POLITECNICO
MILANO 1863

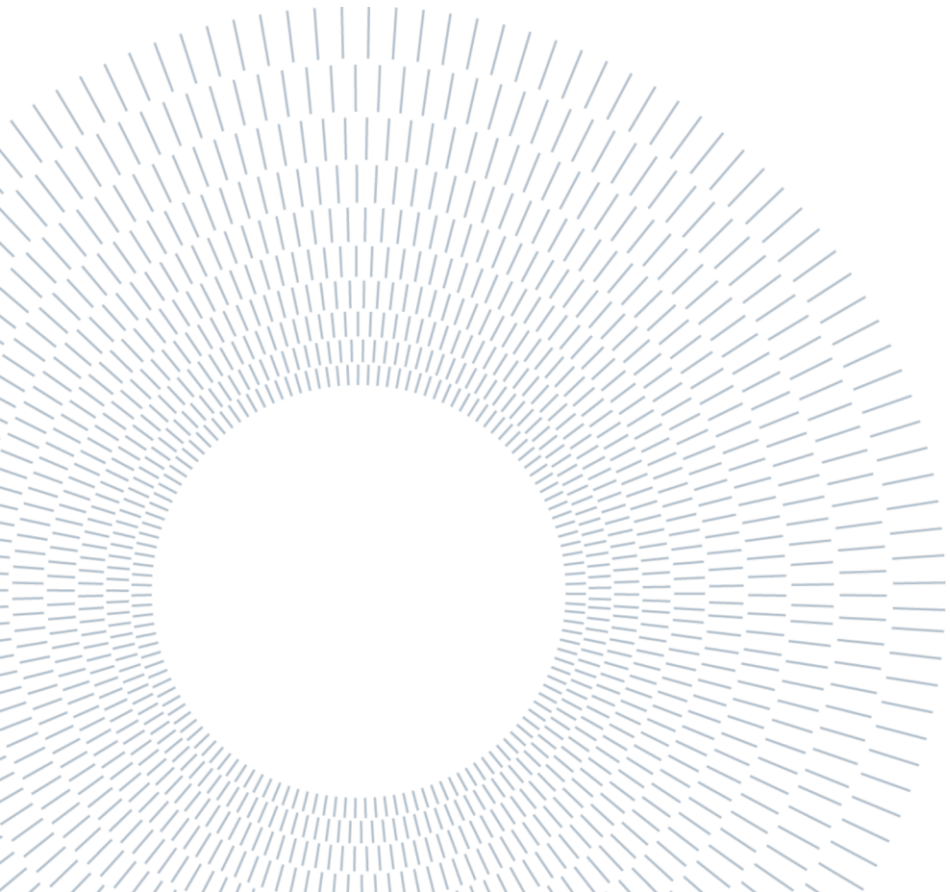
SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

The Typical Load Days: A Clustering Problem

TESI DI LAUREA MAGISTRALE IN
ELECTRICAL ENGINEERING-INGEGNERIA ELETTRICA

Author: **Simone Pivari**

Student ID:	946762
Advisor:	Alberto Berizzi
Co-advisor:	Marina Petrelli
Academic Year:	2021-22



Abstract

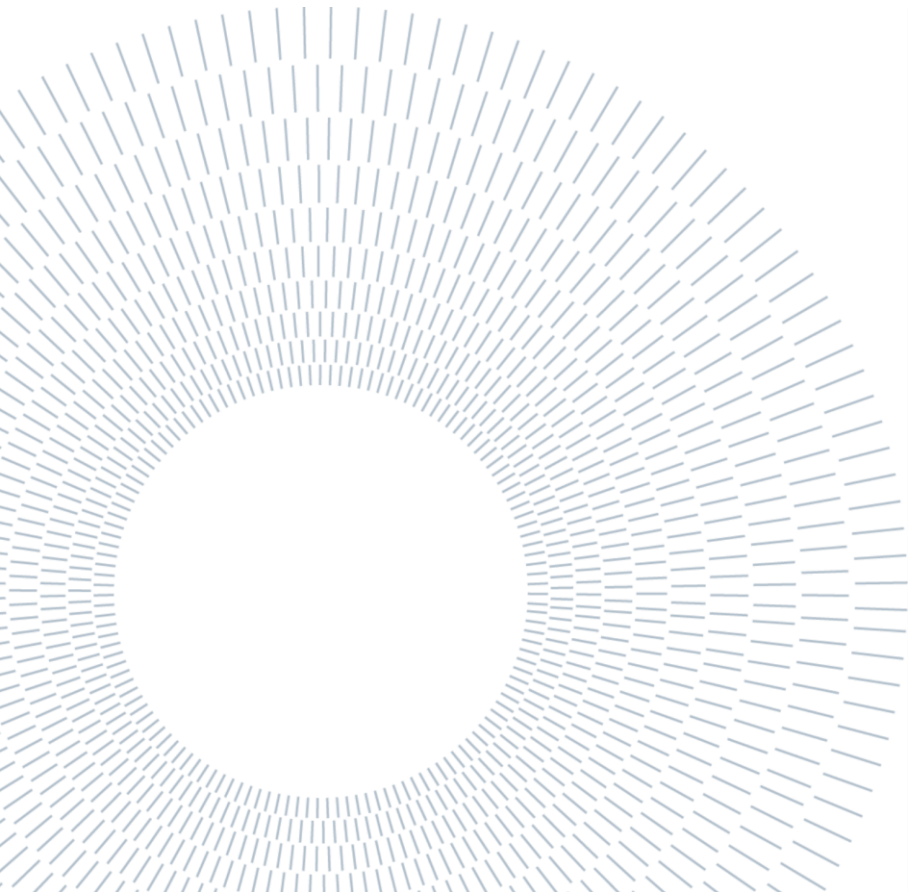
This thesis combines different subjects. Data science, mathematical analysis and electrical engineering collaborate here to determine the representative or typical electrical load days along one year by applying a quite recent technique, under a strong development process: time series clustering. Indeed, though three different clustering algorithms, a dataset composed by 365 daily load curves is clustered into 36 or less clusters, and for each cluster a representative day is selected (typical load day). In the introduction, the importance of clustering in everyday life will be discussed and deepened though real-life examples. Instead, in the first chapter, a solid theoretical basis on time series, similarity measures and clustering will be provided. Moreover, the second chapter will analyze in detail the three employed clustering algorithms, while their practical implementation will be examined in the third chapter to determine the typical load days from a real-life dataset. The considered softwares to perform the clustering process are Matlab, typically used in engineering field, and R-studio, common instead in data-analysis. The obtained results will show that is possible to use only the typical days, instead than the complete dataset, as input to many specific algorithms to greatly reduce the computational time, achieving anyway coherent results. Finally, the conclusion will highlight again the importance of clustering in addressing modern problems and will provide a possible further development of the standard algorithms to solve some issues that are nowadays under research.

Key-words: typical days; representative days; clustering; time series, similarity measure; load curve.

