database_name)
Efficiency = pd.read_sql("select * from Efficiency", conn)
CommodityEmissionFactor = pd.read_sql("select * from CommodityEmissionFactor", conn)

regions=list()
emis_comm=list()
input_comm=list()
tech=list()
vintage=list()
output_comm=list()
emis_act=list()
emis_act_units=list()
emis_act_notes=list()

for i_tech in range(0, len(Efficiency.tech)):
    for i_comm in range(0, len(CommodityEmissionFactor.input_comm)):
        if Efficiency.input_comm[i_tech] == CommodityEmissionFactor.input_comm[i_comm]:
            regions.append(Efficiency.regions[i_tech])
            emis_comm.append(CommodityEmissionFactor.emis_comm[i_comm])
            input_comm.append(CommodityEmissionFactor.input_comm[i_comm])
            tech.append(Efficiency.tech[i_tech])
            vintage.append(Efficiency.vintage[i_tech])
            output_comm.append(Efficiency.output_comm[i_tech])

emis_act.append(float(np.format_float_scientific(CommodityEmissionFactor.ef[i_comm] /
Efficiency.efficiency[i_tech], 2)))
            emis_act_units.append(CommodityEmissionFactor.emis_unit[i_comm])
            emis_act_notes.append('')

EmissionActivity_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "emis_comm": pd.Series(emis_comm, dtype='str'),
        "input_comm": pd.Series(input_comm, dtype='str'),
        "tech": pd.Series(tech, dtype='str'),
        "vintage": pd.Series(vintage, dtype='int'),
        "output_comm": pd.Series(output_comm, dtype='str'),
        "emis_act": pd.Series(emis_act, dtype='float'),
        "emis_act_units": pd.Series(emis_act_units, dtype='str'),
        "emis_act_notes": pd.Series(emis_act_notes, dtype='str')
    }
)
```

```
if tosql_set['EmissionActivity']:
    EmissionActivity_DF.to_sql("EmissionActivity", conn, index=False, if_exists='replace')

if print_set['EmissionActivity']:
    pd.set_option('display.max_rows', len(EmissionActivity_DF))
    pd.set_option('display.max_columns', len(EmissionActivity_DF))
    print("\nEmissionActivity DataFrame\n\n", EmissionActivity_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     EmissionActivity updated...')



# CostInvest

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
CostInvest = pd.read_sql("select * from CostInvest", conn)

regions = list()
tech = list()
vintage = list()
cost_invest = list()
cost_invest_units = list()
cost_invest_notes = list()

tech_already_considered=list()
for i_tech in range(0, len(CostInvest.tech)):
    tech_i = CostInvest.tech[i_tech]

    flag_check = 0
    for check in range(0, len(tech_already_considered)):
        if tech_i == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(CostInvest.tech)):
            if CostInvest.tech[j_tech] == tech_i:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i)

        if flag == 0:  # No other values

            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= CostInvest.vintage[i_tech] and
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:
                    regions.append(CostInvest.regions[i_tech])
                    tech.append(CostInvest.tech[i_tech])
                    vintage.append(int(time_periods.t_periods[i_year]))

cost_invest.append(float(np.format_float_scientific(CostInvest.cost_invest[i_tech], 2)))
                    cost_invest_units.append(CostInvest.cost_invest_units[i_tech])
                    cost_invest_notes.append(CostInvest.cost_invest_notes[i_tech])

        else:
            for i_location in range(0, len(location)-1):
                year1 = CostInvest.vintage[location[i_location]]
                year2 = CostInvest.vintage[location[i_location+1]]
                cost1 = CostInvest.cost_invest[location[i_location]]
                cost2 = CostInvest.cost_invest[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        regions.append(CostInvest.regions[i_tech])
                        tech.append(CostInvest.tech[i_tech])
                        vintage.append(int(year))
                        cost_invest.append(float(np.format_float_scientific(cost1 + (year-
year1)/(year2-year1)*(cost2-cost1), 2)))
                        cost_invest_units.append(CostInvest.cost_invest_units[i_tech])
                        cost_invest_notes.append(CostInvest.cost_invest_notes[i_tech])

            year_last = CostInvest.vintage[location[i_location+1]]
            cost = CostInvest.cost_invest[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-1]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        regions.append(CostInvest.regions[i_tech])
                        tech.append(CostInvest.tech[i_tech])
                        vintage.append(int(year))

cost_invest.append(float(np.format_float_scientific(CostInvest.cost_invest[location[i_loca
tion + 1]], 2)))
                        cost_invest_units.append(CostInvest.cost_invest_units[i_tech])
                        cost_invest_notes.append(CostInvest.cost_invest_notes[i_tech])
            else:
                regions.append(CostInvest.regions[i_tech])
                tech.append(CostInvest.tech[i_tech])
                vintage.append(int(year_last))

cost_invest.append(float(np.format_float_scientific(CostInvest.cost_invest[location[i_loca
tion + 1]], 2)))
                cost_invest_units.append(CostInvest.cost_invest_units[i_tech])
                cost_invest_notes.append(CostInvest.cost_invest_notes[i_tech])

CostInvest_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "tech": pd.Series(tech, dtype='str'),
        "vintage": pd.Series(vintage, dtype='int'),
        "cost_invest": pd.Series(cost_invest, dtype='float'),
        "cost_invest_units": pd.Series(cost_invest_units, dtype='str'),
        "cost_invest_notes": pd.Series(cost_invest_notes, dtype='str')
    }
)

if tosql_set['CostInvest']:
    CostInvest_DF.to_sql("CostInvest", conn, index=False, if_exists='replace')

if print_set['CostInvest']:
    pd.set_option('display.max_rows', len(CostInvest_DF))
    pd.set_option('display.max_columns', len(CostInvest_DF))
    print("\nCostInvest DataFrame\n\n", CostInvest_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     CostInvest updated...')



# CostFixed

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
LifetimeTech = pd.read_sql("select * from LifetimeTech", conn)
LifetimeProcess = pd.read_sql("select * from LifetimeProcess", conn)
CostFixed = pd.read_sql("select * from CostFixed", conn)

regions = list()
periods = list()
```

```
tech = list()
vintage = list()
cost_fixed = list()
cost_fixed_units = list()
cost_fixed_notes = list()

tech_already_considered = list()
for i_tech in range(0, len(CostFixed.tech)):
    tech_i = CostFixed.tech[i_tech]

    flag_check = 0
    for check in range(0, len(tech_already_considered)):
        if tech_i == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        lifetime = 0
        year_lifetime = list()
        lifetime_process = list()
        for i_life in range(0, len(LifetimeTech.life)):
            if LifetimeTech.tech[i_life] == tech_i:
                lifetime = LifetimeTech.life[i_life]
        if lifetime == 0:
            for i_life in range(0, len(LifetimeProcess.life_process)):
                if LifetimeProcess.tech[i_life] == tech_i:
                    year_lifetime.append(LifetimeProcess.vintage[i_life])
                    lifetime_process.append(LifetimeProcess.life_process[i_life])
                else:
                    lifetime = lifetime_default

        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(CostFixed.tech)):
            if CostFixed.tech[j_tech] == tech_i:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i)

        #lifetime_save = lifetime

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= CostFixed.vintage[i_tech] and
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:

                    #lifetime = lifetime_save
                    year_vintage = time_periods.t_periods[i_year]
                    for i in range(0, len(year_lifetime)):
                        if year_vintage == year_lifetime[i]:
                            lifetime = lifetime_process[i]
                    start = year_vintage
                    stop = year_vintage + lifetime

                    for j_year in range(0, len(time_periods)):
                        year_periods = time_periods.t_periods[j_year]
                        if start <= year_periods < stop and year_periods !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                            regions.append(CostFixed.regions[i_tech])
                            periods.append(int(year_periods))
                            tech.append(CostFixed.tech[i_tech])
                            vintage.append(int(year_vintage))

cost_fixed.append(float(np.format_float_scientific(CostFixed.cost_fixed[i_tech], 2)))
                            cost_fixed_units.append(CostFixed.cost_fixed_units[i_tech])
                            cost_fixed_notes.append(CostFixed.cost_fixed_notes[i_tech])

        else:
            year_list = list()
            cost_list = list()
            for i_location in range(0, len(location)-1):
                year1 = CostFixed.periods[location[i_location]]
                year2 = CostFixed.periods[location[i_location+1]]
                cost1 = CostFixed.cost_fixed[location[i_location]]
                cost2 = CostFixed.cost_fixed[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        year_list.append(year)
                        cost_list.append(cost1 + (year-year1)/(year2-year1)*(cost2-cost1))

            year_last = CostFixed.vintage[location[i_location+1]]
            cost_last = CostFixed.cost_fixed[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-2]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        year_list.append(year)
                        cost_list.append(cost_last)
            else:
                year_list.append(year_last)
                cost_list.append(cost_last)

            for i_year in range(0, len(year_list)):
                if year_list[i_year] >= CostFixed.vintage[i_tech]:

