Politecnico di Torino

POLITECNICO DI TORINO

Energy department "Galileo Ferraris"

MASTER's Degree in ENERGY AND NUCLEAR ENGINEERING

October 2021

MASTER's Degree Thesis

Procedures for operational optimization of polygeneration systems

Supervisors

Prof. MAURIZIO REPETTO

Eng. PAOLO LAZZERONI

Company

EGEA S.r.l.

Candidate

SIMONA PELUSO

Abstract

The need for more efficient and less pollutant energy systems can be satisfied only through the deployment of proper optimization tools, capable to improve environmental, economic, and social sustainability.
In this master thesis are illustrated some procedures aimed at including new functionalities to an optimization tool. XEMS13, the tool under consideration, can be used to optimize the operation of polygeneration systems. In this case, it is able to optimize the production of thermal and electrical energy of a group of selected components, to satisfy the energy demands of a district heating network.

The work is divided into two main topics. The first part regards the analysis of the input data of XEMS13. In particular, a clustering procedure is created to derive from the annual thermal demand, a selected number of significant days, which profiles are able to represent the entire annual profile. The scope is to reduce the computational time and improve the accuracy of the simulation of a year.

Subsequently, two post-processing codes are realized with the aim to reconstruct, starting from the XEMS13 simulation results of the representative periods, the annual results profiles.

The second main topic is the realization of procedures that iteratively run the optimization tool by changing a parameter.
The first is a procedure that performs an iterative run for the implementation of white certificates. They are calculated through the result of the simulation and inserted in the successive iteration, by varying the maintenance cost of the cogenerators. The loop continues until convergence is found or a number of maximum iterations is reached.

The last routine is able to launch a parametric run, for which the size of one component is repeatedly changed. In the end, the best configuration is displayed.

The research showed that it is possible to identify a suitable number of input representative days, able to return accurate annual results and that the external iterative procedures are valid tools for multi-run simulations.

These procedures have been realized in cooperation with EGEA S.p.A. firm that kindly provided most of the data present in this master thesis.

To my grandma Carmela

Table of Contents

List of Tables

List of Figures

Acronyms

ANOVA
Analysis of Variance

CAR
Cogenerazione ad Alto Rendimento

CB
Certificati Bianchi

CHP
Combined Heat and Power

DB
Davies-Bouldin score

DC
Duration curve

ESC
Energy saving certificates

GME
Gestore dei Mercati Energetici

IEA
International Energy Agency

MILP
Mixed Integer Linear Programming

PES

Primary Energy Savings

RES

Renewable Energy Sources

RISP

Risparmio di Energia Primaria

SSE

Sum of Squared Errors

TEE

Titoli di Efficienza Energetica

TOE

Tonnes of Oil Equivalent

Chapter 1

Introduction

The evidences of climate changes push to find solutions aimed at modify the way people live, produce and consume. To overcome the challenges caused by greenhouse effect and environmental degradation, different countries in the world, with EU leading, have stipulated many pacts in the last years, beginning from the Kyoto Protocol in 1997. In all these years steps forward have been made on the road to a more sustainable world, but they are not sufficient.
The most recent pact is the European Green Deal, officially presented in December 2019. It is aimed at building a path to follow in order to limit as much as possible the climate changes effects, making European Union an efficient but also competitive economy. The main points of the Green Deal are the followings:

- Zero net emissions of greenhouse gases by 2050;
- Decoupling of the economical growth from the resources consumption;
- Inclusion of all people and places, "no one must be left behind".

Being the production and the use of energy the responsible for the 75% of EU's greenhouse gases emissions, the sector is one of the most relevant in the transition. The decarbonization of the EU energy system is a critical point to meet the long-term objectives, but also to reach the reduction of net greenhouse gas emissions of 55% by 2030 (compared to 1990 levels), which is a midway point.
The increment of the energy efficiency and the growth of the share of renewable are priory, but attention must be given also to a secure and affordable energy supply and an energy market more interconnected and digitalised, [1].

In this master thesis there are different topics which are connected to the sustainability in the energy field.

First of all, the energy system analyzed and on which the work done is based, are district heating systems. District heating is an infrastructure that provides

thermal energy to multiple buildings from a group of central energy plants. Hot water produced at the plant is transmitted through insulated underground piping networks. The thermal energy is transferred to the building's heating system, avoiding the need for boilers in individual buildings. Customers avoid installing expensive boilers and save moneys for their operations, maintenance, repair and replacement. But the main advantage is on the point of view of energy efficiency and emissions reduction. Connecting multiple buildings to a district system, in fact, enables the deployment of more efficient local energy resources. This scale also make possible the integration of cleaner options like CHP, waste to energy, heat pumps, biomass, geothermal, and other renewables which significantly cut emissions, [2].

From IEA statistics, district heating systems is used to meet the 4% of heating demand in the world, with a consistent presence in China, Russia and Europe, especially for space heating. Since 2010, new connections have increased of 3.5% per year, in particular due to China's large network.
On the other hand, significant effort is still needed to reduce the carbon intensity of district heating, which has remained almost unchanged across the globe in recent years, especially due to China's reliance on coal. The share of renewable energy sources in European district energy systems, instead, increased in recent years, especially in Denmark, Finland, France, Latvia and Lithuania. The carbon intensity of district heat production in Europe is around $150 \div 300$ gCO_2/kWh. Decarbonisation efforts are oriented towards the improving of existing networks and the development of fourth- and fifth-generation low-temperature new networks, which allow a greater usage of RES and local waste heat, [3].

In 2019 in Italy there were around 330 district heating systems, with a total of 9,6 GW installed power. Considering the residential sector only, it satisfies the 2% of the heating and domestic hot water demand. In the same year the total thermal energy fed into the network was of 11,9 TWh, the 63% supplied from natural gas, 25% from renewables sources and wastes and 12% from the remaining fossil fuels. In the last years also district cooling has started to spread, [4].

In the Italian perspective, one of the most relevant district heating company is EGEA S.p.a, which kindly provided the data and the information to conduce the example cases in this thesis. EGEA is a multiutility operating in the sectors of electrical energy, district heating, gas distribution, water facilities and public utility services. Based in Alba (CN), it operates predominantly in Piedmont. The company, on May 2019, has been cited by Financial Times as the first Italian multiutility regarding the energy sustainability sector, and it is at the 86° place in Europe. The mention among the "Europe's Climate Leaders 2021" is due to the decrease, from 2014 to 2019, of the 17.3% of the *core* CO_2 emissions.

Among the other technologies to produce the required energy, there are the

cogenerators, which are another relevant topic in the sustainability field encountered in this thesis. Cogeneration is an efficient technology that generates both electricity and heat. For this reason it is also called Combined Heat and Power (CHP). Nowadays cogeneration supplies 11% of electricity and 15% of heat in Europe. EU's targets aims at increasing these shares to 20% and 25% respectively by 2030 and to double cogeneration capacity by 2050. COGEN Europe, indicates as main advantages of cogeneration [5]:

- Increased energy efficiency (even of 40%) with respect to separate generation of heat and power;
- Lower emissions (around 200 Mton of CO2 saved in Europe every year) and reduced energy costs due to the higher energy efficiency that allow to use less fuel;
- Cogeneration can work with renewable fuels in a cost-effective way. Nowadays, 27% of fuels used in cogeneration in Europe are renewable, as for example biomass and biogas;
- It can have different sizes: and can fit to supply a single household or an entire town;
- It is resilient and flexible and makes transmission and distribution costs decrease;
- The cogeneration sector employs 100,000 people in Europe and this number is expected to grow due to European Union investments.

Related to cogenerators, there is another important topic faced during the thesis work, which is the white certificates. In Italy white certificates have been introduced in 2005. They are marketable securities. They certify energy savings obtained from measures aimed at increase the energy efficiency of a system. Each white certificate corresponds to a TOE (Tonnes of Oil Equivalent) saving. They are issued from GME (Gestore dei Mercati Energetici) and can be exchanged on the proper market platform, also managed by GME. They are the main instrument for the energy efficiency promotion in Italy and they are called "certificati bianchi" (CB) or also "Titoli di Efficienza Energetica" (TEE). The cogeneration unit recognized as CAR (Cogenerazione ad Alto Rendimento) have the access to the white certificates market and get the entitled number, [6]. A cogeneration unit is defined as CAR if its primary energy saving (PES, calculated as in formula 5.6) is of almost 10%, or if the plant is of small (< 1 MWe) or micro (< 50 kWe) size. CAR have also priority for the dispatch of the produced electricity and tax benefits for the gas utilization, [7].

Being linked to different energy carriers, cogenerators are key points for sector coupling, which could be defined as the process of progressively inter-linking the electricity, heat and gas sectors. The optimization of the existing synergies in the generation, transport, and distribution of different energy carriers, has as ultimate scope the decarbonization.
So the environmental sustainability of the energy systems goes along with the ability to manage in the optimal way different actors, paying attention to the technical constraint and the demands to satisfy. This critical point needs the help of technology. Digitalization, together with the use of software and support tools, is a valid aide in the challenges that the energy sector must face in these years. The use of optimization tools is essential for a system management able to realize environmental, social and economical sustainability.
The scope of this master thesis is to provide to one of them, XEMS13, some support procedures aimed at increase its functionality. The codes will be introduced in chapter 2 and illustrated in the respective following sections.

Chapter 2

Optimization tool and external procedures

In this master thesis will be presented some external support codes for the software XEMS13, aimed at adding some useful functionalities to the tool.

2.1 XEMS13

XEMS13 is an optimization tool developed by the Energy Department of Politecnico di Torino "Galileo Ferraris" and LINKS. It is able to simulate polygeneration systems and optimize their management, considering all the constraint relative to the problem. The objective function to minimize is the sum of all the operational costs.
In order to work properly, the program needs as input:

- The time profiles of the energy demands (of heating, electricity and, when present, also cooling), the time profiles of the energy prices (electricity purchased and sold, natural gas) and of the generation from renewables (ex. solar thermal). They are in form of hourly values listed in csv files, usually contained in a folder called "Profiles";

- All technical and operational characteristics of the used components (as cogenerators and boilers), inserted in an xml file. For example, it must contain the power levels, the maintenance cost, the fuel used (if not declared natural gas is the default) and other parameters related to each component;

- A netlist text document that resumes information about the simulations, as the title of the profile files, the name of the component used and the length

of the period to simulate. It is contained usually in the folder "Work", where also the result files are created.

After the request of the folders "Work", "Profiles" and "Components" and of the netlist file, the tool starts to elaborate the problem. The equations describing the problem are the balance equations of each energy carrier that ensure the satisfaction of the demands, and the constitutive equations that represent the energy flows of each source. They are all linear or piecewise linear, so the problem can be solved by MILP Mixed Integer Linear Programming, [8]. The approach is steady state and the transient status of the cogenerators is not considered.
If required, also an environmental analysis can be done, which computes the mass of the total CO_2 emitted.

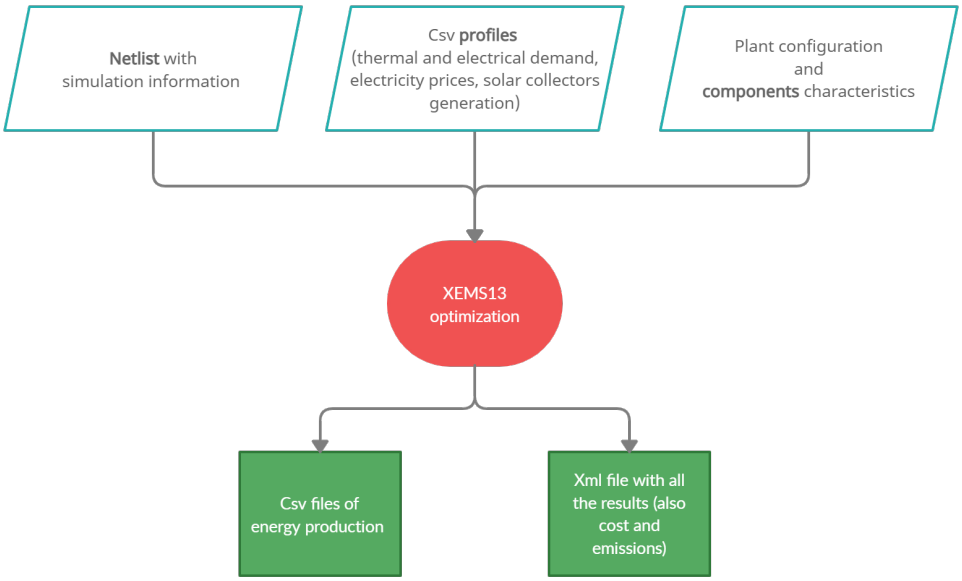


Figure 2.1: Optimization tool inputs and outputs

Once the problem is linearized and all the equations, boundaries and constrains are defined, an ".mps" file is created where the information is condensed and delivered to the MILP solver (such as SCIP, Gurobi or MatLab). The solution is processed and the results are produced, [9].
The tool solution is presented in form of a xml result file and two csv files. The xml contains the value of the objective function, all the values of the hourly energies produced by all the components, the electricity sold and bought, the thermal energy that is stored (if a storage is present) or dissipated and all the other values able to describe the system. The csv files resume, in a tabular form, all the energy

exchanges. There is one for the thermal energy that satisfies the heating demand and one for the electrical energy.

The duration of the period that the tool easily simulate is about one week, so 168 hours. To understand the annual behavior, by now, the methodology most used is to select one week per month and two weeks for the months when the change of the heating period happens (April and October). A total of 14 weeks is so simulated to approximate the results of all the year.

The program has been developed on Matlab, but an executable has been created to use it also with computers without Matlab installed.
The external shell codes, instead, are realized with Python, which is nowadays more spread.

The work done in this master thesis can be divided in four principal topics, briefly introduced in the next sections.

2.2 Clustering of the thermal demand profile

As already said, by now, the way used by EGEA to get the annual results is to execute XEMS13 simulating 14 weeks to represent the entire year. From the annual thermal demand profile, only one week per month is taken, two for the switch months.
However, for complex system, the simulation of 14 weeks can become quite computational heavy. Moreover, it could happen that the selected week does not reflect properly the profile of the month. For example, it could happen that the chosen week, which is usually the first one, is characterized by a demand higher than the rest of the month, causing an overestimation of the load.

In chapter 3, an alternative method is investigated by using the clustering. The clustering algorithms take a set of data and group similar data into clusters, which can be represented by the cluster centroids.
So clustering can be used to derive from an annual thermal demand, a certain number of representative days, each one depicting a group of days among the year. The thermal demand is the most important XEMS13 input profile for EGEA, that uses it to simulate district heating systems. For the other input profiles, as for example the prices ones, are taken the profiles referred to the day in the year most similar to each reference day.

Before a brief description of the main clustering techniques, two in particular are investigated: k-means and Ward algorithms.
For each of the two, a code for the selection of the right number of cluster and a code for the results analysis have been realized.

2.3 Simulations and annual post-processing

In chapter 4, once completed the preliminary steps to run the tool, as the creation of all the others input files, XEMS13 is launched for different cases: for all the reference days derived using different numbers of clusters and for the 14 weeks case. Whether the XEMS13 simulation is run for the 14 weeks, whether for the reference days found with the clustering, the results will be related to each single week or to each single day, so it is necessary a post-processing to derive the annual results.

Two post-processing codes are built for the two cases and discussed in chapter 4. For the 14 weeks case, the results of each month are obtained assuming that all the weeks of the month have the same behaviour of the selected (first) one.
To reconstruct the annual results for the clustering case, at each day of the year are associated the results of the reference day that represent its cluster.

The results so obtained are used to confront the different cases, completing the comparison started in the previous chapter.

2.4 White certificates implementation

Chapter 5 regards the implementation of the white certificates for CAR by an iterative procedure. A part of the topic has been argument of the internship done in EGEA.

White certificates can not be calculated during the XEMS13 simulation, being them dependent on its outputs. In particular, they depend on the useful thermal energy produced by the CHP, as it will be discussed more in detail in the proper chapter.
The aim of the implemented procedure is so to take the output of a netlist simulation, calculate the white certificates using the formulas indicated, and subtract them from the maintenance cost of the cogenerators. The results of the successive simulation will be different and the work is repeated until convergence is found.
With the aid of the post-processing codes, the routine is then transformed in an annual procedure, that calculates white certificates from the approximated yearly results.

2.5 Parametric run

Chapter 6 discusses a procedure able to make a parametric run of XEMS. In general, the routine iteratively changes a parameter inside the xml components file, and simulates XEMS13 to find the optimal selection of that parameter.

In particular, the procedure is focused on the change of the size of one or more components. The component and the range of sizes to test are chosen and then, for the different sizes in the range, the power levels of the component are changed in the relative xml and the tool is launched.
Among the other outcomes, all post-processed to get the annual results, the objective functions are saved for each iteration. At the end they are all compared and the lowest one is used to determine the optimal size.

Chapter 3

Analysis and discretization of temporal series

In the perspective of the optimization of energy systems, one of the encountered issues can be the computational time needed to have hourly results for an year. In order to reduce the computational weight of a simulation, some significant periods can be taken as references instead of the full year, and the simulation can be run only over the selected periods. Then, with proper correlations, the annual results can be extracted.
The selection of the proper reference periods, or of the method to derive them, is a relevant matter since different situations must be taken into account and considered in the final solution: periods of high and low load, seasonal oscillations, critical days.
The method used by now by EGEA is based on the use of 14 representative weeks of the year. Each week represents a months, the only exception is for April and October. Turin and great part of the Piedmont provinces are, in fact, in the E climatic zone. This means that the building heating period goes from the 15 of October to 15 of April [10]. So, in these two months, two reference weeks are taken into account to analyze the half-month behaviour of the system, before and after the start of the heating season for October and conversely for April. However, the simulation of 14 weeks requires a long computational time, especially for the period when the demand is almost null, since the storage management becomes difficult. The total running time can be reduced with the right selection of most significant and shorter time periods.
Different techniques are analyzed in order to choose the most suitable, whose results will be then compared with the current method used by EGEA. The series to sample is the thermal demand of the users of the district heating system.

3.1 Overview of the sampling techniques

There are different methods that can be adopted to choose the suitable reference time periods. The desired optimization model should select a certain number of representative periods and good compromise between accuracy of the results, computational time and complexity must be found for the given situation. The models most used in literature are:

- Heuristic;
- Clustering algorithms;
- Random selections;
- MILP optimization.

In [11] they are briefly described and compared.
With the heuristic method, the reference days that need to represent a period of time (a year or a season) are chosen with a practical method, as, for example, selecting the days with the highest and the lowest load. Heuristic methods are very flexible and simple, but may produce solutions very far from the optimal.

Clustering algorithms are often used and comprehend a large family of different approaches, so they will be analyzed in a separate section.

The random selection method chooses in a casual way a given number of set of representative days and calculates the error (calculated with proper error metrics) associated to the usage of each set. The set with the lowest error is selected. For computational reasons it is not possible to try all the possible combinations of representative days, so the number of set to analyze is limited before the run.

The Mixed Integer Linear Problem optimization method minimises the differences between the duration curve of the real full year data and of the representative year data for each time series (if more than one must be used). The duration curve is, in fact, divided in bins and for each bin the error with the approximated duration curve must be minimized by means of the selection of the best days. The "weight" associated to all the representative days is also calculated. It is the number of repetitions of each day in the reconstructed year, [11].
After, a Mixed Integer Quadratic Program method can be also used to reconstruct the chronological order of the data. It needs as input the time series to discretize, the representative days and their weights, and gives as result the year reconstructed with the selected days chronologically ordered, [12].

3.1.1 Clustering algorithms

> "Clustering is an unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters), [13]."

Clustering procedures can follow a large variety of approaches, the most relevant are hierarchical and partitional.

- Hierarchical methods produce a nested series of partitions using a bottom-up approach. Agglomerative types are the most common, which include the Ward's clustering.

- Partitional methods produce only one series of partitions. k-means clustering is an example, [14].

3.1.2 Preliminary definitions

A table with the meanings of the most used symbols in chapter 3 is presented.

Symbol	Meaning
x	generic sample day
j	subscript for samples
i	subscript for clusters
n	number of samples in a cluster
y	subscript for the cluster closer to i
t	subscript for the samples of the closer cluster
k	number of clusters
c	generic centroid
C	generic cluster
X	set of all the samples
h	subscript for an element of a vector (so an hourly value)
v	element of a vector (so hourly value)
p	generic vector
q	generic vector
f	apex for iterations

Table 3.1: List of symbols

Moreover, some general formulas are indicated.
The centroid of a cluster is the reference day that represents all the sample days in

the cluster. It is the mean vector, and each of its hourly values is calculated as:

$$c_{h,i} = \frac{1}{n_i} \sum_{j=1}^{n_i} v_{h,j,i} \tag{3.1}$$

where $c_{h,i}$ is the value at hour h of the centroid of the cluster i, $v_{h,j,i}$ is the value assumed at the h^{th} hour by the j^{th} day profile of the cluster i, that contains a total of n_i profiles.

The euclidean distance between two generic vectors is defined as:

$$||p - q|| = \sqrt{\sum_{h=1}^{24} (p_h - q_h)^2} \tag{3.2}$$

In the analysis under consideration all the vectors are daily profiles, so they have 24 elements.

Ward's algorithm

Between the hierarchical agglomerative type of algorithms, the Ward's has been chosen as example. Others differentiate mainly on the metric used to calculate the distances. Ward's algorithm proceeds in the following way:

1. For each day of the year a vector with all the hourly values of the data series is created. If there is more than a time series to sample, it is a matrix. The quantities are normalized, scaled according to the maximum value assumed over the year.

2. Initially each observation is a cluster;

3. The algorithm groups the days into cluster in a "rich get richer" way. Most similar clusters are merged. Ward's algorithm iteratively joins the two clusters whose combination results in the smallest error increase.
   Being the sum of squared error of a cluster (called also within-cluster variance) defined as:

   $$SSE_i = \sum_{j=1}^{n_i} ||x_{j,i} - c_i||^2 \tag{3.3}$$

   where $x_{j,i}$ represents the j^{th} observation vector in cluster i, the clusters merged are the two clusters i and y for which:

   $$SEE_{increment,iy} = SSE_{iy} - (SSE_i + SSE_y) \tag{3.4}$$

   is minimized, [15], [16].

4. The algorithm can continue until a certain number of clusters is reached. This number should be a good trade-off between computational time and accuracy, so different runs with different number of clusters should be tried;
5. At each representative day a weight is assigned, in function of its cluster size;
6. Then all the time series are re-scaled to reach the correct annual average, [17].

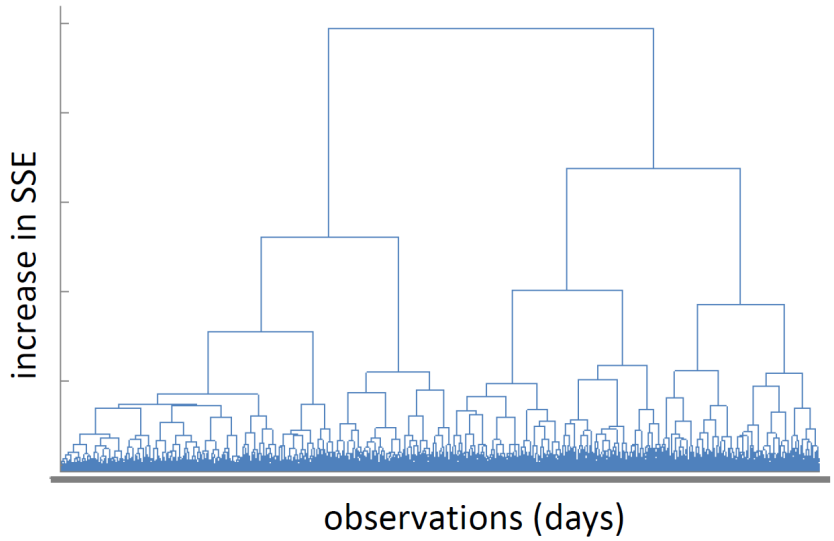


Figure 3.1: Dendrogram of hierarchical clustering algorithm: in this case the algorithm is not stopped until it reaches an unique cluster, [17]

On scikit-learn Python library the agglomerative algorithms can be used thanks to the function *AgglomerativeClustering*. The number of clusters to achieve at the end can be put as input or, in alternative, the distance threshold between which the clusters will not be merged can be inserted.
It is possible also to choose the linkage type to determine how to calculate the distance between sets. As already said, Ward minimizes the variance between clusters and it is the default one. The others are: "average" which uses the average of the distances of each observation of the two sets, "complete" (or "maximum") linkage that uses the maximum distances between all observations of the two sets, and "single" which uses the minimum of the distances between all observations of the two sets. [16].

k-means

k-means algorithm is, instead, a partitional clustering method based on the Lloyd's algorithm, called also Voronoi iteration and named after Stuart P. Lloyd. The

Lloyd algorithm, applied in an Euclidean plane, given a set of points (or seeds), is able to realize a Voronoi diagram. A Voronoi diagram is a partition of a plane into regions (called Voronoi cells) consisting of all points of the plane closer to a seed than to any other. The seeds, in the simplest case, are points on the plane. So, in the final configuration, the seeds are the centroids and each region represents all the samples belonging to a cluster, [16], [18].

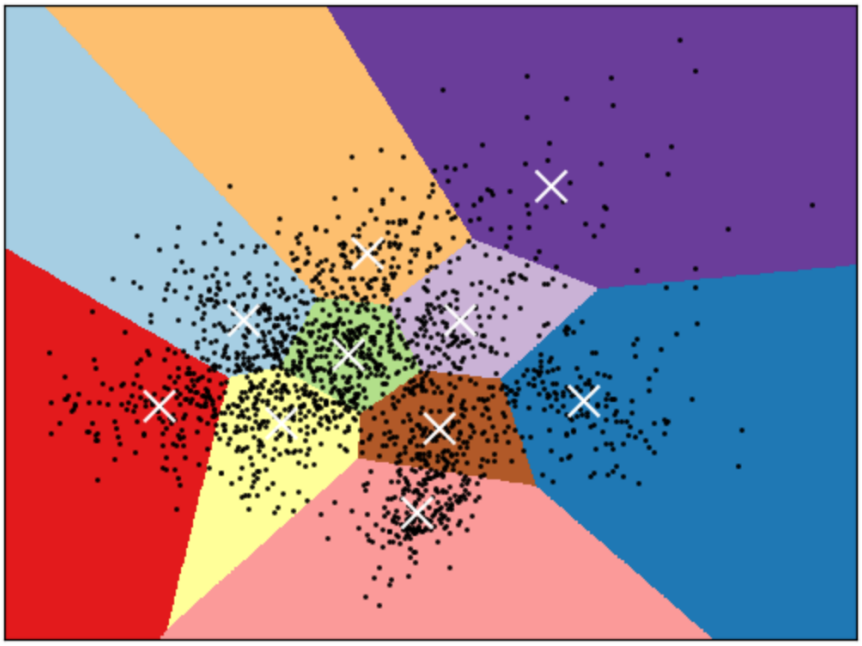


Figure 3.2: Example of the Voronoi diagram representation for a set of points in a 2D space, [19]

k-means algorithm minimizes the sum of the squared error over each cluster (or within-cluster variances), called also *inertia*, [16]. The total inertia, which is a measure of how internally coherent clusters are, can be calculated as:

$$I = \sum_{i=1}^{k} \sum_{j=1}^{n_i} ||x_{j,i} - c_i||^2 \tag{3.5}$$

where k is the number of clusters, n_i the number of observations j in the i^{th} cluster, $x_{j,i}$ is the j^{th} observation of the i^{th} cluster and c_i its centroid, [20]. So the objective function of Ward and k-means are similar, but the approach is different. The procedure, indeed, is the following:

1. Some random (or not) clusters center are chosen among the days;
2. All the observations are associated to the closer cluster;

3. Cluster centers are then recomputed as the mean of the new observations of the cluster;

4. Convergence check: if the selected convergence criterion is not met the loop continues re-starting from point 2.

The k-means algorithm requires as input the number of clusters and the starting points and results will depend on these two parameters. A solution presented in [14] is to build a multi-objective external optimization which minimizes the number of representative periods and the profiles deviation, expressed by some dedicated indicators.

On scikit-learn Python library, k-means algorithm is implemented thanks to the command *KMeans*. The most important input parameters are:

- Number of clusters;

- Method for the initialization. The method "k-means++" is set by default. It initializes the centroids to be distant from each other. In alternative, the starting points can also be random or an array can be manually inserted.

Moreover, it is possible to insert also the maximum number of iterations of the k-means algorithm for a single run, the k-mean algorithm to use ("auto" is the default) and other specifications.

The metric used to evaluate the distances between two observations, so between two day profile vectors, is the euclidean distance.

The algorithm returns the cluster centers, the label of each point (so at which cluster is associated each sample, from which is possible to calculate the weight of each representative day), the total inertia and the number of iterations run.

3.2 Implementation of the k-means technique

The first method used to extract the representative days from the thermal load curve is the k-means.
k-means clusters data minimizing the already cited inertia. The algorithm, during the initialization, chooses the initial centroids from the dataset. Consequently, it loops between the first step, which assigns each sample to its closest centroid and the second step, that creates new centroids by taking the mean value of all of the samples previously assigned to each "old" centroid. The difference between the old and the new centroids are computed and the loop stops when this value is less than a threshold, so when the centroids do not move significantly. More in detail, [21]:

Algorithm 1 k-means pseudo-code

1: *Given a set of elements X and a desired number of clusters k:*
2: Centroids c are initialized with *k-means++*
3: **while** $\sum_{i=1}^{k} ||c_i^{(f)} - c_i^{(f-1)}|| >$ *relative tolerance* so, while euclidean distance between the clusters of two consecutive iterations is higher than the tolerance **do**
4: Each profile is assigned to a cluster. $\forall i \in \{1, ..., k\}$, the cluster C_i is the set of profiles in X that are closer to its centroid $c_i^{(f)}$ than to any other centroid (using Euclidean distance)
5: $\forall i \in \{1, ..., k\}$, the centroid $c_i^{(f+1)}$ of the cluster C_i is set as: $c_i^{(f+1)} = \frac{1}{n_i} \sum_{\forall x \in C_i}^{n_i} x^{(f+1)}$, if the cluster has n_i elements
6: **end while**

3.2.1 Initialization

As first preliminary step, it is necessary to re-organize the set of data in the shape desired from the algorithm. It requires, in fact, as input, the matrix X, whose rows correspond to each sample (so to each day of the year) and whose columns correspond to the hourly data of the thermal load. So, a matrix of dimensions 365x24 is built.
Another important input parameter is the number of clusters. At this point the best number, in terms of accuracy of the solutions and speed of convergence, is unknown, so an external loop that tries different numbers and then compares the solutions on the basis of three different indicators will be implemented. By now, attention is given to the internal k-means loop, and this matter will be investigated later.
KMeans requires also the starting centroids. To define them, the scikit initialization scheme *k-means++* has been used. It initializes the centroids to be distant from each other, leading to better results than random initialization and also speeding up convergence. The algorithm allows also to choose different weights for the samples, but in this case this property has not been used, so at each day the same importance has been assigned. The pseudo-code of *k-means++* is the following:

Algorithm 2 k-means++ initialization pseudo-code

1: An initial center c_1 is chosen at random from the samples set X
2: Being *D(x)* the shortest distance from a data x to the closest center already chosen, the next center $c_i = x' \in X$ is chosen from X with probability $\frac{D(x')^2}{\sum_{x \in X} D(x)^2}$;
3: The previous step is repeated for all the remaining centroids;
4: It is possible to proceed with the standard k-means algorithm.

3.2.2 Outcomes of the algorithm

The interesting results that can be computed are:

- The cluster centers: the vector of the desired reference days, that coincide with the centroids;
- The labels: a vector that at each sample (day of the year) associates the number of the cluster at which it belongs, so the reference day at which it is associated;
- The weights of each reference day: it is not given as output, but it can be easily calculated from the labels. It is useful to understand how big each cluster is, so the importance of each reference day for further considerations;
- The computational time: can be useful to compare different algorithms;
- The Inertia, the Davies-Bouldin score, the Silhouette score: the three index necessary to compare solutions with different number of clusters, analyzed in the next paragraph.