Abstract in lingua italiana

Questa tesi combina diversi ambiti scientifici. In particolare, data-science, analisi matematica e ingegneria elettrica vengono impiegate per determinare i giorni di carico elettrico rappresentativi o tipici durante un anno, applicando una tecnica abbastanza giovane ma che sta subendo un forte processo di sviluppo: il clustering di serie temporali. Infatti, attraverso tre diversi algoritmi di clustering, un dataset composto da 365 curve di carico giornaliere è stato raggruppato in 36 (o meno) cluster, e per ogni cluster viene selezionato un giorno rappresentativo (giorno di carico tipico). Nell'introduzione, l'importanza del clustering nella vita di tutti i giorni sarà discussa e approfondita attraverso esempi di vita reale (per esempio analizzando il caso di Google). Nel primo capitolo, invece, verranno fornite solide basi teoriche su serie temporali, misure di similarità e clustering. Inoltre, il secondo capitolo analizzerà in dettaglio i tre algoritmi di clustering impiegati, mentre la loro implementazione pratica sarà esaminata nel terzo capitolo allo scopo di determinare i giorni di carico tipici da un dataset reale. I software utilizzati per eseguire il processo di clustering sono Matlab, tipicamente impiegato in campo ingegneristico, e R-studio, comune invece nell'analisi dei dati. I risultati ottenuti mostreranno che è possibile utilizzare solo i giorni tipici, invece del dataset completo, come input di molti algoritmi specifici per ridurre notevolmente i tempi di calcolo ma ottenendo comunque risultati coerenti. Infine, la conclusione evidenzierà ancora l'importanza del clustering nell'affrontare i problemi moderni e fornirà un possibile ulteriore sviluppo degli algoritmi standard per risolvere alcuni problemi tutt'oggi oggetto di ricerca.

Parole chiave: giorni tipici, giorni rappresentativi, clustering, serie temporali; misura di similarità; curva di carico.



Contents

Abstract ..... **i**
Abstract in lingua italiana ..... **iii**
Contents ..... **v**
Introduction to Clustering **1**
1. The scientific approach to Clustering **13**
1.1 Time-series analysis 13
1.1.1 Deterministic approach to time series analysis 15
1.1.2 Statistical approach to time series analysis 27
1.2 Similarity and dissimilarity measures 33
1.2.1 Dynamic time warping 39
1.3 Time series clustering 54
2. Clustering algorithms and optimal number of clusters **69**
2.1 K-means clustering algorithm 69
2.2 K-medoids clustering algorithm 74
2.3 Ward hierarchical algorithm 78
2.4 Optimal number of clusters and clustering evaluation 83
3. Clustering algorithms implementation: an Electrical Engineering problem. **91**
3.1 Matlab clustering implementation 94
3.2 R-studio clustering implementation 108
3.3 Future developments 122
4. Conclusion **127**
Bibliography **131**
A. Appendix A **137**
B. Appendix B **141**
List of Figures **143**

List of Tables **147**

2. List of Symbols **149**

3. Acknowledgements **153**

Introduction to Clustering

This thesis has the purpose to determine the so-called *typical* or *representative load days*. More precisely, starting from 365 daily load curves of a real dataset (which will be further examined), a reduced number of load curves will be obtained through a clustering process. The derived curves are the representative load days and are extremely useful to reduce the computational time of many algorithms. Indeed, by considering just the typical days rather than the complete dataset, coherent results can be achieved in a fraction of the required time. In this introduction, the importance of clustering in everyday life will be discussed and deepened though real-life examples. Instead, in the first chapter, a solid theoretical basis on time series, similarity measures and clustering will be provided. Moreover, the second chapter will analyze in detail the three employed clustering algorithms, while their practical implementation will be examined in the third chapter to determine the typical load days from a real-life dataset. The obtained results will show that is possible to use only the typical days, instead than the complete dataset, as input to many specific algorithms to greatly reduce the computational time, achieving anyway coherent results. Finally, the conclusion will highlight again the importance of clustering in addressing modern problems and will provide a possible further development of the standard algorithms to solve some issues that are nowadays under research.

Clustering is a technique used to partition similar data elements into homogeneous groups without advance knowledge of the groups definition. Clusters are formed so that objects have maximum similarity with the other objects in the same group and minimum similarity with objects in the other groups. In particular, assume to have a set of objects to be analyzed. Note that the term “objects” is used instead of the term “elements” to underline that the set could be composed by physical objects or by abstract elements, such as data collected from measurement stations. Moreover, assume that the class is not known for each of the objects in the set. The process of grouping the set in classes with similar objects is denoted as clustering. It is important to underline that, even if the class of the single object in the set is not known, different possible classifications are known. Then, considering all the

possible classification criteria, a grouping method is defined according to the set of objects under consideration. For the sake of clearness, let's consider the following practical examples, which show also that clustering is an essential process in many fields and that it permeates human life in its possible aspect.

Natural clustering in human life

During childhood, each infant learns to distinguish between animal and plant reign and even between different species inside of each reign (e.g., between cats and dogs, both part of the animal reign). Growing up, the child continuously improves its subconscious classification schemes evolving, for instance, from "color-based", "shape-based" or "dimension-based" criteria to more sophisticated classification criteria like "number of legs" or "natural ambient". Moreover, exploiting the knowledge learnt during the grow-up process (school education, self-education, internet), even more theoretical classifications are possible, like "type of respiratory system". This shows up that the clustering process is actually a natural activity that our subconscious performs spontaneously to have a clearer view of a set of objects so that, even if the specific class of each object is not known, it is possible to assign a specific label to a subset of objects, according to a classification criterion.

Clustering in marketing and economy [1], [2]

The clustering process can help companies to determine different groups of customers, classifying the customers set according to different possible criteria (for instance: buying preferences, age, maximum budget, demand elasticity). Moreover, data-clustering can be essential in identifying trends in stock exchange quotations by grouping data points of a considered graph according to some similarity criteria. On the other hand, the clustering process could be useful also to a potential customer and is used by many websites that identify, for instance, groups of car insurances with the same policy or groups of houses in a city according to same characteristics, same value or same location.

Clustering in biology [3]

The clustering process can be exploited to derive plants and animals taxonomies, to catalogue genes with similar functionalities and to examine similar characteristics between populations or species.

Clustering in data mining [4]

The clustering process can be exploited to examine data distributions, defining the most important distributions and focalizing in studying their characteristics.

Alternatively, it can be used as a preprocessing step for data mining algorithms, that will consider only the obtained clusters as input, to reduce the computational time.

The previous examples show that clustering is an essential activity to improve the performance in multiple fields, scientific and non, like subconscious processes which are part of everyday life. Despite the evident advantages, the scientific approach to clustering is a modern subject, under a strong development process. It is now part of many research areas like data mining, statistics, machine learning, space database technology (e.g., identifying terrestrial areas with similar use or characteristics), biology, marketing. Last but not least, the clustering process can be also useful for outliers detection, finding values that are very "distant" from the clusters (for instance, exceptional transactions with a credit card could indicate a fraud or illegal e-commerce activity). Note that the term "distant" is used on purpose, since in the next chapter different possible distance measures will be presented. It has been shown [5] that different works exploiting clustering can be found in many different subjects such as geology [6], bioinformatics [7], biology [8], human motion analysis [9], space exploration [10], handwriting recognition [11], multimedia [12], and finance [13]. Moreover, there are some comprehensive surveys and reviews that focus on comparative aspects of time-series clustering experiments [14], [15] which show a trend of increased activity and research in the last twenty years.