                    #lifetime = lifetime_save
                    year_vintage = year_list[i_year]
                    for i in range(0, len(year_lifetime)):
                        if year_vintage == year_lifetime[i]:
                            lifetime = lifetime_process[i]
                    start = year_vintage
                    stop = year_vintage + lifetime

                    for j_year in range(0, len(year_list)):
                        year_periods = year_list[j_year]
                        if start <= year_periods < stop and year_periods !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                            regions.append(CostFixed.regions[i_tech])
                            periods.append(int(year_periods))
                            tech.append(CostFixed.tech[i_tech])
                            vintage.append(int(year_vintage))

cost_fixed.append(float(np.format_float_scientific(cost_list[j_year], 2)))
                            cost_fixed_units.append(CostFixed.cost_fixed_units[i_tech])
                            cost_fixed_notes.append(CostFixed.cost_fixed_notes[i_tech])

CostFixed_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "periods": pd.Series(periods, dtype='int'),
        "tech": pd.Series(tech, dtype='str'),
        "vintage": pd.Series(vintage, dtype='int'),
        "cost_fixed": pd.Series(cost_fixed, dtype='float'),
        "cost_fixed_units": pd.Series(cost_fixed_units, dtype='str'),
        "cost_fixed_notes": pd.Series(cost_fixed_notes, dtype='str')
    }
)

if tosql_set['CostFixed']:
    CostFixed_DF.to_sql("CostFixed", conn, index=False, if_exists='replace')

if print_set['CostFixed']:
    pd.set_option('display.max_rows', len(CostFixed_DF))
    pd.set_option('display.max_columns', len(CostFixed_DF))
    print("\nCostFixed DataFrame\n\n", CostFixed_DF)
    pd.reset_option('display.max_rows')
```

```
conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     CostFixed updated...')



# CostVariable

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
LifetimeTech = pd.read_sql("select * from LifetimeTech", conn)
LifetimeProcess = pd.read_sql("select * from LifetimeProcess", conn)
CostVariable = pd.read_sql("select * from CostVariable", conn)

regions = list()
periods = list()
tech = list()
vintage = list()
cost_variable = list()
cost_variable_units = list()
cost_variable_notes = list()

tech_already_considered = list()
for i_tech in range(0, len(CostVariable.tech)):
    tech_i = CostVariable.tech[i_tech]

    flag_check = 0
    for check in range(0, len(tech_already_considered)):
        if tech_i == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        lifetime = 0
        year_lifetime = list()
        lifetime_process = list()
        for i_life in range(0, len(LifetimeTech.life)):
            if LifetimeTech.tech[i_life] == tech_i:
                lifetime = float(LifetimeTech.life[i_life])
        if lifetime == 0:
            for i_life in range(0, len(LifetimeProcess.life_process)):
                if LifetimeProcess.tech[i_life] == tech_i:
                    year_lifetime.append(float(LifetimeProcess.vintage[i_life]))
                    lifetime_process.append(float(LifetimeProcess.life_process[i_life]))
                else:
                    lifetime = lifetime_default

        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(CostVariable.tech)):
            if CostVariable.tech[j_tech] == tech_i:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i)

        #lifetime_save = lifetime

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= CostVariable.vintage[i_tech] and
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:

                    #lifetime = lifetime_save
                    year_vintage = float(time_periods.t_periods[i_year])
                    for i in range(0, len(year_lifetime)):
                        if year_vintage == year_lifetime[i]:
                            lifetime = float(lifetime_process[i])
                    start = float(year_vintage)
                    stop = float(year_vintage) + lifetime

                    for j_year in range(0, len(time_periods)):
                        year_periods = time_periods.t_periods[j_year]
                        if start <= year_periods < stop and year_periods !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                            regions.append(CostVariable.regions[i_tech])
                            periods.append(int(year_periods))
                            tech.append(CostVariable.tech[i_tech])
                            vintage.append(int(year_vintage))

cost_variable.append(float(np.format_float_scientific(CostVariable.cost_variable[i_tech],
2)))

cost_variable_units.append(CostVariable.cost_variable_units[i_tech])

cost_variable_notes.append(CostVariable.cost_variable_notes[i_tech])

        else:
            year_list = list()
            cost_list = list()
            for i_location in range(0, len(location)-1):
                year1 = float(CostVariable.periods[location[i_location]])
                year2 = float(CostVariable.periods[location[i_location+1]])
                cost1 = float(CostVariable.cost_variable[location[i_location]])
                cost2 = float(CostVariable.cost_variable[location[i_location+1]])

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        year_list.append(year)
                        cost_list.append(cost1 + (year-year1)/(year2-year1)*(cost2-cost1))

            year_last = float(CostVariable.vintage[location[i_location+1]])
            cost_last = float(CostVariable.cost_variable[location[i_location+1]])
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-2]:
#different by 2050
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        year_list.append(year)
                        cost_list.append(cost_last)
            else: # if year_last=2050
                year_list.append(year_last)
                cost_list.append(cost_last)

            for i_year in range(0, len(year_list)):
                if float(year_list[i_year]) >= float(CostVariable.vintage[i_tech]):

                    #lifetime = lifetime_save
                    year_vintage = year_list[i_year]
                    for i in range(0, len(year_lifetime)):
                        if year_vintage == year_lifetime[i]:
                            lifetime = float(lifetime_process[i])
                    start = float(year_vintage)
                    stop = float(year_vintage) + lifetime
                    for j_year in range(0, len(year_list)):
                        year_periods = year_list[j_year]
                        if start <= year_periods < stop and year_periods !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                            regions.append(CostVariable.regions[i_tech])
                            periods.append(int(year_periods))
                            tech.append(CostVariable.tech[i_tech])
                            vintage.append(int(year_vintage))

cost_variable.append(float(np.format_float_scientific(cost_list[j_year], 2)))

cost_variable_units.append(CostVariable.cost_variable_units[i_tech])
```

```
cost_variable_notes.append(CostVariable.cost_variable_notes[i_tech])

CostVariable_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "periods": pd.Series(periods, dtype='int'),
        "tech": pd.Series(tech, dtype='str'),
        "vintage": pd.Series(vintage, dtype='int'),
        "cost_variable": pd.Series(cost_variable, dtype='float'),
        "cost_variable_units": pd.Series(cost_variable_units, dtype='str'),
        "cost_variable_notes": pd.Series(cost_variable_notes, dtype='str')
    }
)

if tosql_set['CostVariable']:
    CostVariable_DF.to_sql("CostVariable", conn, index=False, if_exists='replace')

if print_set['CostVariable']:
    pd.set_option('display.max_rows', len(CostVariable_DF))
    pd.set_option('display.max_columns', len(CostVariable_DF))
    print("\nCostVariable DataFrame\n\n", CostVariable_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     CostVariable updated...')