3.2.3 External loop for the number of clusters

The algorithm has been inserted inside a loop that ranges the number of clusters between a minimum and a maximum, set initially. At each iteration, the value of Inertia, Davies-Bouldin score and Silhouette score are saved and at the end plotted with respect to the number of cluster to evaluate the best number.

Inertia is the sum of squared distances of samples to their closest cluster center (within-cluster variances), defined in 3.5. The algorithm aims at minimizing it. Inertia tells how far the points within a cluster are from the centroid. It decreases increasing the number of clusters.

Davies-Bouldin score is the average similarity measure of each cluster with its most similar cluster. Similarity is calculated as the ratio of intra-cluster distances (average euclidean distance between each point of a cluster and its centroid) to the distance between the cluster centroid and the centroid of the closest cluster. For a generic cluster i the intra-cluster distance is $s_i = \frac{1}{n_i} \sum_{j=1}^{n_i} ||x_{ij} - c_i||$, where c_i is its centroid, x_{ij} its generic *jth* element and n_i the number of elements in i. The between cluster distance is $d_{i,y} = ||c_i - c_y||$, if y is the cluster closest to i). So, being:

$$R_{i,y} = \frac{s_i + s_y}{d_{i,y}} \tag{3.6}$$

Davies-Bouldin index can be calculated as:

$$DB = \frac{1}{k}\sum_{i=1}^{k} \underset{i\neq y}{m}axR_{i,y} \tag{3.7}$$

A cluster is better if it is far from the others and less dispersed. Lower values indicates best clustering, so this score must be minimized, [16].
The concept behind this index is related to the ANOVA, (Analysis of Variance) that comprehends a series of statistical techniques that allow to confront groups of data comparing the internal groups variance with the variance between groups. The only difference is that in ANOVA the indexes are based on the squared euclidean distances, [22].

Silhouette score tells how far away the samples in one cluster are from the samples in another cluster.
Being (a) the mean distance between a sample and all other samples in the same cluster ($a_{i,j} = \frac{1}{n_i}\sum_{t=1}^{n_i} ||x_{i,j} - x_{i,t}||$), and (b) the mean distance between a sample and all the samples in the next closest cluster ($b_{i,j} = \frac{1}{n_y}\sum_{t=1}^{n_y} ||x_{i,j} - x_{y,t}||$ where n_y is the number of samples in cluster y), the silhouette of a single sample can be calculated as:

$$Silhouette_j = \frac{(b_j - a_j)}{max(a_j, b_j)} \tag{3.8}$$

The total Silhouette is the mean of the Silhouette of each sample.
The range of Silhouette score is from -1 to 1. The best value is 1 and the worst value is -1, because negative values indicate that a sample is closer to a cluster which does not belong, than to its own cluster, so it has been assigned to the wrong cluster. Values near 0, instead, indicate that some clusters are overlapping, [16].

Plotting these three index, is possible to choose a good trade-off that minimizes Inertia and Davies-Bouldin score and keeps Silhouette closest ad possible to 1. Once the right cluster number has been chosen, *KMeans* is run again for it and the results are further investigated.

3.3 Example cases with k-means

To implement the k-means method two codes have been written: one to select the best number of clusters on the basis of the three indexes already cited, the one other that repeats the k-means algorithm for the chosen number of clusters and executes some post-processing analysis. A profile related to the thermal demand curve own by EGEA has been used as test case. It contains the hourly thermal load of 2020.

3.3.1 Selection of the number of clusters

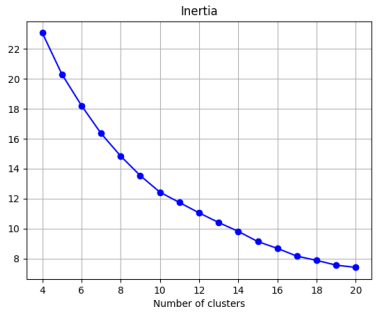
The code *"k_means_n_clusters"* has been developed for the correct choice of the number of reference days.
As first thing, it is possible to set certain parameters, which are:

- The number of the sample days (or periods) in the real profile. In this case, the whole period is of one year, and being the 2020 a leap year, the days to sample are 366. The choice to make this parameter an independent variable is to allow to sampling different types of periods, from season to groups of years;
- The minimum number of clusters: in this case is set to 4, so the year will be represented at least by four representative days;
- The maximum number of clusters: it has been set to 20.
- The number of hours of each sample period: in this case the aim is to find reference days, so it is 24, but days can also be grouped together (in this instance also the number of the sample period must be changed. For example, if in another situation there is the need to find one representative week for an entire season, the number of sample periods is the number of weeks in that season and the hours are the total hours in a week, so 168. The only limit is that the total period must be a multiple of the number of sample periods).

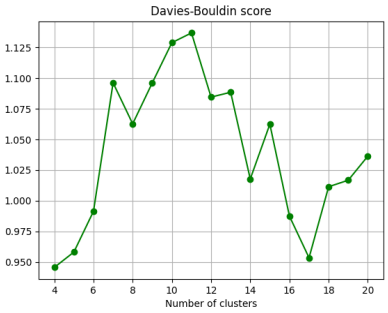
After the initialization, a separate function is called. This function, built with the package "tkinter" is able to open a window allowing the user to select the csv file from which the profile is taken. This file, for sake of simplicity, must contain a unique column, where the first row contains a title and the others the hourly values in chronological order. After the selection of the file, the window must be closed, so the code can continue to run.
From the annual profile, the matrix X is derived. It is the input of the k-means algorithm and contains the profiles of all the days. In this case is 366x24. All the values are normalized over the maximum, which is memorized in a variable.
Inside a *for* cycle, the k-means is run for all the number of clusters in the selected range and some parameters are displayed, like the computational time and the number of k-means iterations run for each number of cluster.

The values of Inertia, Davies-Bouldin score and Silhouette score are calculated and plotted with respect of the number of clusters selection they refer. The results are shown in fig 3.3.

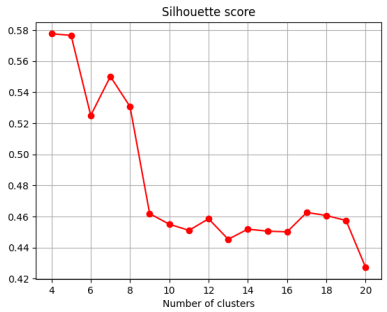
The Silhouette score results very high up to five clusters, than decreases and oscillates. This value should be as close as possible to the value of 1 to indicate that all points are associated to the right cluster. Good scores are achieved for 7, but also 6 and 8 clusters.



(a) Inertia



(b) Davies-Bouldin score



(c) Silhouette score

Figure 3.3: Plots of the three indexes for the selection of the number of clusters

Davies-Bouldin score, which indicates how much the clusters are dispersed, should be minimized. Good performances are achieved with 17, but there are local minimum also for 14 and 8 clusters.
Since Inertia, which represents the sum of squared distance of each point to its cluster centroid, always decreases increasing the number of clusters, the number of 8 clusters could be the right choice for representation of this year. It is, in fact, quite near to the elbow of the curve after which the curve starts to flatten, after 10 further increment in the number of cluster results in smaller decrements of the inertia.

3.3.2 K-Means and post-processing

With the code *"k_means_pp"*, the algorithm is repeated only for the selected number of clusters. In fact in this case, the parameters to set are:

- The number of clusters, to set after the observation of the results of the code "k_means_n_clusters". In this example is 8;
- The number of days to sample, that, as already said, in this example case are 366.
- The path of the folders in which the required csv and text files will be created.

Initially the function to select the csv file with the thermal profile is called and the matrix X is built. The k-means algorithm is launched and the reference days are obtained.
From the vectors of label also the weight of each reference day can be calculated. The reconstructed annual profile is also built by taking for each sample day its reference day.

One important data from the original curve that must be preserved is the total amount of thermal energy required in the entire year. Using the clustering, this information could be not perfectly maintained, being the real and the reconstructed profiles different. The reconstructed profile is built by taking for each sample day its reference day. To solve this problem, the reference profiles and the reconstructed curve are divided by the sum of the energy demands of the fictitious year and multiplied by the sum of the energy demands of the real year. This is a precautionary step, since actually this ratio resulted in a value very near the unit, with an error of only 1.71e-14.

The first charts shown are the profiles of all the reference days. In the plots of figure 3.4 all the grey curves in each graph represent the profiles of all the sample days of the year, with the thermal load in MWh in the y-axis and the hours of the day in the x-axis. The profiles in dark grey are the ones which belong to that

cluster, the others are in light grey. The green curve in each graph is the profile of the reference day that represents the cluster.

The figure 3.5(a) shows instead in a pie chart the weight of each reference day, so the fraction of days represented by that centroid.
The graph 3.5(b) associates at each day of the year the corresponding cluster number, so it tells from which reference day is represented each sample day. The enumeration of the clusters is done randomly by the algorithm.

How it is possible to observe, the representative days well approximate the different periods of the year:

- The central days are represented with the centroid 1, that alone accounts for the 50% of the total weight. It corresponds to a reference day with a very low demand. In fact the days that are in the range 106-298 are the days between the 15 of April and the 15 of October, where the demand is almost null;

- The rest of the year, that is the heating period, is represented by the remaining seven types of reference periods. The reference day number 5, which is the one with the highest peak experienced, fits well the first days of the year, so the January days, generally colder. The reference days 2, 3 and 6 are also used for the winter, with 2 and 3 (highest after 5) used especially for February and December, which are the coldest after January;

- The days just after and just before the non-heating period are in line with the centroids 4, 7 and 8 that in fact, have a mid demand profile.

- The clusters with days with high and low demand are quite coherent: they contain sample days with very similar profiles.
  There is a cluster with medium demand which seems to collect the days whose profiles does not match with the other reference days. In 3.4 cluster number 7, indeed, contains profiles that, after 15 p.m., assume very different shapes.

- In all the reference days the highest peak is around the 8÷10 a.m. in the morning, there is another around 14 p.m. and the last one is the evening peak (only exception for the summer reference day 1);

- From the labels image, it is possible to see that there are some days which belong to a different cluster of the days before and after. This discontinuities may be due to different environmental conditions, like a sudden change in the external temperature.

The analysis is completed with the plot of the entire annual profiles of the reconstructed curve and the real curve (3.6) and of the two duration curves (3.7). The reconstructed profile, as expected, results as more step-wise and with less oscillations then the real one.

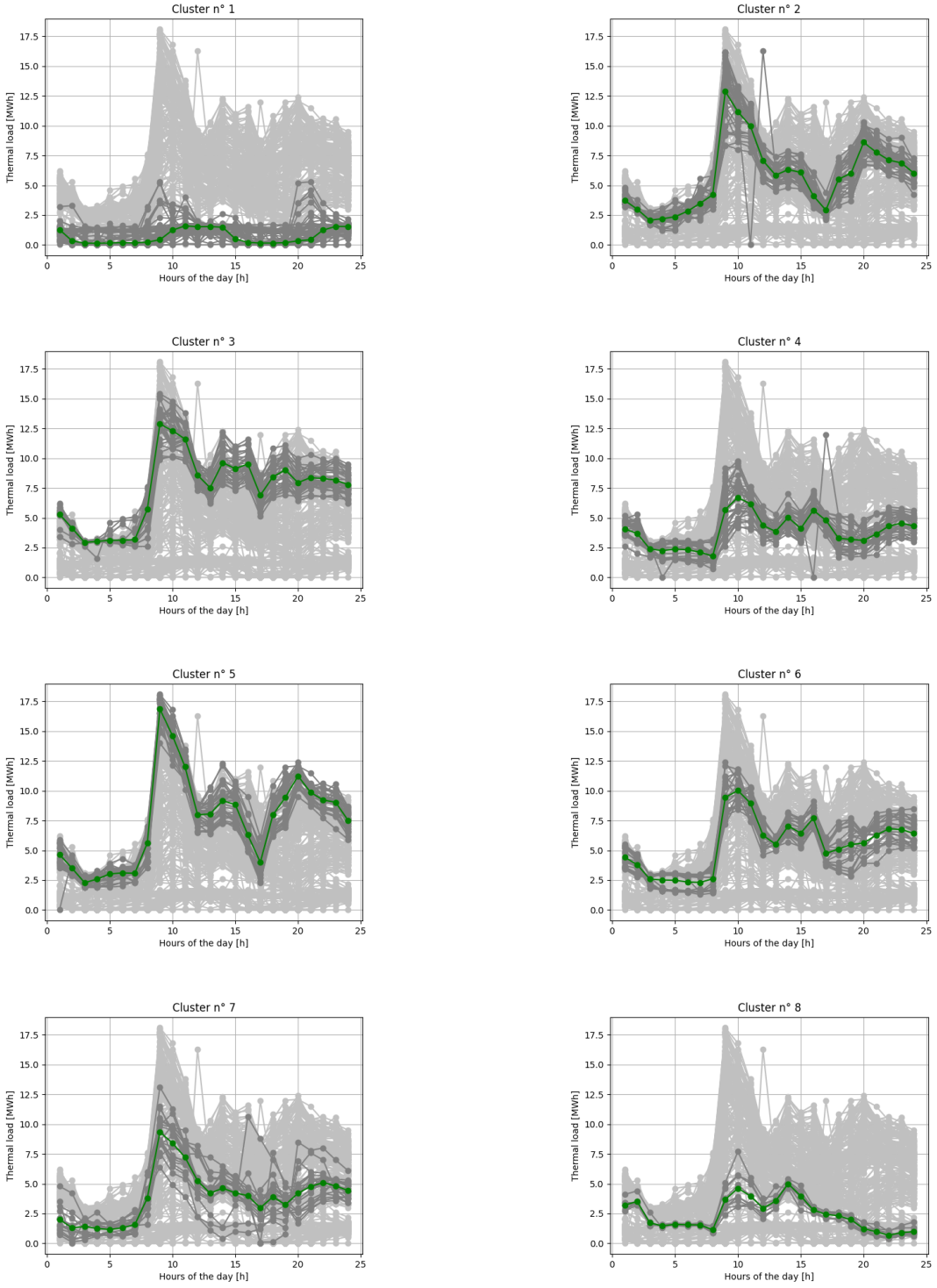
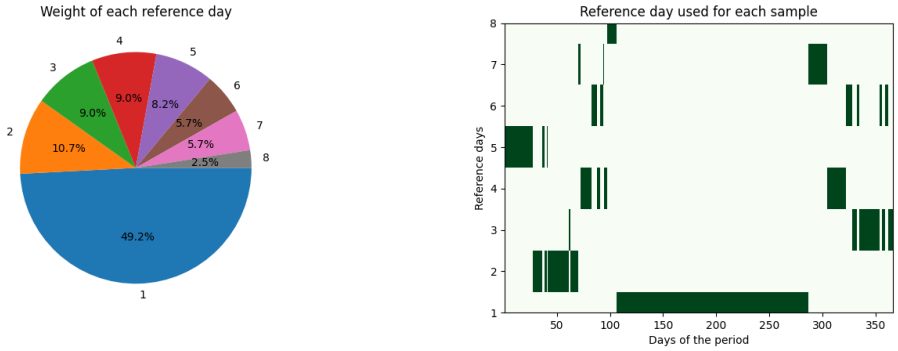


Figure 3.4: Reference days profiles



(a) Weights of each reference day

(b) Reference day corresponding for each sample

Figure 3.5: Weights and labels

The duration curve is the graphical representation of the relationship of all the values assumed by the thermal demand and their respective duration, normalized over the year. For the construction of the duration curves the hourly values are disposed in descending order. Each point of the curve tells that the correspondent load value on its ordinate is overcome for a percentage of the year equal to its abscissa.
How it is possible to see, the region between the two duration curves are coloured in red. The regions in which the reconstructed duration curves overcomes the real one have the same area than the regions in which the opposite occurs, to make the total area below the two graphs equal and the total energy demand the same.

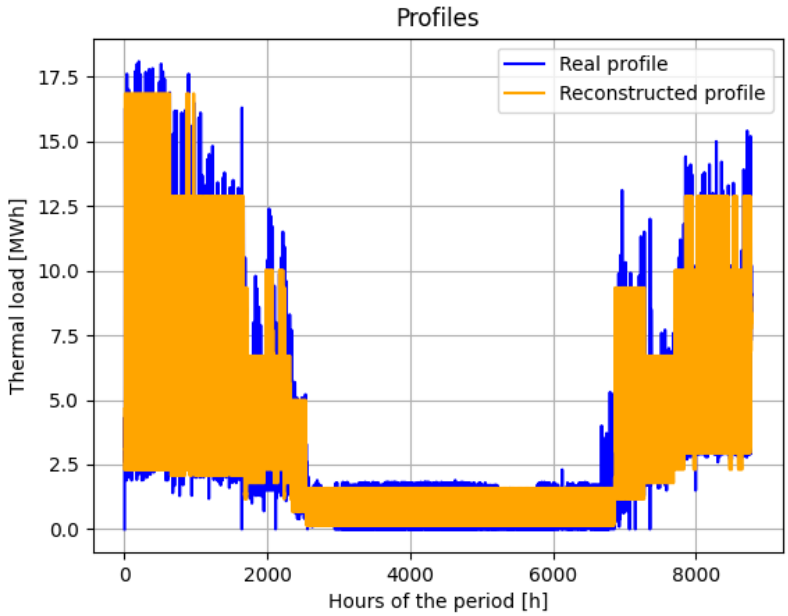


Figure 3.6: Real and reconstructed annual profiles

Another possible analysis is related to the difference in the profiles of the two duration curves. Despite the total energy demand of the year has been constrained to be the same, there could be significant variations in the profiles of the two curves, that could result in dissimilar solutions of the consequent optimization problem. An example is if for the real DC the same thermal load has an occurrence very different from the reconstructed DC.
To investigate this matter, an analysis based on [11] is carried out. The demand is normalized on the maximum and it is divided in a certain number of bins N_{bin}, for example 20, as in fig. 3.8. In correspondence of each bin the duration in % of the respective normalized thermal load is obtained for both the curves. Then the error

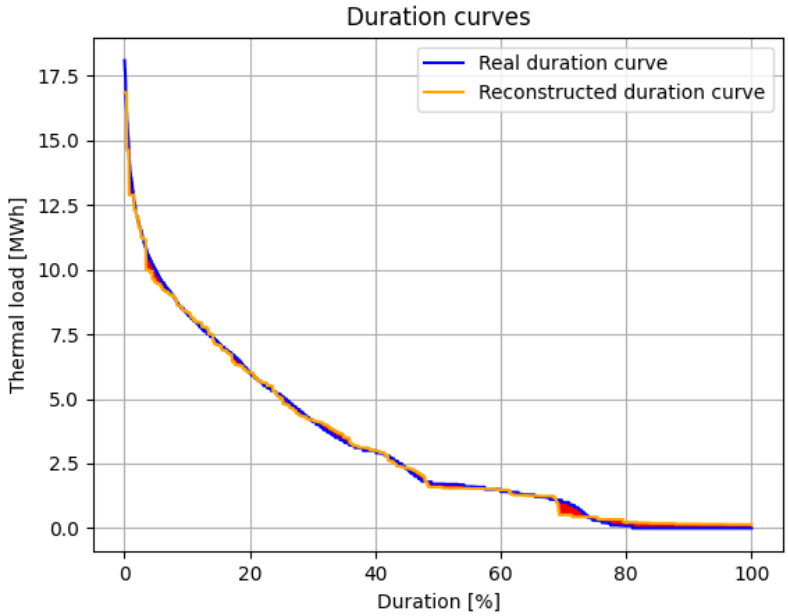


Figure 3.7: Real and reconstructed duration curves

for that bin is calculated as their absolute difference. The error associated to each bin is then plotted as in fig. 3.9(a).

$$\sum_{bin=0}^{N_{bin}} |L_{bin} - A_{bin}| \tag{3.9}$$

The formula 3.9 shows the total error that in [11] is chosen as variable to minimize to solve the MILP problem. Despite in this analysis the k-means method is used, this error can be used for further investigation. L is the share of time during which the real DC curve exceeds the value of the corresponding bin. So, for that bin, it is the horizontal distance between the blue triangular point and the y axis. A is the share of time during which the reconstructed DC curve exceeds the value of the corresponding bin. So, for that bin, it is the horizontal distance between the yellow squared point and the y axis. The error associated to each bin is the discrepancy in the share of time correspondent to the normalized thermal load of that bin for the two DC curves that is the horizontal distance between the two curves for that bin.

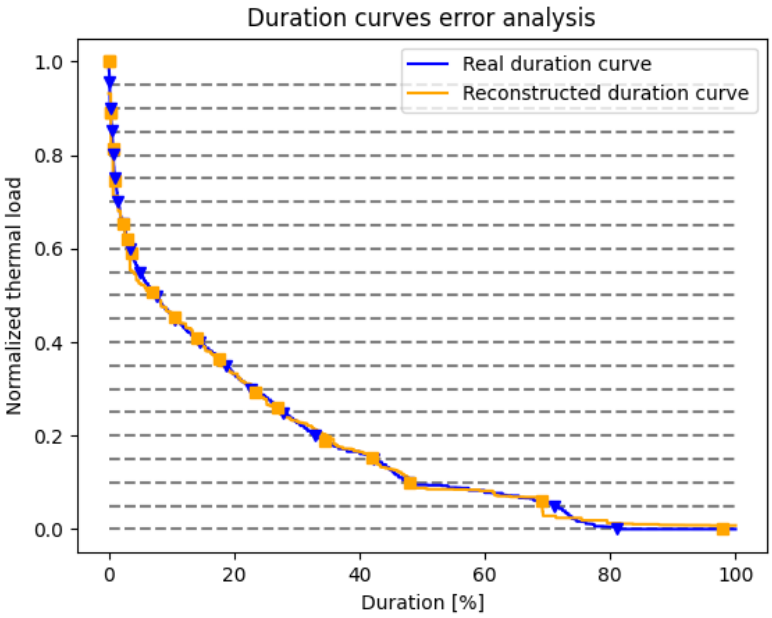
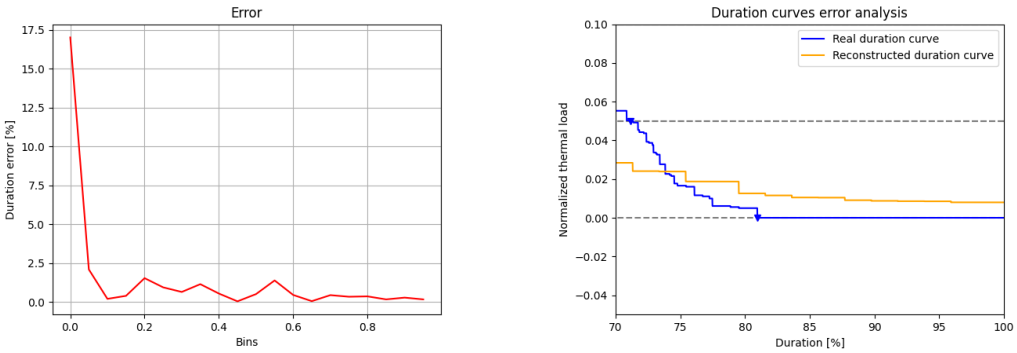


Figure 3.8: DC graphs with bin analysis

As can be seen from fig. 3.9 (a), that shows the error associated to each bin (called duration error), the error remains well under a duration difference of 2.5% for all the bins, except for the very first one, which is associated to the null thermal load. Observing more in detail what occurs with a zoom in this part of the graph, it

is possible to note that for the real DC the thermal demand is not null only for the 81% of time, so for the remaining hours is zero. The reconstructed curve, instead, never reaches values of zero demand. This could mean that the days approximated in the worst way are the days with null demand, so the summer days.



(a) Error associated to each bin **(b)** Zoom of the graph in fig. 3.8(a)

Figure 3.9: Discrepancies in the DC curves

Repeating the analysis with a different number of bins, no significant differences are detected, the error profiles is almost the same, with the highest value for the bin associated to the null thermal load.
Actually, dissimilarity are present also for the hours of very high thermal load. The reconstructed curve never touches value of normalized thermal demand of 100%. The maximum hourly thermal load, that is 18.09 MWh, is not present in the reference days profiles, in which the maximum value reached is 16.85 MWh. This is not so important in terms of the final result, since in the real profile this value is overcome only for 17 hours, so for the 0.2% of time.
Since the summer days are less relevant in the optimization of a district heating system, and the hours of very high thermal demand that are not perfectly represented are few, this result can be considered acceptable.

The part of code that calculates this type of error can be implemented also in the code that iterates the k-means for different cluster numbers. The sum of the errors related to each bin is calculated and plotted for each cluster number.

In fig. 3.10 can be seen that this value starts to be considerably lower for 9 clusters, where there is a mean value of a difference in the share of time of real and reconstructed DCs of 2.5%. In the figure 3.3 Silhouette score and Davies-Bouldin score returned worst value for 9 clusters, but 9 cluster is closer to the elbow of the inertia curve, so a trial can be done following the minimization of the duration curve error and observing the differences.
Repeating the k-means and post-processing for 9 clusters can be noted that, in

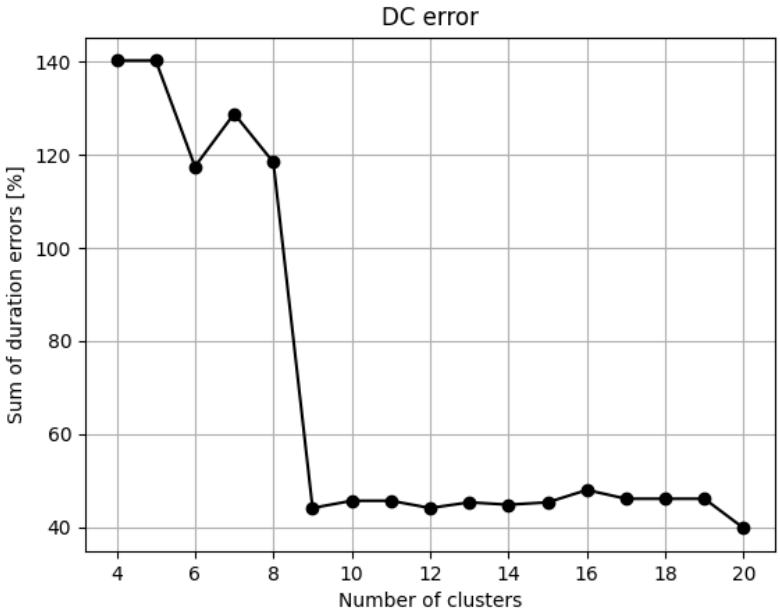


Figure 3.10: Error in the DC for different number of clusters used

comparison with the case with 8 clusters:

- The non-heating period days are represented with an additional reference day, with a totality of two (2 and 7).
- The January days with the highest demand are represented by the reference day 3;
- The remaining clusters are used to represent the winter and the mid-demand period.
- In this case, the cluster with the less coherent profiles is the number 1, again representing mid demand days.

It is really important to notice that the clusters are not the same as in the previous analysis and even if there are some cluster very similar, they are associated to a different number.
It can be observed also that with 9 clusters, the error in correspondence of the null demand still high, instead decreases for higher loads. The algorithm finds that is still not worth it to create a representative days with perfectly null demand.

The higher number of reference days results in representing in a more accurate way the low demand period, capturing better the non-heating season oscillations.

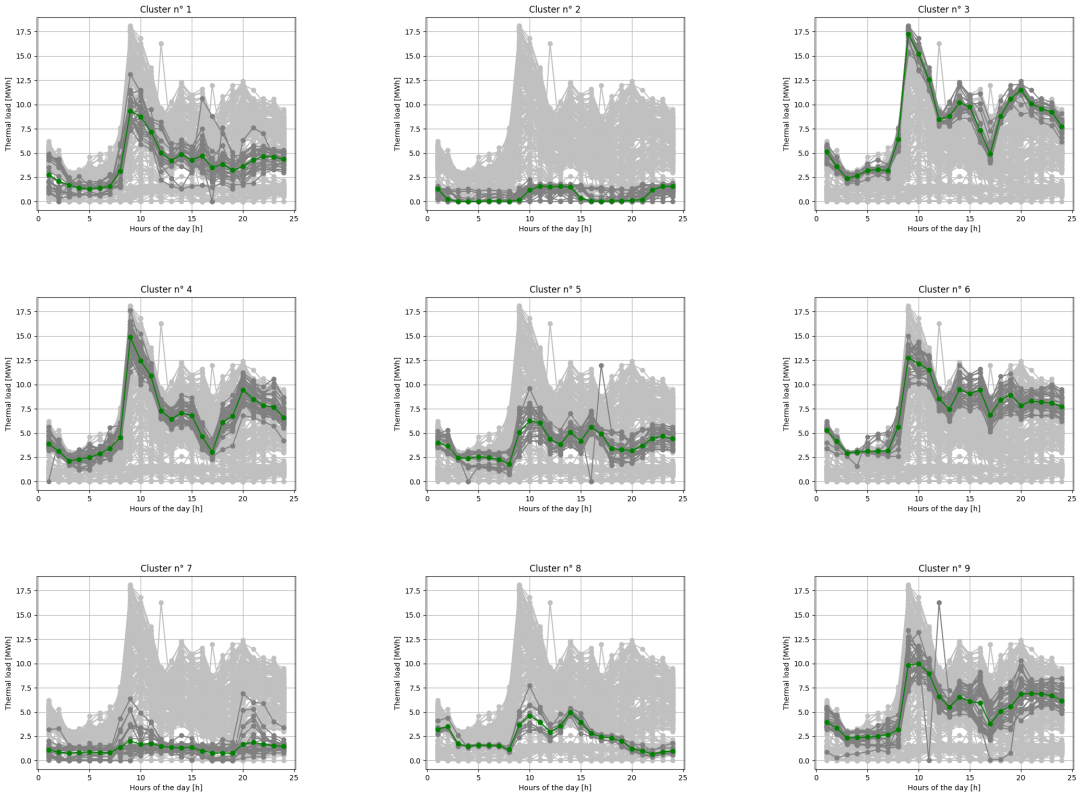


Figure 3.11: Reference days profiles for 9 clusters

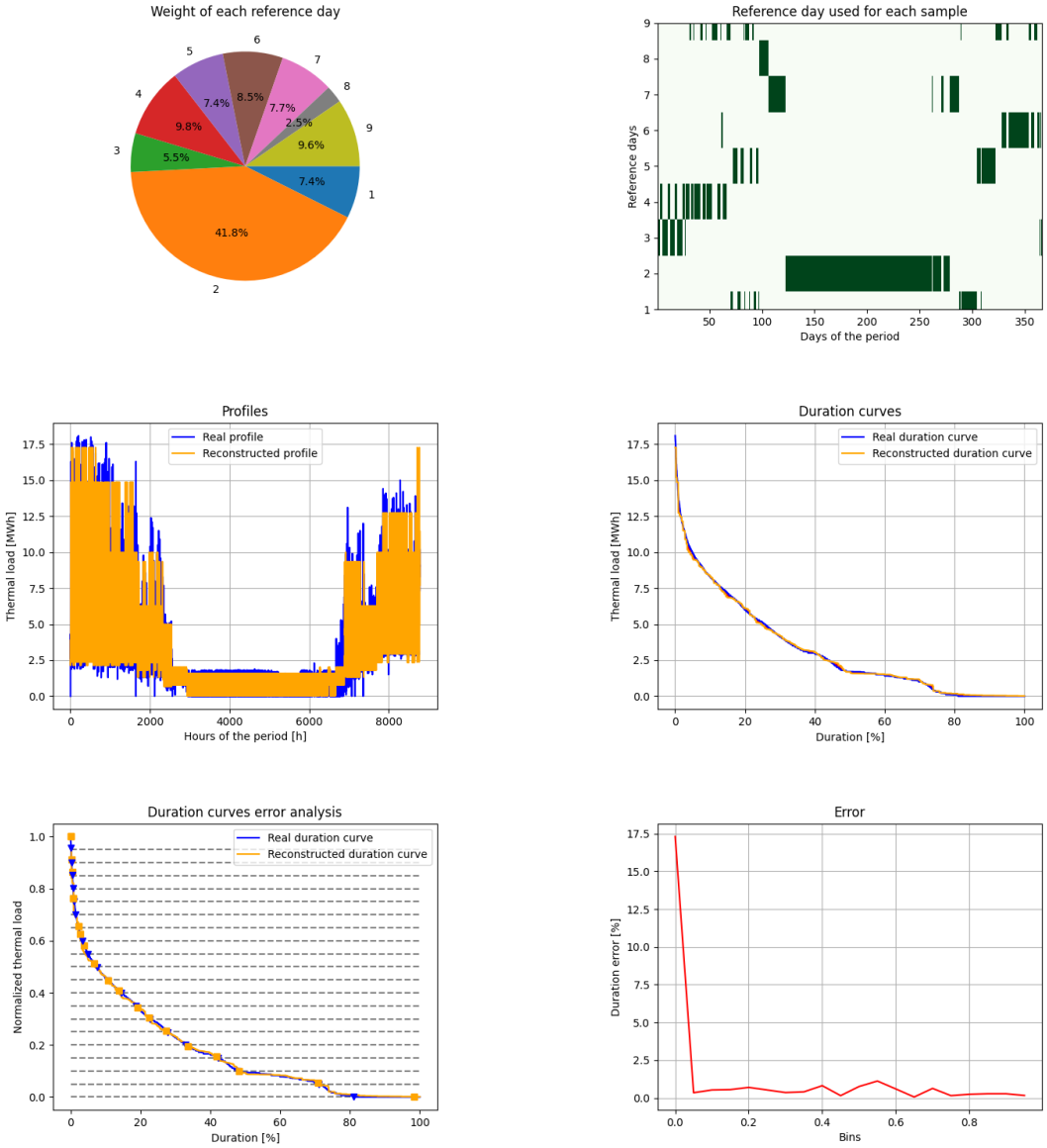


Figure 3.12: Results for 9 clusters

The observation of the inertia values, of the Davies-Bouldin score and of the Silhouette score that is a used and validated methodology, can be supplemented with other types of indicators, like the clustering dispersion indicator, the scatter index, the mean-adequacy index, the variance ratio criterion and several others, [23]. In this analysis also a method based on the discrepancies between the real and the reconstructed DC curves is utilized. The best number of representative days should met different needs, from a fast optimization to a more accurate solution. In the last case, a choice could be also clustering separately different periods of the year, as single seasons.