As the previous examples show, clustering is an extremely important process that permeates everyday life in its possible aspect. Among all the research areas, this thesis focuses on data clustering, also denoted as time series clustering. A formal definition of time series will be provided in the next chapter; so, for now, it is enough to state that a time series is composed by a list of two-coordinates points, in which the first coordinate is the time instant while the second coordinate is the value of the series at the considered instant. According to this unformal definition, it is clear that a time series can be represented as a two-dimensional graph on a xy plane, where the time instants are on the x-axis and the values of the time series are on the y-axis. Clustering of time series data is mostly used to discover interesting patterns in the time series dataset. More precisely, two main tasks can be defined:

- Find patterns that appear frequently in the dataset;
- Find patterns that appear surprisingly in the dataset.

As the title suggest, this thesis is focused on the first task. Indeed, the main purpose of this work is to determine a good method for clustering electrical load and generation data, in order to reduce the computational time of many optimization algorithm which are often employed in electrical engineering field. Frequently, these

algorithms receive as input $365y$ electrical load/generation curves (one curve for every day of the year, for a selected number of years y) and exploit them to optimize the size of the machines/components to be installed or to make predictions about future development. Typically, the computational complexity for these algorithms is in the order of $O(n^2)$. For the sake of clearness, it is recalled that time complexity is a function of the number of data points [16] and is commonly estimated by counting the number of elementary operations performed by the algorithm, supposing that each elementary operation takes a fixed amount of time to perform. Thus, the amount of time employed to complete the algorithm and the number of elementary operations performed by the algorithm itself are related by a constant factor. Since this function is generally difficult to compute exactly, and the running time for small inputs is usually not consequential, the behaviour of the complexity becomes relevant when the input size increases. Thus, only the asymptotic behaviour of the complexity is of interest, so the time complexity is typically expressed using big O notation. The following figure shows a graph of the main functions commonly used in algorithms analysis to express the computational complexity.

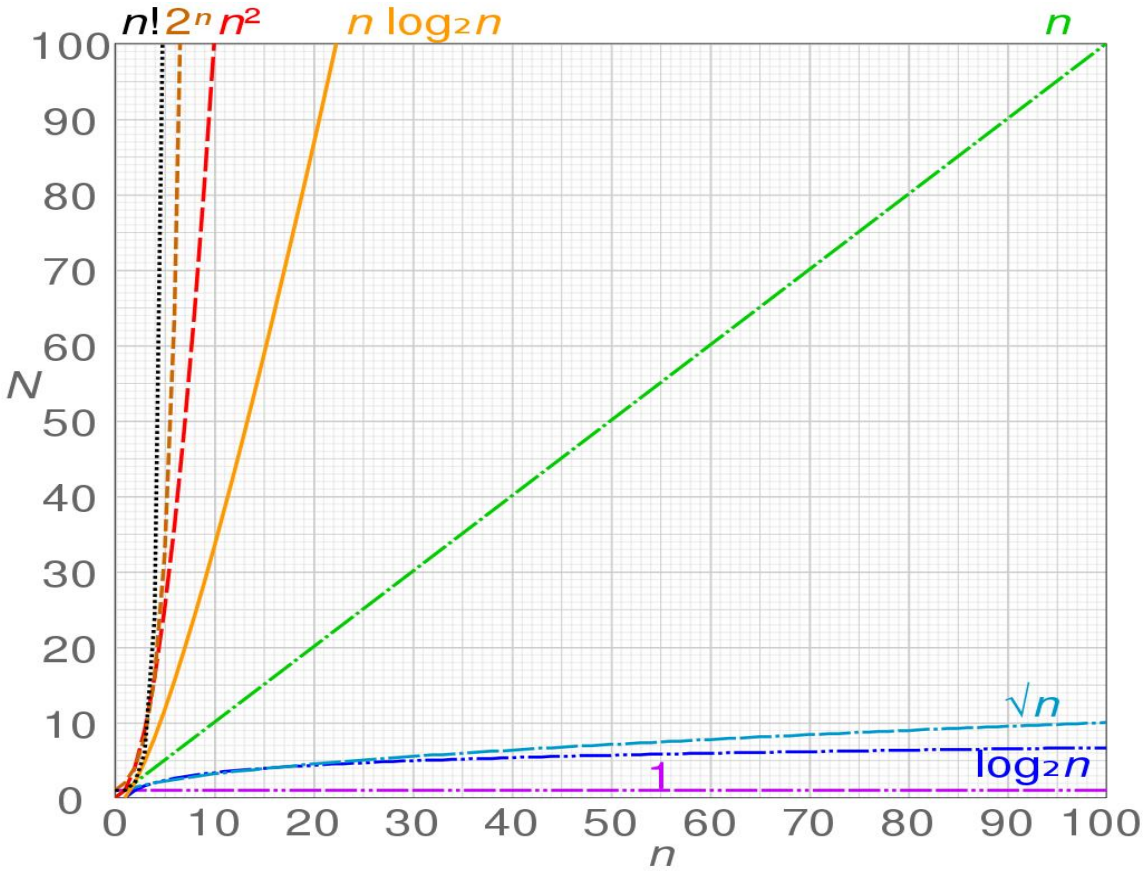


Figure 0.1: Commonly used time complexity functions in algorithm analysis

In figure 0.1 the input data size n is on the x-axis, while the number of operations N of the algorithm is on the y-axis. As it is possible to observe, different behaviors occur for different algorithms. Since the main purpose of this thesis is to reduce the complexity of an optimization algorithm, as already discussed, the red function will be selected as reference to determine the reduction rate of the computational time after the clustering process.

The fact that typically optimization algorithms have a computational complexity in the order of $O(n^2)$ implies that the computational time has a quadratic asymptotic behaviour, that becomes more and more precise for larger and larger datasets. In this perspective, halving the number of datapoints will make the computational time almost one quarter of the original one, while using one tenth of datapoints will make it almost one hundredth of the original one. This makes clear, again, that time series clustering is an essential pre-processing activity that greatly improves the performance of whichever algorithm, leading to coherent and consistent results in a fraction of the original computational time. In this thesis, different clustering techniques will be employed to reduce the computational time of an optimization algorithm which receives as input 365 load duration curves and/or 365 generation curves, one for every day of the year. By clustering the 365 curves in 36 curves (approximately one tenth), the most representative days of the year will be determined. The selection of the number of curves to be obtained is not random: it derives from the computation of some quality indexes of the clustering result, exploited to determine the optimal number of clusters as a compromise between the needs to reduce the computational time and to well represent the original data. The determined representative days are called *typical load/generation days* and are extremely important since they allow to obtain similar and consistent results as using the complete 365 days in the optimization algorithm, but also reducing the computational time as:

$$O\left(\left(\frac{n}{10}\right)^2\right) = O\left(\frac{n^2}{100}\right) = \frac{1}{100}O(n^2) \tag{0.1}$$

As equation 0.1 shows, the proposed approach can reduce the computational time by almost one hundredth, of course provided that the results are coherent with the use of the complete 365 curves. In the next chapter, a more theoretical and rigorous

explanation of time series, clustering techniques and their practical implementation will be provided.