# MinCapacity

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
MinCapacity = pd.read_sql("select * from MinCapacity", conn)

regions = list()
periods = list()
tech = list()
mincap = list()
mincap_units = list()
mincap_notes = list()

tech_already_considered=list()
for i_tech in range(0, len(MinCapacity.tech)):
    tech_i = MinCapacity.tech[i_tech]

    flag_check = 0
    tech_i_check = tech_i
    for check in range(0, len(tech_already_considered)):
        if tech_i_check == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(MinCapacity.tech)):
            tech_j_check = MinCapacity.tech[j_tech]
            if tech_j_check == tech_i_check:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i_check)

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= MinCapacity.periods[i_tech] and
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:
                    regions.append(MinCapacity.regions[i_tech])
                    periods.append(int(time_periods.t_periods[i_year]))
                    tech.append(MinCapacity.tech[i_tech])

mincap.append(float(np.format_float_scientific(MinCapacity.mincap[i_tech], 2)))
                    mincap_units.append(MinCapacity.mincap_units[i_tech])
                    mincap_notes.append(MinCapacity.mincap_notes[i_tech])

        else:
            for i_location in range(0, len(location)-1):
                year1 = MinCapacity.periods[location[i_location]]
                year2 = MinCapacity.periods[location[i_location+1]]
                min_cap1 = MinCapacity.mincap[location[i_location]]
                min_cap2 = MinCapacity.mincap[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        regions.append(MinCapacity.regions[i_tech])
                        periods.append(int(year))
                        tech.append(MinCapacity.tech[i_tech])
                        mincap.append(float(np.format_float_scientific(min_cap1 + (year-
year1)/(year2-year1)*(min_cap2-min_cap1), 2)))
                        mincap_units.append(MinCapacity.mincap_units[i_tech])
                        mincap_notes.append(MinCapacity.mincap_notes[i_tech])

            year_last = MinCapacity.periods[location[i_location+1]]
            min_cap = MinCapacity.mincap[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-1]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        regions.append(MinCapacity.regions[i_tech])
                        periods.append(int(year))
                        tech.append(MinCapacity.tech[i_tech])

mincap.append(float(np.format_float_scientific(MinCapacity.mincap[location[i_location +
1]], 2)))
                        mincap_units.append(MinCapacity.mincap_units[i_tech])
                        mincap_notes.append(MinCapacity.mincap_notes[i_tech])
#            else:
#                regions.append(MinCapacity.regions[i_tech])
#                periods.append(int(year_last))
#                tech.append(MinCapacity.tech[i_tech])
#
mincap.append(float(np.format_float_scientific(MinCapacity.mincap[location[i_location +
1]], 2)))
#                mincap_units.append(MinCapacity.mincap_units[i_tech])
#                mincap_notes.append(MinCapacity.mincap_notes[i_tech])

MinCapacity_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "periods": pd.Series(periods, dtype='int'),
        "tech": pd.Series(tech, dtype='str'),
        "mincap": pd.Series(mincap, dtype='float'),
        "mincap_units": pd.Series(mincap_units, dtype='str'),
        "mincap_notes": pd.Series(mincap_notes, dtype='str')
    }
)

if tosql_set['MinCapacity']:
    MinCapacity_DF.to_sql("MinCapacity", conn, index=False, if_exists='replace')

if print_set['MinCapacity']:
    pd.set_option('display.max_rows', len(MinCapacity_DF))
    pd.set_option('display.max_columns', len(MinCapacity_DF))
    print("\nMinCapacity DataFrame\n\n", MinCapacity_DF)
    pd.reset_option('display.max_rows')
```

```
conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     MinCapacity updated...')



# MinActivity

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
MinActivity = pd.read_sql("select * from MinActivity", conn)

regions = list()
periods = list()
tech = list()
minact = list()
minact_units = list()
minact_notes = list()

tech_already_considered=list()
for i_tech in range(0, len(MinActivity.tech)):
    tech_i = MinActivity.tech[i_tech]

    flag_check = 0
    tech_i_check = tech_i
    for check in range(0, len(tech_already_considered)):
        if tech_i_check == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(MinActivity.tech)):
            tech_j_check = MinActivity.tech[j_tech]
            if tech_j_check == tech_i_check:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i_check)

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= MinActivity.periods[i_tech] and
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:
                    regions.append(MinActivity.regions[i_tech])
                    periods.append(int(time_periods.t_periods[i_year]))
                    tech.append(MinActivity.tech[i_tech])

minact.append(float(np.format_float_scientific(MinActivity.minact[i_tech], 2)))
                    minact_units.append(MinActivity.minact_units[i_tech])
                    minact_notes.append(MinActivity.minact_notes[i_tech])

        else:
            for i_location in range(0, len(location)-1):
                year1 = MinActivity.periods[location[i_location]]
                year2 = MinActivity.periods[location[i_location+1]]
                min_act1 = MinActivity.minact[location[i_location]]
                min_act2 = MinActivity.minact[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        regions.append(MinActivity.regions[i_tech])
                        periods.append(int(year))
                        tech.append(MinActivity.tech[i_tech])
                        minact.append(float(np.format_float_scientific(min_act1 + (year-
year1)/(year2-year1)*(min_act2-min_act1), 2)))
                        minact_units.append(MinActivity.minact_units[i_tech])
                        minact_notes.append(MinActivity.minact_notes[i_tech])
            year_last = MinActivity.periods[location[i_location+1]]
            min_act = MinActivity.minact[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-1]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year != 
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        regions.append(MinActivity.regions[i_tech])
                        periods.append(year)
                        tech.append(MinActivity.tech[i_tech])

minact.append(float(np.format_float_scientific(MinActivity.minact[location[i_location + 
1]], 2)))
                        minact_units.append(MinActivity.minact_units[i_tech])
                        minact_notes.append(MinActivity.minact_notes[i_tech])
#            else:
#                regions.append(MinActivity.regions[i_tech])
#                periods.append(int(year_last))
#                tech.append(MinActivity.tech[i_tech])
#
minact.append(float(np.format_float_scientific(MinActivity.minact[location[i_location + 
1]], 2)))
#                minact_units.append(MinActivity.minact_units[i_tech])
#                minact_notes.append(MinActivity.minact_notes[i_tech])

MinActivity_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "periods": pd.Series(periods, dtype='int'),
        "tech": pd.Series(tech, dtype='str'),
        "minact": pd.Series(minact, dtype='float'),
        "minact_units": pd.Series(minact_units, dtype='str'),
        "minact_notes": pd.Series(minact_notes, dtype='str')
    }
)

if tosql_set['MinActivity']:
    MinActivity_DF.to_sql("MinActivity", conn, index=False, if_exists='replace')

if print_set['MinActivity']:
    pd.set_option('display.max_rows', len(MinActivity_DF))
    pd.set_option('display.max_columns', len(MinActivity_DF))
    print("\nMinActivity DataFrame\n\n", MinActivity_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     MinActivity updated...')



# MaxCapacity

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
MaxCapacity = pd.read_sql("select * from MaxCapacity", conn)

regions = list()
periods = list()
tech = list()
maxcap = list()
maxcap_units = list()
maxcap_notes = list()

tech_already_considered=list()
for i_tech in range(0, len(MaxCapacity.tech)):
    tech_i = MaxCapacity.tech[i_tech]

    flag_check = 0
    tech_i_check = tech_i
    for check in range(0, len(tech_already_considered)):
        if tech_i_check == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(MaxCapacity.tech)):
            tech_j_check = MaxCapacity.tech[j_tech]
            if tech_j_check == tech_i_check:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i_check)

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= MaxCapacity.periods[i_tech] and
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:
                    regions.append(MaxCapacity.regions[i_tech])
                    periods.append(int(time_periods.t_periods[i_year]))
                    tech.append(MaxCapacity.tech[i_tech])

maxcap.append(float(np.format_float_scientific(MaxCapacity.maxcap[i_tech], 2)))
                    maxcap_units.append(MaxCapacity.maxcap_units[i_tech])
                    maxcap_notes.append(MaxCapacity.maxcap_notes[i_tech])

        else:
            for i_location in range(0, len(location)-1):
                year1 = MaxCapacity.periods[location[i_location]]
                year2 = MaxCapacity.periods[location[i_location+1]]
                max_cap1 = MaxCapacity.maxcap[location[i_location]]
                max_cap2 = MaxCapacity.maxcap[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        regions.append(MaxCapacity.regions[i_tech])
                        periods.append(int(year))
                        tech.append(MaxCapacity.tech[i_tech])
                        maxcap.append(float(np.format_float_scientific(max_cap1 + (year-
year1)/(year2-year1)*(max_cap2-max_cap1), 2)))
                        maxcap_units.append(MaxCapacity.maxcap_units[i_tech])
                        maxcap_notes.append(MaxCapacity.maxcap_notes[i_tech])

            year_last = MaxCapacity.periods[location[i_location+1]]
            max_cap = MaxCapacity.maxcap[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-1]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        regions.append(MaxCapacity.regions[i_tech])
                        periods.append(int(year))
                        tech.append(MaxCapacity.tech[i_tech])

maxcap.append(float(np.format_float_scientific(MaxCapacity.maxcap[location[i_location +
1]], 2)))
                        maxcap_units.append(MaxCapacity.maxcap_units[i_tech])
                        maxcap_notes.append(MaxCapacity.maxcap_notes[i_tech])
#            else:
#                regions.append(MaxCapacity.regions[i_tech])
#                periods.append(int(year_last))
#                tech.append(MaxCapacity.tech[i_tech])
#
maxcap.append(float(np.format_float_scientific(MaxCapacity.maxcap[location[i_location +
1]], 2)))
#                maxcap_units.append(MaxCapacity.maxcap_units[i_tech])
#                maxcap_notes.append(MaxCapacity.maxcap_notes[i_tech])