The results show that the k-means algorithm with both 8 and 9 clusters approximate well the annual profile. For further considerations the optimization algorithm should be run for both the cases and then the system results and the computational times should be compared. This will be the next step, done in Chapter 4.

3.4 Ward implementation and example cases

Before the prosecution of the study with the XEMS simulation of the found reference days, the procedure has been repeated with the Ward algorithm.

3.4.1 Algorithm description

Ward hierarchical agglomerative clustering is a method that builds a hierarchy of clusters with a "bottom-up" approach: at the beginning each observation is a cluster, subsequently pairs of clusters are merged as it goes up on the hierarchy. As already said, the logic with which clusters are merged is the minimization of the increase of the sum of squared errors between observations and centroids, 3.4. The agglomerative behaviour leads to uneven cluster sizes, but Ward is the type of linkage that returns the most regular sizes, [16].

Algorithm 3 Ward's pseudocode

1: *Given a set of elements X and a desired number of clusters k:*
2: Each observation is a cluster
3: **while** number of clusters > k **do**
4: The two clusters C_i and C_y whose merging returns the minimum SSE (as in 3.4) are joined
5: **end while**

The preliminary phase with the construction of the observations matrix is identical to the previous case. The initialization does not require the starting centroid as in the k-means case, since each sample day initially is a reference day.

Then the algorithm continues merging similar reference days until the selected number of cluster is reached.

The outcomes that the algorithm provides are not exactly the same of the k-means.
The cluster centers, indeed, are not provided, so they are calculated afterwards as a mean of the days belonging to each cluster, as in 3.1. The days of each cluster are known since the labels are provided.
Another property that is not available with Ward is the Inertia, the validation of the number of cluster must follow only the other indexes.
So the external loop for the choice of the number of cluster is based on the Davies-Bouldin score, the Silhouette score and the additional duration curve error.

3.4.2 Small Ward's example case

For a better comprehension of the algorithm, a small example is shown.
Four profiles are taken from the annual thermal demand, shown in fig 3.13: a high demand profile from Winter, a Spring low demand profile, an almost null demand from Summer and a mid demand profile from Autumn.

The Ward's algorithm is applied to cluster them in three clusters and the three reference profiles obtained are those in fig. 3.14.

It is evident that the reference profile 2 (b) represents the cluster having only the Autumn profile (fig. 3.13 (d)) and the reference profile 3 (c), the Winter profile (fig. 3.13 (a)). The Spring (fig. 3.13 (b)) and Summer (fig. 3.13 (c)) profiles have been insert in the same cluster and their representative profile is the one in fig. 3.14 (a)). So the increase in SEE due to their merging is the lowest and the centroid resulting is the mean of the two vectors.

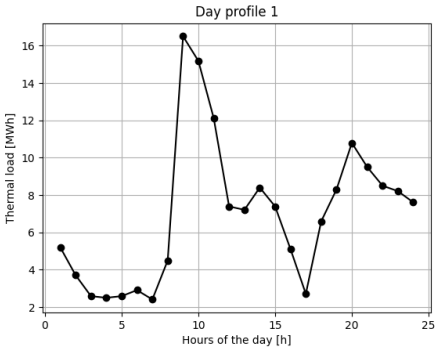
If the Ward's algorithm is done without stopping when a certain number of cluster is reached, the full dendrogram tree can be plotted and the result is in fig. 3.15.

After the low and almost null demand profiles, the mid and high demand profiles are merged, since they contribute to the lowest increase in the SEE. The final merging in a unique cluster make the SEE increase considerably.

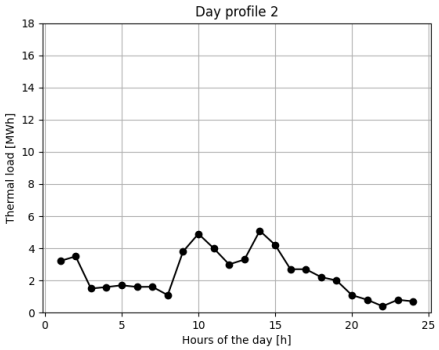
3.4.3 Results

In this section all the Ward's results related to the Carmagnola case are presented. The plots of the external algorithm for the choice of the number of clusters are shown in fig. 3.16.

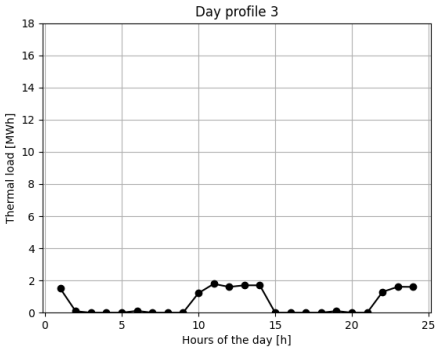
In this case, Davies-Bouldin score suggests good results for 13 clusters.
The profiles of the Silhouette score and the duration curve error are very similar,



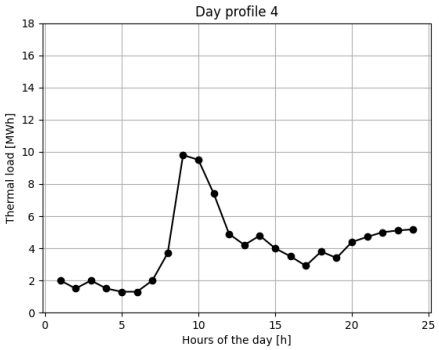
(a) High-demand Winter profile



(b) Low-demand Spring profile

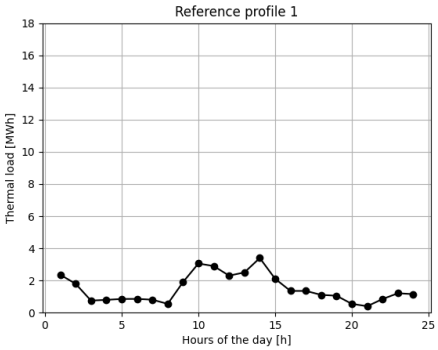


(c) Almost null demand Summer profile

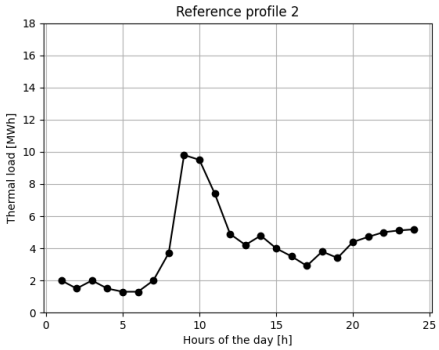


(d) Mid-demand Autumn profile

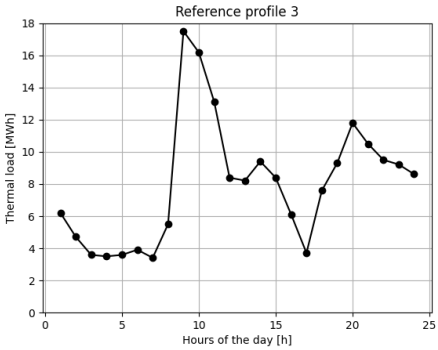
Figure 3.13: Initial profiles, Ward's example



(a) Reference profile 1



(b) Reference profile 2



(c) Reference profile 3

Figure 3.14: Reference profiles for 3 clusters, Ward's example

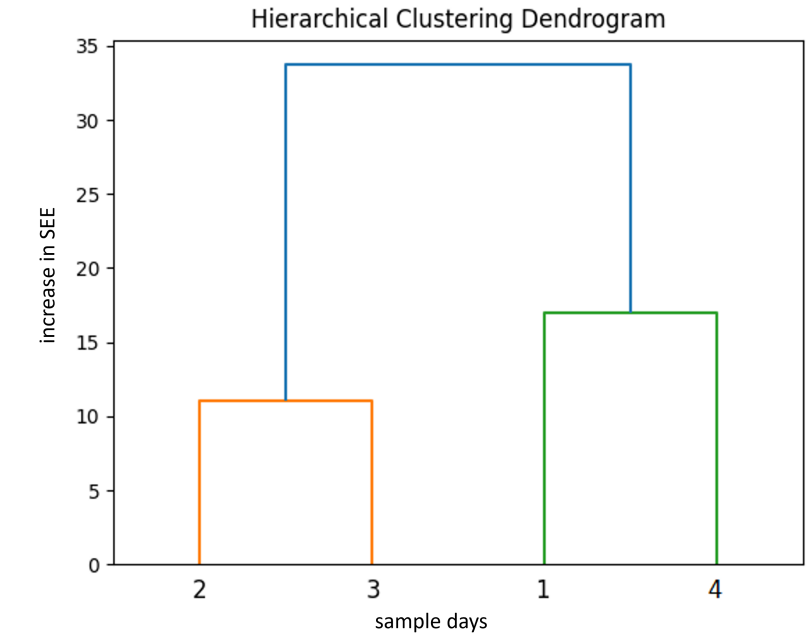
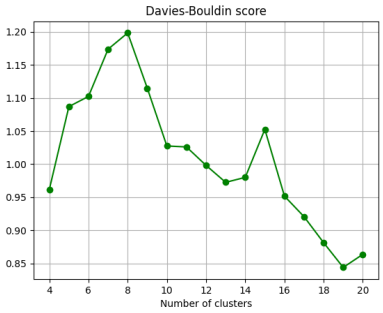
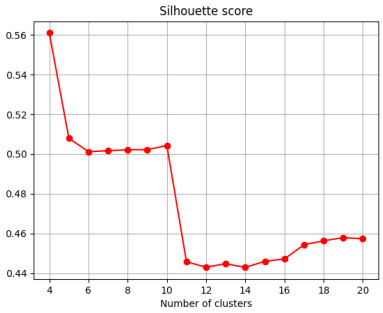


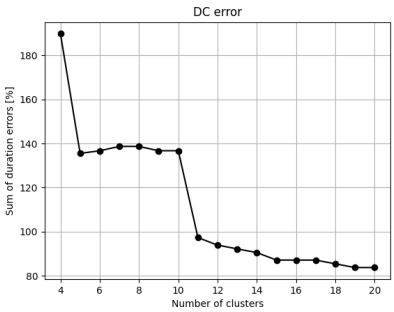
Figure 3.15: Dendrogram of Ward's example case



(a) Davies-Bouldin score



(b) Silhouette score score

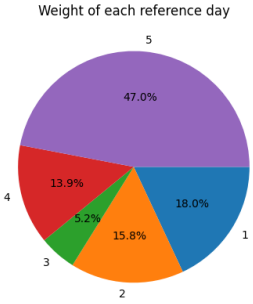


(c) Duration curve error

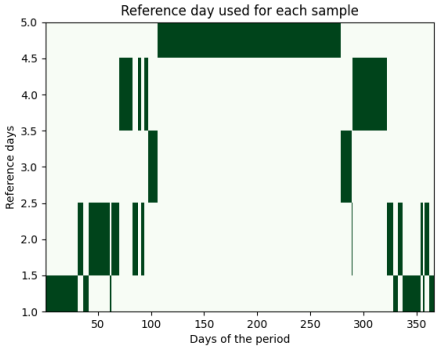
Figure 3.16: Plots of the indexes for the selection of the number of clusters

but it is an inconvenience since the first should be maximized and the second minimized. So, it would be interesting to see the behaviour both before and after the big drop in the profiles: 10 and 11 clusters.

Comparing the k-means and the Ward clustering using the "best" number found separately with the indexes analysis could be not so fair, since with Ward the higher number of reference days could lead to more accurate results. On the other hand, it would be unfair also to use the same number, since 8 and 9 clusters with Ward return a very high David-Bouldin score.
Since the computational time of the algorithm is very reduced, it is possible to execute different trials with 5, 8, 10, 11 and 14 clusters.
Weights of each reference day, labels and profiles are plotted for all the cases.



(a) Weights of each reference day

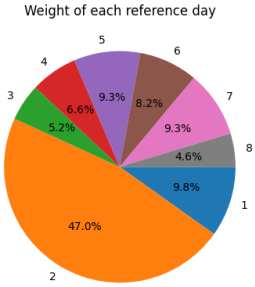


(b) Reference day corresponding for each sample

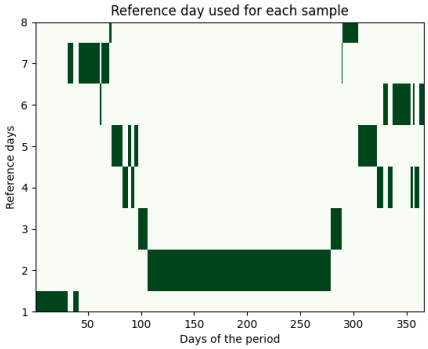
Figure 3.17: Weights and labels with 5 ref. days

Some considerations can be done about the labels and the weights graphs :

- The "heaviest" reference day, so the day that represents the larger cluster and is repeated more time during the reconstructed year, is always the day of the non-heating period, with a weight in the range 40÷50%.
- From 11 clusters and above, there are two days representing the non-heating period, but one is used to represent only some days at its beginning and at its end;
- The vast majority of reference days is used to approximate the profile of the heating period. The largest part of the additional reference day from 5 to 14 is added at the two edges of the year, so during the winter season. It is also the most energetically relevant period, so this result is fitting, since accuracy is increased when demand is higher.

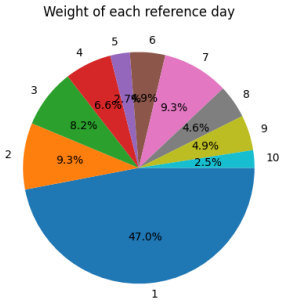


(a) Weights of each reference day

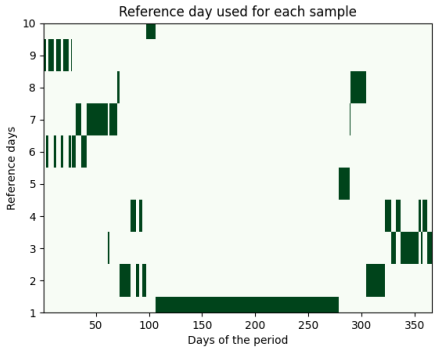


(b) Reference day corresponding for each sample

Figure 3.18: Weights and labels with 8 ref. days

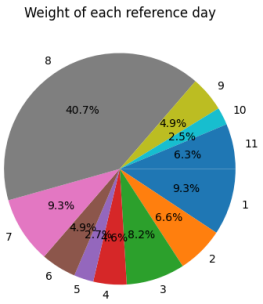


(a) Weights of each reference day

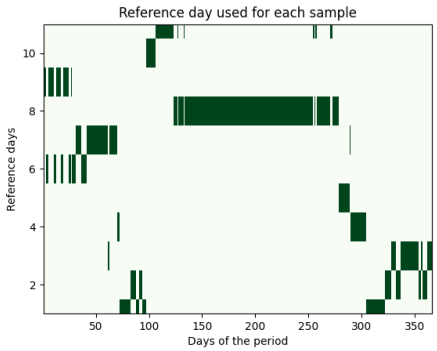


(b) Reference day corresponding for each sample

Figure 3.19: Weights and labels with 10 ref. days

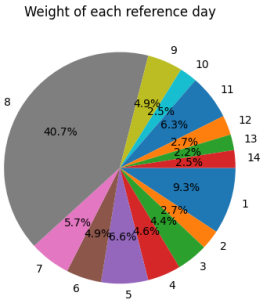


(a) Weights of each reference day

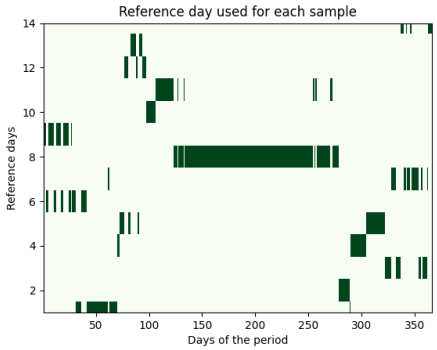


(b) Reference day corresponding for each sample

Figure 3.20: Weights and labels with 11 ref. days

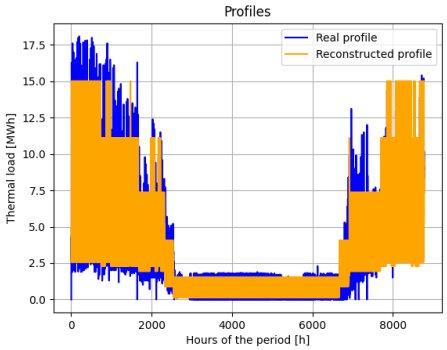


(a) Weights of each reference day

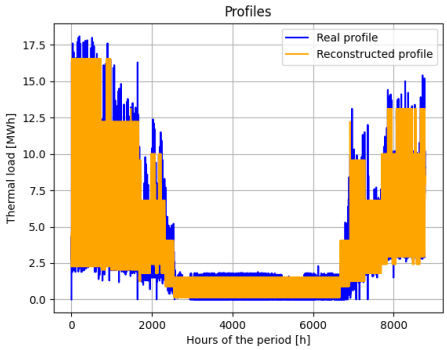


(b) Reference day corresponding for each sample

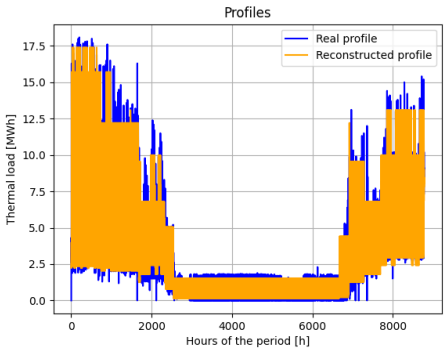
Figure 3.21: Weights and labels with 14 ref. days



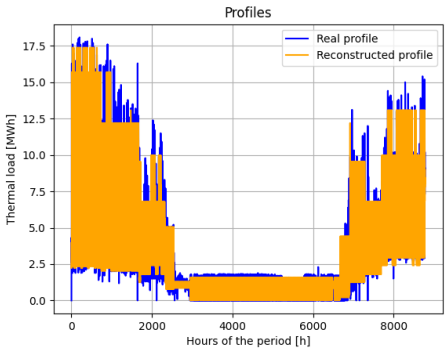
(a) 5 ref. days



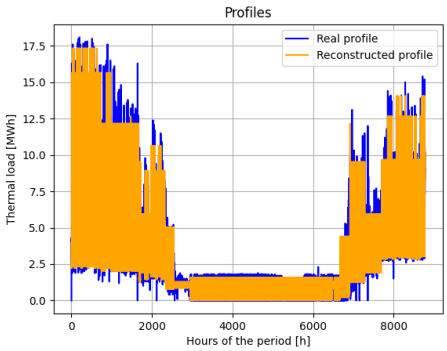
(b) 8 ref. days



(c) 10 ref. days



(d) 11 ref. days



(e) 14 ref. days

Figure 3.22: Profiles using the Ward clustering with different number of clusters

In fig. 3.22 the reconstructed profiles are shown for the different number of clusters. Higher is the number of reference days, more accurate is the annual thermal demand profile.

- The 8 reference days profile (b) in comparison with the 5 ones (a), approximates better the high load of January days, that instead with 5 clusters were represented by the same cluster of the December load. Also in correspondence of October (around hour 7000) the profile is approximated in a best way;
- The 14 reference days clustering (e), with respect to the 11 day one (d), increases the accuracy especially during the March-April period (around hour 2000) and again in the last part of the year;
- Profiles realized with the use of 8, 10 and 11 reference days are not distinguishable. In spite of the very different index scores that are associated, they seem to return very similar results, at least on the point of view of the annual thermal demand profiles.

Since the 8 reference days clustering returns results similar to the ones obtained optimizing the error indexes, the comparison with the k-means can continue with this number. In fig. 3.23 are illustrated the profiles of the 8 reference days.

The similarity with the graphs 3.4 is very high. It seems that, with different numeration, the two algorithms return twin centroids, in such a way that each centroid of Ward could be paired with the correspondent derived with k-means. For example, the first in the Ward case is very close to the fifth with the k-means clustering with 8 clusters (they are the ones with the highest demand). In table 3.2 are reported the coupled reference days.

Ward ref. day (fig. 3.23)	k-means ref. day (fig. 3.4)
1	5
2	1
3	8
4	6
5	4
6	3
7	2
8	7

Table 3.2: Matching between the 8 reference days resulting with the Ward and the k-means algorithms

How it is possible to note from figure 3.24, also the duration curve and the bin error associated to it are pretty similar to the ones obtained with the k-means clustering at fig. 3.7 and fig. 3.9(a).

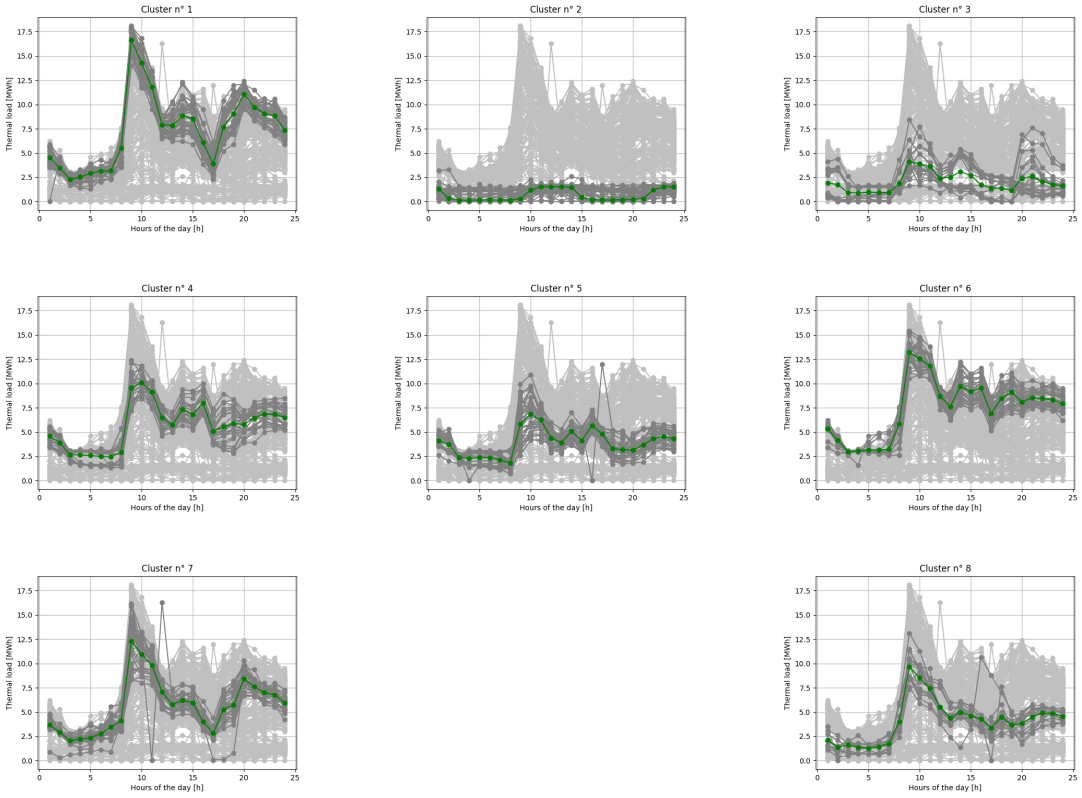
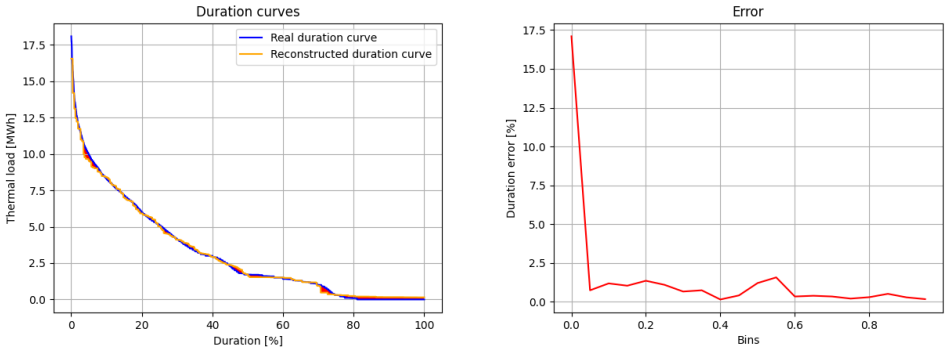


Figure 3.23: Reference days profiles for Ward with 8 clusters



(a) DC curve **(b)** Error associated to the DC curve

Figure 3.24: DC curve and its error with Ward 8 ref. days

In spite of the very different logic and the diverse way of operation, the two algorithms, Ward and k-means, returned very similar results. Since no evident difference have been noticed, the run of the optimization tool will be executed only with different number of reference days obtained with the k-means.

Chapter 4

Simulation with the reference profiles

4.1 Preliminary steps

To run the optimization algorithm XEMS, it is necessary to do some preliminary steps, so the last part of the code *"kmeans_pp"* builds the csv files needed by the software.

- The hourly thermal demand for each reference day in the proper csv format is built by calling a function at the end of the code. From the profiles of the obtained reference days it builds as many csv files as the number of reference days;

- The hourly electrical load is obtained from the thermal demand, taking the 3.5% of it. Since it is due mainly to the pumps work, this percentage has been recognized as effective, [24];

- The hourly prices of the sold electrical energy are derived from the GME database. At each reference day the price profile of its most similar real day in the year has been associated. The real day closest to each reference day is found exploiting a scikit function that calculates the root mean squared error RMSE between two lists. The closest day is the one for which the function returns the lower value. Then the prices for that day are extracted from the csv database with another function "ad hoc" built. They are the PNord prices;

- The prices of the purchased electrical energy are derived by summing 100 €/MWh to the prices of the PUN profile for the correspondent day and hour, as suggested by EGEA [25].

To create all these profiles the code calls two functions contained in the code *"Create_csv"*.
It contains also the functions needed to build the correspondent profiles for the case with 14 weeks. If they are called, the prices and the thermal and electrical demands are taken for the first seven days of all the months and, for April and October, also for the first seven days of the second half of the month.
Also the creation of the netlist has been automatized at the end of the code. Finally a text information file is build containing necessary data for the simulation post-processing.

In this section the XEMS optimization has been run for different cases and the results are then compared:

- 5 reference days;
- 8 reference days;
- 9 reference days;
- 20 reference days;
- 14 representative weeks, as done routinely by now by EGEA;
- All the weeks of the year.

The cases with 8 and 9 clusters have been chosen because are the ones analyzed in the previous paragraph, a case with a lower and a higher number of clusters are added for further considerations. The case with the 14 representative weeks is important because allows a comparison with the actual EGEA procedure.
To make the comparison more effective, a case as accurate as possible should be plotted. So the XEMS simulation has also been launched for all the periods of the year. Since the optimization can not run for periods too long, each month has been divided into four periods (the first three weeks and the remaining days, for a total of 48 periods) and then all of them are simulated. This is the case that obviously will approximate in the best way the real results, but it is also very computational heavy, so it is done only for a research purpose. The thermal load profile, which is the one used also in the previous paragraph, represents the demand of Carmagnola.

4.1.1 Test case

Since for this city the xml components file wasn't already built, it has been created. It is a file which represents the operation of the components utilized. The city of Carmagnola has two cogenerators of around 1500 kWe and three boilers (two of 5000 kWth and one 10000 kWth), which are the components that have been implemented. The operations at partial and full load are derived from the datasheet

or provided by EGEA, [26] [25]. The only value that is not provided at partial load is the thermal power provided by the CHP. To calculate it, it has been hypothesized that the thermal efficiency at 75% of load is higher of 2% with respect to the full load case, and that at 50% is 4% higher. This relationships, in fact, have been often encountered in these types of components. Boiler are assumed to have the same efficiencies also at partial loads, , [24].

Other parameters to set are the minimum on time (minimum hours that the component must be on when started) and the minimum shut-down time (minimum hours that the component must be off when shut down). They are imposed to avoid too frequent on-off alternations that could damage the component. For example, this could happen if the price of electricity decreases for an hour and it is more economically convenient to shut down.
For the cogenerators they are set to 4 and 2 hours respectively, for the boilers both to 1, because they can withstand more state changes.

The values of the others parameters, like the emissions, are taken from other similar components in already existing xml files used by EGEA.
Another relevant quantity set in the xml file is the so called *"etoVdef"*, a parameter used to calculate the volume of the tax-free gas for the cogenerators. It is derived by multiplying the parameter to the produced electrical energy. When the electrical efficiency overcomes the value of 46%, instead, all the gas is tax-free, but it is not the case. *"etoVdef"* is 0.22, [24].

Once completed these preliminary steps, it has been possible to run the simulations for all the different cases.

4.2 Optimization and post-processing

The first difference that emerges is the computational time needed for each case. In table 4.1 all the timeframes are resumed.

Case	Hours simulated by each netlist	Total time
5 reference days	24	22”
8 reference days	24	38”
9 reference days	24	40”
20 reference days	24	1’20”
14 weeks	168	6’40”

Table 4.1: Computational times needed by XEMS in the different cases

The simulation of one day requires about 4÷5 seconds, so obviously the case with 14 weeks, that simulates a total of 98 days, is the one that requires more time.

This difference can be not so relevant if a simple simulation is done, but it is crucial when parametric simulations are run, for example when a component is repeatedly substituted to determine the best configuration.