As already discussed, clustering of time series data is mostly used to discover interesting patterns in the time series dataset. More precisely, four main applications can be defined:

1. Anomaly detection: methods to discover unusual and unexpected patterns which occur surprisingly in the dataset. As an example, it can be used in sensor databases obtained by sensor readings to discover unexpected or critical events.
2. Recognizing dynamic changes in the time series: methods to detect correlation between time series. As an example, it can be used in financial databases to find companies with similar stock price move.

3. Prediction and recommendation: a hybrid technique combining clustering and function approximation per cluster that can help users to predict and recommend the future data trend. As an example, it can be used in scientific databases by addressing problems such as finding the patterns of solar magnetic wind to predict its daily development.

4. Pattern discovery: methods to discover interesting patterns in a database. As an example, it can be used in marketing databases to find different daily patterns of sale for a specific product in a store.

The following table reports some applications of time series clustering, in different domains [1], to address real world problems.

Reference	Dataset	Objective
Košmelj & Batagelj, (1990)	Country's energy consumption	Energy Consumption pattern of 23 European Countries (commercial consumption)
VanWijk & Van Selow(1999)	Daily power consumption	Discovering consumer power consumption patterns
Ramoni, Sebastiani, & Cohen(2000)	Robot sensor data	Prototypal representations of robot's experiences

Fu et al. (2001)	Stock market data	Discovery patterns from stock time-series
Golay et al., (1998); Wismüller et al., (2002)	Functional MRI(fMRI)	Detecting brain activity
Tran & Wagner (2002)	Speechtime-series	Speaker verification
M. Kumar & Patel (2002)	Sales data from several departments of a major retail chain	Finding seasonality patterns (Retail pattern)
Steinbach, Tan, Kumar, Klooster, & Potter, (2003)	Climate time-series	Discovery of climate indices
Möller-Levet, Klawonn, Cho, & Wolkenhauer, (2003)	Gene expression	Identification of functionally related genes
Bagnall, Janacek, De la Iglesia, & Zhang, (2003)	Time-series representing the per capita personal income	Personal income pattern
Shumway,(2003)	Earthquake	Analyzing potential violations of a Comprehensive Test Ban Treaty (CTBT)
Guan & Jiang, (2007)	Financial data	Creating an efficient portfolio (a group of stocks owned by a particular person or company)
C. Guo, Jia, & Zhang, (2008)	Stock exchange data	Discovery patterns from stock time-series
Rebbapragada, Protopapas, Brodley, & Alcock, (2009)	Astronomical data (star light curves)	Pre-processing for outlier detection
Gullo et al., (2011)	Mass spectra data	Exploring, identifying, and discriminating pathological cases from MS clinical samples
Kurbalija et al., (2012)	Human behavior data	Analysis of human behavior in psychological domain

Table 0.1: Some applications of time series clustering in different domains

As table 0.1 shows, time series clustering is an essential task in many different research areas, that helps in discovering and analysing useful and interesting patterns that represent the data trend, leading to optimal results in a fraction of the necessary time. However, its usage requires an important amount of resources and is subject to many requirements. In the following pages, the most typical requirements for a good clustering model are defined.

- Scalability: the clustering algorithm must work fine for small and large datasets. This is very important in big data analysis, where the databases are composed by hundreds of millions of data. Thus, many clustering methods that are fitted for small databases must work well also for big databases.

- Capability to treat different data attributes: many clustering algorithms are designed to cluster numerical data. However, the possible applications can require clustering other data types (e.g., binary, categorical, a combination of different data attributes…).

- Definition of clusters according to different metrics: many clustering algorithms define clusters according to Euclidean distance. These methods tend to create spherical clusters with similar dimensions and density. However, a cluster can generally have a generic shape, different from the spherical one. Therefore, it is important to define clustering algorithms able to work with different distance metrics, defining cluster with an arbitrary shape.

- Minimum requests number to determine input parameters: many clustering algorithms require to the user to define a priori some parameters (for instance, the number of desired clusters). Since clustering results are quite sensitive to input parameters, and these parameters are very difficult to estimate with precision, it is important that a good clustering model requires a number of inputs as little as possible, not to bind users and make clustering quality difficult to control.

- Robustness: capability to treat noisy data. Typically, measurements dataset contains missing, unknown or noisy data. A good clustering algorithm must have a low sensibility to these data, avoiding to determine low quality clusters.

- Incremental clustering ability: capability to incorporate new data in already existent clusters, avoiding to re-perform the whole clustering process in case of addition of new data. This requirement is typically not so essential, but

anyway it would be important to develop algorithms able to update the clusters incrementally.

- Insensibility to input data order: a good clustering model must not generate dramatically different clusters if the order of the input data changes.

- Multidimensionality: capability to cluster dataset containing many different variables, with one or more time series for every variable.

- Capability to include constraints: in real world applications, many constraints can be imposed to the clustering process. A good clustering model should be able to include these constraints in the algorithm.

- Interpretability and usability: clustering results must be interpretable, understandable and usable

As it is possible to note, a good clustering model must comply with many requirements. In this thesis, two main clustering algorithms have been considered: *k-means clustering* and *Ward hierarchical clustering*. These algorithms have some advantages and disadvantages which will be discussed furthermore but anyway they generally comply with all the previously presented requirements [17]. Moreover, both clustering techniques exploit a dissimilarity matrix between data to perform the clustering process. More precisely, it is possible to divide clustering algorithms in two categories, according to the matrix exploited to perform:

- Clustering algorithms exploiting a raw data matrix

  A data matrix represents n objects, with p variables for each object. Indeed, data are represented by a $n \times p$ matrix, as shown below.

$$\begin{pmatrix} x_{11} & \dots & \dots & x_{1p} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ x_{n1} & \dots & \dots & x_{np} \end{pmatrix} \tag{0.2}$$

This structure is called *object-by-variable* since the matrix relates every object with its variables. Indeed, the first row contains the p observations of the first of the n objects, and so on for the successive rows. As an example, n can be a certain number of people and p could be a set of measurements for every people, like height, weight, ethnicity and so on.

- Clustering algorithms exploiting a dissimilarity matrix

  A dissimilarity matrix contains the degree of dissimilarity between each couple of objects. Indeed, it is represented by a $n \times n$ matrix, as shown in the next page.

$$\begin{pmatrix} 0 & d(1,2) & \dots & d(1,n) \\ d(2,1) & 0 & \dots & \dots \\ \dots & \dots & 0 & \dots \\ d(n,1) & d(n,2) & \dots & 0 \end{pmatrix} \tag{0.3}$$

  In the matrix 0.3, $d(i,j)$ denotes the dissimilarity between object i and object j. This parameter can be computed in many different ways, but typically a distance metric is exploited, as Euclidean distance or more sophisticated metrics. In this thesis, a particular but extremely important distance metric is considered: *dynamic time warping (DTW)*. As it will be clarified in the next chapter, the use of DTW allows to greatly improve the precision of the clustering model, since it exploits the Euclidean distance multiple times to find the minimum/maximum dissimilarity between every possible object couple, leading to a more precise estimation of the dissimilarity matrix. Eventually, it is important to recall that in matrix 0.3 $d(i,j) = d(j,i)$ and $d(i,i) = 0$. This structure is called *object-by-object* since rows and columns of the matrix represent the same entity (the objects).