MaxCapacity_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "periods": pd.Series(periods, dtype='int'),
        "tech": pd.Series(tech, dtype='str'),
        "maxcap": pd.Series(maxcap, dtype='float'),
        "maxcap_units": pd.Series(maxcap_units, dtype='str'),
        "maxcap_notes": pd.Series(maxcap_notes, dtype='str')
    }
)

if tosql_set['MaxCapacity']:
    MaxCapacity_DF.to_sql("MaxCapacity", conn, index=False, if_exists='replace')

if print_set['MaxCapacity']:
    pd.set_option('display.max_rows', len(MaxCapacity_DF))
    pd.set_option('display.max_columns', len(MaxCapacity_DF))
    print("\nMaxCapacity DataFrame\n\n", MaxCapacity_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     MaxCapacity updated...')



# MaxActivity

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
MaxActivity = pd.read_sql("select * from MaxActivity", conn)

regions = list()
periods = list()
tech = list()
maxact = list()
maxact_units = list()
maxact_notes = list()

tech_already_considered=list()
for i_tech in range(0, len(MaxActivity.tech)):
    tech_i = MaxActivity.tech[i_tech]

    flag_check = 0
    tech_i_check = tech_i
    for check in range(0, len(tech_already_considered)):
        if tech_i_check == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(MaxActivity.tech)):
            tech_j_check = MaxActivity.tech[j_tech]
            if tech_j_check == tech_i_check:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i_check)

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= MaxActivity.periods[i_tech] and
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:
                    regions.append(MaxActivity.regions[i_tech])
                    periods.append(int(time_periods.t_periods[i_year]))
                    tech.append(MaxActivity.tech[i_tech])
```

```
maxact.append(float(np.format_float_scientific(MaxActivity.maxact[i_tech], 2)))
                    maxact_units.append(MaxActivity.maxact_units[i_tech])
                    maxact_notes.append(MaxActivity.maxact_notes[i_tech])

        else:
            for i_location in range(0, len(location)-1):
                year1 = MaxActivity.periods[location[i_location]]
                year2 = MaxActivity.periods[location[i_location+1]]
                max_act1 = MaxActivity.maxact[location[i_location]]
                max_act2 = MaxActivity.maxact[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        regions.append(MaxActivity.regions[i_tech])
                        periods.append(int(year))
                        tech.append(MaxActivity.tech[i_tech])
                        maxact.append(float(np.format_float_scientific(max_act1 + (year-
year1)/(year2-year1)*(max_act2-max_act1), 2)))
                        maxact_units.append(MaxActivity.maxact_units[i_tech])
                        maxact_notes.append(MaxActivity.maxact_notes[i_tech])

            year_last = MaxActivity.periods[location[i_location+1]]
            max_act = MaxActivity.maxact[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-1]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        regions.append(MaxActivity.regions[i_tech])
                        periods.append(int(year))
                        tech.append(MaxActivity.tech[i_tech])

maxact.append(float(np.format_float_scientific(MaxActivity.maxact[location[i_location +
1]], 2)))
                        maxact_units.append(MaxActivity.maxact_units[i_tech])
                        maxact_notes.append(MaxActivity.maxact_notes[i_tech])
#             else:
#                 regions.append(MaxActivity.regions[i_tech])
#                 periods.append(int(year_last))
#                 tech.append(MaxActivity.tech[i_tech])
#
maxact.append(float(np.format_float_scientific(MaxActivity.maxact[location[i_location +
1]], 2)))
#                 maxact_units.append(MaxActivity.maxact_units[i_tech])
#                 maxact_notes.append(MaxActivity.maxact_notes[i_tech])

MaxActivity_DF = pd.DataFrame(np.transpose([regions, periods, tech, maxact, maxact_units,
maxact_notes]),
                             columns=["regions", "periods", "tech", "maxact",
"maxact_units", "maxact_notes"]);

MaxActivity_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "periods": pd.Series(periods, dtype='int'),
        "tech": pd.Series(tech, dtype='str'),
        "maxact": pd.Series(maxact, dtype='float'),
        "maxact_units": pd.Series(maxact_units, dtype='str'),
        "maxact_notes": pd.Series(maxact_notes, dtype='str')
    }
)

if tosql_set['MaxActivity']:
    MaxActivity_DF.to_sql("MaxActivity", conn, index=False, if_exists='replace')

if print_set['MaxActivity']:
    pd.set_option('display.max_rows', len(MaxActivity_DF))
    pd.set_option('display.max_columns', len(MaxActivity_DF))
    print("\nMaxActivity DataFrame\n\n", MaxActivity_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     MaxActivity updated...')



# AvailabilityFactor

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
AvailabilityFactor = pd.read_sql("select * from AvailabilityFactor", conn)

regions = list()
tech = list()
vintage = list()
af = list()
af_notes = list()

tech_already_considered=list()
for i_tech in range(0, len(AvailabilityFactor.tech)):
    tech_i = AvailabilityFactor.tech[i_tech]

    flag_check = 0
    tech_i_check = tech_i
    for check in range(0, len(tech_already_considered)):
        if tech_i_check == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(AvailabilityFactor.tech)):
            tech_j_check = AvailabilityFactor.tech[j_tech]
            if tech_j_check == tech_i_check:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i_check)

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= AvailabilityFactor.vintage[i_tech]
and time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-
1]:
                    regions.append(AvailabilityFactor.regions[i_tech])
                    tech.append(AvailabilityFactor.tech[i_tech])
                    vintage.append(int(time_periods.t_periods[i_year]))

af.append(float(np.format_float_scientific(AvailabilityFactor.af[i_tech], 2)))
                    af_notes.append(AvailabilityFactor.af_notes[i_tech])

        else:
            for i_location in range(0, len(location)-1):
                year1 = AvailabilityFactor.vintage[location[i_location]]
                year2 = AvailabilityFactor.vintage[location[i_location+1]]
                af1 = AvailabilityFactor.af[location[i_location]]
                af2 = AvailabilityFactor.af[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        regions.append(AvailabilityFactor.regions[i_tech])
                        tech.append(AvailabilityFactor.tech[i_tech])
                        vintage.append(int(year))
                        af.append(float(np.format_float_scientific(af1 + (year-
year1)/(year2-year1)*(af2-af1), 2)))
```

```
                        af_notes.append(AvailabilityFactor.af_notes[i_tech])

            year_last = AvailabilityFactor.vintage[location[i_location+1]]
            eff = AvailabilityFactor.af[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-1]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year != 
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        regions.append(AvailabilityFactor.regions[i_tech])
                        tech.append(AvailabilityFactor.tech[i_tech])
                        vintage.append(int(year))

af.append(float(np.format_float_scientific(AvailabilityFactor.af[location[i_location + 
1]], 2)))
                        af_notes.append(AvailabilityFactor.af_notes[i_tech])
            else:
                regions.append(AvailabilityFactor.regions[i_tech])
                tech.append(AvailabilityFactor.tech[i_tech])
                vintage.append(int(year_last))

af.append(float(np.format_float_scientific(AvailabilityFactor.af[location[i_location + 
1]], 2)))
                af_notes.append(AvailabilityFactor.af_notes[i_tech])

AvailabilityFactor_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "tech": pd.Series(tech, dtype='str'),
        "vintage": pd.Series(vintage, dtype='int'),
        "af": pd.Series(af, dtype='float'),
        "af_notes": pd.Series(af_notes, dtype='str')
    }
)

if tosql_set['AvailabilityFactor']:
    AvailabilityFactor_DF.to_sql("AvailabilityFactor", conn, index=False, 
if_exists='replace')

if print_set['AvailabilityFactor']:
    pd.set_option('display.max_rows', len(AvailabilityFactor_DF))
    pd.set_option('display.max_columns', len(AvailabilityFactor_DF))
    print("\nAvailabilityFactor DataFrame\n\n", AvailabilityFactor_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     AvailabilityFactor updated...')