In order to adequately compare the optimization results, two post-processing codes are built: one for the case with 14 weeks and another one for the case with a defined number of reference days.

The first thing that the code *"Post_processing_14_weeks"* does, is asking for the fourteen weeks xml files that contain the results of the optimization. If the number of files provided is correct, the procedure continues. In the same folder with the xml (so, the folder "Work", which is the one containing also the netlists), it creates a csv file that resumes all the annual results, called with the name of the first xml file plus the caption *post_processing*.
In order to produce it, each xml file is parsed and the most relevant hourly quantities are summed to obtain the total results for the week. The calculated values are the weekly: thermal, electrical and feed energy of the CHPs, thermal energy produced by the boilers, thermal load, dissipation, total energy entered in the storage, emissions and objective function results.
To get the monthly results, the quantities of each week are divided by seven and multiplied by the number of days of the correspondent month. The only exception is for April and October, having two representative weeks. The results of the two weeks are summed and then divided by fourteen and multiplied by the number of days of the month. Thermal, electrical and global efficiencies are calculated using the obtained correspondent monthly energy values.
In the csv all the properties are written for each month and at the end of the file there are also the annual results. In the annual results row are present also the RISP (Risparmio di Energia Primaria) and the PES (Primary Energy Savings), that are useful for considerations about the white certificates, done in the next chapters. At the end a plot of the energy balances is shown, with at the x-axis the months of the year. There is a plot for the thermal energies and one for the electrical ones.

The code *"Post_processing_ref_days"* is the code built to post-process the result obtained by the XEMS optimization of the reference days of an year.
As first thing, a text information file is required. It is build automatically in the folder "Work" at the end of the *"kmeans_pp"* code, and it is called *"Info_"* plus the number of clusters selected. It contains the number of days of the year, the number of reference days selected, the label of each sample day and the weight of each reference day. All these information are used later by the code to build a csv post-processing file similar to the one built for the 14 weeks case.
Afterwards, the xml results files are requested and, if their number is equal to the number of reference days, the procedure continues. All the quantities already

mentioned are calculated for each reference day. Subsequently, they are multiplied with the correspondent weight and summed to obtained the annual results.
To get the monthly results, the labels vector has been used. At each day of the year are associated the quantities calculated for its correspondent reference day. To obtain the values of each month the correspondent daily values are summed.
The results for each reference day, the annual results and the monthly results are all written in the csv post-processing file built in the netlist folder.
Also in this case a plot of the desired quantities can be shown.

The table 4.2 reports the annual results obtained with the different simulations (the value of the total energy entered in the storage is calculated in the code, but not reported here since the storage is not present in Carmagnola). Next to each value there is the error percentage with respect to the "all weeks" case.

Case	Thermal CHP [kWh]	Electrical CHP [kWh]	Fed energy CHP[kWh]	Dissipation [kWh]	Boiler thermal [kWh]	Thermal load [kWh]
All weeks (48 periods)	14005757	13316428	31473464	922648	14918553	28000820
14 weeks	14554144 +3.9%	13910690 +4.5%	32831598 +4.3%	1109301 +20.2%	15317120 +2.7%	28761963 +2.7%
5 ref. days	12818679 -8.5%	11964931 -10.1%	28420240 -9.7%	147990 -84.0%	15330131 +2.8%	28000820
8 ref. days	13864006 -1.0%	13339780 +0.2%	31419443 -0.2%	240358 -73.9%	14377172 -3.6%	28000820
9 ref. days	13751266 -1.8%	13194155 -0.9%	31102012 -1.2%	104333 -88.7%	14353887 -3.8%	28000820
20 ref. days	13878598 -0.9%	13265413 -0.4%	31303234 -0.5%	247112 -73.2%	14369334 -3.7%	28000820

Case	Global efficiency	Thermal efficiency	Electrical efficiency	CO_2 emissions [kg]	Obj. function [€]	RISP [kWh]	PES
All weeks (48 periods)	0.838787	0.445002	0.4231	3924884	623795.5	12012081	0.276232
14 weeks	0.833207 -0.7%	0.443297 -0.4%	0.423698 +0.1%	3941559 +0.4%	623030.9 -0.1%	12347747 +2.8%	0.273305 -1.1%
5 ref. days	0.866834 +3.3%	0.45104 +1.4%	0.421 -0.5%	4052219 +3.2%	677434.4 +8.6%	11669024 -2.9%	0.291076 +5.4%
8 ref. days	0.858177 +2.3%	0.441256 -0.8%	0.424571 +0.3%	3953236 +0.7%	620476 -0.5%	12717466 +5.9%	0.288137 +4.3%
9 ref. days	0.863002 +2.9%	0.442134 -0.6%	0.424222 +0.3%	3937046 +0.3%	627280.5 +0.6%	12744193 +6.1%	0.290657 +5.2%
20 ref. days	0.859237 +2.4%	0.44336 -0.4%	0.423771 +0.2%	3944221 +0.5%	639801 +2.6%	12680716 +5.6%	0.288303 +4.4%

Table 4.2: Comparison of the annual results

In general the results are quite similar for all the cases, but at first view it seems that the case with the 14 reference weeks and those with the highest number of clusters approximate better the real results.
The objective function that XEMS minimizes, which is the cost, in some cases results lower than the actual. This is not because a better solution is found, but due to an imperfect approximation of the prices profiles.

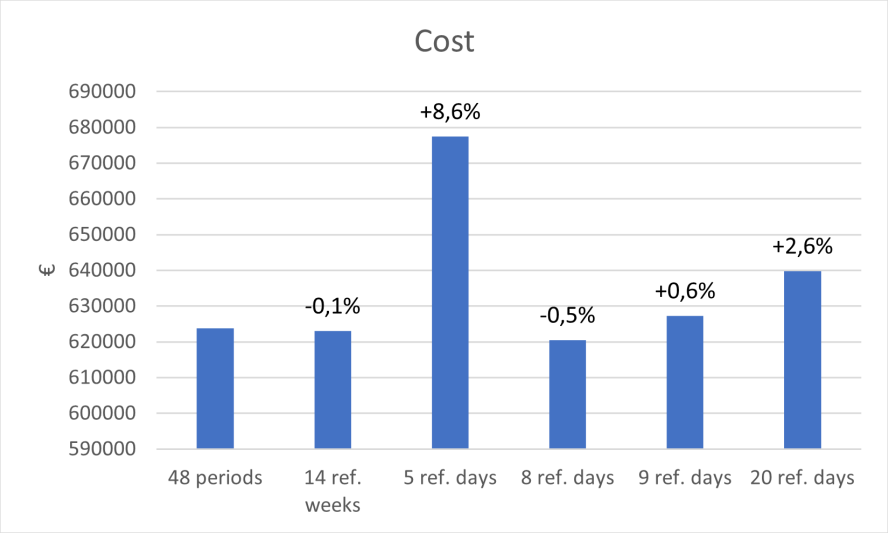


Figure 4.1: Comparison of the objective function in all the cases

In fig. 4.1 it can be noticed that, except for the 5 cluster cases, the costs are closed to the real ones.
To understand in a clearer way the differences between the annual solutions, an Excel graph has been realized for the annual energy values comparison.

How it is possible to see from fig. 4.2, the annual thermal loads found with the reference days are equal to the real one. For the reference weeks case, instead, the value is slightly different (+2.7%). This is because in the clustering code it has been imposed, instead, if the simulations are run for the first days of each month, it can not be imposed.
For the other values it seems that the case with 5 reference days returns the worst results, quite different from the real ones. The other cases obtained with the clustering, instead, does not show big differences on an annual level, and especially the 8 days case return results very close to the real annual ones, meaning that the indicators used to choose the number of reference days on the previous paragraph are valid.

It can be also interesting to see the monthly behaviours in all the cases, so the plots realized with the Python codes are reported.

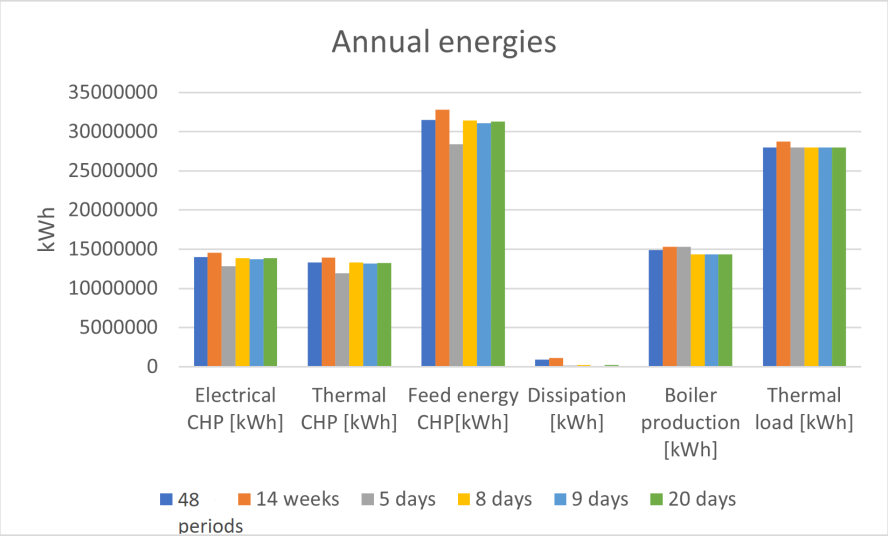


Figure 4.2: Comparison of the annual energies exchanged in all the cases

For a clearer comparison, some Excel graphs have been also developed to compare the monthly results of thermal load, thermal energy produced by CHPs and thermal energy produced by boilers.

As shown in fig. 4.3 and fig.4.4, the clustering-derived profiles are more similar to the real ones than the profiles derived from the 14 weeks simulation. There are some months that are not represented well by their first week, as March, July and November when there are relevant differences with the real case, both in the thermal load profile and in the optimization results.
For the clustering cases the thermal load is always represented very well, but the other values, resulted from the optimization, have some small differences. This can also be due to the prices profiles used, which are the ones of the day closest to each reference day. Maybe a solution could be clustering more temporal series together, but the results are quite satisfying also with this approximation.

The electrical energy balance in fig. 4.5 does not add many relevant considerations. The shape of the electrical load has been derived from a percentage of the thermal one, so it replicates it in a smaller scale. Almost all the electricity produced by the cogenerators is sold to the grid, being the demand to satisfy very small. No electricity is purchased in any case.

Resuming, some considerations can be done:

- The clustering cases in the results do not show the same July increment in the CHP production as the real simulation. This is due to the lower representation of the summer days among the reference days.

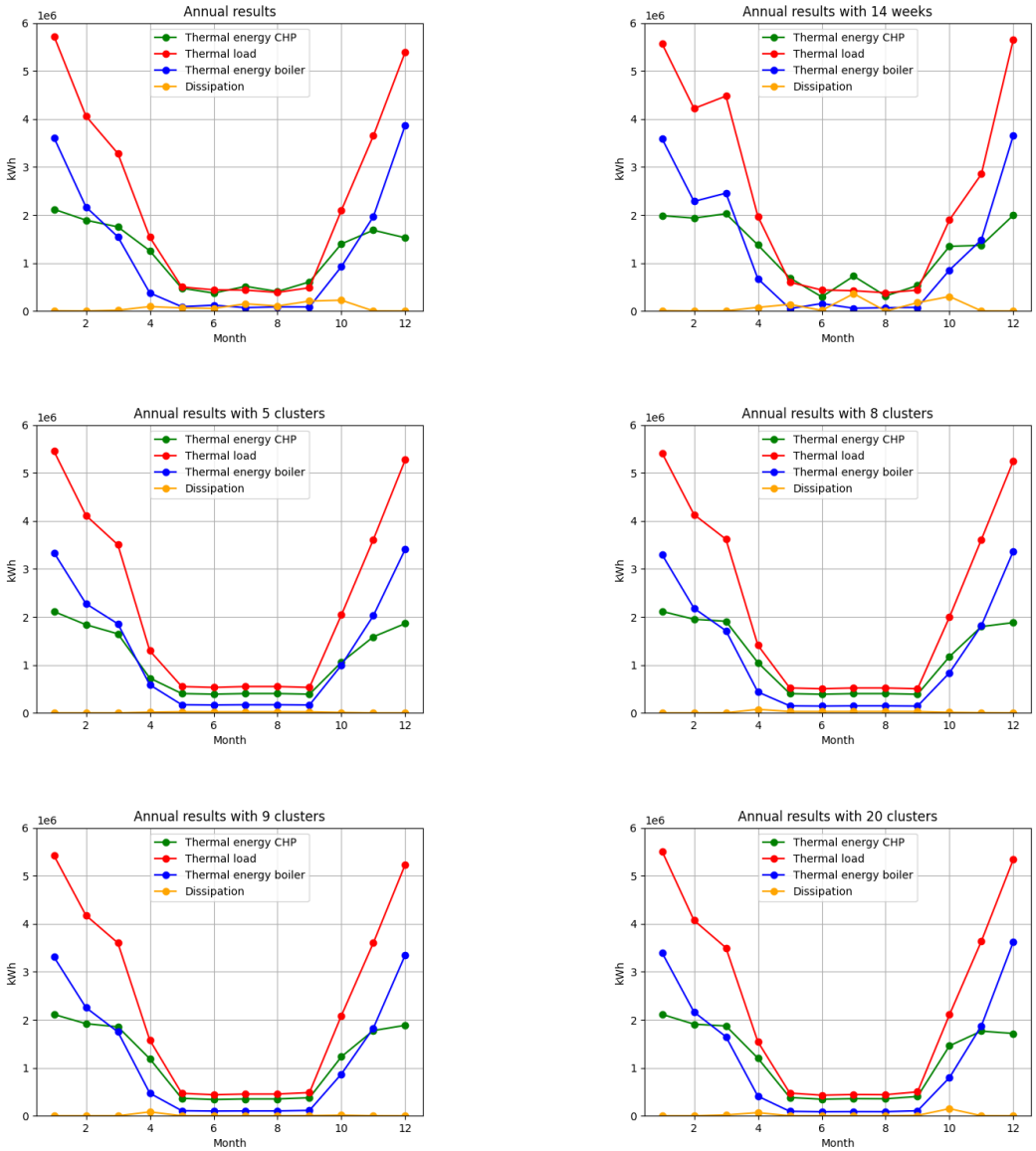


Figure 4.3: Comparison of the thermal production

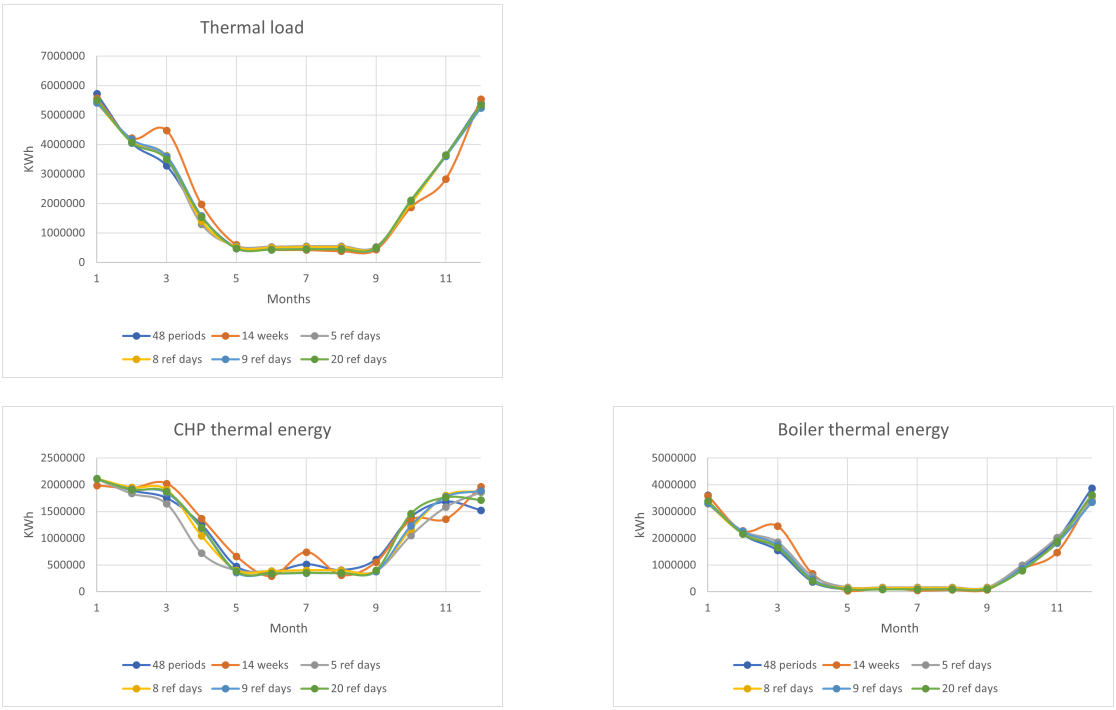


Figure 4.4: Comparison of the thermal production

- A month that is not approximated very well in all the cases is December. In December the best solution for the real case is to diminish the CHP usage and increment the boiler production. This trend is well caught only by the 20 reference day case: in this case the days of the month for which this behaviour is the optimal one are represented by an own cluster.
- In the 14 reference weeks case there are some spikes in the graphs: representing an entire month by only using its first seven days is risky.
- The combined usage of the Inertia, Davies-Bouldin and Silhouette scores is a valid indicator in the choice of the number of clusters: the 8 days reference case is a fair approximation of the real one.

In conclusion, it seems that the usage of reference days allows to reduce the computational time, and can provide an improvement of the results of the simulations with respect to the 14 weeks case, if it is done with the proper number of clusters.

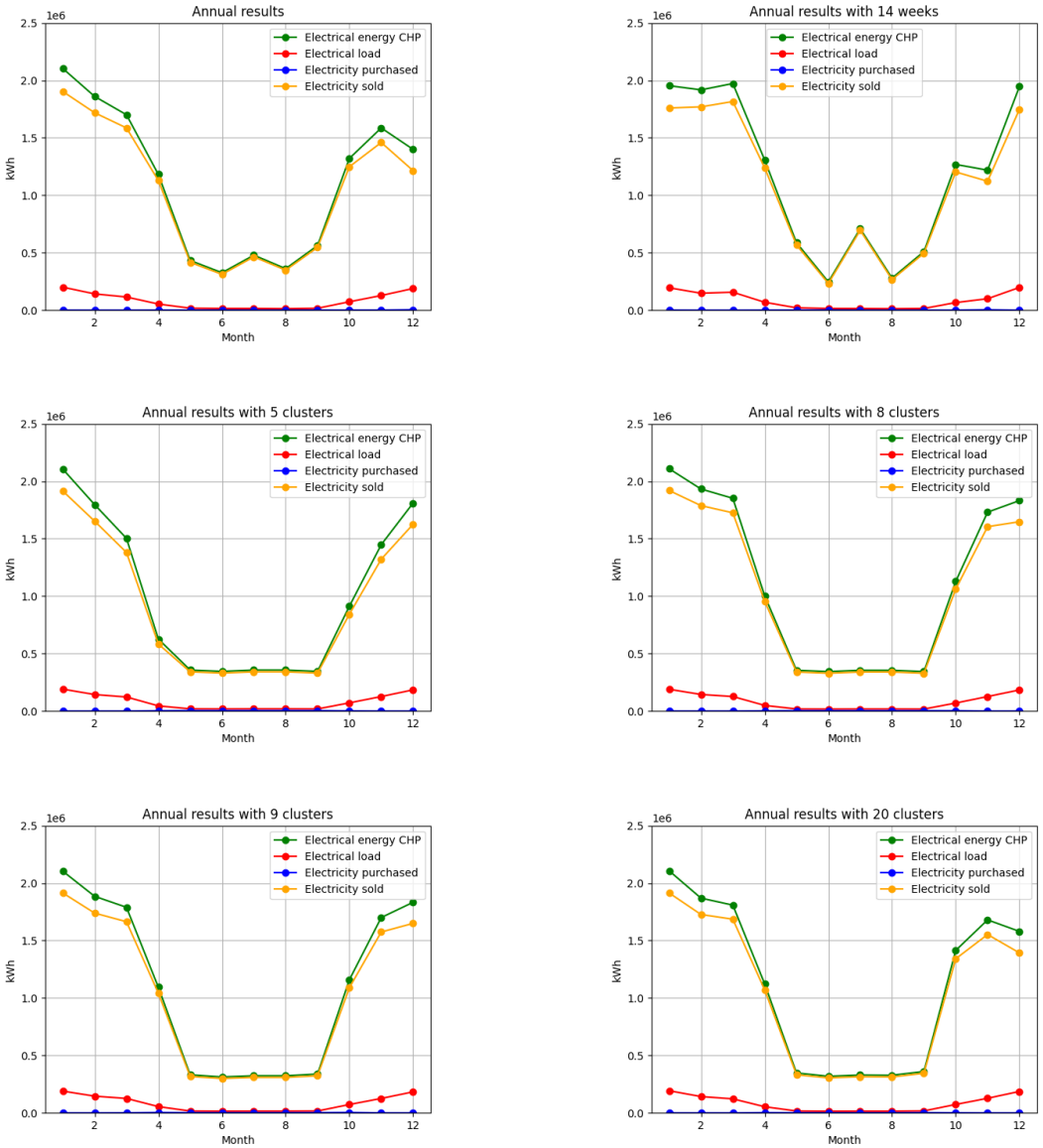


Figure 4.5: Comparison of the electricity balance

Chapter 5

Implementation of the white certificates

The aim of the procedure is to consider white certificates and their effects inside the simulation, in order to make the optimal solution take them into account. Earning from white certificates, in fact, depends on the useful energy produced and, to take into account white certificates effects, it is necessary to implement an iterative procedure that starts the optimization, uses the results to calculate white certificates and re-starts the optimization with the new input data that depend on them, until convergence is found.

In order to consider earnings from white certificates, it is necessary to distinguish the amount of thermal energy produced that is effectively utilized (useful) from the amount that is dissipated. The final saving is in fact dependent on a parameter β, which expresses the ratio between thermal power that is dissipated by the plant and the total produced. [27].

The easier way to insert the economical savings derived from white certificates, is to modify the maintenance cost of the components. As a consequence, the simulation will return an optimal result different from the previous and, for this reason, an iterative procedure is necessary. It re-starts the simulation until the difference with the previous results is lower than a tolerance or the maximum number of iterations is reached.

The procedure is described in this chapter.

5.1 Procedure

The aim is to create a Python program that, as first thing, starts the XEMS13 executable file. During the first iteration it is necessary to ask for the folders

"Work", "Profiles" and "Components". Subsequently it is appropriate to modify the configuration file in a way that for the following iterations the folder are not requested anymore, since their paths are already been saved.
Once XEMS results are obtained, these are used to get the savings derived from white certificates. Following, in the xml file that contains the components characteristic data, the maintenance values of cogenerators are modified, in order to take into account the savings. Thereby the simulation can be repeated, thanks to a while loop that ends when the difference between the results of the previous iteration is lower than the selected tolerance or when the maximum number of iterations is reached.

Following, a flow diagram that shows the described procedure.

The Python code which implements the procedure can be divided in four parts:

- The main code, called *"CB"*, that calls all the others. It is the one that launches the XEMS13 main for the first time, saves the real maintenance values, changes the configuration file in a way that netlist, profiles and components folders are not requested anymore, and starts the iterative loop;
- *"Find_in_netlist"* it supports CB when it is necessary to read the netlist text file, that contains information about the simulation. It has two functions, one is able to find the name of the component file and the other is able to find all the cogenerators to utilize for the simulation;
- *"Result_for_CB"* it contains the function that is called to read the results of the simulation from the correspondent xml file. It returns values necessary to evaluate the outcome of the procedure;
- *"Modify_maint* it contains the function which rewrites the updated values of the maintenance in the components xml, in such a way that the iteration *i+1* considers earnings from white certificates, calculated with the results of the iteration *i*.

5.1.1 CB

The code must be in the same folder of the executable file.
As first thing inside the code, the maximum number of iterations is set. It is set to 6, this means that after the fist simulation, maximum other 5 simulations will be run and then the procedure will stop, even if it has not yet arrived to convergence. So, the first simulation is started.

Subsequently, the configuration file *"XEMS13cfg.txt"* is re-written, in such a way that to the indication "dialog" corresponds the value "0". This file contains setup information and this setting avoids the request of the folders at the next

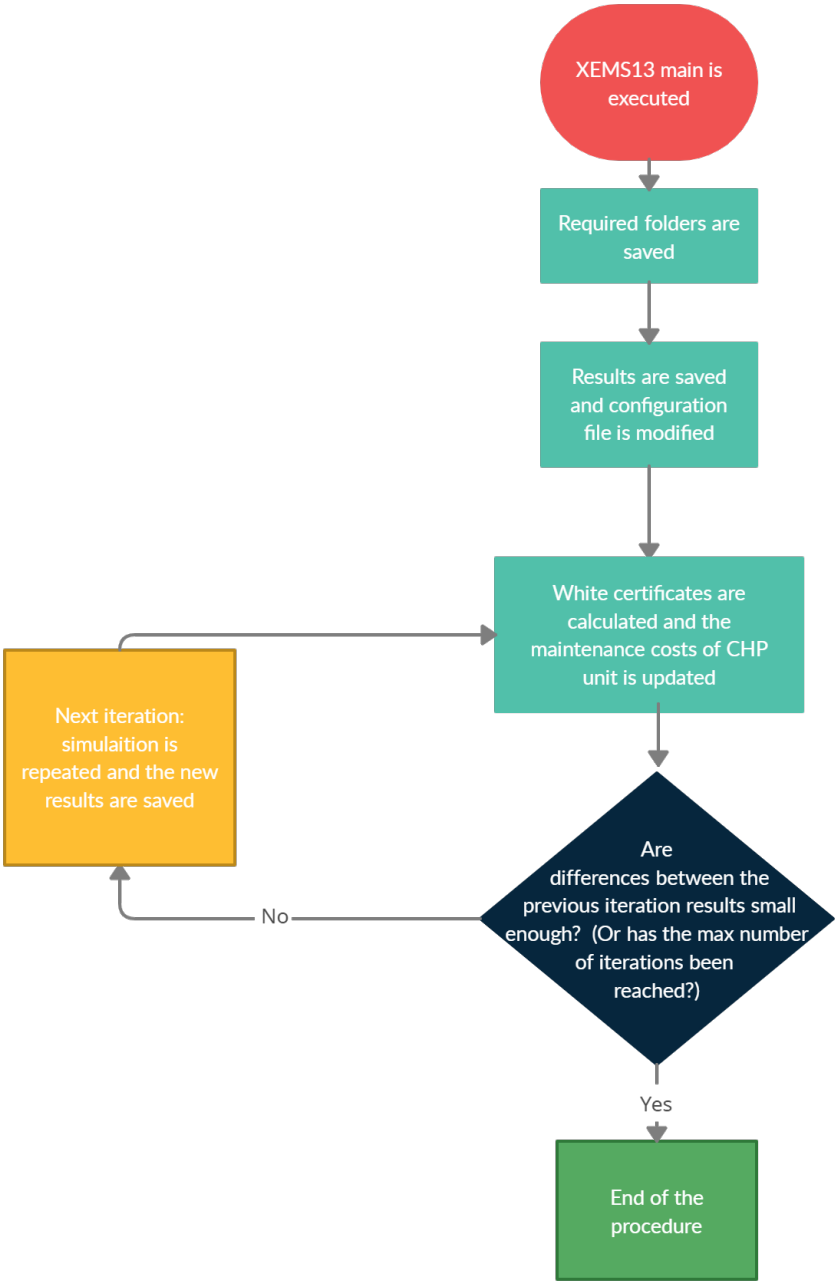


Figure 5.1: Iterative procedure flow diagram

XEMS executions, since their paths are already contained in another text file, *"defaultDIR.txt"*, that is read immediately after. The latter contains also the netlist name with the extension txt, that corresponds to the file name from which all the other files generated by XEMS take information.
The homonym file with the extension .xml contains instead the simulation results. All these information are useful and are saved.

These information are immediately used in order to call the functions contained in *"Find_in_netlist"*. Path and netlist file name are given to the functions, that return the name of the component file or all the cogenerators present in the simulation. More precisely, the function that regards the cogenerators, creates two lists, one containing all the modality (CHP, CHPLE, CHPS) and the other containing the size of each unit (ex. "CHP_1200").

Following, it is possible to move to the components folder and read the xml file, since its name has been just obtained. The parse of the xml file is done to read the maintenance of the cogenerators present inside the lists and another list is created, which contains all the original values, not yet modified.

Subsequently the function "results", contained in *"Results_for_CB"* is called. It requires the path of the folder with the executable and the initial maintenance and it returns the values updated of the maintenance that take into account the earnings from white certificates associated to the obtained results. It returns also a series of other results, among which the coefficient β and the RISP (in Italian "risparmio di energia primaria"), that are used inside the iterative cycle. The other results are useful for afterwards considerations.

Following the csv file with the iterations results is created, which name is composed by the simulation name plus the Italian statement *risultati_iterazioni.csv*. Each parameter name is written in a different column, then the file will be updated at each iteration with the correspondent values.

The loop can start. It is a while loop based mainly to the control of the difference of the coefficient β between one iteration and the previous. When "diff_beta" is less than 0.01 the loop is interrupted. An absolute difference has be chosen instead of a relative one because for very small numbers a negligible variation would result as too influential and the difference would be higher than the tolerance.
However, in the complex cases that consider also a thermal storage, the total dissipation is null or very near to 0, and so the same for β. As a consequence, the loop would end after the first two iterations. To solve the problem, an additional control had been added. It is performed only if β value is very near to zero. In this case, the RISP is controlled and if the relative difference between two consecutive iterations is less than 1%, the loop is interrupted. Otherwise the next iteration will be performed, since "beta_diff" is appositely set to 1.
A further control is done on the maximum number of iterations: if it is reached,

the loop is exited.
The first time the loop is executed, the executable file is not launched and results are not searched with their function, because they are already been saved. Instead, the function *modify_maint* is called. It requires the components file name, the cogenerators lists and the "new" maintenance values. It updates the values inside the xml components file, in a way that they will be used in the next iteration. For all the following iterations, instead, before this step, the XEMS main is executed and the new results are saved.
Finally, for all the iterations, saved results are written in the csv relative to the iterations results.

In the end, out of the while loop, the configuration file is re-written in such a way that at the next executable start, folders are requested again.

5.1.2 Find_in_netlist

This subcode contains the definition of two functions, both with the aim to analyze the netlist text file. Both require as input its name and its folder path (the folder "Work"). The functions are:

- *Find_components_file_name*: searches for and saves the name of the components file;
- *Find_CHP_name*: searches and saves the rows containing information about the cogenerators.

Find_components_file_name reads the rows of the netlist text file and compares them with the stripe *"@XML Library Components File"*. When the read line corresponds to the stripe, the number of the following line is saved together with its containing, because it is the xml component file name.

Find_CHP_name reads the rows and compares them to the stripe *"@Dispatchable Electric Input"*. When they are equal, it saves the number of all the successive rows until there is a void row. Each saved line contains information of one cogenerator.
It is important to note that actually there are two stripes *"@Dispatchable Electric Input"* inside the text file. First time it is followed by information regarding the grid and the second time about cogenerators. Relevant information are the ones about cogenerators, as a consequence, the fact that what follows the statements the first time is overwritten by what follows it the second time is not a problem.
Inside each line referred to a cogenerator, there are multiple information. The saved ones are the modality (CHP, CHPLE o CHPS) and the typology. So, two lists are created, each one with the correspondent element for each cogenerator.

5.1.3 Results_for_CB

This subcode contains the function *results*, which has as input the folder with the main and the original maintenance list. The output is a series of results.