Most clustering algorithms perform exploiting the dissimilarity matrix. Thus, if data are represented in the object-by-variable form, it is necessary to move from the raw data matrix to the dissimilarity matrix before running the algorithms. This topic will be developed in detail in the next chapters.

This introduction had the purpose to remark the importance of clustering in everyday life, showing its applications and its advantages. Moreover, the most

important requirements of a good clustering algorithm have been defined and an important distinction between algorithms exploiting a raw data matrix and algorithms exploiting a dissimilarity matrix has been analysed. In the first chapter, a more theoretical and formal definition of time series, distance metrics (with DTW) and time series clustering will be provided. In the second chapter, instead, the two already mentioned algorithms, that will be employed in this thesis, will be analysed in details and a mathematical proof will be provided, when possible, to show the effectiveness of the methods. Moreover, in the third chapter, the adopted methodology to exploit the clustering algorithms in an electrical engineering problem will be explained, as well as the implemented codes, and the experimental results will be discussed. Finally, the last chapter will focus on the chronological ordering of clustered data, an open problem which affects the clustering process in its possible application. Again, some methodology will be proposed and the results will be discussed.

Eventually, to conclude the introduction and to remark the importance of clustering, it is recalled that many real-life companies exploit data clustering to reach better performances and improve the user experience. In particular, the Google case can be discussed [18]. Indeed, at Google, clustering is used for generalization, data compression and privacy preservation in products such as YouTube videos, Play apps and Music tracks. More precisely:

- Generalization: when some objects in a cluster have missing feature data, it is possible to infer the missing data from other objects in the cluster. For instance, less popular videos can be clustered with more popular videos to improve video recommendations.

- Data compression: as discussed, data in a cluster can be replaced by their cluster ID. This replacement simplifies the feature data and saves storage. These benefits become significant when scaled to large datasets. Further, machine learning systems can use the cluster ID as input instead of the entire feature dataset., reducing the complexity of input data and making the machine learning model simpler and faster to train. For instance, feature data for a single YouTube video can include: viewer data on location, time and demographics, comment data with timestamp, text and user ID, video tags. Clustering YouTube videos makes possible to replace this set of features with a single cluster ID, thus compressing the data.

- Privacy preservation: it is possible to preserve privacy by clustering users and associating user data with a cluster ID instead of with specific users. For instance, if the video history for YouTube users has to be added to a ML model, instead of relying on the user ID, it is possible to cluster users and rely only on the cluster ID. Now the model cannot associate the video history with a specific user, but only with a cluster ID that represents a large group of users.

This example shows that even an extremely important real-life companies like Google exploits clustering in its everyday activity to improve the performance and guarantee the privacy of its users. Again, this underlines the effectiveness and advantages of the clustering process, which will be discussed and remarked also in the next chapters. In particular, chapter 1 will provide a more theoretical and formal definition of time series, distance metrics (with DTW) and time series clustering, while chapter 2 will focus on the two implemented algorithms, to make the reader having a clear view of the proposed methodology and a full comprehension of the obtained results, presented in chapter 3.

1. The scientific approach to Clustering

In this chapter, most of the formal and mathematical aspects of time series, dissimilarity measures and of the clustering process will be developed in detail. Even if it could result quite long, it is extremely important to provide a scientific approach to the subject, to give the reader all the necessary information to have a clear understanding of the implemented algorithms, which apply time series clustering to an electrical engineering problem. More precisely, three sections are considered below: time-series (section 1.1), dissimilarity measures (section 1.2) and time-series clustering (section 1.3). Each section is subdivided in subsections to better distinguish between the different concepts, metrics or algorithms that will be presented.

1.1 Time-series analysis

The term *time series* denotes a succession of values obtained by observation of a phenomenon, ordered according to the time variable. In formal terms:

$$Y(t) = \{y_1, y_2, \dots, y_t, \dots, y_N\} \tag{1.1}$$

Equation (1.1) shows that the time series $Y(t)$ is composed by a set of values, ordered according to a time variable going from 1 to N. This last term is called duration of the series and corresponds to the number of observations. It is important to remark that each observation y_i can be composed by multiple variables. For instance, as already discussed, $Y(t)$ could be a time series regarding a company and y_i could be a set of observations (values) related to multiple variables such as number of employees, incomes, outcomes, presence or absence of certain condition (binary variable), most diffused ethnicity between employees (categorical variable) and so on at the time $t = i$. This type of series, in which each observation y_i is a vector containing multiple variables, are called *multivariate time* series and, for the sake of simplicity, will not be considered in this thesis. In this perspective, a single-variable time series, which has

just one variable with N observations, can be represented as a two-dimensional graph on a xy plane, where the time instants are on the x-axis and the values of the time series are on the y-axis; this type of graph is called *run chart*. For instance, the following figure shows the run chart of a time series representing the total Italian load of 10/02/2022 from 12:00 a.m. to 2:00 p.m., whit a period of observation of 15 minutes.