# Demand

conn = sqlite3.connect(database_name)
Allocation = pd.read_sql("select * from Allocation", conn)
Demand = pd.read_sql("select * from Demand", conn)
Driver = pd.read_sql("select * from Driver", conn)
Elasticity = pd.read_sql("select * from Elasticity", conn)
regions=list()
periods=list()
demand_comm=list()
demand=list()
demand_units=list()
demand_notes=list()

for i in range(0, len(Demand.demand_comm)):
    regions.append(Demand.regions[i])
    periods.append(int(Demand.periods[i]))
    demand_comm.append(Demand.demand_comm[i])
    demand.append(Demand.demand[i])
    demand_units.append(Demand.demand_units[i])
    demand_notes.append(Demand.demand_notes[i])
    for j in range(0, len(Allocation.demand_comm)):
        if Allocation.demand_comm[j] == Demand.demand_comm[i]:
            for k in range(0, len(Driver.periods)):
                for l in range(0, len(Elasticity.periods)):
                    if Driver.driver_name[k] == Allocation.driver_name[j] and
Elasticity.demand_comm[l] == Demand.demand_comm[i] and Driver.periods[k] ==
Elasticity.periods[l]:
                        regions.append(Elasticity.regions[l])
                        periods.append(int(Elasticity.periods[l]))
                        demand_comm.append(Elasticity.demand_comm[l])
                        if not Driver.periods[k] == 2006:

demand.append(float(np.format_float_scientific(demand[len(demand)-
1]*(1+(Driver.driver[k]/Driver.driver[k-1]-1)*Elasticity.elasticity[l]), 2)))
                            demand_units.append(demand_units[len(demand_units)-1])
                        demand_notes.append('')

Demand_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "periods": pd.Series(periods, dtype='int'),
        "demand_comm": pd.Series(demand_comm, dtype='str'),
        "demand": pd.Series(demand, dtype='float'),
        "demand_units": pd.Series(demand_units, dtype='str'),
        "demand_notes": pd.Series(demand_notes, dtype='str')
    }
)

for i in range(0, len(Demand_DF)):
    if Demand_DF.loc[i, lambda df: "periods"] == 2006:
        Demand_DF = Demand_DF.drop(index = [i])

if tosql_set['Demand']:
    Demand_DF.to_sql("Demand", conn, index=False, if_exists='replace')

if print_set['Demand']:
    pd.set_option('display.max_rows', len(Demand_DF))
    pd.set_option('display.max_columns', len(Demand_DF))
    print("\nDemand DataFrame\n\n", Demand_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     Demand updated...')



# CapacityFactorProcess

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
CapacityFactorProcess = pd.read_sql("select * from CapacityFactorProcess", conn)

regions = list()
season_name = list()
tech = list()
vintage = list()
time_of_day_name = list()
cf_process = list()
cf_process_notes = list()

tech_already_considered=list()
for i_tech in range(0, len(CapacityFactorProcess.tech)):
    tech_i = CapacityFactorProcess.tech[i_tech]

    flag_check = 0
    tech_i_check = CapacityFactorProcess.season_name[i_tech] + tech_i +
CapacityFactorProcess.time_of_day_name[i_tech]
```

```
    for check in range(0, len(tech_already_considered)):
        if tech_i_check == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(CapacityFactorProcess.tech)):
            tech_j_check = CapacityFactorProcess.season_name[j_tech] +
CapacityFactorProcess.tech[j_tech] + CapacityFactorProcess.time_of_day_name[j_tech]
            if tech_j_check == tech_i_check:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i_check)

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= CapacityFactorProcess.vintage[i_tech]
and time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-
1]:
                    regions.append(CapacityFactorProcess.regions[i_tech])
                    season_name.append(CapacityFactorProcess.season_name[i_tech])

time_of_day_name.append(CapacityFactorProcess.time_of_day_name[i_tech])
                    tech.append(CapacityFactorProcess.tech[i_tech])
                    vintage.append(int(time_periods.t_periods[i_year]))

cf_process.append(float(np.format_float_scientific(CapacityFactorProcess.cf_process[i_tech
], 2)))

cf_process_notes.append(CapacityFactorProcess.cf_process_notes[i_tech])

        else:
            for i_location in range(0, len(location)-1):
                year1 = CapacityFactorProcess.vintage[location[i_location]]
                year2 = CapacityFactorProcess.vintage[location[i_location+1]]
                cf1 = CapacityFactorProcess.cf_process[location[i_location]]
                cf2 = CapacityFactorProcess.cf_process[location[i_location+1]]

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        regions.append(CapacityFactorProcess.regions[i_tech])
                        season_name.append(CapacityFactorProcess.season_name[i_tech])

time_of_day_name.append(CapacityFactorProcess.time_of_day_name[i_tech])
                        tech.append(CapacityFactorProcess.tech[i_tech])
                        vintage.append(int(year))
                        cf_process.append(float(np.format_float_scientific(cf1 + (year-
year1)/(year2-year1)*(cf2-cf1), 2)))

cf_process_notes.append(CapacityFactorProcess.cf_process_notes[i_tech])

            year_last = CapacityFactorProcess.vintage[location[i_location+1]]
            eff = CapacityFactorProcess.cf_process[location[i_location+1]]
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-1]:
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        regions.append(CapacityFactorProcess.regions[i_tech])
                        season_name.append(CapacityFactorProcess.season_name[i_tech])

time_of_day_name.append(CapacityFactorProcess.time_of_day_name[i_tech])
                        tech.append(CapacityFactorProcess.tech[i_tech])
                        vintage.append(int(year))
```

```
cf_process.append(float(np.format_float_scientific(CapacityFactorProcess.cf_process[locati
on[i_location + 1]], 2)))

cf_process_notes.append(CapacityFactorProcess.cf_process_notes[i_tech])
            else:
                regions.append(CapacityFactorProcess.regions[i_tech])
                season_name.append(CapacityFactorProcess.season_name[i_tech])
                time_of_day_name.append(CapacityFactorProcess.time_of_day_name[i_tech])
                tech.append(CapacityFactorProcess.tech[i_tech])
                vintage.append(int(year_last))

cf_process.append(float(np.format_float_scientific(CapacityFactorProcess.cf_process[locati
on[i_location + 1]], 2)))
                cf_process_notes.append(CapacityFactorProcess.cf_process_notes[i_tech])

CapacityFactorProcess_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "season_name": pd.Series(season_name, dtype='str'),
        "time_of_day_name": pd.Series(time_of_day_name, dtype='str'),
        "tech": pd.Series(tech, dtype='str'),
        "vintage": pd.Series(vintage, dtype='int'),
        "cf_process": pd.Series(cf_process, dtype='float'),
        "cf_process_notes": pd.Series(cf_process_notes, dtype='str')
    }
)

if tosql_set['CapacityFactorProcess']:
    CapacityFactorProcess_DF.to_sql("CapacityFactorProcess", conn, index=False,
if_exists='replace')

if print_set['CapacityFactorProcess']:
    pd.set_option('display.max_rows', len(CapacityFactorProcess_DF))
    pd.set_option('display.max_columns', len(CapacityFactorProcess_DF))
    print("\nCapacityFactorProcess DataFrame\n\n", CapacityFactorProcess_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     CapacityFactorProcess updated...')