As first, the file *"defaultDIR.txt"* is re-opened in order to read the folder "Work" path and the simulation name. This time it is used to build the name of the xml results file, that is also situated inside the folder "Work".
This file is opened and parsed. All the hourly values of thermal power produced by all the cogenerators are summed, and total thermal energy produced by all the cogenerators is obtained. The same procedure is executed for electrical energy, feed energy, dissipated thermal energy and total thermal energy that enters in the storage. Also the value of the global emissions is read. It is present in the xml only if the optimization is "ECOENVI". Otherwise, if the optimization is "ECO" only, the emissions value is reported as "undefined". The dissipation coefficient β, so defined, is calculated:

$$\beta = \frac{Dissipation}{Thermal\,energy\,produced} \tag{5.1}$$

Following thermal and electrical efficiencies as calculated:

$$\eta_{thermal} = \frac{E.\,thermal\,produced}{E.\,feed} \tag{5.2}$$

$$\eta_{electrical} = \frac{E.\,electrical\,produced}{E.\,feed} \tag{5.3}$$

Global efficiency:

$$\eta_{global} = \frac{E.\,electrical\,produced\,+\,E.\,thermal\,useful}{E.\,feed} \tag{5.4}$$

It is important to note that in the global efficiency there is not the total thermal energy produced, but rather the useful energy, calculated by subtracting from the total the dissipation.
Following the RISP is calculated:

$$RISP = \frac{E.\,thermal\,useful}{\eta_{thermal,\,RIF}} + \frac{E.\,electrical}{\eta_{electrical,\,RIF}} - E.\,feed \tag{5.5}$$

and the indicator primary energy savings, which is the dimensionless expression of the primary energy saving realized with the cogenerative plant, with respect to the traditional separated plants, [28]:

$$PES = 1 - \frac{1}{\frac{E.\,thermal\,useful}{E\,feed\cdot\eta_{ter,\,RIF}} + \frac{E.\,electrical}{E\,feed\cdot\eta_{ele,\,RIF}}} = \frac{RISP}{\frac{E.\,thermal\,useful}{\eta_{thermal,\,RIF}} + \frac{E.\,electrical}{\eta_{electrical,\,RIF}}} \tag{5.6}$$

Finally the earning from white certificates can be calculated as:

$$c_{CB} = \frac{a \cdot CB_{valore,euro}}{\eta_{electrical}} \cdot \left(\frac{\eta_{electrical}}{\eta_{electrical,RIF}} + \frac{\eta_{thermal} \cdot (1-\beta)}{\eta_{thermal,RIF}} - 1 \right) \tag{5.7}$$

The value is expressed in €/MWh, as a consequence it must be divided by 1000 and then it can be subtracted to the original maintenance values, that are expressed in €/kWh. So a list is created, in which each element is equal to the correspondent element of the real maintenance list, minus the value c_{CB}.

Values returned to the main code are: updated maintenance list, thermal and electrical produced energies, useful energy, feed energy, total dissipation, total energy entering the storage, the three efficiencies, RISP, PES and global emissions.

Formulation and reference values are taken from [27].

5.1.4 Modify_maint

In this subcode there is the homonym function. It needs the path of the folder with the executable, the name of the xml components file, the two lists regarding the cogenerators and the list with the updated maintenance values. The aim is, in fact, to modify the old maintenance values in the xml, substituting them with the updated ones of the list.

The function does the parsing of the components xml. Subsequently, for each cogenerator inside the lists, it searches the one with the same modality and typology inside the xml and substitutes its maintenance value.
It is important to note that, in a simulation, it is possible to have two identical cogenerator units, so with the same indicators in the correspondent lists. For example, in the first complex case there are two unit called CHPLE_4401. In the xml, the maintenance value referred to this typology will be substituted twice. Nevertheless, this is not a problem, since inside the updated maintenance list the value are necessarily the same. In fact, as already said, both are calculated subtracting from the original maintenance (that must be equal because it is referred to the same typology) the value c_{CB} (equal for all the cogenerators).

It is important to underline that in the components xml file at each iteration all the maintenance values are updated. As a consequence, at the end of the simulation, the file will be modified and it will not have the effective values anymore. If the procedure must be repeated, it is necessary to substitute the modified file with another with the original values.

5.2 Results

The csv document related to the iterations results is created in the folder "Work", that contains already the netlist and the other csv results. Following, the results of some simulations are presented. At the end of the section the tables with the iterations results are reported for the four analyzed cases.

5.2.1 Simple cases

Two simulations have been performed in "easy" mode, meaning with only one cogenerator, without heat recover at low temperature, without thermal storage and of a single day.
First simulation is of a January day, the second of a March day.

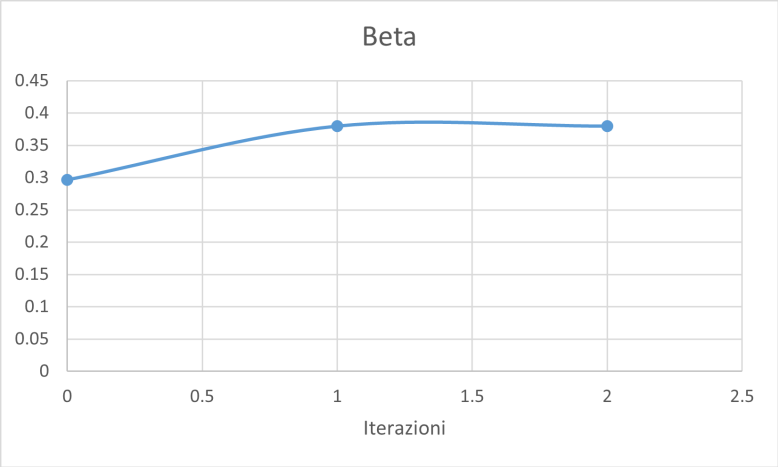


Figure 5.2: β convergence in the first case

First case needs to launch XEMS13 only three times. After the first implementation of the white certificates, β settles down to a value around the 38% and the value still the same for the next iteration. The parameter increases with respect to the first simulation, in which it resulted 29%. This because all values of thermal, electrical and feed energy increase. As a consequence, keeping unchanged the demand and increasing the heat production, the dissipation and β necessarily rise. With a lower maintenance value it worth it to produce more, until a dissipation of the 38% of the thermal energy produced. The global efficiency decreases from 75% to 71%. The RISP rises of 271 kWh (about +2%), instead, the primary energy savings PES decreases from about 22% to 19%, remaining by the way over the threshold value of 10%. This can be explained with the fact that the PES numerator, that coincides with the RISP, increases, but the denominator (at which

the feed energy is not subtracted), increases even more, causing a PES reduction. Feed energy, in fact, rises significantly (about +20%).

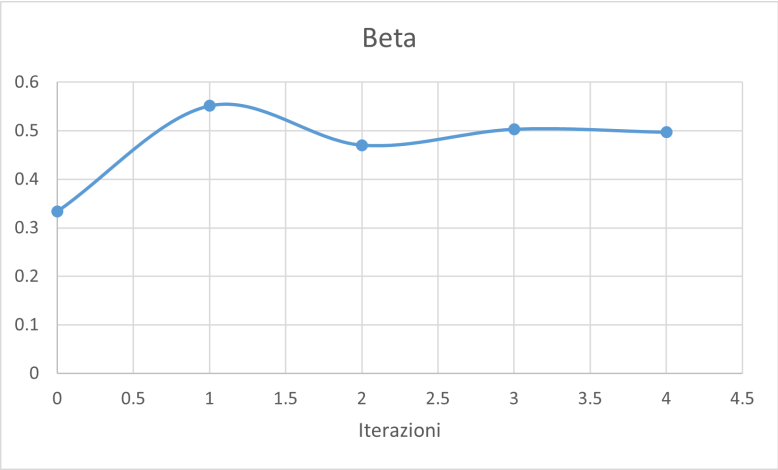


Figure 5.3: β convergence in the second case

March simulations need XEMS13 launching five times. All the values oscillate, until they settle when β is around 50% (starting from a value of 33%). Also in this case the feed and produced energy increase and the efficiencies decrease. RISP rises of 4% and PES goes form 20% to 15%.

The results of these cases suggest a bigger economical advantage with a bigger productivity and, as a consequence, dissipation. One may think that on an environmental sustainability point of view, this leads to a disadvantage. However, how it is possible to note, global emissions decreases. With a more detailed analysis of the complete results of the simulations, in fact, it can be seen that the thermal energy produced by the boilers decreases (in the first case of the 11%, in the second one of the 40%). Accordingly, even if the total thermal energy produced increases, the emissions decrease, being the boilers more pollutants.

It is possible to see it clearly with a Sankey diagram, for example for the April case. The text file that allows to easily build it, is created in the folder "Work" by XEMS13 together with the other results.

With these type of plots in fig. 5.4 and 5.5 is highlighted the decrease of the boiler production (blue top flux) in favour of a higher cogenerators activity (orange bottom flux) on the left sources side.
On the right part with the final outputs, instead, a higher thermal energy dissipation and higher values of electricity sold to the grid can be noticed.

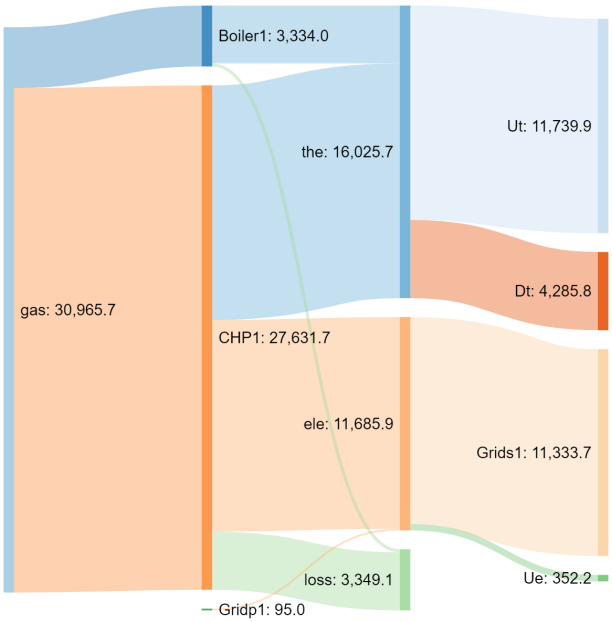


Figure 5.4: Sankey diagram without white certificates

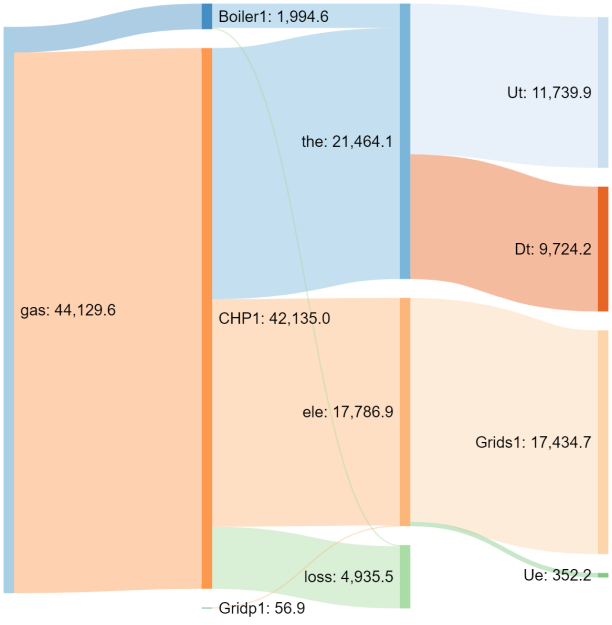


Figure 5.5: Sankey diagram with white certificates

5.2.2 Cases with storage

Complex cases simulations consider the use of more than one cogenerative unit with heat recover and a thermal energy storage. They simulate the activity of a week. Thanks to the heat storage presence, in these simulations, the dissipation is null or almost null and the results is a β value equal to 0. As a consequence, the parameter to confront is the RISP. Two simulations has been executed, one for a January week, and the other for an April week.

In the first case, the executable is started only twice. All the values, in fact, slightly variate after the first maintenance modification, RISP comprehended, which increases of less than 0.1%. Despite the rise of the thermal energy produced by the cogenerators, total energy stored decreases. It could seem a contradictory result, however comparing the complete results of the simulation with and without white certificates, it can be seen that, in the case in which white certificates are not included, thermal energy produced with heat pumps is higher. As a consequence, considering all the sources, total thermal energy decreases with the implementation of white certificates and for this reason also the storage usage decreases.
This is the only case in which emissions slightly increases (+0.6%). It could be due to the less smart storage utilization, however all these variation are not significant.

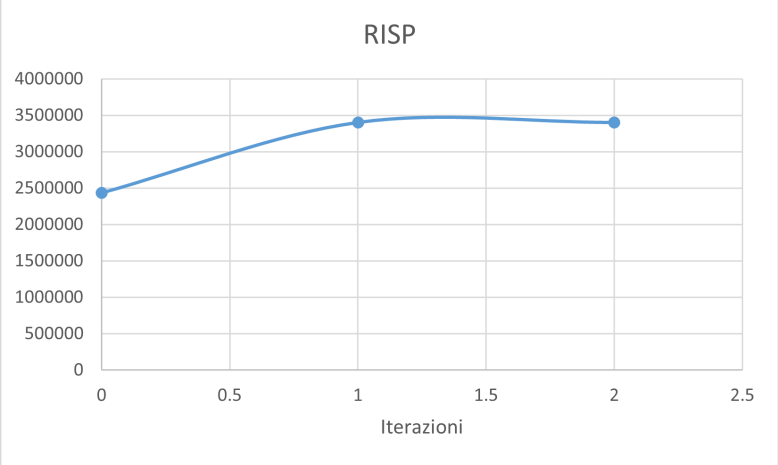


Figure 5.6: RISP convergence in the second complex case

In the April week simulation, three iterations are needed. The final RISP is bigger than the initial of almost 1 GWh (so, it increases of about 40%), while PES decreases only from 33.12% to 32.98%. The storage usage rises, with an increment of the total thermal energy introduced of almost 2 Gwh (+40%). It is important to note that also dissipation increases, and goes from being nonexistent to 26 MWh, a value that is, by the way, very low with respect to the produced thermal energy,

and that results in a β lower than 0.1%. Global emissions decreases of 26%.

1
Caso_1_C_feriale_1

Iteration	Dissipation [kWh]	Thermal energy [kWh]	Feed energy [kWh]	Electrical energy [kWh]	β	Energy entering storage [kWh]	Electrical efficiency	Thermal efficiency	Global efficiency	RISP [kWh]	PES	Global emissions [kg]	Maintenance (€/kWh]
0	7839.66	26420	56980	24040	0.296732	0	0.421902	0.463671	0.747888	15925.69	0.218442	6742.02	[0.01]
1	12030.27	31704	68376	28848	0.379456	0	0.421902	0.463671	0.709631	16196.74	0.191513	6530.595	[-0.005045519]
2	12030.27	31704	68376	28848	0.379456	0	0.421902	0.463671	0.709631	16196.74	0.191513	6530.595	[-0.002275105]

Table 5.1: Results for January simple case

2
Caso_1_C_feriale_3

Iteration	Dissipation [kWh]	Thermal energy [kWh]	Feed energy [kWh]	Electrical energy [kWh]	β	Energy entering storage [kWh]	Electrical efficiency	Thermal efficiency	Global efficiency	RISP [kWh]	PES	Global emissions [kg]	Maintenance (€/kWh]
0	4285.838	12858.09	27631.67	11590.84	0.333318	0	0.419476	0.465339	0.729709	7090.517	0.204207	2490.657	[0.01]
1	13113.15	23778	51282	21636	0.551482	0	0.421902	0.463671	0.629867	7602.62	0.12911	2152.771	[-0.00389303]
2	8582.363	18248	39286	16528	0.470318	0	0.42071	0.464491	0.666742	7384.032	0.158218	2290.129	[0.00201968]
3	9970.189	19815	42735	18030	0.503164	0	0.421902	0.463671	0.652271	7399.331	0.14759	2311.34	[-0.00014629]
4	9724.189	19569	42135	17790	0.496918	0	0.420779	0.464436	0.654144	7347.157	0.148481	2286.033	[0.00067968]

Table 5.2: Results for March simple case

3
Caso_2_1

Iteration	Dissipation [kWh]	Thermal energy [kWh]	Feed energy [kWh]	Electrical energy [kWh]	β	Energy entering storage [kWh]	Electrical efficiency	Thermal efficiency	Global efficiency	RISP [kWh]	PES	Global emissions [kg]	Maintenance (€/kWh]
0	0	3149066	7104778	3277912	0	661719.5	0.461367	0.443224	0.904591	3520014	0.331302	427241.9	[0.0092 0.0092 0.0092 0.0123]
1	0	3152352	7112616	3283376	0	661280	0.461346	0.443206	0.904552	3523423	0.331272	429692.8	[-0.01518827 -0.01518827 -0.01518827 -0.01518827 -0.01208827]

Table 5.3: Results for January complex case

4
Caso_2_4

Iteration	Dissipation [kWh]	Thermal energy [kWh]	Feed energy [kWh]	Electrical energy [kWh]	β	Energy entering storage [kWh]	Electrical efficiency	Thermal efficiency	Global efficiency	RISP [kWh]	PES	Global emissions [kg]	Maintenance (€/kWh]
0	-9.20E-07	2163584	4845184	2226719	-4.25E-13	318471.3	0.459574	0.446543	0.906117	2399492	0.331208	141318.7	[0.0092 0.0092 0.0092 0.0123]
1	25378.52	3041979	6854867	3162536	0.008343	465270.8	0.461356	0.443769	0.901423	3377089	0.329719	165779.7	[-0.01527303 -0.01527303 -0.01527303 -0.01527303 -0.01217303]
2	25866.54	3066961	6909305	3187955	0.008434	465237.4	0.4614	0.443889	0.901545	3400026	0.329801	164938.4	[-0.01501501 -0.01501501 -0.01501501 -0.01501501 -0.01191501]

Table 5.4: Results for April complex case

5.3 Annual version

Primary energy savings, and so also the economical earnings deriving from the white certificates, have more meaning if calculated on a yearly base instead than related to a single week.
With the help of the post-processing codes already elaborated for both the 14 reference weeks case and the clustering case, it is possible with few modification of the code.

The *CB* code has been modified only by saving the names of all the netlist launched and calling a different function to calculate the results.

The functions in *Find_in_netlist* do not require modifications, since the assumption that the same components are used all the year has been made.

The function in *Results_CB* is substituted with two different versions to utilize in diverse situations: *Results_CB_an* to use if a run with the 14 reference weeks is made, and *Results_CB_an_rdays* to use when the run is done using the clustering outputs. They contain part of the correspondent post-processing codes already analyzed: *Post_processing_14_weeks* and *Post_processing_ref_days*.
After the parsing of all the xml results files and the calculation of the required annual values, also the white certificates savings and the new values of maintenance are computed, as demanded to the function.

The function that modifies the maintenance in the xml files, contained in *modify_maint* is kept the same, since the xml components file is in common for all the netlists of the same year.

In order to validate the code, a run has been executed with the thermal profile of Carmagnola, choosing the 8 day reference case. It had shown good results in chapter 4 and requires the simulation of only 8 days for each iteration, so it is not too computational heavy.
After the first run, the procedure requires only other three iterations and then it finds convergence, as the picture 5.7 shows.

With the implementation of white certificates, β coefficient grows from 2% to 26%. The thermal energy produced by the cogenerators increases of the 46%, instead that produced by boiler slightly decreases (-9%). It implies a rise of the total dissipation, as shown in figure 5.8. So the additional thermal energy produced by CHPs in part causes a decrease of the usage of boilers and in part an increase of dissipation.

With a small increment of the CO_2 emissions (+6%) there is a huge drop of the total cost (-39%). As expected, The primary energy saved RISP increases (+5%) and the PES indicator decreases from 29% to 22%, being affected by the increment of the cogenerators fed energy.
All iterations results are reported in table 5.5. In the last row, so the row of the

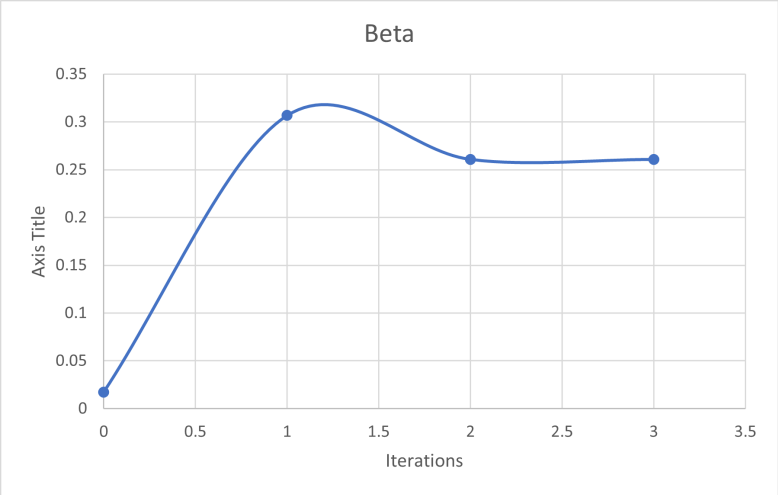


Figure 5.7: β convergence in the Carmagnola 8 ref. days annual case

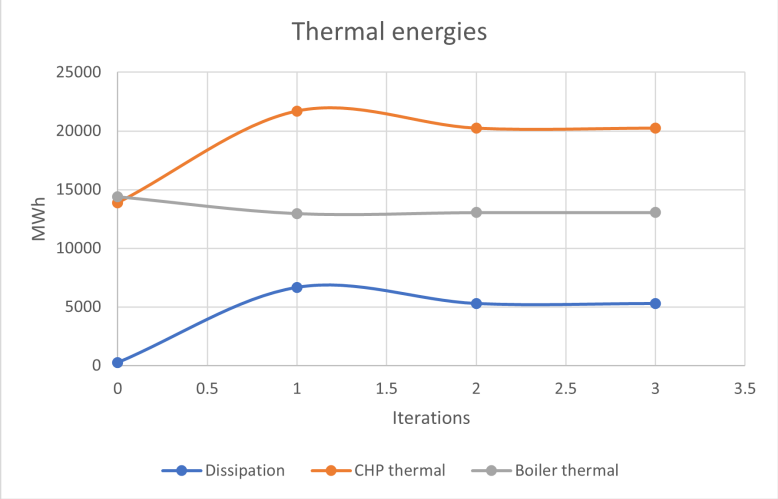


Figure 5.8: Thermal energy produced CHPs, boilers and dissipated

convergent iteration, there are also the differences in percentage between the case without white certificates.

Iteration	Dissipation [kWh]	Thermal CHP [kWh]	Fed energy CHP[kWh]	Electrical CHP [kWh]	Boiler thermal [kWh]	β	Maintenance [€/kWh]
0	240358	13864006	31419443	13339780	14377172	0.017337	[0.01 0.01]
1	6652568	21689312	50601572	21705380	12964076	0.306721	[-0.011651 -0.011651]
2	5277616	20231784	47120309	20197655	13046652	0.260858	[-0.003907 -0.003907]
3	5277616 +2096%	20231784 +46%	47123395 +50%	20199184 +51%	13046652 -9%	0.260858 +1405%	[-0.005071 -0.005071]

Iteration	Electrical efficiency	Thermal efficiency	Global efficiency	RISP [kWh]	PES [€/kWh]	CO_2 emissions [kg]	Obj. function [€]
0	0.424571	0.441256	0.858177	12717466	0.288137	3953236	620476
1	0.428947	0.428629	0.726106	13291531	0.208028	4283535	239281
2	0.42864	0.429364	0.746002	13403378	0.221457	4186300	399770
3	0.428644 +1%	0.429336 -3%	0.745985 -13%	13403616 +5%	0.221449 -23%	4186415 +6%	376261 -39%

Table 5.5: Annual results of CB iterations

Chapter 6

Parametrical optimization

The aim of this chapter is to describe a parametric procedure for the substitution of a component inside a system. It can happen that the choice of the components is not the optimal one. In order to check if a component in a system has effectively the most suitable size, another parametric procedure has been implemented in Python.

Given one or more components of the same type, and given a range of sizes that the analysis should investigate, the code is able to run the XEMS13 tool for all the desired sizes and to return the configuration that gives the lowest cost.
The code is directly been built for the annual version with the post-processing (of the 14 weeks or of the reference days) already incorporated. The routine is described by the flow chart in fig. 6.1.

6.1 Procedure

The procedure is similar to the one implemented for the white certificates and is divided in more codes, all referring to a main one that calls the functions contained in the others. They are:

- *"Parametric"*: the main code. It allows to write the component to change and the sizes range. After the first run with the original size, a *for* loop continues to change the xml component file and re-run XEMS13 until all the sizes have been tested. The costs with the different configurations are compared and the one that returns the lowest is printed;

- *"Find_in_netlist"*: the code is the same already used in the white certificates routine. In this case, only the function able to find the name of the xml components file is utilized;

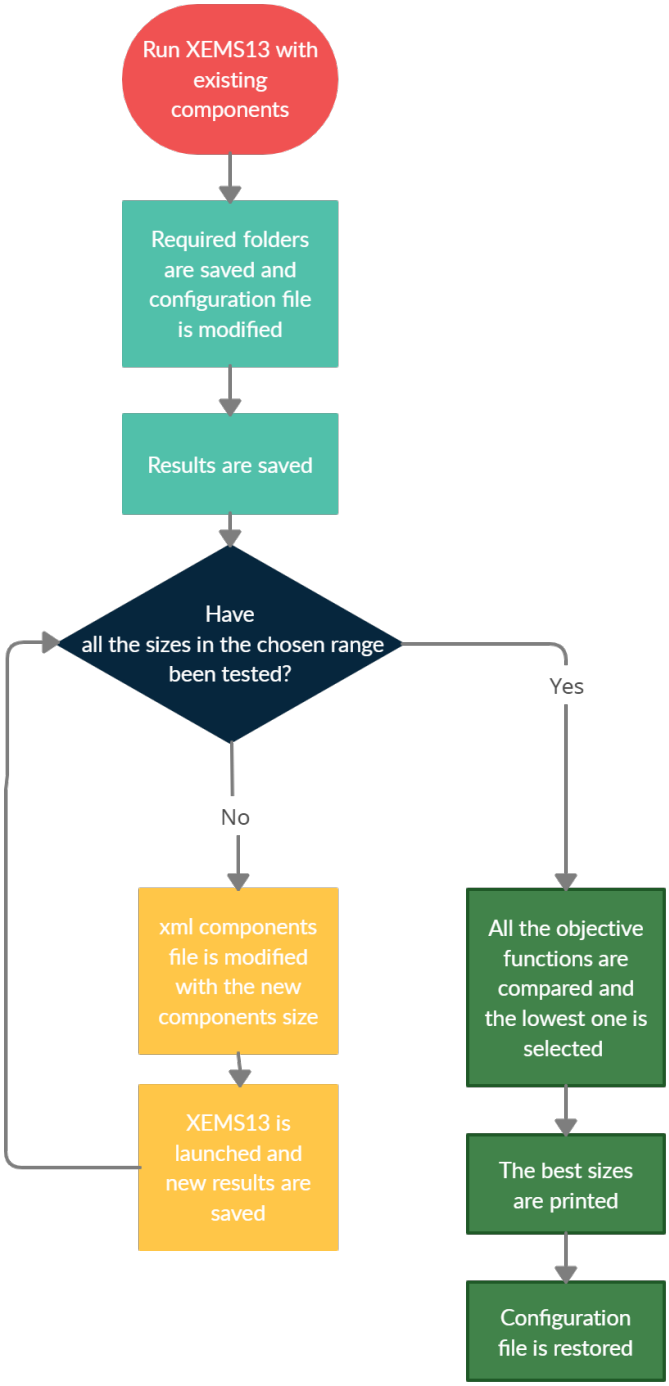


Figure 6.1: Iterative procedure

- *"Results_for_param_an"* or *"Results_for_param_an_r_days"*: are the two codes built for the post-processing of the XEMS13 results. The first obtains the annual values from the 14 reference weeks run, the second from the reference days run. They are similar to their equivalents for the white certificates, but they do not calculate white certificates and new maintenance values;

- *Modify_size*: it contains the function to modify the xml component file. It is called as many times as the sizes to test.

6.1.1 Parametric

This code must be contained in the same folder of the XEMS13 tool.

As first thing, it is possible to choose the components to change in the parametric run. If more than one components are chosen, they must be of the same type (for example two cogenerators) and they will have the same size in each iteration. In the example case with Carmagnola profile, the two cogenerators have been chosen. For each component, different parameters must be inserted:

- The types: in this example the two "CHP" components have been chosen;

- The names: they coincide with the component original sizes; "CHP_1413" and "CHP_1487" are the ones taken into consideration;

- Efficiencies related to the each component. Since the cogenerators have two efficiencies (thermal and electrical) two lists are created, both containing the two values for the two cogenerators. They are needed to change the sizes in the proper way;

- The CO_2 emissions at full load with the original size;

- Minimum size: smallest size of the components to simulate; 1000 kWe is selected;

- Maximum size: biggest size of the components to simulate; 2000 kWe is selected;

- Size gap: gap between two successive sizes. 100 kWe is selected, so there will be 11 run with fictitious sizes;

Subsequently, the XEMS13 main is started, the first run is performed with the original sizes. The configuration file is modified so the required folders are not requested again and the *"defaultDIR.txt"* file is read in order to save their paths. A csv file is created where the results of the parametric runs will be written.

Following, the function *Find_components_file_name* is called to get the name of the component xml file from the netlist. This function is inside the already cited *Find_in_netlist* code, that won't be analyzed again.

After this, the function *results* in the code *Results_for_param_an_r_days*, is called to derive from the xml results file the first outcomes, which are written in the csv file. This function requires the main path and the netlists name. It parses the xml results of all the reference days and does the post-processing to obtain the annual results, with a procedure equal to the one already described previously. It returns the objective function, the electrical and the thermal energies produced by the cogenerators, the thermal one produced by the boilers, the dissipation, the global efficiency, the indicators RISP, PES and the total CO_2 emissions. The objective function is saved in a list.
There is also a twin function inside the code *Results_for_param_an* that does the same for the 14 weeks post-processing case. The two functions will be no more discussed since they are very similar to the ones already cited.

The *for* cycle can start. For each size in the selected range, the function *modify_size* is called and the components xml is updated with the new values. Then, the proper function gets the results, which are written in the csv.

After the end of the cycle, the minimum value among all the objective functions is extracted, and the size that gave it as result (so, the best one) is printed.
At the end, the configuration file is restored with the option that makes XEMS13 ask again the folders.
At end the xml components file will remain modified, so if it is required to re-run the procedure, a new original xml file should be used to substitute the modified one.

6.1.2 Modify_size

The code contains the homonym function.
It needs the main path, the name of the components xml file, the types and names of the components to change, their electrical and thermal efficiency and the size to write in the xml. It does not return anything.

The function does the parsing of all the xml of the components.
For each of the components to change size there is the following procedure:

1. The component with the same type and name is found in the xml;

2. Its thermal and electrical efficiencies are got from the correspondent lists, as well as the emissions with its original size;

3. For the power levels going from 50% to 75% to 100% of load the correspondent

electrical power, thermal power and fed power are calculated and written in the xml.

At each iteration, the electrical power at 100% of load corresponds to the size in kWe. To derive the thermal and the fed powers, the hypothesis of constant thermal and electrical efficiencies for all the sizes has been made.
By now, this hypothesis can be considered as sufficient because in this phase the aim is to create fictitious cogenerators varying the parameters with continuity. Following, it will be possible also to use a series of real components as substitutes. So the fed power has been calculated by dividing the electrical power by the electrical efficiency and the thermal one by multiplying the fed power for the thermal efficiency.

For the thermal and fed power at the lower power levels, it has been made again the hypothesis that at 50% of load the thermal efficiency is higher of +4% points with respect to that at 100% and the electrical efficiency decreases of 4%.
At 75% of load the thermal efficiency increases of 2% and the electrical one decreases of 2%, [24].

The CO_2 emissions are, by hypothesis, proportional to the size of the component. So, for each size and each load, they are derived through a linear proportion with the original value. The other emissions are neglected.