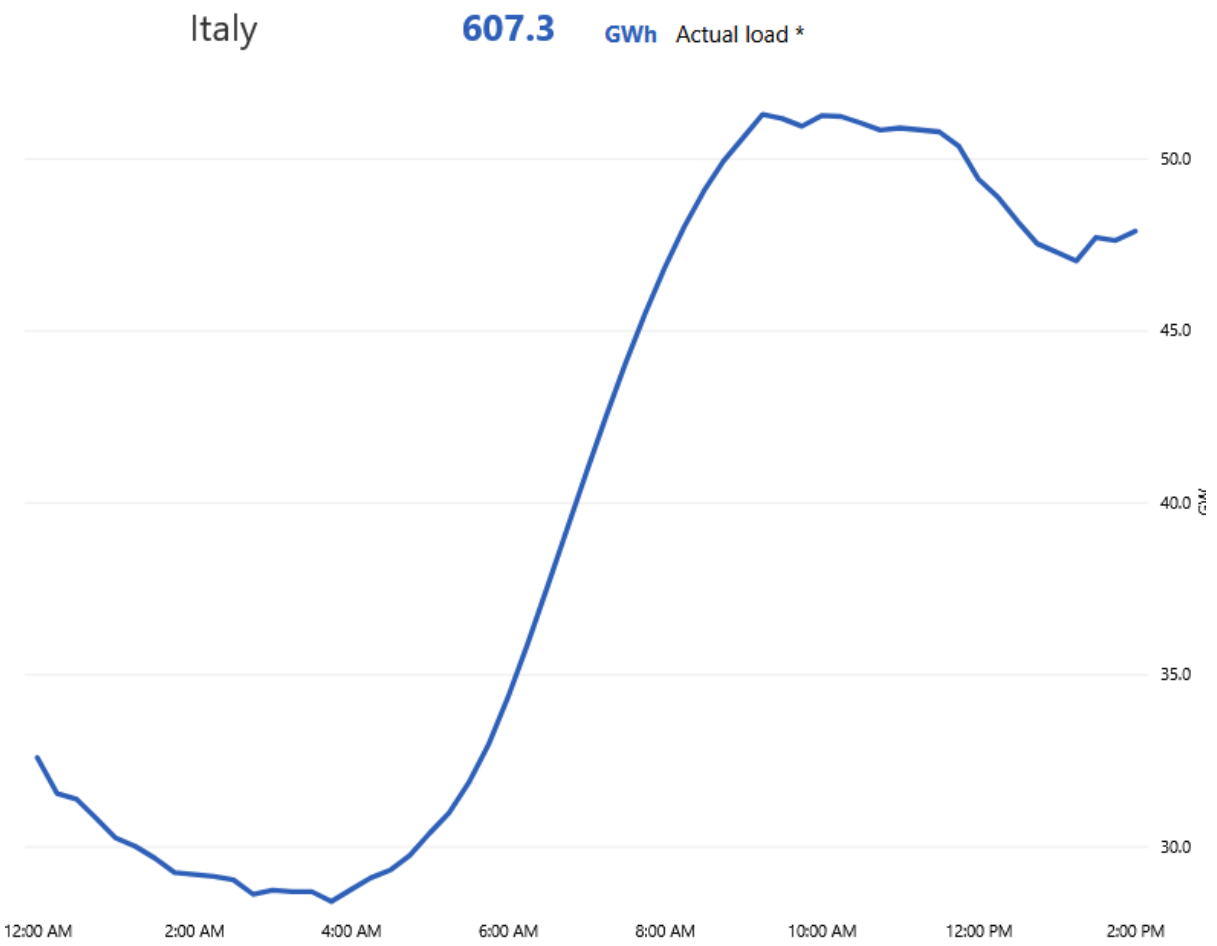


Figure 1.1: The total Italian load curve is an example of run chart of a single variable time series

The period of observation can be defined according to a succession of time instants (for instance: the employees of a company at the end of the month) or to time intervals, equally spaced or not (for instance: the electrical energy demand of a certain nation, monitored every hour). In the first case, the series is denoted as *state* or *positional time series* while in the second case it is denoted as *flux time series*.

Two different approaches are possible to study a time series: the "classical" approach or *deterministic approach* and the "modern" approach or *statistical approach.* In the first case, it is assumed that the process represented by the series has a deterministic nature, which allows to decompose the series in four components that can be estimated. The second approach, instead, assumes that the series has been generated by a stochastic process, so that each observation is the realization of a random variable Y_t. In the following pages, the two approaches will be analyzed separately, focusing on the deterministic one for the sake of simplicity.

1.1.1 Deterministic approach to time series analysis

As already discussed, the fundamental hypothesis for this approach is that the series is generated by a process which has a deterministic nature. In this perspective, it is possible to state that generally time series show oscillations around a long-period behavior. This leads to an important decomposition which can be applied to any time-series. Indeed, a time series can be decomposed in four components, called *virtual components of the series*:

- T_t: trend component of the series at time instant t. This component reflects the long-term progression of the series, which highlights a structural evolution of the observed phenomenon due to causes acting systematically on it. Typically, the trend component is non-zero when there is a persistent increasing or decreasing of the data, linearly or not. To remark that this component reflects the long-term behaviour of the series, it is also denoted as secular variation component.

- C_t: cyclical component of the series at time instant t. This component reflects repeated but not periodic fluctuations, which duration depends on the time series nature. In economy, where the time series could represent stock prices, typically these fluctuations are caused by the occurrence of favourable or negative conditions, such as the expansion or contraction of the economic context where the studied phenomenon develops.

- S_t: seasonal component of the time series at time instant t. As the name suggests, this component reflects seasonal variation of the time series. Of course, a seasonal pattern exists only when the time series is affected by seasonal factors. Generally, these factors are climatic conditions or social events. For instance, in the electrical engineering field, the electrical energy

demand of a certain nation increases during summer due to a massive usage of air conditioning systems.

- I_t: irregular component of the time series at time instant t. This component reflects random or irregular behaviour of the time series, caused by accidental events or measurement noise.

In formal terms, it is theoretically possible to define the following relationship for a considered time series:

$$Y_t = f(T_t, C_t, S_t, I_t) \tag{1.2}$$

where $t = 1, \dots, N$.

This decomposition highlights that, in the classic approach, the series is supposed to be composed by a systematic or deterministic pattern, with random oscillations or disturbances superposed. Moreover, two main models are typically employed:

- Additive model: $Y_t = T_t + C_t + S_t + I_t$ (1.3)

- Multiplicative model: $Y_t = T_t \times C_t \times S_t \times I_t$ (1.4)

Typically, the additive model is appropriate when the amplitude of the seasonal oscillation does not vary with the variation of the level of the series, that in this case is called *additive time series*. Moreover, all the four components are expressed in the same unit of Y_t. Instead, the multiplicative model is appropriate when the seasonal oscillation increases (reduces) proportionally to the increase (reduction) of the level of the series, and only T_t and C_t have the same unit of Y_t, while S_t and I_t are expressed as indexes with respect to $T_t \times C_t$ For the sake of clearness, figures 1.2, 1.3, 1.4 in the next page [19] show an additive time series (1.2), in which the oscillations are the same even if the level of the series is increasing, and a non-additive time series, in which the oscillations increase or decrease as the level of the series increases (1.3) or decreases (1.4).