# CapacityCredit

conn = sqlite3.connect(database_name)
time_periods = pd.read_sql("select * from time_periods", conn)
LifetimeTech = pd.read_sql("select * from LifetimeTech", conn)
LifetimeProcess = pd.read_sql("select * from LifetimeProcess", conn)
CapacityCredit = pd.read_sql("select * from CapacityCredit", conn)

regions = list()
periods = list()
tech = list()
vintage = list()
cf_tech = list()
cf_tech_notes = list()

tech_already_considered = list()
for i_tech in range(0, len(CapacityCredit.tech)):
    tech_i = CapacityCredit.tech[i_tech]

    flag_check = 0
    for check in range(0, len(tech_already_considered)):
        if tech_i == tech_already_considered[check]:
            flag_check = 1

    if flag_check == 0:
        lifetime = 0
        year_lifetime = list()
        lifetime_process = list()
        for i_life in range(0, len(LifetimeTech.life)):
            if LifetimeTech.tech[i_life] == tech_i:
                lifetime = float(LifetimeTech.life[i_life])
        if lifetime == 0:
            for i_life in range(0, len(LifetimeProcess.life_process)):
                if LifetimeProcess.tech[i_life] == tech_i:
                    year_lifetime.append(float(LifetimeProcess.vintage[i_life]))
                    lifetime_process.append(float(LifetimeProcess.life_process[i_life]))
                else:
                    lifetime = lifetime_default

        # Checking if other values are present for the technology
        flag = 0
        location = list()
        location.append(i_tech)
        for j_tech in range(i_tech + 1, len(CapacityCredit.tech)):
            if CapacityCredit.tech[j_tech] == tech_i:
                flag = 1
                location.append(j_tech)
                tech_already_considered.append(tech_i)

        #lifetime_save = lifetime

        if flag == 0:  # No other values
            for i_year in range(0, len(time_periods)):
                if time_periods.t_periods[i_year] >= CapacityCredit.vintage[i_tech] and 
time_periods.t_periods[i_year] != time_periods.t_periods[len(time_periods.t_periods)-1]:

                    #lifetime = lifetime_save
                    year_vintage = float(time_periods.t_periods[i_year])
                    for i in range(0, len(year_lifetime)):
                        if year_vintage == year_lifetime[i]:
                            lifetime = float(lifetime_process[i])
                    start = float(year_vintage)
                    stop = float(year_vintage) + lifetime

                    for j_year in range(0, len(time_periods)):
                        year_periods = time_periods.t_periods[j_year]
                        if start <= year_periods < stop and year_periods != 
time_periods.t_periods[len(time_periods.t_periods)-1]:
                            regions.append(CapacityCredit.regions[i_tech])
                            periods.append(int(year_periods))
                            tech.append(CapacityCredit.tech[i_tech])
                            vintage.append(int(year_vintage))
                            
cf_tech.append(float(np.format_float_scientific(CapacityCredit.cf_tech[i_tech], 2)))
                            cf_tech_notes.append(CapacityCredit.cf_tech_notes[i_tech])

        else:
            year_list = list()
            cost_list = list()
            for i_location in range(0, len(location)-1):
                year1 = float(CapacityCredit.periods[location[i_location]])
                year2 = float(CapacityCredit.periods[location[i_location+1]])
                cost1 = float(CapacityCredit.cf_tech[location[i_location]])
                cost2 = float(CapacityCredit.cf_tech[location[i_location+1]])

                for i_year in range(0, len(time_periods)):
                    year = time_periods.t_periods[i_year]
                    if year1 <= year < year2:
                        year_list.append(year)
                        cost_list.append(cost1 + (year-year1)/(year2-year1)*(cost2-cost1))

            year_last = float(CapacityCredit.vintage[location[i_location+1]])
            cost_last = float(CapacityCredit.cf_tech[location[i_location+1]])
            if year_last != time_periods.t_periods[len(time_periods.t_periods)-2]: 
#different by 2050
                for i_year in range(0, len(time_periods.t_periods)):
                    year = time_periods.t_periods[i_year]
                    if year >= year_last and year !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                        year_list.append(year)
                        cost_list.append(cost_last)
            else: # if year_last=2050
                year_list.append(year_last)
                cost_list.append(cost_last)

            for i_year in range(0, len(year_list)):
                if float(year_list[i_year]) >= float(CapacityCredit.vintage[i_tech]):

                    #lifetime = lifetime_save
                    year_vintage = year_list[i_year]
                    for i in range(0, len(year_lifetime)):
                        if year_vintage == year_lifetime[i]:
                            lifetime = float(lifetime_process[i])
                    start = float(year_vintage)
                    stop = float(year_vintage) + lifetime
                    for j_year in range(0, len(year_list)):
                        year_periods = year_list[j_year]
                        if start <= year_periods < stop and year_periods !=
time_periods.t_periods[len(time_periods.t_periods)-1]:
                            regions.append(CapacityCredit.regions[i_tech])
                            periods.append(int(year_periods))
                            tech.append(CapacityCredit.tech[i_tech])
                            vintage.append(int(year_vintage))

cf_tech.append(float(np.format_float_scientific(cost_list[j_year], 2)))
                            cf_tech_notes.append(CapacityCredit.cf_tech_notes[i_tech])

CapacityCredit_DF = pd.DataFrame(
    {
        "regions": pd.Series(regions, dtype='str'),
        "periods": pd.Series(periods, dtype='int'),
        "tech": pd.Series(tech, dtype='str'),
        "vintage": pd.Series(vintage, dtype='int'),
        "cf_tech": pd.Series(cf_tech, dtype='float'),
        "cf_tech_notes": pd.Series(cf_tech_notes, dtype='str')
    }
)

if tosql_set['CapacityCredit']:
    CapacityCredit_DF.to_sql("CapacityCredit", conn, index=False, if_exists='replace')

if print_set['CapacityCredit']:
    pd.set_option('display.max_rows', len(CapacityCredit_DF))
    pd.set_option('display.max_columns', len(CapacityCredit_DF))
    print("\nCapacityCredit DataFrame\n\n", CapacityCredit_DF)
    pd.reset_option('display.max_rows')

conn.close()
print_i = print_i + 1
print('[',print_i,'/',len(print_set),']     CapacityCredit updated.')
```

B. Commodity-based emission factors table

```
CREATE TABLE "CommodityEmissionFactor" (
        "input_comm"    text,
        "emis_comm"     text,
        "ef"            real,
        "emis_unit"     text,
        "ef_notes"      text,
        PRIMARY KEY("input_comm","ef","emis_comm"),
        FOREIGN KEY("input_comm") REFERENCES "commodities"("comm_name"),
        FOREIGN KEY("emis_comm") REFERENCES "commodities"("comm_name")
);
```

C. Tables for service demands projection

```
CREATE TABLE "Driver" (
    "regions"       text,
    "periods"   integer,
        "driver_name"   text,
        "driver"        real,
        "driver_notes"  text,
        PRIMARY KEY("regions", "periods", "driver_name"),
        FOREIGN KEY("regions") REFERENCES "regions"("regions"),
        FOREIGN KEY("periods") REFERENCES "time_periods"("t_periods")
);

CREATE TABLE "Allocation" (
    "regions"       text,
        "demand_comm"   text,
        "driver_name"   text,
        "allocation_notes"  text,
        PRIMARY KEY("regions", "demand_comm", "driver_name"),
        FOREIGN KEY("regions") REFERENCES "regions"("regions"),
        FOREIGN KEY("demand_comm") REFERENCES "commodities"("comm_name"),
        FOREIGN KEY("driver_name") REFERENCES "Driver"("driver_name"),
);

CREATE TABLE "Elasticity" (
    "regions"       text,
    "periods"   integer,
        "demand_comm"   text,
        "elasticity"    real,
        "elaticity_notes"  text,
        PRIMARY KEY("regions", "periods", "demand_comm"),
        FOREIGN KEY("regions") REFERENCES "regions"("regions"),
        FOREIGN KEY("periods") REFERENCES "time_periods"("t_periods"),
        FOREIGN KEY("demand_comm") REFERENCES "commodities"("comm_name")
);
```

D. Database postprocessing

```
import pandas as pd
import numpy as np
import sqlite3

database_name = "Temoa_Italia.sqlite"
years = np.array([2007, 2008, 2010, 2012, 2014, 2016, 2018, 2020, 2022, 2025, 2030, 2040,
2050])