6.2 Results

The example case is the parametric run of the Carmagnola case with the 8 reference days approximation, varying the size of the two cogenerators.
Since they originally have similar sizes (1413 kWe and 1487 kWe), it is reasonable to change with continuity their sizes together from 1000 to 2000 kWe, increasing by 100 kWe.

In table 6.1 are listed all the results. In the objective function column there is also the percentage increase or decrease with respect to the original case. The sizes that return the lowest objective function value are the highest, so 2000 kWe per cogenerator. It must be underlined that in the table the size indicated is the one of each cogenerator, instead in the following graphs is the total cumulative size of all the CHPs.

Sizes [kWe]	Obj. function [€]	Ele. CHP [kWh]	The. CHP[kWh]	Boiler the. [kWh]	Dissipation [kWh]	Global efficiency	RISP [kWh]	PES	CO_2 emissions [kg]
originals	620476	13339780	13864006	14377172	240358	0.858177	12717466	0.288137	3935981.541
1000	687306 11%	10194039	10435485	17702869	137533.6	0.852016	9551932	0.284257	4329579.771
1100	673622.7 9%	11013325	11316290	16873013	188482.2	0.850411	10270455	0.282884	4229329.752
1200	659623.9 6%	11595214	11833540	16296993	129712.8	0.853462	10911792	0.285565	4164773.44
1300	645521.8 4%	12413456	12595009	15562642	156830.7	0.852696	11661196	0.285772	4082011.339
1400	631705.1 2%	13011600	12992642	15190930	182752.1	0.851829	12206298	0.287077	3979471.666
1500	618303.3 0%	14024682	14287507	13941692	228379.2	0.851241	13118064	0.284497	3884137.274
1600	606291.6 -2%	14713755	15068388	13225084	292652.4	0.850074	13713405	0.283312	3795120.932
1700	595153.1 -4%	15044756	15394189	12765026	158395.6	0.853902	14173281	0.285552	3731308.068
1800	583870.9 -6%	15627623	16049399	12134569	183147.4	0.853443	14700134	0.284874	3650468.623
1900	572899.5 -8%	16243969	16693624	11555890	248694.3	0.851887	15212771	0.283899	3582739.858
2000	562123.7 -9%	16827426	17300014	11008948	308142.3	0.850692	15706180	0.283192	3518943.403

Table 6.1: Comparison of the annual results for the different cogenerators sizes

Indeed, it results as more convenient to produce more thermal energy from cogenerators than from boiler, as shows the image 6.2 that illustrates the variation of the thermal energies produced by CHPs and boilers increasing the cumulative size of the cogenerator unit.

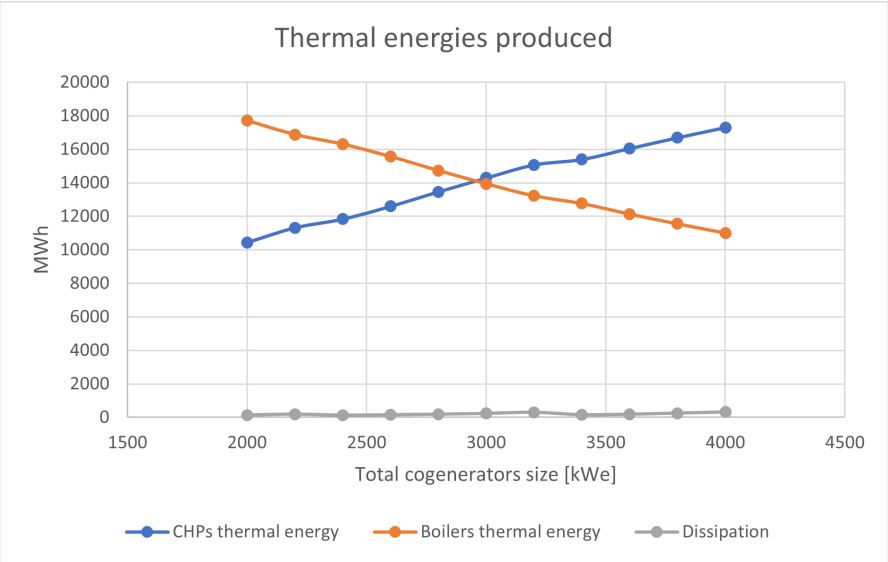


Figure 6.2: Thermal energies produced by CHPs and boilers and dissipated

It can be useful also to compare the number of equivalent hours of the cogenerators in each simulation. It can be calculated as the ratio between the annual total electrical energy produced over the nominal electrical power (the size), [29]:

$$h_{eq} = \frac{E_{ele}}{P_{nom}} \tag{6.1}$$

Considering the cogenerators as a single unit, the equivalent hours are plotted in function of the size in fig. 6.3.

As it can be expected, the equivalent hours decrease increasing the size of the cogenerators, reaching a value of around 4206 h, so about half of the year.
From the csv results of the reference days, it can be seen that with the optimal size during the winter days the cogenerators work almost always near the 100% of load. During the non-heating period, instead, there is only one cogenerator working at partial load to satisfy the small peak.

As it could be expected from the increase of the cogenerators use to the detriment of boilers, the emissions of CO_2 decrease linearly with the size increase.
With a total size of 400 kWe, emissions decrease of 10.5% with respect to emissions with original size.

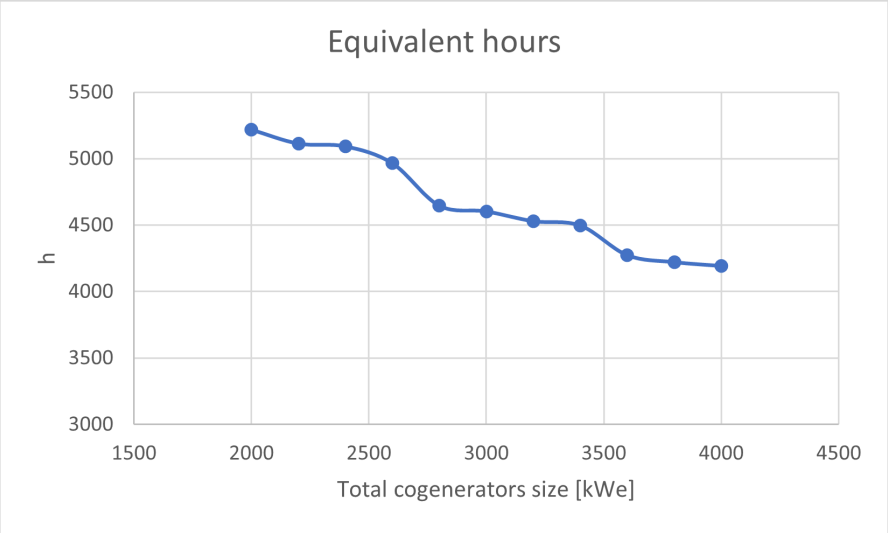


Figure 6.3: Thermal energies produced by CHPs and boilers and dissipated

The range of sizes to variate has been chosen to be realistic, but it can be extended in order to check the behaviour of the system also for larger sizes. Increasing them further, at a certain point the objective functions start increasing again. This happens when the cogenerators have a size of around 8000 kWe each.

Doing the test with the boiler sizes make no sense. In fact, when it operates at partial load there is the assumption of constant efficiency, as well as for the size scale. So the procedure does not see any differences, for example, in using a boiler of 5000 kW at 50% of load or a boiler of 2500 kW at 100%. Anyway, the possibility to change also the boiler sizes has been implemented for the case in which real boilers taken from an external list must be compared.

In conclusion, this analysis is quite approximate and it is done only as an example for the procedure. In order to obtain more reliable results, more precise information should be given, preferably taken from real components. In this case, with few modification of the code is possible to realize a procedure that, taking real data from a file, iteratively substitutes a component with the ones whom information are contained in the file.
Moreover, in order to understand if it actually worth to change the components, a more accurate analysis should be performed, taking in consideration also the capital fixed costs that in the procedure are ignored since the simulation optimizes only the operation of the system.

Chapter 7

Conclusions

In this master thesis different codes to add functionalities to the optimization tool XEMS13 have been presented. They have been tested using the data related to the Carmagnola district heating system, kindly provided by EGEA.

The first topic that has been illustrated is the clustering, that can allow to simulate shorter periods without compromising the annual results of the simulation, and sometimes improving them. The use of the right number of reference days, based on the observation of some main indicators (Inertia, Silhouette and Davis-Bouldin score, and one additional, the DC error), allows to reconstruct the annual thermal profile with a shape fitting with the real one. The tested algorithms are k-means and Ward, that even if characterized by different logic, return similar solutions. Since no evident differences have been detected, the comparison continued only between different number of cluster obtained with k-means and the actual method of 14 reference weeks.

In chapter 4 the analysis has been pursued by running the optimization with the profiles got in the previous chapter. Before, some preliminary steps were necessary in order to complete the input profiles needed by XEMS13, as the electrical demands, taken as percentage of the derived thermal one, and the electricity prices, taken from the day most similar to each reference day. By means of two post-processing algorithms, aimed at building the annual results from the single netlists results, the outcomes are then compared. The results obtained with the clustering reference days are confronted with the results obtained by running all the year divided in periods and with those obtained running 14 reference weeks. The comparison showed that, in the Carmagnola case, 8 reference days is already enough to obtain quite accurate results, not much different from the complete year case and with a very reduced computational time in comparison with the 14 weeks reference case.

The study then focused on the codes for the implementation of the white certificates. The XEMS13 procedure does not take them into account, so an

external procedure was necessary to include them in the final solution. The earning derived from the white certificates is calculated from the output of the simulation, than they are implemented by subtracting them to the maintenance cost of the cogenerators. The simulation is run again and the routine is repeated until convergence of the dissipation coefficient β (or RISP in presence of a storage) is found or a given maximum number of iterations is reached. The procedure has been implemented for a single netlist run, or also for annual run (with multiple netlist, one for each reference period and then a post-processing of the results). In the proven cases the procedure stopped before the fifth iteration and returned consistent results. White certificates allow to reduce the total costs, but not always emissions decreased, since a certain margin of thermal dissipation is allowed and sometimes the solution suggests to run the cogenerators more then necessary, without a consistent reduction of the boiler work. The presence of the storage can be a great advantage in this case, avoiding the energy waste. Generally, its utilization is mean to increase thanks to white certificates implementation.

In chapter 6, the discussion is about a parametric procedure with an iterative substitution of a component. The routine can change the size of a component (or more, but by now sizes are changed in parallel) and run for each size in the chosen range the optimization. The results, in particular the objective functions, are then compared to obtain the best configuration. The results got with the modification of the size of the cogenerators are interesting. They suggest a bigger size for the Carmagnola cogenerators, but more information regarding the components are needed to got more accurate outcomes and evaluate a real substitution or supplements.

The results of the procedures are satisfying and leave room for further possible insights in the field of data analysis and modelling of energy systems.

Appendix A

Codes

A.1 Table of codes

Note: the codes indicated with * contain only the definition of functions called in other codes.

Code	Input	Output	Description
kmean_n_cluster	Minimum and maximum number of clusters, number of sample days and hour of each reference period, annual profile to cluster	Plots of Inertia, Davies-Bouldin, Silhouette score and duration curve error	Used to select the right number of cluster for k-means
Select_Ut *	Annual hourly demand profile	Vector of the annual profile	Takes the profile from csv and creates a vector

k_means_pp	Number of clusters, number of sample days and hour of each reference period, paths of the folders of netlists and profiles	Reference days. Plots of: reference day for each cluster, weights, labels, annual profiles and DC curves comparisons, duration curve error. csv profiles of each reference day, netlist of each reference day. A text file containing useful information for the post-processing	Main code for k-means algorithm
Create_csv *	Number of clusters, reference days profiles, labels of the day closest to the centroids, the profiles folder path, the annual PNord and PUN prices profiles	The csv profiles of thermal and electrical demand and the prices profiles in the profiles folder	Used to create automatically the csv profiles
ward_n_cluster	Minimum and maximum number of clusters, number of sample days and hour of each reference period, annual profile to cluster	Plots of Davies-Bouldin, Silhouette score and duration curve error	Used to select the right number of cluster for Ward

Ward_pp	Number of clusters, number of sample days and hour of each reference period, paths of the folders of netlists and profiles	Reference days. Plots of: the reference day for each cluster, the weights, the labels, the annual profiles and DC curves comparisons, duration curve error	Main code for Ward's algorithm
Post_processing_14_weeks	The xml results files of the 14 simulations of the weeks	A csv file with all the main annual results and the plots of the annual profiles of the thermal and electrical energies balance	Used for the post-processing of the 14 reference weeks case
Post_processing_ref_days	The xml results files of all the reference days	A csv file with all the main annual results and the plots of the annual profiles of the thermal and electrical energies balance	Used for the post-processing clustering case
CB	Maximum number of iterations, netlists, profiles and components folders, netlist file	xml and csv results files, a csv file with the main results at each iteration	Main code for the implementation of white certificates on a single netlist
CB_an	Maximum number of iterations, netlists, profiles and components folders, netlist files	xml and csv results files, a csv file with the annual main results at each iteration	Main code for the implementation of white certificates over an year

Find_in_netlist *	Path of the netlist folder, netlist file name	The name of the xml components file, the name and the type of the co-generators used	Functions that search information through the netlist text file
Results_for_CB *	Netlist folder path, original maintenance vector	Main results of the iteration, included the vector of the updated maintenance	Calculates the results of the iteration for a single netlist
Results_for_CB_an *	Netlist folder path, original maintenance vector	Main annual results of the iteration, included the vector of the updated maintenance	Calculates the annual results of the iteration for the 14 weeks case
Results_for_CB_an_rdays *	Netlist folder path, original maintenance vector	Main annual results of the iteration, included the vector of the updated maintenance	Calculates the annual results of the iteration for the clustering case
Modify_maint *	Path of the netlist folder, name of the components file, used cogenerators, updated maintenance vector	None	Updates the cogenerators maintenance in the xml components file
Parametric	Names and types of the components to modify and their efficiencies and emissions at 100% of load. The range of sizes to test.	A csv with the main results for each size and the configurations that returns the lowest cost	Main code for the parametric run

Results_for_ param_an *	Path of the netlist folder, name of the netlist file	Main annual results	Calculates the main annual results in the 14 reference weeks case
Results_for_ param_an_ r_days *	Path of the netlist folder, name of the netlist file	Main annual results	Calculates the main annual results in the clustering case
Modify_size *	Path of the netlist folder, name of the components file, name, types, efficiencies and emissions of the components to test, new size	None	Updates the xml components file with the data relative to the new size to test

Table A.1: All the used codes

A.2 Codes relative to chapter 3

Only the codes using k-means algorithm are shown, since the Ward codes are very similar.

kmean_n_clusters: to find the optimal number of clusters for the clustering. It plots also the indexes.

```
from sklearn.cluster import KMeans
import numpy as np
import time as time
from sklearn.metrics import davies_bouldin_score
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import Select_Ut


# Parameters to set
sample_days = 366
min_clusters = 4
max_clusters = 20
hours = 24
```

```
15
16 Real = Select_Ut.Real_profile()
17 n = 0
18 X = np.ones([sample_days, hours])
19 for i in range(0, sample_days):
20     for j in range(0, hours):
21         X[i, j] = Real[n + j]
22     n = n + hours
23
24 # Find the maximum value and normalize
25 Max = 0
26
27 for i in range(0, sample_days):
28     List_day = X[i, :].tolist()
29     Maximum = max(List_day)
30     if Maximum > Max:
31         Max = Maximum
32 X = X / Max
33
34 max_clusters_plus = int(max_clusters + 1)
35 Inertia = []
36 D_B = []
37 Silhouette = []
38 error_list = []
39
40 for n_clusters in range(min_clusters, max_clusters_plus):
41     st = time.time()
42     kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(X)
43     elapsed_time = time.time() - st
44
45     # Cluster centroids, so reference days
46     # print("Reference days are: ", kmeans.cluster_centers_)
47
48     # Label: vector tells at which cluster belongs each sample
49     # print("Label of each sample:", kmeans.labels_)
50     labels = kmeans.labels_.tolist()
51     weights = []
52     for t in range(0, n_clusters):
53         weights.append(labels.count(t))
54     total_w = sum(weights)
55     for t in range(0, n_clusters):
56         weights[t] = weights[t] / total_w  # Weight of each reference
       day
57     print("Weight of each reference day:", weights)
58
59     print("Number of iterations run:", kmeans.n_iter_)
60     Inertia.append(kmeans.inertia_)
61     print("Elapsed time:", elapsed_time, "s")
62     D_B.append(davies_bouldin_score(X, labels))
```

```
63      Silhouette.append(silhouette_score(X, labels))
64      Reference_days = kmeans.cluster_centers_
65      Reconstructed = []
66      for label in labels:
67          Rec_day = Reference_days[label]
68          Rec_day = Rec_day.tolist()
69          Reconstructed = Reconstructed + Rec_day
70      Reconstructed = np.array(Reconstructed)
71      Reconstructed = Reconstructed / sum(Reconstructed) * sum(Real)
72      Reference_days = Reference_days / sum(Reconstructed) * sum(Real)
73      Real_DC = np.sort(Real, kind="stable")[::-1]
74      Reconstructed_DC = np.sort(Reconstructed)[::-1]
75      time1 = np.arange(1, hours * sample_days + 1, 1)
76      timen = time1 / (sample_days * hours) * 100
77      n_bin = 20
78      bins = np.arange(0, 1, 1 / n_bin)
79      index_L_list = []
80      index_A_list = []
81      error = 0
82
83      for bin in bins:
84          Real_DC_0 = abs(Real_DC - bin)
85          Reconstructed_DC_0 = abs(Reconstructed_DC - bin)
86          Real_DC_list = Real_DC_0.tolist()
87          Reconstructed_DC_list = Reconstructed_DC_0.tolist()
88          x = min(Real_DC_list)
89          y = min(Reconstructed_DC_list)
90          index_L = Real_DC_list.index(min(Real_DC_list))
91          index_A = Reconstructed_DC_list.index(min(
    Reconstructed_DC_list))
92          index_L_list.append(index_L)
93          index_A_list.append(index_A)
94          L = timen[index_L]
95          A = timen[index_A]
96          error_bin = abs(L - A)
97          error = error + error_bin
98      error_list.append(error)
99
100 # Plots:
101 Num_clusters = range(min_clusters, max_clusters_plus)
102
103 plt.plot(Num_clusters, Inertia, marker="o", color='blue', label="
    Inertia")
104 plt.title("Inertia")
105 plt.xlabel("Number of clusters")
106 plt.grid()
107 plt.show()
108
```

```
plt.plot(Num_clusters, D_B, marker="o", color='green', label="Davies-Bouldin score")
plt.title("Davies-Bouldin score")
plt.xlabel("Number of clusters")
plt.grid()
plt.show()

plt.plot(Num_clusters, Silhouette, marker="o", color='red', label="Silhouette score")
plt.title("Silhouette score")
plt.xlabel("Number of clusters")
plt.grid()
plt.show()

plt.plot(Num_clusters, error_list, marker="o", color='black', label="Total error")
plt.title("DC error")
plt.xlabel("Number of clusters")
plt.ylabel("Sum of duration errors [%]")
plt.grid()
plt.show()
```

Select_Ut: contains a function that extracts the annual thermal load from the csv file.

```
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog as fd
import os
import pandas as pd


def Real_profile():
    global filenames
    root = tk.Tk()
    root.title('File selection')
    root.resizable(False, False)
    root.geometry('300x150')

    def select_files():  # Ask the Ut csv file
        global filenames

        filetypes = (
            ('csv files', '*.csv'),
            ('All files', '*.*')
        )

        filenames = fd.askopenfilenames(
```

```
            title='Open files',
            initialdir='/',
            filetypes=filetypes)

    # open button
    open_button = ttk.Button(
        root,
        text='Choose thermal load file',
        command=select_files
    )

    open_button.pack(expand=True)

    root.mainloop()
    name = "undefined"
    path = "undefined"
    for filename in filenames:
        name_complete = os.path.basename(os.path.normpath(filename))
        name = name_complete.rsplit("/", 1)[0]  # name of the
    simulation
        path = filename[0:(len(filename) - len(name_complete) - 1)]
    # path of the simulation

    os.chdir(path)
    df = pd.read_csv(name)
    Demands = df.to_numpy()
    Real = Demands.ravel()

    return Real
```

kmeans_pp: main code for the generation of the reference days.

```
from sklearn.cluster import KMeans
import numpy as np
from numpy import zeros
import time as time
import matplotlib.pyplot as plt
import Select_Ut
import Create_csv
from sklearn.metrics import mean_squared_error
import os

# Parameters to set
n_clusters = 9
sample_days = 366
hours = 24

Real = Select_Ut.Real_profile()
```

```
17
18 n = 0
19 X = np.ones([sample_days, hours])
20 for i in range(0, sample_days):
21     for j in range(0, hours):
22         X[i, j] = Real[n + j]
23     n = n + hours
24
25 st = time.time()
26 kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(X)
27 elapsed_time = time.time() - st
28
29 # Cluster centroids, so reference days
30 Reference_days = kmeans.cluster_centers_
31 # print("Reference days are: ", Reference_days)
32
33 # Label: vector tells at which cluster belongs each sample
34 # print("Label of each sample:", kmeans.labels_)
35 labels = kmeans.labels_.tolist()
36 weights = []
37 for t in range(0, n_clusters):
38     weights.append(labels.count(t))
39 total_w = sum(weights)
40 for t in range(0, n_clusters):
41     weights[t] = weights[t] / total_w  # Weight of each reference day
42 print("Weight of each reference day:", weights)
43 print("Elapsed time:", elapsed_time, "s")
44 print("Number of iterations run:", kmeans.n_iter_)
45
46 Reconstructed = []
47 for label in labels:
48     Rec_day = Reference_days[label]
49     Rec_day = Rec_day.tolist()
50     Reconstructed = Reconstructed + Rec_day
51 Reconstructed = np.array(Reconstructed)
52 print("Error in the ratio between real total energy and reconstructed
       profile total energy:", 1 - sum(Real) / sum(Reconstructed))
53 Reconstructed = Reconstructed / sum(Reconstructed) * sum(Real)
54 Reference_days = Reference_days / sum(Reconstructed) * sum(Real)
55
56 # Plots
57
58 # To find the closest day to each reference day
59 Closest_days = np.arange(1, n_clusters + 1, 1)  # matrix with the
      profiles of the closest days
60 for reference_day in range(0, n_clusters):
61     min_MSE = 100
62     reference_profile = Reference_days[reference_day][:].tolist()
63     for sample_day in range(0, sample_days):
```

```
64          day_profile = X[sample_day][:].tolist()
65          RMSE = mean_squared_error(day_profile, reference_profile,
    squared="False")
66          if RMSE < min_MSE:
67              min_MSE = RMSE
68              Closest_days[reference_day] = sample_day + 1
69
70 # Profiles plots
71 time = np.arange(1, hours + 1, 1)
72 for reference_day in range(0, n_clusters):
73      for profile in range(0, sample_days):
74          profilo = X[profile, :]
75          if labels[profile] != reference_day:
76              plt.plot(time, profilo, marker="o", color='silver', label
    ="profile")
77      for profile in range(0, sample_days):
78          profilo = X[profile, :]
79          if labels[profile] == reference_day:
80              plt.plot(time, profilo, marker="o", color='grey', label="
    profile")
81      plt.xlabel("Hours of the day [h]")
82      plt.ylabel("Thermal load [MWh]")
83      plt.plot(time, Reference_days[reference_day], marker="o", color='
    green', label="reference day")
84      title = ("Cluster nr. ", str(reference_day + 1))
85      title = ''.join(title)
86      plt.title(title)
87      plt.grid()
88      plt.show()
89
90 # Pie chart
91 graph_labels = range(1, n_clusters + 1, 1)
92 plt.pie(weights, labels=graph_labels, autopct='%1.1f%%', counterclock
    =False, normalize=True)
93 plt.title("Weight of each reference day")
94 plt.show()
95
96 # Labels
97 days = range(1, sample_days + 1, 1)
98 labels_plus = np.array(labels) + 1
99 Matrix = zeros((n_clusters + 1, sample_days + 1))
100 for i in range(0, sample_days):
101     for j in range(0, n_clusters):
102         if labels[i] == j:
103             Matrix[j+1][i+1] = 1
104
105 plt.imshow(Matrix, interpolation='nearest', cmap="Greens", aspect='
    auto')
106 plt.xlabel("Days of the period")
```

```
plt.ylabel("Reference days")
plt.axis([1, sample_days, 1, n_clusters])
plt.title("Reference day used for each sample")
plt.show()

# Real and reconstructed profiles
time = np.arange(1, hours * sample_days + 1, 1)
plt.plot(time, Real, color='blue', label="Real profile")
plt.plot(time, Reconstructed, color='orange', label="Reconstructed
    profile")
plt.xlabel("Hours of the period [h]")
plt.ylabel("Thermal load [MWh]")
plt.fill_between(time, Real, Reconstructed, facecolor="red")
plt.legend()
plt.title("Profiles")
plt.grid()
plt.show()

# Duration curves
time = time / (sample_days * hours) * 100
Real_DC = np.sort(Real, kind="stable")[::-1]
Reconstructed_DC = np.sort(Reconstructed)[::-1]
plt.plot(time, Real_DC, color='blue', label="Real duration curve")
plt.plot(time, Reconstructed_DC, color='orange', label="Reconstructed
     duration curve")
plt.xlabel("Duration [%]")
plt.ylabel("Thermal load [MWh]")
plt.fill_between(time, Real_DC, Reconstructed_DC, facecolor="red")
plt.legend()
plt.title("Duration curves")
plt.grid()
plt.show()

# Error DC analysis
Max = 0
for i in range(0, sample_days):
    List_day = X[i, :].tolist()
    Maximum = max(List_day)
    if Maximum > Max:
        Max = Maximum
X = X / Max
Reconstructed_DCn = Reconstructed_DC / Max
Real_DCn = Real_DC / Max

n_bin = 20
bins = np.arange(0, 1, 1 / n_bin)
error = []
index_L_list = []
index_A_list = []
```

```
154 for bin in bins:
155     Real_DC_0 = abs(Real_DCn - bin)
156     Reconstructed_DC_0 = abs(Reconstructed_DCn - bin)
157     Real_DC_list = Real_DC_0.tolist()
158     Reconstructed_DC_list = Reconstructed_DC_0.tolist()
159     x = min(Real_DC_list)
160     y = min(Reconstructed_DC_list)
161     index_L = Real_DC_list.index(min(Real_DC_list))
162     index_A = Reconstructed_DC_list.index(min(Reconstructed_DC_list))
163     index_L_list.append(index_L)
164     index_A_list.append(index_A)
165     L = time[index_L]
166     A = time[index_A]
167     error_bin = abs(L - A)
168     error.append(error_bin)
169 plt.plot(bins, error, color='red', label="Error")
170 plt.xlabel("Bins")
171 plt.ylabel("Duration error [%]")
172 plt.grid()
173 plt.title("Error")
174 plt.show()
175
176 plt.plot(time, Real_DCn, color='blue', label="Real duration curve")
177 plt.plot(time, Reconstructed_DCn, color='orange', label="
        Reconstructed duration curve")
178 plt.xlabel("Duration [%]")
179 plt.ylabel("Normalized thermal load")
180 for number_bin, bin in enumerate(bins):
181     plt.hlines(y=bin, xmin=0, xmax=100, color="grey", ls="--")
182     x = index_L_list[number_bin]
183     y = index_A_list[number_bin]
184     plt.plot(time[x], Real_DCn[x], marker="v", color="blue")
185     plt.plot(time[y], Real_DCn[y], marker="s", color="orange")
186 plt.hlines(y=0.05, xmin=0, xmax=100, color="grey", ls="--")
187 # plt.axis([70, 100, -0.05, 0.1]) # To zoom a part of the graph
188 plt.legend()
189 plt.title("Duration curves error analysis")
190 plt.show()
191
192 # To build the csv files
193 path = "C:/Users/simx_/Documents/Università/PoliTo/Tesi_Tirocinio
        /2020_9_Carmagnola/Profiles"  # Folder where to save the files
194 Create_csv.Build_Ut_Ue(path, Reference_days, n_clusters)
195 Create_csv.Build_cp_cs(path, n_clusters, Closest_days, hours)
196
197 # To save useful informations
198 os.chdir("C:/Users/simx_/Documents/Università/PoliTo/Tesi_Tirocinio
        /2020_9_Carmagnola/Work")
199 text_name = "Info_" + str(n_clusters) + "_clusters.txt"
```

```
with open(text_name, "w") as text_file:
    text_file.write(str(sample_days))
    text_file.write("\n")
    text_file.write(str(n_clusters))
    text_file.write("\n")
    for element in labels:
        text_file.write(str(element))
        text_file.write(" ")
    text_file.write("\n")
    for element in weights:
        text_file.write(str(element))
        text_file.write(" ")

os.chdir("C:/Users/simx_/Documents/Università/PoliTo/Tesi_Tirocinio
    /2020_9_Carmagnola/Work")
with open("ex.txt", "r") as netlist:
    example = netlist.readlines()
for i in range(0, n_clusters):
    name_txt = str(1+i) + ".txt"
    with open(name_txt, "w") as text:
        for row in example:
            if row == "0 1 Ut Ut\n":
                row = str("0 1 Ut Ut_" + str(i + 1) + "\n")
            if row == "0 1 Ue Ue\n":
                row = str("0 1 Ue Ue_" + str(i + 1) + "\n")
            if row == "0 1 Cs Cs\n":
                row = str("0 1 Cs Cs_" + str(i + 1) + "\n")
            if row == "0 1 Cp Cp\n":
                row = str("0 1 Cp Cp_" + str(i + 1) + "\n")
            text.write(row)
```

A.3 Codes related to chapter 4

Create_csv: contains all the functions needed to create the csv profile, inputs of XEMS13.

```
import csv
import os


def Build_Ut_Ue(path, Reference_days, n_clusters):
    os.chdir(path)
    for reference_day in range(0, n_clusters):      # Ut
        ref_day = reference_day + 1
        csv_name = "Ut_" + str(ref_day) + ".csv"
        Ut = Reference_days[reference_day][:]
```

```
11          with open(csv_name, "w", newline="") as csv_file:
12              for i, ut in enumerate(Ut):
13                  writer = csv.writer(csv_file, delimiter=",")
14                  writer.writerow(["0", "0", "0", "0", i + 1, ut *
    1000])
15      for reference_day in range(0, n_clusters):      # Ue
16          ref_day = reference_day + 1
17          csv_name = "Ue_" + str(ref_day) + ".csv"
18          Ue = Reference_days[reference_day][:] * 0.035
19          with open(csv_name, "w", newline="") as csv_file:
20              for i, ue in enumerate(Ue):
21                  writer = csv.writer(csv_file, delimiter=",")
22                  writer.writerow(["0", "0", "0", "0", i + 1, ue *
    1000])
23      return
24
25
26  def Build_cp_cs(path, n_clusters, Closest_days, hours):
27
28      os.chdir("C:/Users/simx_/Documents/Università/PoliTo/
    Tesi_Tirocinio")  # Path with prices file
29      with open("prezzi_PNord.csv", "r") as prices_file:          # Cs
30          prices = prices_file.readlines()
31      for reference_day in range(0, n_clusters):
32          ref_day = reference_day + 1
33          os.chdir(path)
34          csv_name = "Cs_" + str(ref_day) + ".csv"
35          Cs = []
36          day = Closest_days[reference_day]
37          for hour in range(0, hours):
38              cs = str(prices[day*hours + hour + 1]).strip("\n")
39              Cs.append(cs)
40          with open(csv_name, "w", newline="") as csv_file:
41              for i, cs in enumerate(Cs):
42                  writer = csv.writer(csv_file, delimiter=",")
43                  writer.writerow(["0", "0", "0", "0", i + 1, cs])
44
45      os.chdir("C:/Users/simx_/Documents/Università/PoliTo/
    Tesi_Tirocinio")
46      with open("prezzi_PUN.csv", "r") as prices_file:            # Cp
47          prices = prices_file.readlines()
48      for reference_day in range(0, n_clusters):
49          ref_day = reference_day + 1
50          os.chdir(path)
51          csv_name = "Cp_" + str(ref_day) + ".csv"
52          Cp = []
53          day = Closest_days[reference_day]
54          for hour in range(0, hours):
```

```
55              cp = float(str(prices[day*hours + hour + 1]).strip("\n"))
    + 100
56              Cp.append(cp)
57          with open(csv_name, "w", newline="") as csv_file:
58              for i, cp in enumerate(Cp):
59                  writer = csv.writer(csv_file, delimiter=",")
60                  writer.writerow(["0", "0", "0", "0", i + 1, cp])
61
62      return
```