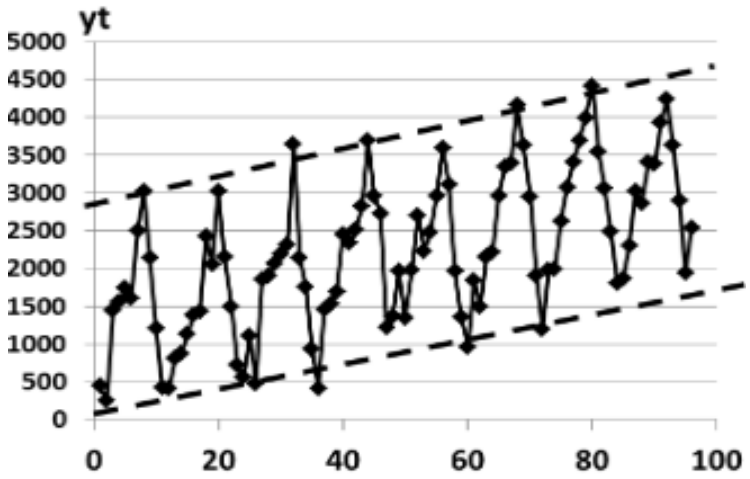


Figure 1.2: additive time series

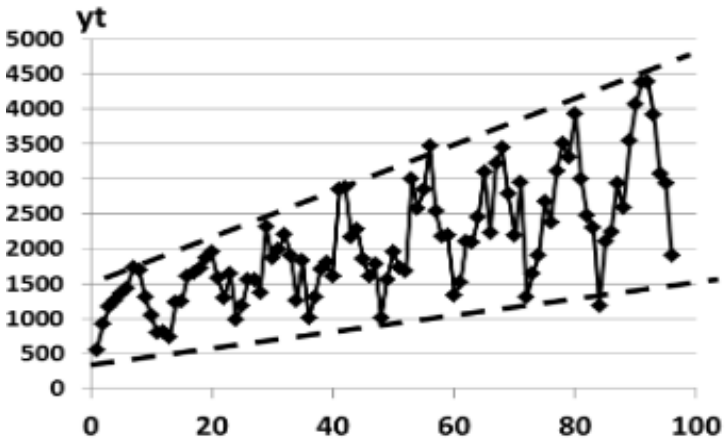


Figure 1.3: non-additive increasing time series

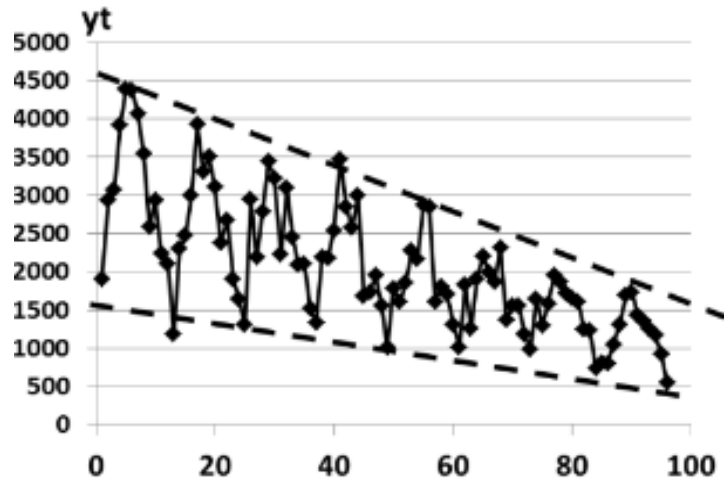


Figure 1.4: non-additive decreasing time series

Before analyzing the techniques to determine the four virtual components of the series, it is important to remark that since the cyclical component is typically very difficult to estimate, the analysis will consider a single component T'_t, given by the sum (additive model) or the multiplication (multiplicative model) of T_t and C_t. More precisely:

- Additive model: $T'_t = T_t + C_t$

(1.5)

- Moltiplicative model: $T'_t = T_t \times C_t$

(1.6)

This new component is called *trend-cycle* component, since it includes both contributions. Eventually, it is important to remark that in the additive model the irregular component can assume positive and negative values and has zero as neutral value, while in the multiplicative model the irregular component can assume only positive values and has one as neutral value. Moreover, it is possible to move from the multiplicative to the additive model by applying the natural logarithm to the decomposed time series as follows:

$$\ln(Y_t) = \ln\left(T'_t \times S_t \times I_t\right) \tag{1.7}$$

$$\ln(Y_t) = \ln(T'_t) + \ln(S_t) + \ln\left(I_t\right) \tag{1.8}$$

Equation (1.8) shows that instead of applying a multiplicative model on the original data of the series, it is possible to apply an additive model on their natural logarithm. In the following pages, figures 1.5, 1.6 and 1,7 [19] show the run chart of a time series representing the sold bottles of a famous drink (1.5), the run chart of the seasonal component (1.6) and the run chart of the original time series without the seasonal component (seasonally adjusted data). The estimation of the components has been obtained with an additive model, thus the seasonally adjusted data can be obtained by subtracting the seasonal component from the original time series.

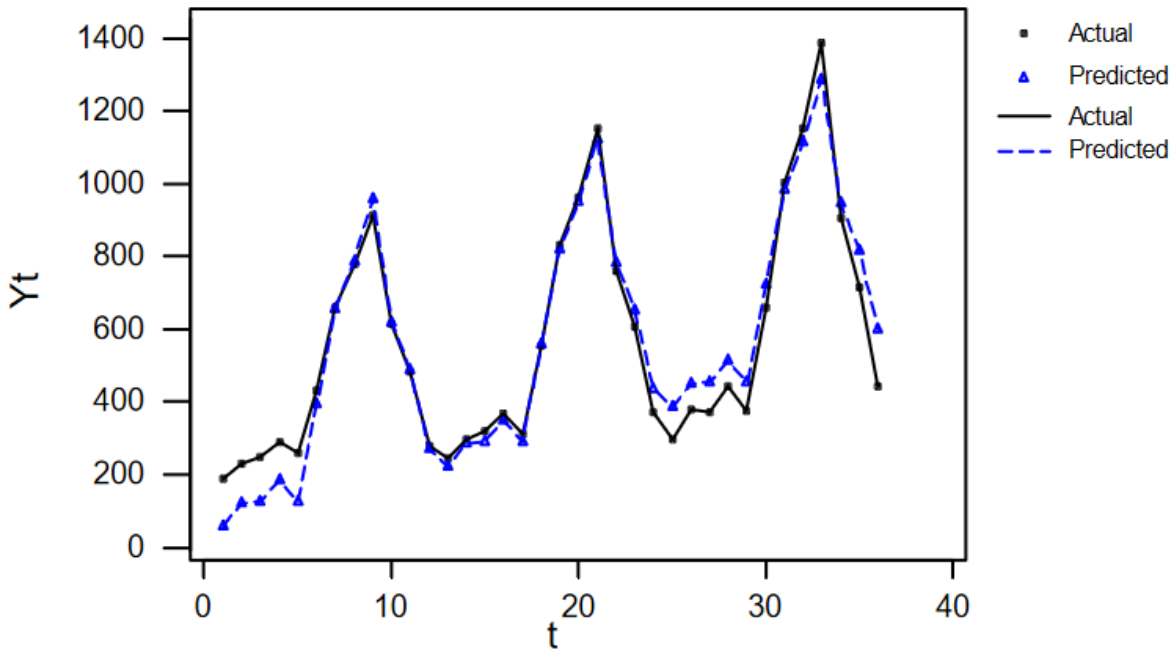


Figure 1.5: original time series (actual) and prediction (predicted)