# Following 4 lines should be used to set the export results.
# Set the flags to 1 to split by technologies/commodities, to 0 to not split.
# Set to_excel_flag to 1 to export data into a Excel file
# The arrays are used to select the technologies/commodities that should be considered.

to_excel_flag = 0
technologies_flag = 0
commodities_flag = 1

technologies = ['']
commodities = ['']


# Input

comm = list()
tech = list()
vflow_in_2007 = list()
vflow_in_2008 = list()
vflow_in_2010 = list()
vflow_in_2012 = list()
```

```
vflow_in_2014 = list()
vflow_in_2016 = list()
vflow_in_2018 = list()
vflow_in_2020 = list()
vflow_in_2022 = list()
vflow_in_2025 = list()
vflow_in_2030 = list()
vflow_in_2040 = list()
vflow_in_2050 = list()

# To classify by technologies, commodities or both
if technologies_flag == 1 and commodities_flag == 1:
    for i_tech in range(0, len(technologies)):
        for i_comm in range(0, len(commodities)):
            conn = sqlite3.connect(database_name)
            Output_VFlow_In = pd.read_sql("select * from Output_VFlow_In where input_comm
= '" + commodities[i_comm] + "' and tech = '" + technologies[i_tech] + "'", conn)
            conn.close()
            vflow_in = np.zeros_like(years, dtype=float)
            for i_year in range(0, len(years)):
                for i in range(0, len(Output_VFlow_In.t_periods)):
                    if years[i_year] == Output_VFlow_In.t_periods[i]:
                        vflow_in[i_year] = vflow_in[i_year] + Output_VFlow_In.vflow_in[i]
            comm.append(commodities[i_comm])
            tech.append(technologies[i_tech])
            vflow_in_2007.append(vflow_in[0])
            vflow_in_2008.append(vflow_in[1])
            vflow_in_2010.append(vflow_in[2])
            vflow_in_2012.append(vflow_in[3])
            vflow_in_2014.append(vflow_in[4])
            vflow_in_2016.append(vflow_in[5])
            vflow_in_2018.append(vflow_in[6])
            vflow_in_2020.append(vflow_in[7])
            vflow_in_2022.append(vflow_in[8])
            vflow_in_2025.append(vflow_in[9])
            vflow_in_2030.append(vflow_in[10])
            vflow_in_2040.append(vflow_in[11])
            vflow_in_2050.append(vflow_in[12])

elif technologies_flag == 0 and commodities_flag == 1:
    for i_comm in range(0, len(commodities)):
        vflow_in = np.zeros_like(years, dtype=float)
        for i_tech in range(0, len(technologies)):
            conn = sqlite3.connect(database_name)
            Output_VFlow_In = pd.read_sql("select * from Output_VFlow_In where input_comm
= '" + commodities[i_comm] + "' and tech = '" + technologies[i_tech] + "'", conn)
            conn.close()
            for i_year in range(0, len(years)):
                for i in range(0, len(Output_VFlow_In.t_periods)):
                    if years[i_year] == Output_VFlow_In.t_periods[i]:
                        vflow_in[i_year] = vflow_in[i_year] + Output_VFlow_In.vflow_in[i]
        comm.append(commodities[i_comm])
        tech.append('#')
        vflow_in_2007.append(vflow_in[0])
        vflow_in_2008.append(vflow_in[1])
        vflow_in_2010.append(vflow_in[2])
        vflow_in_2012.append(vflow_in[3])
        vflow_in_2014.append(vflow_in[4])
        vflow_in_2016.append(vflow_in[5])
        vflow_in_2018.append(vflow_in[6])
        vflow_in_2020.append(vflow_in[7])
        vflow_in_2022.append(vflow_in[8])
        vflow_in_2025.append(vflow_in[9])
        vflow_in_2030.append(vflow_in[10])
        vflow_in_2040.append(vflow_in[11])
        vflow_in_2050.append(vflow_in[12])

elif technologies_flag == 1 and commodities_flag == 0:
    for i_tech in range(0, len(technologies)):
        vflow_in = np.zeros_like(years, dtype=float)
        for i_comm in range(0, len(commodities)):
            conn = sqlite3.connect(database_name)
            Output_VFlow_In = pd.read_sql("select * from Output_VFlow_In where input_comm
= '" + commodities[i_comm] + "' and tech = '" + technologies[i_tech] + "'", conn)
            conn.close()
            for i_year in range(0, len(years)):
                for i in range(0, len(Output_VFlow_In.t_periods)):
                    if years[i_year] == Output_VFlow_In.t_periods[i]:
                        vflow_in[i_year] = vflow_in[i_year] + Output_VFlow_In.vflow_in[i]
        comm.append('#')
        tech.append(technologies[i_tech])
        vflow_in_2007.append(vflow_in[0])
        vflow_in_2008.append(vflow_in[1])
        vflow_in_2010.append(vflow_in[2])
        vflow_in_2012.append(vflow_in[3])
        vflow_in_2014.append(vflow_in[4])
        vflow_in_2016.append(vflow_in[5])
        vflow_in_2018.append(vflow_in[6])
        vflow_in_2020.append(vflow_in[7])
        vflow_in_2022.append(vflow_in[8])
        vflow_in_2025.append(vflow_in[9])
        vflow_in_2030.append(vflow_in[10])
        vflow_in_2040.append(vflow_in[11])
        vflow_in_2050.append(vflow_in[12])

# To find rows with only zero elements
delete_index = list()
for i_comm in range(0, len(comm)):
    flag_zero = 0
    if vflow_in_2007[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2008[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2010[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2012[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2014[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2016[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2018[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2020[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2022[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2025[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2030[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2040[i_comm] != 0:
        flag_zero = 1
    elif vflow_in_2050[i_comm] != 0:
        flag_zero = 1

    if flag_zero == 0:
        delete_index.append(i_comm)

# To remove rows with only zeros elements
for i_delete in range(0, len(delete_index)):
    comm.pop(delete_index[i_delete])
    tech.pop(delete_index[i_delete])
    vflow_in_2007.pop(delete_index[i_delete])
    vflow_in_2008.pop(delete_index[i_delete])
    vflow_in_2010.pop(delete_index[i_delete])
    vflow_in_2012.pop(delete_index[i_delete])
    vflow_in_2014.pop(delete_index[i_delete])
    vflow_in_2016.pop(delete_index[i_delete])
    vflow_in_2018.pop(delete_index[i_delete])
    vflow_in_2020.pop(delete_index[i_delete])
    vflow_in_2022.pop(delete_index[i_delete])
    vflow_in_2025.pop(delete_index[i_delete])
    vflow_in_2030.pop(delete_index[i_delete])
    vflow_in_2040.pop(delete_index[i_delete])
    vflow_in_2050.pop(delete_index[i_delete])

    for j_delete in range(0, len(delete_index)):
        delete_index[j_delete] = delete_index[j_delete] - 1

# Building and printing the table
vflow_in_DF = pd.DataFrame(
    {
        "input_comm": pd.Series(comm, dtype='str'),
        "tech": pd.Series(tech, dtype='str'),
        "2007": pd.Series(vflow_in_2007, dtype='float'),
        "2008": pd.Series(vflow_in_2008, dtype='float'),
        "2010": pd.Series(vflow_in_2010, dtype='float'),
        "2012": pd.Series(vflow_in_2012, dtype='float'),
        "2014": pd.Series(vflow_in_2014, dtype='float'),
        "2016": pd.Series(vflow_in_2016, dtype='float'),
        "2018": pd.Series(vflow_in_2018, dtype='float'),
        "2020": pd.Series(vflow_in_2020, dtype='float'),
        "2022": pd.Series(vflow_in_2022, dtype='float'),
        "2025": pd.Series(vflow_in_2025, dtype='float'),
        "2030": pd.Series(vflow_in_2030, dtype='float'),
        "2040": pd.Series(vflow_in_2040, dtype='float'),
        "2050": pd.Series(vflow_in_2050, dtype='float'),
    }
)

pd.set_option('display.max_rows', len(vflow_in_DF))
pd.set_option('display.max_columns', 16)
pd.set_option('display.precision', 2)
print(vflow_in_DF)
print("\n")
pd.reset_option('display.max_rows')