Post_processing_ref_days: XEMS13 post-processing to obtain the annual results from the reference days outcomes.

```
1  import tkinter as tk
2  import os.path
3  import matplotlib.pyplot as plt
4  import numpy as np
5  from numpy import mean
6  import csv
7  import xml.etree.ElementTree as ET
8  from tkinter import ttk
9  from tkinter import filedialog as fd
10
11 # Create root window to ask text file
12 root = tk.Tk()
13 root.title('File selection')
14 root.resizable(False, False)
15 root.geometry('300x150')
16 filenames = "undefined"
17
18 flag = 0
19
20
21 def select_files():
22     global filenames
23
24     filetypes = (
25         ('text files', '*.txt'),
26         ('All files', '*.*')
27     )
28
29     filenames = fd.askopenfilenames(
30         title='Open files',
31         initialdir='/',
32         filetypes=filetypes)
33
34
35 # open button
```

```
36 open_button = ttk.Button(
37     root,
38     text='Choose files with clusters information',
39     command=select_files
40 )
41
42 open_button.pack(expand=True)
43
44 root.mainloop()
45 week = 0
46 n_days = int(0)
47 name = "undefined"
48 path = "undefined"
49
50 for filename in filenames:
51     with open(filename, "r") as text_file:
52         Text = text_file.readlines()
53         sample_days = int(Text[0].strip("\n"))
54         n_clusters = int(Text[1].strip("\n"))
55         labels = Text[2].split()
56         weights = Text[3].split()
57
58 # Create the root window to ask xml files
59 root = tk.Tk()
60 root.title('File selection')
61 root.resizable(False, False)
62 root.geometry('300x150')
63 filenames = "undefined"
64
65 flag = 0
66
67
68 def select_files():                                    # Ask the xml files
69     global filenames
70
71     filetypes = (
72         ('xml files', '*.xml'),
73         ('All files', '*.*')
74     )
75
76     filenames = fd.askopenfilenames(
77         title='Open files',
78         initialdir='/',
79         filetypes=filetypes)
80
81
82 # open button
83 open_button = ttk.Button(
84     root,
```

```
85      text='Choose files to post-process',
86      command=select_files
87  )
88
89  open_button.pack(expand=True)
90
91  root.mainloop()
92  day = 0
93  n_days = int(0)
94  name = "undefined"
95  path = "undefined"
96
97  for filename in filenames:
98      n_days = n_days + 1     # number of reference periods
99      if name == "undefined":
100         name_complete = os.path.basename(os.path.normpath(filename))
101         name = name_complete.rsplit(".", 1)[0]                          #
        name of the simulation
102         path = filename[0:(len(filename)-len(name_complete)-1)]          #
        path of the simulation
103
104 os.chdir(path)
105 name_post_processing = name + "_post_processing.csv"
106 Emissions = []
107 E_thermal = []
108 E_electrical = []
109 Dissipation = []
110 E_entered_storage = []
111 Energy_fed = []
112 Boiler_the = []
113 Demand_the = []
114 Demand_ele = []
115 eta_E = []
116 eta_T = []
117 eta_G = []
118 Obj_f = []
119 Ele_p = []
120 Ele_s = []
121
122 rendimento_E_rif = float(0.46)
123 rendimento_T_rif = float(0.9)
124
125 with open(name_post_processing, "w", newline="") as risultati:
126     writer = csv.writer(risultati, delimiter=";")
127     writer.writerow(
128         ["Reference day", "Thermal energy CHP [kWh]", "Electrical
    energy CHP [kWh]", "Feed energy CHP [kWh]", "Dissipation [kWh]", "
    Boiler production [kWh]", "Ut Thermal load [kWh]",
```

```
129         "In energy stored [kWh]", "Global efficiency CHP", "Thermal
    efficiency CHP", "Electrical efficiency CHP", "CO2 emissions [kg]"
    , "Objective function [€]", "RISP [kWh]", "PES"])
130
131 E_elettrica = 0
132 E_termica = 0
133 E_in = 0
134 E_storage_in = 0
135 Dissipazione = 0
136 Emissioni = 0
137 E_ter_boiler = 0
138 Utenza_ter = 0
139 Utenza_ele = 0
140 Month = 0
141 num_mese = 0
142 num_period = 0
143 E_utile = 0
144 Costo = 0
145 E_p = 0
146 E_s = 0
147
148 if n_days == n_clusters:
149
150     for day, filename in enumerate(filenames):
151         E_elettrica = 0
152         E_termica = 0
153         E_in = 0
154         E_storage_in = 0
155         Dissipazione = 0
156         Emissioni = 0
157         E_ter_boiler = 0
158         Utenza_ter = 0
159         Utenza_ele = 0
160         Costo = 0
161         E_p = 0
162         E_s = 0
163
164         tree = ET.parse(filename)                                    #
    read all values in xml file
165         for valore in tree.findall('.//Costo'):
166             Costo = float(valore.text)
167
168         for valore in tree.findall('.//EmissioniGlobali'):
169             Emissioni = float(valore.text)
170
171         for valore in tree.findall('.//Node_1/CHP/istanza/Pt/VAL'):
172             Pot = float(valore.text)
173             E_termica = E_termica + Pot
174             # legge le potenze termiche prodotte dal CHP
```

```
175         for valore in tree.findall('.//Node_1/CHP/istanza/Pe/VAL'):
176             Pot = float(valore.text)
177             E_elettrica = E_elettrica + Pot
178             # legge le potenze elettriche prodotte dal CHP
179         for valore in tree.findall('.//Node_1/CHP/istanza/Pc/VAL'):
180             Pot = float(valore.text)
181             E_in = E_in + Pot
182             # legge le potenze in ingresso al CHP
183 
184             # legge i risultati dei CHP con recupero di calore a
    bassa temperatura
185         for valore in tree.findall('.//Node_1/CHPLE/istanza/Ptle/VAL'
    ):
186             Pot = float(valore.text)
187             E_termica = E_termica + Pot
188             # legge le potenze termiche prodotte dal CHPLE
189         for valore in tree.findall('.//Node_1/CHPLE/istanza/Pele/VAL'
    ):
190             Pot = float(valore.text)
191             E_elettrica = E_elettrica + Pot
192             # legge le potenze elettriche prodotte dal CHPLE
193         for valore in tree.findall('.//Node_1/CHPLE/istanza/Pcle/VAL'
    ):
194             Pot = float(valore.text)
195             E_in = E_in + Pot
196             # legge le potenze in ingresso al CHPLE
197 
198             # legge i risultati dei CHPS
199         for valore in tree.findall('.//Node_1/CHPS/istanza/Pts/VAL'):
200             Pot = float(valore.text)
201             E_termica = E_termica + Pot
202             # legge le potenze termiche prodotte dal CHP
203         for valore in tree.findall('.//Node_1/CHPS/istanza/Pes/VAL'):
204             Pot = float(valore.text)
205             E_elettrica = E_elettrica + Pot
206         # legge le potenze elettriche prodotte dal CHP
207         for valore in tree.findall('.//Node_1/CHPS/istanza/Pcs/VAL'):
208             Pot = float(valore.text)
209             E_in = E_in + Pot
210             # legge le potenze in ingresso al CHP
211 
212         for valore in tree.findall('.//Node_1/Stt/istanza/PSttin/VAL'
    ):
213             Pot = float(valore.text)
214             E_storage_in = E_storage_in + Pot
215             # legge le potenze in ingresso allo storage (se c'è)
216 
217         for valore in tree.findall('.//Node_1/Dt/Dt/VAL'):
218             Pot = float(valore.text)
```

```
219             Dissipazione = Dissipazione + Pot
220         # legge la dissipazione termica (per tutti i casi)
221 
222         for valore in tree.findall('.//Node_1/Boiler/istanza/Bt/VAL')
    :
223             Pot = float(valore.text)
224             E_ter_boiler = E_ter_boiler + Pot
225             # legge le potenze termiche prodotte dal CHP
226 
227         for valore in tree.findall('.//Node_1/Ut/P/VAL'):
228             Pot = float(valore.text)
229             Utenza_ter = Utenza_ter + Pot
230         # legge la domanda termica (per tutti i casi)
231 
232         for valore in tree.findall('.//Node_1/Ue/P/VAL'):
233             Pot = float(valore.text)
234             Utenza_ele = Utenza_ele + Pot
235         # legge la domanda elettrica (per tutti i casi)
236 
237         for valore in tree.findall('.//Node_1/Grid/istanza/Ps/VAL'):
238             Pot = float(valore.text)
239             E_s = E_s + Pot
240             # legge elettricità venduta alla rete
241 
242         for valore in tree.findall('.//Node_1/Grid/istanza/Pp/VAL'):
243             Pot = float(valore.text)
244             E_p = E_p + Pot
245             # legge elettricità acquistata dalla rete
246 
247         E_utile = E_termica - Dissipazione
248 
249         E_thermal.append(E_termica)                              # update
    the vectors
250         E_electrical.append(E_elettrica)
251         E_entered_storage.append(E_storage_in)
252         Dissipation.append(Dissipazione)
253         Energy_fed.append(E_in)
254         Boiler_the.append(E_ter_boiler)
255         Demand_the.append(Utenza_ter)
256         Demand_ele.append(Utenza_ele)
257         rendimento_T = E_termica / E_in
258         rendimento_E = E_elettrica / E_in
259         rendimento_globale = E_utile / E_in + E_elettrica / E_in
260         eta_T.append(rendimento_T)
261         eta_E.append(rendimento_E)
262         eta_G.append(rendimento_globale)
263         Obj_f.append(Costo)
264         Emissions.append(Emissioni)
265         Ele_p.append(E_p)
```

```
266         Ele_s.append(E_s)
267
268         with open(name_post_processing, "a", newline="") as results:
                # update csv file
269             writer = csv.writer(results, delimiter=";")
270             writer.writerow(
271                 [day + 1, E_termica, E_elettrica, E_in, Dissipazione,
     E_ter_boiler, Utenza_ter, E_storage_in,
272                  rendimento_globale, rendimento_T, rendimento_E,
    Emissioni, Costo, "-", "-"])
273
274     with open(name_post_processing, "a", newline="") as results:  #
    update csv file with annual values
275         writer = csv.writer(results, delimiter=";")
276         writer.writerow("Annual results")
277         E_thermal_w = [0] * n_clusters
278         E_electrical_w = [0] * n_clusters
279         Energy_fed_w = [0] * n_clusters
280         Dissipation_w = [0] * n_clusters
281         Boiler_the_w = [0] * n_clusters
282         E_entered_storage_w = [0] * n_clusters
283         Emissions_w = [0] * n_clusters
284         Costs_w = [0] * n_clusters
285         Demand_the_w = [0] * n_clusters
286         Demand_ele_w = [0] * n_clusters
287         Ele_s_w = [0] * n_clusters
288         Ele_p_w = [0] * n_clusters
289
290         for day in range(0, n_days):
291             E_thermal_w[day] = E_thermal[day] * float(weights[day]) *
     sample_days
292             E_electrical_w[day] = E_electrical[day] * float(weights[
    day]) * sample_days
293             Energy_fed_w[day] = Energy_fed[day] * float(weights[day])
     * sample_days
294             Dissipation_w[day] = Dissipation[day] * float(weights[day
    ]) * sample_days
295             Boiler_the_w[day] = Boiler_the[day] * float(weights[day])
     * sample_days
296             E_entered_storage_w[day] = E_entered_storage[day] * float
    (weights[day]) * sample_days
297             Emissions_w[day] = Emissions[day] * float(weights[day]) *
     sample_days
298             Costs_w[day] = Obj_f[day] * float(weights[day]) *
    sample_days
299             Demand_the_w[day] = Demand_the[day] * float(weights[day])
     * sample_days
300             Demand_ele_w[day] = Demand_ele[day] * float(weights[day])
     * sample_days
```

```
301             Ele_s_w[day] = Ele_s[day] * float(weights[day]) *
    sample_days
302             Ele_p_w[day] = Ele_p[day] * float(weights[day]) *
    sample_days
303
304         E_thermal_a = sum(E_thermal_w)
305         E_electrical_a = sum(E_electrical_w)
306         Energy_fed_a = sum(Energy_fed_w)
307         Dissipation_a = sum(Dissipation_w)
308         Boiler_the_a = sum(Boiler_the_w)
309         Demand_the_a = sum(Demand_the_w)
310         Demand_ele_a = sum(Demand_ele_w)
311         E_entered_storage_a = sum(E_entered_storage_w)
312         Emissions_a = sum(Emissions_w)
313         Cost_a = sum(Costs_w)
314         Ele_s_a = sum(Ele_s_w)
315         Ele_p_a = sum(Ele_p_w)
316         E_useful_a = E_thermal_a - Dissipation_a
317         RISP = E_electrical_a / rendimento_E_rif + E_useful_a /
    rendimento_T_rif - Energy_fed_a
318         PES = RISP / (E_electrical_a / rendimento_E_rif + E_useful_a
    / rendimento_T_rif)
319         eta_T_a = E_thermal_a / Energy_fed_a
320         eta_E_a = E_electrical_a / Energy_fed_a
321         eta_G_a = eta_E_a + E_useful_a / Energy_fed_a
322         writer.writerow(
323             ["Annual results", E_thermal_a, E_electrical_a,
    Energy_fed_a, Dissipation_a, Boiler_the_a, Demand_the_a,
324              E_entered_storage_a,
325              eta_G_a, eta_T_a, eta_E_a, Emissions_a, Cost_a, RISP,
    PES])
326
327     # Monthly results
328     Emissions_y = []
329     E_thermal_y = []
330     E_electrical_y = []
331     Dissipation_y = []
332     E_entered_storage_y = []
333     Energy_fed_y = []
334     Boiler_the_y = []
335     Demand_the_y = []
336     Demand_ele_y = []
337     eta_E_y = []
338     eta_T_y = []
339     eta_G_y = []
340     Obj_f_y = []
341     rendimento_T_y = []
342     rendimento_E_y = []
343     rendimento_globale_y = []
```

```
344     Ele_s_y = []
345     Ele_p_y = []
346
347     for sample_day in range(0, sample_days):
348         index = int(labels[sample_day])
349         E_thermal_y.append(E_thermal[index])
350         E_electrical_y.append(E_electrical[index])
351         E_entered_storage_y.append(E_entered_storage[index])
352         Dissipation_y.append(Dissipation[index])
353         Energy_fed_y.append(Energy_fed[index])
354         Boiler_the_y.append(Boiler_the[index])
355         Obj_f_y.append(Obj_f[index])
356         rendimento_T_y.append(eta_T[index])
357         rendimento_E_y.append(eta_E[index])
358         rendimento_globale_y.append(eta_G[index])
359         Emissions_y.append(Emissions[index])
360         Demand_the_y.append(Demand_the[index])
361         Demand_ele_y.append(Demand_ele[index])
362         Ele_p_y.append(Ele_p[index])
363         Ele_s_y.append(Ele_s[index])
364
365     E_thermal_my = []
366     E_electrical_my = []
367     Energy_fed_my = []
368     Boiler_the_my = []
369     Demand_the_my = []
370     Demand_ele_my = []
371     Dissipation_my = []
372     Ele_p_my = []
373     Ele_s_my = []
374
375     with open(name_post_processing, "a", newline="") as results:  #
    update csv file with annual values
376         writer = csv.writer(results, delimiter=";")
377         writer.writerow("Monthly results")
378         first = "undefined"
379         last = "undefined"
380         for month in range(0, 12):
381             if month == 0:
382                 mese = "January"
383                 first = 0
384                 last = 31
385             if month == 1:
386                 mese = "February"
387                 first = 31
388                 last = 60
389             if month == 2:
390                 mese = "March"
391                 first = 60
```

```
392                 last = 91
393             if month == 3:
394                 mese = "April"
395                 first = 91
396                 last = 121
397             if month == 4:
398                 mese = "May"
399                 first = 121
400                 last = 152
401             if month == 5:
402                 mese = "June"
403                 first = 152
404                 last = 182
405             if month == 6:
406                 mese = "July"
407                 first = 182
408                 last = 213
409             if month == 7:
410                 mese = "August"
411                 first = 213
412                 last = 244
413             if month == 8:
414                 mese = "September"
415                 first = 244
416                 last = 274
417             if month == 9:
418                 mese = "October"
419                 first = 274
420                 last = 305
421             if month == 10:
422                 mese = "November"
423                 first = 305
424                 last = 335
425             if month == 11:
426                 mese = "December"
427                 first = 335
428                 last = 366
429
430             E_thermal_m = sum(E_thermal_y[first:last])
431             E_electrical_m = sum(E_electrical_y[first:last])
432             Energy_fed_m = sum(Energy_fed_y[first:last])
433             Dissipation_m = sum(Dissipation_y[first:last])
434             Boiler_the_m = sum(Boiler_the_y[first:last])
435             Demand_the_m = sum(Demand_the_y[first:last])
436             Demand_ele_m = sum(Demand_ele_y[first:last])
437             E_entered_storage_m = sum(E_entered_storage_y[first:last
    ])
438             a = eta_T_y[first:last]
439             eta_G_m = mean(rendimento_globale_y[first:last])
```

```
440             eta_T_m = mean(rendimento_T_y[first:last])
441             eta_E_m = mean(rendimento_E_y[first:last])
442             Emissions_m = sum(Emissions_y[first:last])
443             Obj_f_m = sum(Obj_f_y[first:last])
444             Ele_p_m = sum(Ele_p_y[first:last])
445             Ele_s_m = sum(Ele_s_y[first:last])
446
447             writer.writerow(
448                 [month + 1, E_thermal_m, E_electrical_m, Energy_fed_m
    , Dissipation_m, Boiler_the_m, Demand_the_m,
449                  E_entered_storage_m,
450                  eta_G_m, eta_T_m, eta_E_m, Emissions_m, Obj_f_m, "-"
    , "-"])
451
452             E_thermal_my.append(E_thermal_m)
453             E_electrical_my.append(E_electrical_m)
454             Energy_fed_my.append(Energy_fed_m)
455             Boiler_the_my.append(Boiler_the_m)
456             Demand_the_my.append(Demand_the_m)
457             Demand_ele_my.append(Demand_ele_m)
458             Dissipation_my.append(Dissipation_m)
459             Ele_s_my.append(Ele_s_m)
460             Ele_p_my.append(Ele_p_m)
461
462     # Plots:
463
464     E_thermal_y = np.array(E_thermal_y)
465     E_electrical_y = np.array(E_electrical_y)
466     Dissipation_y = np.array(Dissipation_y)
467     Energy_fed_y = np.array(Energy_fed_y)
468     Boiler_the_y = np.array(Boiler_the_y)
469     Time = x = np.arange(1, 13, 1)
470     E_useful_y = E_thermal_y - Dissipation_y
471     plt.plot(Time, E_thermal_my, marker="o", color='green', label="
    Thermal energy CHP")
472     plt.plot(Time, Demand_the_my, marker="o", color='red', label="
    Thermal load")
473     # plt.plot(Time, Energy_fed_my, marker="o", color='black', label
    ="Energy fed")
474     plt.plot(Time, Boiler_the_my, marker="o", color='blue', label="
    Thermal energy boiler")
475     plt.plot(Time, Dissipation_my, marker="o", color='orange', label=
    "Dissipation")
476     plt.ylim(0, 6e6)
477     plt.legend()
478     titolo = "Annual results with " + str(n_clusters) + " clusters"
479     plt.title(titolo)
480     plt.xlabel("Month")
481     plt.ylabel("kWh")
```

```
482     plt.grid()
483     plt.show()
484 
485     plt.plot(Time, E_electrical_my, marker="o", color='green', label=
    "Electrical energy CHP")
486     plt.plot(Time, Demand_ele_my, marker="o", color='red', label="
    Electrical load")
487     # plt.plot(Time, Energy_fed_my, marker="o", color='black', label
    ="Energy fed")
488     plt.plot(Time, Ele_p_my, marker="o", color='blue', label="
    Electricity purchased")
489     plt.plot(Time, Ele_s_my, marker="o", color='orange', label="
    Electricity sold")
490     plt.ylim(0, 2.5e6)
491     plt.legend()
492     titolo = "Annual results with " + str(n_clusters) + " clusters"
493     plt.title(titolo)
494     plt.xlabel("Month")
495     plt.ylabel("kWh")
496     plt.grid()
497     plt.show()
498 
499 else:
500     print("Numero incorretto di file selezionati")  # if there is not
    the correct number of selected files
```

Post_processing_14_weeks: XEMS13 post-processing to obtain the annual results from the 14 weeks outcomes.

```
1  import tkinter as tk
2  import os.path
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import csv
6  import xml.etree.ElementTree as ET
7  from tkinter import ttk
8  from tkinter import filedialog as fd
9 
10 
11 # create the root window
12 root = tk.Tk()
13 root.title('File selection')
14 root.resizable(False, False)
15 root.geometry('300x150')
16 filenames = "undefined"
17 
18 flag = 0
19 
```

```
20
21 def select_files():                                  # Ask the 14 xml files
22     global filenames
23
24     filetypes = (
25         ('xml files', '*.xml'),
26         ('All files', '*.*')
27     )
28
29     filenames = fd.askopenfilenames(
30         title='Open files',
31         initialdir='/',
32         filetypes=filetypes)
33
34
35 # open button
36 open_button = ttk.Button(
37     root,
38     text='Choose files to post-process',
39     command=select_files
40 )
41
42 open_button.pack(expand=True)
43
44 root.mainloop()
45 week = 0
46 n_weeks = int(0)
47 name = "undefined"
48 path = "undefined"
49
50 for filename in filenames:
51     n_weeks = n_weeks + 1     # number of periods analyzed
52     if name == "undefined":
53         name_complete = os.path.basename(os.path.normpath(filename))
54         name = name_complete.rsplit(".", 1)[0]                          #
     name of the simulation
55         path = filename[0:(len(filename)-len(name_complete)-1)]      #
     path of the simulation
56
57 os.chdir(path)
58 name_post_processing = name + "_post_processing.csv"
59 Emissions = []
60 E_thermal = []
61 E_electrical = []
62 Dissipation = []
63 E_entered_storage = []
64 Energy_fed = []
65 Boiler_the = []
66 Demand_the = []
```

```
67 eta_E = []
68 eta_T = []
69 eta_G = []
70 RISP = []
71 PES = []
72 Obj_f = []
73 Ele_demand = []
74 Ele_s = []
75 Ele_p = []
76 rendimento_E_rif = float(0.46)
77 rendimento_T_rif = float(0.9)
78
79 with open(name_post_processing, "w", newline="") as risultati:
80     writer = csv.writer(risultati, delimiter=";")
81     writer.writerow(
82         ["Mese", "Energia termica CHP [kWh]", "Energia elettrica CHP
    [kWh]", "Energia di alimentazione CHP [kWh]", "Dissipazione [kWh]"
    , "Produzione caldaie [kWh]", "Utenza termica [kwh]",
83          "Energia accumulo ingresso [kWh]", "Rendimento globale CHP",
     "Rendimento termico", "Rendimento elettrico", "Emissioni CO2 [kg]
    ", "Objective function [€]", "RISP [kWh]", "PES"])
84
85 E_elettrica = 0
86 E_termica = 0
87 E_in = 0
88 E_storage_in = 0
89 Dissipazione = 0
90 Emissioni = 0
91 E_ter_boiler = 0
92 Utenza_ter = 0
93 Month = 0
94 num_mese = 0
95 num_period = 0
96 Costo = 0
97 Utenza_ele = 0
98 E_p = 0
99 E_s = 0
100
101 if n_weeks == 14:
102
103     for week, filename in enumerate(filenames):
104
105         if week != 4 and week != 11:
106             E_elettrica = 0
107             E_termica = 0
108             E_in = 0
109             E_storage_in = 0
110             Dissipazione = 0
111             Emissioni = 0
```

```
112             E_ter_boiler = 0
113             Utenza_ter = 0
114             Costo = 0
115             Utenza_ele = 0
116             E_p = 0
117             E_s = 0
118 
119             if week <= 3:
120                 Month = week + 1
121             elif 4 <= week <= 10:
122                 Month = week
123             else:
124                 Month = week - 1
125 
126         tree = ET.parse(filename)                                    #
    read all values in xml file
127         for valore in tree.findall('.//Costo'):
128             Costo = float(valore.text)
129 
130         for valore in tree.findall('.//EmissioniGlobali'):
131             Emissioni = float(valore.text)
132 
133         for valore in tree.findall('.//Node_1/CHP/istanza/Pt/VAL'):
134             Pot = float(valore.text)
135             E_termica = E_termica + Pot
136             # legge le potenze termiche prodotte dal CHP
137         for valore in tree.findall('.//Node_1/CHP/istanza/Pe/VAL'):
138             Pot = float(valore.text)
139             E_elettrica = E_elettrica + Pot
140             # legge le potenze elettriche prodotte dal CHP
141         for valore in tree.findall('.//Node_1/CHP/istanza/Pc/VAL'):
142             Pot = float(valore.text)
143             E_in = E_in + Pot
144             # legge le potenze in ingresso al CHP
145 
146             # legge i risultati dei CHP con recupero di calore
147         for valore in tree.findall('.//Node_1/CHPLE/istanza/Ptle/VAL'
    ):
148             Pot = float(valore.text)
149             E_termica = E_termica + Pot
150             # legge le potenze termiche prodotte dal CHP
151         for valore in tree.findall('.//Node_1/CHPLE/istanza/Pele/VAL'
    ):
152             Pot = float(valore.text)
153             E_elettrica = E_elettrica + Pot
154             # legge le potenze elettriche prodotte dal CHP
155         for valore in tree.findall('.//Node_1/CHPLE/istanza/Pcle/VAL'
    ):
156             Pot = float(valore.text)
```

```
157             E_in = E_in + Pot
158             # legge le potenze in ingresso al CHP
159 
160             # legge i risultati dei CHPS
161         for valore in tree.findall('.//Node_1/CHPS/istanza/Pts/VAL'):
162             Pot = float(valore.text)
163             E_termica = E_termica + Pot
164             # legge le potenze termiche prodotte dal CHP
165         for valore in tree.findall('.//Node_1/CHPS/istanza/Pes/VAL'):
166             Pot = float(valore.text)
167             E_elettrica = E_elettrica + Pot
168         # legge le potenze elettriche prodotte dal CHP
169         for valore in tree.findall('.//Node_1/CHPS/istanza/Pcs/VAL'):
170             Pot = float(valore.text)
171             E_in = E_in + Pot
172             # legge le potenze in ingresso al CHP
173 
174         for valore in tree.findall('.//Node_1/Stt/istanza/PSttin/VAL'
    ):
175             Pot = float(valore.text)
176             E_storage_in = E_storage_in + Pot
177             # legge le potenze in ingresso allo storage (se c'è)
178 
179         for valore in tree.findall('.//Node_1/Dt/Dt/VAL'):
180             Pot = float(valore.text)
181             Dissipazione = Dissipazione + Pot
182         # legge la dissipazione termica (per tutti i casi)
183 
184         for valore in tree.findall('.//Node_1/Boiler/istanza/Bt/VAL')
    :
185             Pot = float(valore.text)
186             E_ter_boiler = E_ter_boiler + Pot
187             # legge le potenze termiche prodotte dal CHP
188 
189         for valore in tree.findall('.//Node_1/Ut/P/VAL'):
190             Pot = float(valore.text)
191             Utenza_ter = Utenza_ter + Pot
192         # legge la domanda termica (per tutti i casi)
193 
194         for valore in tree.findall('.//Node_1/Ue/P/VAL'):
195             Pot = float(valore.text)
196             Utenza_ele = Utenza_ele + Pot
197         # legge la domanda termica (per tutti i casi)
198 
199         for valore in tree.findall('.//Node_1/Grid/istanza/Ps/VAL'):
200             Pot = float(valore.text)
201             E_s = E_s + Pot
202             # legge elettricità venduta alla rete
203 
```

```
204         for valore in tree.findall('.//Node_1/Grid/istanza/Pp/VAL'):
205             Pot = float(valore.text)
206             E_p = E_p + Pot
207             # legge elettricità acquistata dalla rete
208 
209         E_utile = E_termica - Dissipazione
210         if week == 3 or week == 10:
211             continue
212 
213         if week == 0 or week == 2 or week == 5 or week == 7 or week
    == 8 or week == 13:
214             num_mese = 31
215             num_period = 7
216         elif week == 1:
217             num_mese = 29
218             num_period = 7
219         elif week == 6 or week == 9 or week == 12:
220             num_mese = 30
221             num_period = 7
222         elif week == 4:
223             num_period = 14
224             num_mese = 30
225         elif week == 11:
226             num_period = 14
227             num_mese = 31
228 
229         E_termica = E_termica / num_period * num_mese          #
    calculate monthly values
230         E_elettrica = E_elettrica / num_period * num_mese
231         E_storage_in = E_storage_in / num_period * num_mese
232         E_storage_in = E_storage_in / num_period * num_mese
233         Dissipazione = Dissipazione / num_period * num_mese
234         Utenza_ele = Utenza_ele / num_period * num_mese
235         E_in = E_in / num_period * num_mese
236         Emissioni = Emissioni / num_period * num_mese
237         E_ter_boiler = E_ter_boiler / num_period * num_mese
238         E_utile = E_utile / num_period * num_mese
239         Utenza_ter = Utenza_ter / num_period * num_mese
240         Costo = Costo / num_period * num_mese
241         E_p = E_p / num_period * num_mese
242         E_s = E_s / num_period * num_mese
243 
244         E_thermal.append(E_termica)                            # update
    the vectors
245         E_electrical.append(E_elettrica)
246         E_entered_storage.append(E_storage_in)
247         Dissipation.append(Dissipazione)
248         Emissions.append(Emissioni)
249         Energy_fed.append(E_in)
```

```
250             Boiler_the.append(E_ter_boiler)
251             Demand_the.append(Utenza_ter)
252             Ele_demand.append(Utenza_ele)
253             Obj_f.append(Costo)
254             Ele_p.append(E_p)
255             Ele_s.append(E_s)
256             rendimento_T = E_termica / E_in
257             rendimento_E = E_elettrica / E_in
258             rendimento_globale = E_utile / E_in + E_elettrica / E_in
259             eta_T.append(rendimento_T)
260             eta_E.append(rendimento_E)
261             eta_G.append(rendimento_globale)
262
263             with open(name_post_processing, "a", newline="") as risultati
    :                 # update csv file
264                 writer = csv.writer(risultati, delimiter=";")
265                 writer.writerow(
266                     [Month, E_termica, E_elettrica, E_in, Dissipazione,
    E_ter_boiler, Utenza_ter, E_storage_in,
267                      rendimento_globale, rendimento_T, rendimento_E,
    Emissioni, Costo, "-", "-"])
268
269 else:
270     print("Numero incorretto di file selezionati")  # if there are
    not 14 files selected
271
272 E_thermal = np.array(E_thermal)
273 E_electrical = np.array(E_electrical)
274 E_entered_storage = np.array(E_entered_storage)
275 Dissipation = np.array(Dissipation)
276 Energy_fed = np.array(Energy_fed)
277 Time = x = np.arange(1, 13, 1)
278 E_useful = E_thermal - Dissipation
279
280 with open(name_post_processing, "a", newline="") as results:  #
    update csv file with annual values
281     writer = csv.writer(results, delimiter=";")
282     E_thermal_a = sum(E_thermal)
283     Demand_ele_a = sum(Ele_demand)
284     E_electrical_a = sum(E_electrical)
285     Energy_fed_a = sum(Energy_fed)
286     Dissipation_a = sum(Dissipation)
287     Boiler_the_a = sum(Boiler_the)
288     Demand_the_a = sum(Demand_the)
289     E_entered_storage_a = sum(E_entered_storage)
290     Emissions_a = sum(Emissions)
291     Obj_f_a = sum(Obj_f)
292     E_useful_a = sum(E_useful)
```

```
293     RISP = E_electrical_a / rendimento_E_rif + E_useful_a /
        rendimento_T_rif - Energy_fed_a
294     PES = RISP / (E_electrical_a / rendimento_E_rif + E_useful_a /
        rendimento_T_rif)
295     eta_T_a = E_thermal_a / Energy_fed_a
296     eta_E_a = E_electrical_a / Energy_fed_a
297     eta_G_a = eta_E_a + E_useful_a / Energy_fed_a
298     writer.writerow(
299         ["Annual results", E_thermal_a, E_electrical_a, Energy_fed_a,
        Dissipation_a, Boiler_the_a, Demand_the_a, E_entered_storage_a,
300          eta_G_a, eta_T_a, eta_E_a, Emissions_a, Obj_f_a, RISP, PES])
301
302 # Plots:
303
304 E_the = plt.plot(Time, E_thermal, marker="o", color='green', label="
        Thermal energy CHP")
305 E_ele = plt.plot(Time, Demand_the, marker="o", color='red', label="
        Thermal load")
306 E_boi = plt.plot(Time, Boiler_the, marker="o", color='blue', label="
        Thermal energy boiler")
307 E_diss = plt.plot(Time, Dissipation, marker="o", color='orange',
        label="Dissipation")
308 plt.ylim(0, 6e6)
309 plt.legend()
310 plt.title("Annual results with 14 weeks")
311 plt.xlabel("Month")
312 plt.ylabel("kWh")
313 plt.grid()
314 plt.show()
315
316 plt.plot(Time, E_electrical, marker="o", color='green', label="
        Electrical energy CHP")
317 plt.plot(Time, Ele_demand, marker="o", color='red', label="Electrical
         load")
318 # plt.plot(Time, Energy_fed_my, marker="o", color='black', label="
        Energy fed")
319 plt.plot(Time, Ele_p, marker="o", color='blue', label="Electricity
        purchased")
320 plt.plot(Time, Ele_s, marker="o", color='orange', label="Electricity
        sold")
321 plt.ylim(0, 2.5e6)
322 plt.legend()
323 titolo = "Annual results with 14 weeks"
324 plt.title(titolo)
325 plt.xlabel("Month")
326 plt.ylabel("kWh")
327 plt.grid()
328 plt.show()
```

A.4 Codes related to chapter 5

Only the codes related to the implementation of white certificates using single netlist are reported.