# Output

comm = list()
tech = list()
vflow_out_2007 = list()
vflow_out_2008 = list()
vflow_out_2010 = list()
vflow_out_2012 = list()
vflow_out_2014 = list()
vflow_out_2016 = list()
vflow_out_2018 = list()
vflow_out_2020 = list()
vflow_out_2022 = list()
vflow_out_2025 = list()
vflow_out_2030 = list()
vflow_out_2040 = list()
vflow_out_2050 = list()

# To classify by technologies, commodities or both
if technologies_flag == 1 and commodities_flag == 1:
    for i_tech in range(0, len(technologies)):
        for i_comm in range(0, len(commodities)):
            conn = sqlite3.connect(database_name)
            Output_VFlow_Out = pd.read_sql("select * from Output_VFlow_Out where 
output_comm = '" + commodities[i_comm] + "' and tech = '" + technologies[i_tech] + "'", 
conn)
            conn.close()
            vflow_out = np.zeros_like(years, dtype=float)
            for i_year in range(0, len(years)):
                for i in range(0, len(Output_VFlow_Out.t_periods)):
                    if years[i_year] == Output_VFlow_Out.t_periods[i]:
                        vflow_out[i_year] = vflow_out[i_year] +
Output_VFlow_Out.vflow_out[i]
            comm.append(commodities[i_comm])
            tech.append(technologies[i_tech])
            vflow_out_2007.append(vflow_out[0])
            vflow_out_2008.append(vflow_out[1])
            vflow_out_2010.append(vflow_out[2])
            vflow_out_2012.append(vflow_out[3])
            vflow_out_2014.append(vflow_out[4])
            vflow_out_2016.append(vflow_out[5])
            vflow_out_2018.append(vflow_out[6])
            vflow_out_2020.append(vflow_out[7])
            vflow_out_2022.append(vflow_out[8])
            vflow_out_2025.append(vflow_out[9])
            vflow_out_2030.append(vflow_out[10])
            vflow_out_2040.append(vflow_out[11])
            vflow_out_2050.append(vflow_out[12])

elif technologies_flag == 0 and commodities_flag == 1:
    for i_comm in range(0, len(commodities)):
        vflow_out = np.zeros_like(years, dtype=float)
        for i_tech in range(0, len(technologies)):
            conn = sqlite3.connect(database_name)
            Output_VFlow_Out = pd.read_sql("select * from Output_VFlow_Out where
output_comm = '" + commodities[i_comm] + "' and tech = '" + technologies[i_tech] + "'",
conn)
            conn.close()
            for i_year in range(0, len(years)):
                for i in range(0, len(Output_VFlow_Out.t_periods)):
                    if years[i_year] == Output_VFlow_Out.t_periods[i]:
                        vflow_out[i_year] = vflow_out[i_year] +
Output_VFlow_Out.vflow_out[i]
        comm.append(commodities[i_comm])
        tech.append('#')
        vflow_out_2007.append(vflow_out[0])
        vflow_out_2008.append(vflow_out[1])
        vflow_out_2010.append(vflow_out[2])
        vflow_out_2012.append(vflow_out[3])
        vflow_out_2014.append(vflow_out[4])
        vflow_out_2016.append(vflow_out[5])
        vflow_out_2018.append(vflow_out[6])
        vflow_out_2020.append(vflow_out[7])
        vflow_out_2022.append(vflow_out[8])
        vflow_out_2025.append(vflow_out[9])
        vflow_out_2030.append(vflow_out[10])
        vflow_out_2040.append(vflow_out[11])
        vflow_out_2050.append(vflow_out[12])

elif technologies_flag == 1 and commodities_flag == 0:
    for i_tech in range(0, len(technologies)):
        vflow_out = np.zeros_like(years, dtype=float)
        for i_comm in range(0, len(commodities)):
            conn = sqlite3.connect(database_name)
            Output_VFlow_Out = pd.read_sql("select * from Output_VFlow_Out where
output_comm = '" + commodities[i_comm] + "' and tech = '" + technologies[i_tech] + "'",
conn)
            conn.close()
            for i_year in range(0, len(years)):
                for i in range(0, len(Output_VFlow_Out.t_periods)):
                    if years[i_year] == Output_VFlow_Out.t_periods[i]:
                        vflow_out[i_year] = vflow_out[i_year] +
Output_VFlow_Out.vflow_out[i]
        comm.append('#')
        tech.append(technologies[i_tech])
        vflow_out_2007.append(vflow_out[0])
        vflow_out_2008.append(vflow_out[1])
        vflow_out_2010.append(vflow_out[2])
        vflow_out_2012.append(vflow_out[3])
        vflow_out_2014.append(vflow_out[4])
        vflow_out_2016.append(vflow_out[5])
        vflow_out_2018.append(vflow_out[6])
        vflow_out_2020.append(vflow_out[7])
        vflow_out_2022.append(vflow_out[8])
        vflow_out_2025.append(vflow_out[9])
        vflow_out_2030.append(vflow_out[10])
        vflow_out_2040.append(vflow_out[11])
        vflow_out_2050.append(vflow_out[12])

# To find rows with only zero elements
delete_index = list()
for i_comm in range(0, len(comm)):
    flag_zero = 0
    if vflow_out_2007[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2008[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2010[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2012[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2014[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2016[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2018[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2020[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2022[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2025[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2030[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2040[i_comm] != 0:
        flag_zero = 1
    elif vflow_out_2050[i_comm] != 0:
        flag_zero = 1

    if flag_zero == 0:
        delete_index.append(i_comm)

# To remove rows with only zeros elements
for i_delete in range(0, len(delete_index)):
    comm.pop(delete_index[i_delete])
    tech.pop(delete_index[i_delete])
    vflow_out_2007.pop(delete_index[i_delete])
    vflow_out_2008.pop(delete_index[i_delete])
    vflow_out_2010.pop(delete_index[i_delete])
    vflow_out_2012.pop(delete_index[i_delete])
    vflow_out_2014.pop(delete_index[i_delete])
    vflow_out_2016.pop(delete_index[i_delete])
    vflow_out_2018.pop(delete_index[i_delete])
    vflow_out_2020.pop(delete_index[i_delete])
    vflow_out_2022.pop(delete_index[i_delete])
    vflow_out_2025.pop(delete_index[i_delete])
    vflow_out_2030.pop(delete_index[i_delete])
    vflow_out_2040.pop(delete_index[i_delete])
    vflow_out_2050.pop(delete_index[i_delete])

    for j_delete in range(0, len(delete_index)):
        delete_index[j_delete] = delete_index[j_delete] - 1

# Building and printing the table
vflow_out_DF = pd.DataFrame(
    {
        "output_comm": pd.Series(comm, dtype='str'),
        "tech": pd.Series(tech, dtype='str'),
        "2007": pd.Series(vflow_out_2007, dtype='float'),
        "2008": pd.Series(vflow_out_2008, dtype='float'),
        "2010": pd.Series(vflow_out_2010, dtype='float'),
        "2012": pd.Series(vflow_out_2012, dtype='float'),
        "2014": pd.Series(vflow_out_2014, dtype='float'),
        "2016": pd.Series(vflow_out_2016, dtype='float'),
        "2018": pd.Series(vflow_out_2018, dtype='float'),
        "2020": pd.Series(vflow_out_2020, dtype='float'),
        "2022": pd.Series(vflow_out_2022, dtype='float'),
        "2025": pd.Series(vflow_out_2025, dtype='float'),
        "2030": pd.Series(vflow_out_2030, dtype='float'),
        "2040": pd.Series(vflow_out_2040, dtype='float'),
        "2050": pd.Series(vflow_out_2050, dtype='float'),
    }
)
pd.set_option('display.max_rows', len(vflow_out_DF))
pd.set_option('display.max_columns', 16)
pd.set_option('display.precision', 2)
print(vflow_out_DF)
print("\n")
pd.reset_option('display.max_rows')

if to_excel_flag == 1:
    vflow_in_DF.to_excel("Export_Input.xlsx", sheet_name='Inputs')
    vflow_out_DF.to_excel("Export_Output.xlsx", sheet_name='Outputs')
```