CB: main code for white certificates.

```
import csv
import os
import xml.etree.ElementTree as ET
import numpy as np
import Results_for_CB
import Find_in_netlist
import Modify_maint

max_iterazioni = 6  # insert maximum number of iterations

main_path = os.getcwd()
os.system(main_path + "\main_XEMS13_v2_7.exe")  # first XEMS13 lauch

with open('XEMS13cfg.txt', 'w') as ConfigurationFile:
    ConfigurationFile.write("CSVdelimiter=,\n")
    ConfigurationFile.write("dialog=0\n")
    ConfigurationFile.write("print_var=0\n")
    ConfigurationFile.write("Sankey=sankeymatic\n")
    # to modify the configuration file, setting dialog=0
    # to avoid the tool asks again the folders

with open("defaultDIR.txt", "r", encoding='utf8') as directories:
    Paths = directories.readlines()
    netlist_name = str(Paths[3].strip("\n"))  # netlist name
    netlist_path = str(Paths[0].strip("\n"))  # netlist folder path
    components_path = str(Paths[2].strip("\n"))
    # components folder path
    netlist_only_name = Paths[3].rsplit(".", 1)[0]
    # netlist name without extension

os.chdir(netlist_path)  # move to netlist folder

components_name = Find_in_netlist.Find_components_file_name(
    netlist_path, netlist_name)
CHP_type, CHP_tag = Find_in_netlist.Find_CHP_name(netlist_path,
    netlist_name)
# CHP_type is the list of all the typologies of cogenerators present
    (ex. CHP, CHPLE)
# CHP_tag  is the list of all the names (sizes) of the cogenerators
    present (ex. CHP_1203)

```

```
38 os.chdir(components_path)  # move to components folder
39 tree = ET.parse(components_name)
40 root = tree.getroot()
41 old_maintenance = []
42 # to save the values of the original maintenance:
43 for numero_cogeneratore, cogeneratore in enumerate(CHP_type):
44     if cogeneratore == "CHP":
45         for components_CHP in tree.findall(".//CHP"):
46             if components_CHP.attrib["name"] == CHP_tag[
    numero_cogeneratore]:
47                 old_maintenance.append(float(components_CHP.attrib["
    maint"]))
48     elif cogeneratore == "CHPLE":
49         for components_CHP in tree.findall(".//CHPLE"):
50             if components_CHP.attrib["name"] == CHP_tag[
    numero_cogeneratore]:
51                 old_maintenance.append(float(components_CHP.attrib["
    maint"]))
52     elif cogeneratore == "CHPS":
53         for components_CHP in tree.findall(".//CHPS"):
54             if components_CHP.attrib["name"] == CHP_tag[
    numero_cogeneratore]:
55                 old_maintenance.append(float(components_CHP.attrib["
    maint"]))
56     else:
57         print("CHP non riconosciuto")
58 old_maint = np.array(old_maintenance)
59
60 new_maint, Dissipazione, E_termica, E_in, E_elettrica, beta,
    E_storage_in, rendimento_E, rendimento_T, rendimento_globale, RISP
    , PES, EmissioniGlobali = Results_for_CB.results(
61     main_path, old_maint)  # calculates white certificates
62 previous_maint = old_maint
63 os.chdir(netlist_path)
64 Risultati_file_name = str(netlist_only_name + "_risultati_iterazioni.
    csv")
65 # name of iterations results csv file
66
67 with open(Risultati_file_name, "w") as Risultati_iterazioni:
68     writer = csv.writer(Risultati_iterazioni, delimiter=";")
69     writer.writerow(
70         ["Iterazione", "Dissipazione [kWh]", "Energia termica [kWh]",
     "Energia di alimentazione [kWh]", "Energia elettrica [kWh]", "
    Beta",
71          "Energia storage ingresso [kWh]", "Rendimento elettrico", "
    Rendimento termico", "Rendimento globale", "RISP [kWh]", "PES",
72          "Emissioni CO2 [kg]", "Maintenance [euro/kWh]"])
73
74 iterazione = int(0)
```

```
beta_diff = 0.5
# ciclo while iterativo
while beta_diff > 0.01:
    previous_beta = beta
    previous_RISP = RISP
    if iterazione == 0:
        Modify_maint.modify_maint(main_path, components_name, CHP_type, CHP_tag, new_maint)
        pass
    else:
        os.chdir(main_path)
        os.system(main_path + "\main_XEMS13_v2_7.exe")
        new_maint, Dissipazione, E_termica, E_in, E_elettrica, beta, E_storage_in, rendimento_E, rendimento_T, rendimento_globale, RISP, PES, EmissioniGlobali = Results_for_CB.results(
            main_path, old_maint)
        Modify_maint.modify_maint(main_path, components_name, CHP_type, CHP_tag, new_maint)

        beta_diff = abs(previous_beta - beta)

    os.chdir(netlist_path)

    with open(Risultati_file_name, "a", newline="") as Risultati_iterazioni:
        writer = csv.writer(Risultati_iterazioni, delimiter=";")
        writer.writerow([iterazione, Dissipazione, E_termica, E_in, E_elettrica, beta, E_storage_in, rendimento_E, rendimento_T, rendimento_globale, RISP, PES, EmissioniGlobali, previous_maint])

    previous_maint = new_maint
    if -0.000001 < previous_beta < 0.000001:
        beta_diff = 1  # to make it remain in the loop
        if iterazione > 0:
            scarto_RISP = abs((previous_RISP - RISP) / previous_RISP)
            if scarto_RISP < 0.01:
                break

    iterazione = iterazione + 1

    if iterazione == max_iterazioni:
        print("Massimo numero di iterazioni raggiunto")
        break

os.chdir(main_path)
with open('XEMS13cfg.txt', 'w') as ConfigurationFile:
    ConfigurationFile.write("CSVdelimiter=,\n")
    ConfigurationFile.write("dialog=1\n")
    ConfigurationFile.write("print_var=0\n")
```

```
117     ConfigurationFile.write("Sankey=sankeymatic\n")
118     # to modify configuration file to make XEMS13 ask the folders
        again
119     # at the next launch
```

Find_in_netlist: support functions that finds element in the netlist text file.

```
1  import os
2
3
4  def Find_components_file_name(netlist_path, netlist_name):
5      os.chdir(netlist_path)
6      with open(netlist_name, "r", encoding='utf8') as netlist:
7          Lines = netlist.readlines()
8          for line_counter, line in enumerate(Lines):
9              linea = str(line.strip("\n"))
10             if linea == "@XML Library Components File":
11                 num_components_file_name = line_counter + 1
12                 components_name = str(Lines[num_components_file_name
       ].strip("\n")) + ".xml"
13                 # saves the components xml file name
14     return components_name
15
16
17 def Find_CHP_name(netlist_path, netlist_name):
18     os.chdir(netlist_path)
19     primo_CHP = int(0)
20     ultimo_CHP = int(0)
21     CHP_type = []
22     CHP_tag = []
23     with open(netlist_name, "r", encoding='utf8') as netlist:
24         Lines = netlist.readlines()
25         for line_counter, line in enumerate(Lines):
26             linea = str(line.strip("\n"))
27             if linea == "@Dispatchable Electric Input":
28                 primo_CHP= line_counter + 1
29             if primo_CHP != 0 and ultimo_CHP < primo_CHP and linea ==
       '':
30                 ultimo_CHP = line_counter
31
32     for num_CHP in range(primo_CHP, ultimo_CHP):
33       CHP_name = str(Lines[num_CHP].strip("\n"))
34       # saves the names of the used CHP
35       CHP_names = CHP_name.split()
36       CHP_type.append(CHP_names[2])  # CHP, CHPLE o CHPS
37       CHP_tag.append(CHP_names[3])
38
39     return(CHP_type, CHP_tag)
```

Results_for_CB: analyzes the simulation results for a single netlist.

```
import os
import os.path
import xml.etree.ElementTree as ET


def results(main_path, old_maint):
    E_termica = 0
    E_elettrica = 0
    Dissipazione = 0
    E_in = 0
    E_storage_in = 0
    EmissioniGlobali = "undefined"

    a = float(0.086)
    rendimento_E_rif = float(0.46)
    rendimento_T_rif = float(0.9)
    CB_val = float(264.08)  # euro per CB
    # values taken from "Certificati Bianchi"

    os.chdir(main_path)

    with open("defaultDIR.txt", "r", encoding='utf8') as directories:
        Paths = directories.readlines()
        # Paths[2] contains the path of the components directory
        # Paths[0] contains the path of the work directory with the results
        # Paths[3] contains name of the netlist .txt
        netlist_path = str(Paths[0].strip("\n"))  # netlist folder path
        AllResults = Paths[3].rsplit(".", 1)[0] + ".xml"  # name of xml results file

    os.chdir(netlist_path)  # move to the netlist folder to read the results
    tree = ET.parse(AllResults)

    # CHP
    for valore in tree.findall('.//Node_1/CHP/istanza/Pt/VAL'):
        Pot = float(valore.text)
        E_termica = E_termica + Pot
        # reads thermal powers produced by CHP
    for valore in tree.findall('.//Node_1/CHP/istanza/Pe/VAL'):
        Pot = float(valore.text)
        E_elettrica = E_elettrica + Pot
        # reads electrical powers produced by CHP
    for valore in tree.findall('.//Node_1/CHP/istanza/Pc/VAL'):
```

```
43          Pot = float(valore.text)
44          E_in = E_in + Pot
45          # reads powers entering CHP
46
47          # CHPLE
48      for valore in tree.findall('.//Node_1/CHPLE/istanza/Ptle/VAL'):
49          Pot = float(valore.text)
50          E_termica = E_termica + Pot
51          # reads thermal powers produced by CHPLE
52      for valore in tree.findall('.//Node_1/CHPLE/istanza/Pele/VAL'):
53          Pot = float(valore.text)
54          E_elettrica = E_elettrica + Pot
55          # reads electrical powers produced by CHPLE
56      for valore in tree.findall('.//Node_1/CHPLE/istanza/Pcle/VAL'):
57          Pot = float(valore.text)
58          E_in = E_in + Pot
59          # reads powers entering CHPLE
60
61          # CHPS
62      for valore in tree.findall('.//Node_1/CHPS/istanza/Pts/VAL'):
63          Pot = float(valore.text)
64          E_termica = E_termica + Pot
65          # reads thermal powers produced by CHPS
66      for valore in tree.findall('.//Node_1/CHPS/istanza/Pes/VAL'):
67          Pot = float(valore.text)
68          E_elettrica = E_elettrica + Pot
69          # reads electrical powers produced by CHPS
70      for valore in tree.findall('.//Node_1/CHPS/istanza/Pcs/VAL'):
71          Pot = float(valore.text)
72          E_in = E_in + Pot
73          # reads powers entering CHPS
74
75      for valore in tree.findall('.//Node_1/Stt/istanza/PSttin/VAL'):
76          Pot = float(valore.text)
77          E_storage_in = E_storage_in + Pot
78          # reads powers enetering the storage (if present)
79
80      for valore in tree.findall('.//Node_1/Dt/Dt/VAL'):
81          Pot = float(valore.text)
82          Dissipazione = Dissipazione + Pot
83          # reads thermal dissipation
84
85      for valore in tree.findall('.//EmissioniGlobali'):
86          EmissioniGlobali = float(valore.text)
87      # reads CO2 emissions
88
89      beta = Dissipazione / E_termica
90
91      rendimento_E = E_elettrica / E_in
```

```
92      rendimento_T = E_termica / E_in
93      E_utile = E_termica - Dissipazione
94      rendimento_globale = (E_elettrica + E_utile) / E_in
95      RISP = E_elettrica / rendimento_E_rif + E_utile /
    rendimento_T_rif - E_in  # Risparmio di energia primaria
96      PES = RISP / (E_elettrica / rendimento_E_rif + E_utile /
    rendimento_T_rif)  # primary energy saving
97      Ccb = a * CB_val / rendimento_E * (rendimento_E /
    rendimento_E_rif + (
98              1 - beta) * rendimento_T / rendimento_T_rif - 1) / 1000
     #  euro / kWh
99      # calculates the white certificates
100
101     new_maint = old_maint - Ccb  # update maintenance of each
    cogenerator
102
103     os.chdir(main_path)
104
105     return (new_maint, Dissipazione, E_termica, E_in, E_elettrica,
    beta, E_storage_in, rendimento_E, rendimento_T,
106             rendimento_globale, RISP, PES, EmissioniGlobali)
```

Modify_maint: modifies maintenance values of cogenerators in the xml components file.

```
1  import os
2  import xml.etree.ElementTree as ET
3
4
5  def modify_maint(main_path, components_name, CHP_type, CHP_tag,
       new_maint):
6       os.chdir(main_path)
7       with open("defaultDIR.txt", "r", encoding='utf8') as directories:
8           Paths = directories.readlines()
9           components_path = str(Paths[2].strip("\n"))  # components
       folder path
10          os.chdir(components_path)  # moves to the components folder
11          tree = ET.parse(components_name)
12
13          for num_cogeneratore, cogeneratore in enumerate(CHP_type):
14              new_maintenance = new_maint[num_cogeneratore]
15              if cogeneratore == "CHP":
16                  for components_CHP in tree.findall(".//CHP"):
17                      if components_CHP.attrib["name"] == CHP_tag[
       num_cogeneratore]:
18                          components_CHP.attrib["maint"] = str(
       new_maintenance)
19
```

```
            elif cogeneratore == "CHPLE":
                for components_CHP in tree.findall(".//CHPLE"):
                    if components_CHP.attrib["name"] == CHP_tag[
    num_cogeneratore]:
                        components_CHP.attrib["maint"] = str(
    new_maintenance)

            elif cogeneratore == "CHPS":
                for components_CHP in tree.findall(".//CHPS"):
                    if components_CHP.attrib["name"] == CHP_tag[
    num_cogeneratore]:
                        components_CHP.attrib["maint"] = str(
    new_maintenance)

        tree.write(components_name, encoding="utf-8")
    pass
```

A.5 Codes relative to chapter 6

Parametric: main code for the parametric run

```
import csv
import os
import Find_in_netlist
import Results_for_param_an_r_days
import Results_for_param_an
import Modify_size


Component_types = []
Component_names = []

Component_types.append("CHP")          # append type of first
    component
Component_names.append("CHP_1413")     # append size of first
    component
Component_types.append("CHP")          # and so on...
Component_names.append("CHP_1487")
eta_E_list = [0.428831563, 0.429768786]
eta_T_list = [0.435811836, 0.419364162]
Emissions_list = [111.147, 111.147]

Min_size = 1000                        # insert minimum size (kWe)
Max_size = 2000                        # insert maximum size
Gap_size = 100                         # insert difference between two
     iterations
```

```
23
24 ########################################
25
26 main_path = os.getcwd()
27 os.system(main_path + "\main_XEMS13_v2_7.exe")  # first XEMS run with
       the existing components
28
29 with open('XEMS13cfg.txt', 'w') as ConfigurationFile:
30     ConfigurationFile.write("CSVdelimiter=,\n")
31     ConfigurationFile.write("dialog=0\n")
32     ConfigurationFile.write("print_var=0\n")
33     ConfigurationFile.write("Sankey=sankeymatic\n")
34     # file is modified with dialog=0 so the folders are not requested
       anymore
35
36 with open("defaultDIR.txt", "r", encoding='utf8') as directories:
37     Paths = directories.readlines()
38     netlist_names = Paths[3:len(Paths)]  # salva il nome delle
      netlist
39     netlist_path = str(Paths[0].strip("\n"))  # il percorso della
      cartella delle netlist
40     components_path = str(Paths[2].strip("\n"))  # il percorso della
      cartella dei componenti
41     netlist_only_names = [0] * (len(Paths) - 3)
42     for i, element in enumerate(netlist_names):
43         netlist_names[i] = str(element).strip("\n")
44         netlist_only_names[i] = str(element).rsplit(".", 1)[0]  #
      senza estensione
45
46 os.chdir(netlist_path)
47 Risultati_file_name = str(
48     netlist_only_names[0]) + "_risultati_parametric.csv"  # name of
      the csv file
49 with open(Risultati_file_name, "w") as Risultati_iterazioni:
50     writer = csv.writer(Risultati_iterazioni, delimiter=";")
51     writer.writerow(
52         ["Sizes", "Obj. f. [€]","CHP electrical [kWh]", "CHP thermal
      [kWh]", "Boiler thermal [kWh]", "Dissipation [kWh]", "Global
      efficiency", "RISP [kWh]", "PES", "CO2 emissions [kg]"])
53
54 components_name = Find_in_netlist.Find_components_file_name(
      netlist_path, netlist_names[0])
55 Obj_f_list = []  # list of all the objective functions
56 Obj_f, E_ele_CHP, E_the_CHP, E_ter_boiler, Dissipation, eta_G, RISP,
      PES, Emissions = Results_for_param_an_r_days.results(
57     main_path, netlist_only_names)
58 Obj_f_list.append(Obj_f)
59 os.chdir(netlist_path)
```

```
60 with open(Risultati_file_name, "a", newline="") as
       Risultati_iterazioni:
61     writer = csv.writer(Risultati_iterazioni, delimiter=";")
62     Size = "originals"
63     writer.writerow(['Size', 'Obj_f', 'E_ele_CHP', 'E_the_CHP', 'E_ter_boiler',
       'Dissipation', 'eta_G', 'RISP', 'PES', 'Emissions'])
64
65 Range = Max_size - Min_size
66 iterations = int(Range / Gap_size)
67 for i in range(0, iterations + 1):
68     Size = Min_size + i * Gap_size
69     Modify_size.modify_size(main_path, components_name,
       Component_names, Component_types, Size, eta_E_list, eta_T_list,
       Emissions_list)
70     os.system(main_path + "\main_XEMS13_v2_7.exe")
71     Obj_f, E_ele_CHP, E_the_CHP, E_ter_boiler, Dissipation, eta_G,
       RISP, PES, Emissions = Results_for_param_an_r_days.results(
           main_path, netlist_only_names)
73     Obj_f_list.append(Obj_f)
74     os.chdir(netlist_path)
75     with open(Risultati_file_name, "a", newline="") as
       Risultati_iterazioni:
76         writer = csv.writer(Risultati_iterazioni, delimiter=";")
77         writer.writerow(['Size', 'Obj_f', 'E_ele_CHP', 'E_the_CHP',
       E_ter_boiler, Dissipation, eta_G, RISP, PES, Emissions])
78
79 Minimum_o_f = min(Obj_f_list)
80 Index = Obj_f_list.index(Minimum_o_f)
81 if Index == 0:
82     print("The configuration that returns the lowest value of the
       objective function is the original one")
83 else:
84     print("The configuration that returns the lowest value of the
       objective function is the one with the sizes: ", Min_size + (Index
       - 1) * Gap_size)
85
86 os.chdir(main_path)
87 with open('XEMS13cfg.txt', 'w') as ConfigurationFile:
88     ConfigurationFile.write("CSVdelimiter=,\n")
89     ConfigurationFile.write("dialog=1\n")
90     ConfigurationFile.write("print_var=0\n")
91     ConfigurationFile.write("Sankey=sankeymatic\n")
92     # modifica il file impostando nuovamente dialog=1 alla fine delle
       iterazioni
```

Modify_size: modifies sizes of the chosen components in the xml components file.

```
import os
import xml.etree.ElementTree as ET


def modify_size(main_path, components_name, Component_names,
    Component_types, Size, eta_E_list, eta_T_list, Emissions_list):
    os.chdir(main_path)
    with open("defaultDIR.txt", "r", encoding='utf8') as directories:
        Paths = directories.readlines()
        components_path = str(Paths[2].strip("\n"))  # il percorso
    della cartella dei componenti
        os.chdir(components_path)  # va nella cartella dei componenti
     per modificare l'xml
    os.chdir(components_path)
    tree = ET.parse(components_name)
    root = tree.getroot()

    for i, component in enumerate(Component_types):
        if component == "CHP":
            for element in root.findall(component):
                if element.attrib["name"] == Component_names[i]:
                    eta_T = eta_T_list[i]
                    eta_E = eta_E_list[i]
                    Original_emi = Emissions_list[i]
                    Original_size = Component_names[i]
                    Original_size = float(Original_size[4:len(
    Original_size)])

                    pl = 0.5
                    for PowerLevel in element.findall("CHP_PowerLevel
    "):
                        if pl == 0.5:
                            eta_E_ = eta_E - 0.04
                            eta_T_ = eta_T + 0.04
                            PL = PowerLevel.find("CHP_PowerLevel1")
                            PL.text = str(Size * pl)
                            PL = PowerLevel.find("CHP_PowerLevel2")
                            PL.text = str(Size * pl / eta_E_ * eta_T_
    )
                            PL = PowerLevel.find("CHP_PowerLevel3")
                            PL.text = str(Size * pl / eta_E_)
                            PL = PowerLevel.find("CHP_PowerLevel6")
                            PL.text = str( pl * Size * Original_emi /
     Original_size)

                        if pl == 0.75:
                            eta_E_ = eta_E - 0.02
                            eta_T_ = eta_T + 0.02
```

```
42                              PL = PowerLevel.find("CHP_PowerLevel1")
43                              PL.text = str(Size * pl)
44                              PL = PowerLevel.find("CHP_PowerLevel2")
45                              PL.text = str(Size * pl / eta_E_ * eta_T_
    )
46                              PL = PowerLevel.find("CHP_PowerLevel3")
47                              PL.text = str(Size * pl / eta_E_)
48                              PL = PowerLevel.find("CHP_PowerLevel6")
49                              PL.text = str( pl * Size * Original_emi /
     Original_size)
50
51                          if pl == 1:
52                              PL = PowerLevel.find("CHP_PowerLevel1")
53                              PL.text = str(Size)
54                              PL = PowerLevel.find("CHP_PowerLevel2")
55                              PL.text = str(Size / eta_E * eta_T)
56                              PL = PowerLevel.find("CHP_PowerLevel3")
57                              PL.text = str(Size / eta_E)
58                              PL = PowerLevel.find("CHP_PowerLevel6")
59                              PL.text = str( pl * Size * Original_emi /
     Original_size)
60
61                          pl = pl + 0.25
62          elif component == "Boiler":
63              for element in root.findall(component):
64                  if element.attrib["name"] == Component_names[i]:
65                      eta_E = eta_E_list[i]
66                      pl = 0
67                      for PowerLevel in element.findall("
    Boiler_PowerLevel"):
68                          if pl == 0:
69                              PL = PowerLevel.find("Boiler_PowerLevel1"
    )
70                              PL = PowerLevel.find("Boiler_PowerLevel2"
    )
71
72                          if pl == 1:
73                              eta_E_ = eta_E
74                              PL = PowerLevel.find("Boiler_PowerLevel1"
    )
75                              PL.text = str(Size)
76                              PL = PowerLevel.find("Boiler_PowerLevel2"
    )
77                              PL.text = str(Size / eta_E_)
78
79                          pl = pl + 1
80
81      tree.write(components_name, encoding="utf-8")
82      os.chdir(main_path)
```

```
83      return
```

Bibliography

[1] European Union. *A European Green Deal* (cit. on p. 1).

[2] International District Heating Association. *District Heating* (cit. on p. 2).

[3] Thibaut Abergel and Chiara Delmastro. *Heating*. Tech. rep. International Energy Agency, 2020 (cit. on p. 2).

[4] GSE. *TELERISCALDAMENTO E TELERAFFRESCAMENTO IN ITALIA: ONLINE IL RAPPORTO GSE* (cit. on p. 2).

[5] COGEN Europe. online slides. 2020 (cit. on p. 3).

[6] GSE - Gestore Servizi Energetici. *Certificati Bianchi - cosa sono e a cosa servono* (cit. on p. 3).

[7] GSE. *COGENERAZIONE AD ALTO RENDIMENTO* (cit. on p. 3).

[8] Maurizio Repetto, Sergio Olivero, Paolo Lazzeroni, Federico Stirano, and Vittorio Verda. *Design of a polygeneration system with optimal management for a district heating and cooling network*. Tech. rep. Politecnico di Torino and LINKS, 2019 (cit. on p. 6).

[9] Maurizio Repetto, Paolo Lazzeroni, and Nicolas Perez-Mora. *XEMS13: An Hybrid-Energy Generation Management System*. Tech. rep. University of Balearic Islands, Istituto Superiore sui Sistemi Territoriali per l'Innovazione, and Politecnico di Torino, 2016 (cit. on p. 6).

[10] Presidente della Repubblica. *DECRETO DEL 26 agosto 1993, n. 412*. 1993 (cit. on p. 10).

[11] K. Poncelet, H. Höschle, E. Delarue, A. Virag, and W. D'haeseleer. *Selecting representative days for capturing the implications of integrating intermittent renewables in generation expansion planning problems*. Tech. rep. Ku Leuven Energy Institute, 2016 (cit. on pp. 11, 26, 28).

[12] Bram van der Heijde, Annelies Vandermeulen, Robbe Salenbien, and Lieve Helsen. *Representative days selection for district energy system optimization: a solar district heating system with seasonal storage*. Tech. rep. KU Leuven, VITO, and EnergyVille, 2019 (cit. on p. 11).

[13] A. K. Jain, M. N. Marty, and P. J. Flynn. *Data clustering: A review*. ACM Computing Surveys, 1999 (cit. on p. 12).

[14] Samira Fazlollahi, Stephane Laurent Bungener, Pierre Mandel, Gwenaelle Becker, and François Marèchal. *Multi-objectives, multi-period optimization of district energy systems: I. Selection of typical operating periods*. Tech. rep. Polytechnique de Lausanne and Veolia Environment Recherche et Innovation, 2014 (cit. on pp. 12, 16).

[15] Trudie Strauss and Michael Johan von Maltitz. *Generalising Ward's Method for Use with Manhattan Distances*. URL: `https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0168288` (cit. on p. 13).

[16] Scikit-learn. *2.3. Clustering*. URL: `https://scikit-learn.org/stable/modules/clustering.html#clustering` (cit. on pp. 13–15, 19, 33).

[17] Paul Nahmmacher, Eva Schmid, Lion Hirth, and Brigitte Knopf. *Carpe diem: A novel approach to select representative days for long-term power system models with high shares of renewable energy sources*. Tech. rep. Potsdam Institute for Climate Impact Research et al., 2014 (cit. on p. 14).

[18] Wikipedia. *Lloyd's algorithm*. URL: `https://en.wikipedia.org/wiki/Lloyd%5C%27s_algorithm` (cit. on p. 15).

[19] Serafeim Loukas. *K-Means Clustering: How It Works & Finding The Optimum Number Of Clusters In The Data*. URL: `https://ai.plainenglish.io/k-means-clustering-how-it-works-finding-the-optimum-number-of-clusters-in-the-data-eff16dd868dd` (cit. on p. 15).

[20] Nagesh Perumandla. *Unsupervised learning: Clustering Algorithms*. URL: `https://medium.com/analytics-vidhya/unsupervised-learning-clustering-algorithms-k-means-hierarchical-and-dbscan-clustering-3981c23efb93` (cit. on p. 15).

[21] David Arthur and Sergei Vassilvitskii. *k-means++: The Advantages of Careful Seeding*. Tech. rep. 2006 (cit. on p. 16).

[22] Wikipedia. *Analysis of variance*. URL: `https://en.wikipedia.org/wiki/Analysis_of_variance` (cit. on p. 19).

[23] Gianfranco Chicco. *Overview and performance assessment of the clustering methods for electrical load pattern grouping*. Tech. rep. Politecnico di Torino, 2012 (cit. on p. 33).

[24] Paolo Lazzeroni. personal communication. Aug. 2021 (cit. on pp. 46, 48, 77).

[25] Giorgia Olivero. personal communication. Aug. 2021 (cit. on pp. 46, 48).

[26] Jenbacher. *Datasheet*. 2013 (cit. on p. 48).

[27] Repetto Maurizio and Lazzeroni Paolo. *Calcolo proventi certificati bianchi* (cit. on pp. 57, 63).

[28] Gazzetta ufficiale dell'Unione europea. *DIRETTIVA 2004/8/CE DEL PARLAMENTO EUROPEO E DEL CONSIGLIO sulla promozione della cogenerazione basata su una domanda di calore utile nel mercato internobdell'energia.* 2004 (cit. on p. 62).

[29] Nicola Pedroni. *Centrali termoelettriche.* slides. 2019 (cit. on p. 79).

Acknowledgements

First of all I would like to thank professor Maurizio Repetto, who has given me the opportunity to deepen into topics that I had not the chance to study during my courses, but which are very useful for the energy field and which I hope to explore further in my next career.
I would like also to thank Eng. Paolo Lazzeroni, always very helpful and kind.
A special thank goes to the company EGEA for the support that Eng. Giorgia Olivero gave my during the thesis and Eng. Cravero Giancarlo and Eng. Federico Mollo during my internship.

I want to thank my parents Rossella and Riccardo for being always supportive. A thank for my "old" friends Alessia and Serena (far only physically), and to all Turin friends, collegians and colleagues who contributed making these years a special experience, first among them my boyfriend Antonio for encouraging and always believing in me.

It has been a wonderful journey.