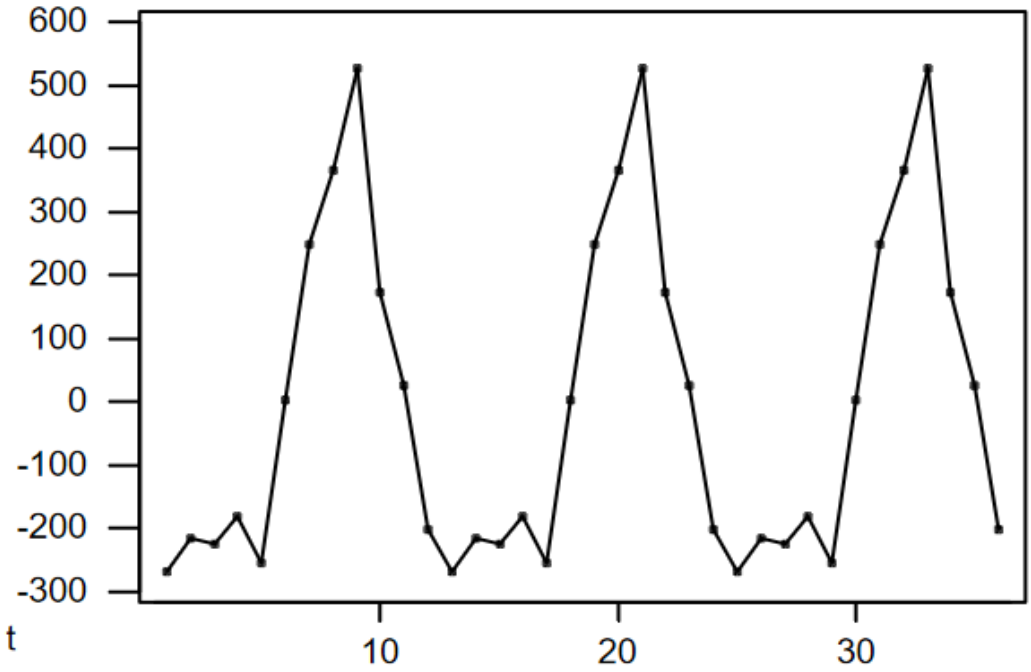


Figure 1.6: seasonal component of the time series

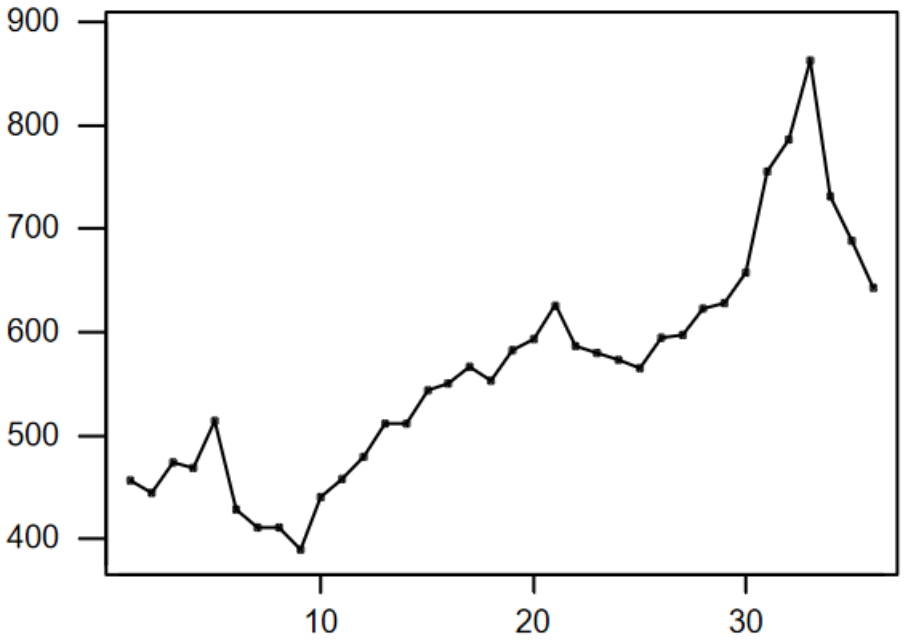


Figure 1.7: seasonally adjusted time series (no seasonal component)

For the sake of clearness, it is recalled that seasonally adjusted data can be derived after the estimation of the seasonal component as follows:

- Additive model: $Y'_t = Y_t - S_t$

(1.9)

- Moltiplicative model: $Y'_t = Y_t / S_t$

(1.10)

The seasonally adjusted data should have a more or less flat behavior, without the oscillations that are typical of the series seasonality. Therefore, figure 1.7 shows that the additive model is not appropriate for analyzing the original time series, reported in figure 1.5. Indeed, there is an over-adjustment in the first periods (the peaks are inverted with respect to the original series) while there is an under-adjustment in the last periods (there is still a very high peak). The reason behind this is that the additive model, as already discussed, assumes that the seasonal component has an amplitude more or less constant along the observation period and this is probably not true for the case under examination. Therefore, the estimation of the seasonality, reported in figure 1.6, cannot be considered valid and a multiplicative model must be adopted.

It is now possible to discuss the techniques used to estimate the virtual components of a time series. These components can be obtained using empirical or analytic methods. The two cases will be treated separately for a better readability.

Empirical methods

These estimation techniques exploit the so-called *moving average (MA)*. Indeed, a new time series is derived from the original one by substituting every point with its moving average. Therefore, this parameter is computed in the original series for every time instant and the result is saved in the correspondent time instant of a new series. For the sake of clearness, it is recalled that the moving average is defined as the mean of k contiguous terms of the considered time series. More precisely:

- If k is odd:

$$MA_k(Y_t) = \frac{1}{k} \sum_{i=t-k+2}^{t+1} Y_i \tag{1.11}$$

- If k is even: it is necessary to compute the mean between the two contiguous moving averages of the considered time instant, calculated with the same formula as for odd k. For instance, if $k = 4$:

$$MA_4(Y_{t-1}) = \frac{Y_t + (Y_{t-1} + Y_{t+1}) + Y_{t-2}}{4} \tag{1.12}$$

$$MA_4(Y_{t+1}) = \frac{Y_t + (Y_{t-1} + Y_{t+1}) + Y_{t+2}}{4} \tag{1.13}$$

$$MA_4(Y_t) = \frac{MA_4(Y_{t-1}) + MA_4(Y_{t+1})}{2} \tag{1.14}$$

Of course, the first and last data of the original time series correspond to their moving average, since a previous or successive term cannot be found for the first and last element of the series, respectively.

By substituting every point of a time series with its moving average of length k, a new series is obtained in which the value correspondent to a considered time instant is smoothed with respect to the correspondent value in the original series. Due to this, this technique is also denoted as *local adaptation method*. The length k of the moving average has a strong impact on the result of the process: the higher is k, the smoother will be the new obtained series. As an example, let's consider the following dataset, that reports the monthly selling of shampoo Y_t, in liters and for three years, of a famous company [20]. Moreover, the moving averages of length 3, 5 and 7 have been computed with the expressions presented in the previous page. Of course, if $k = 5$, also the second term of the original series is equal to its moving average, since it is not possible to define the term $t - 2$. In the same fashion, if $k = 7$, the term $t - 3$ cannot be defined therefore also the third term of the series coincides with its moving average.

month	Y_t	$MA_3(Y_t)$	$MA_5(Y_t)$	$MA_7(Y_t)$
1	266.0	-	-	-
2	145.9	198.3	-	-
3	183.1	149.4	178.9	-
4	119.3	160.9	159.4	185.0
5	180.3	156.0	176.6	179.1
6	168.5	193.5	184.9	185.8
7	231.8	208.3	199.6	177.2
8	224.5	216.4	188.1	208.2
9	192.8	180.1	221.7	209.0
10	122.9	217.4	212.5	212.7
11	336.5	215.1	206.5	200.9
12	185.9	238.9	197.8	198.9
1	194.3	176.6	215.3	210.4
2	149.5	184.6	202.6	220.1
3	210.1	211.0	203.7	213.1
4	273.3	224.9	222.3	218.8
5	191.4	250.6	237.6	234.4
6	287.0	234.8	256.3	254.5
7	226.0	272.2	259.6	284.7
8	303.6	273.2	305.6	283.4
9	289.9	338.4	301.1	305.0

10	421.6	325.3	324.4	312.5
11	264.5	342.8	331.6	343.1
12	342.3	315.5	361.7.	344.9
1	339.7	374.1	340.6	366.2
2	440.4	365.3	375.5	363.3
3	315.9	398.5	387.3	388.0
4	439.3	385.5	406.9	421.4
5	401.3	426.0	433.9	431.1
6	437.4	471.4	452.2	465.5
7	575.5	473.5	500.8	488.3
8	407.6	555.0	515.6	508.6
9	682.0	521.6	544.3	543.7
10	475.3	579.5	558.6	-
11	581.3	567.8	-	-
12	646.9	-	-	-

Table 1.1: Shampoo selling of a company and moving averages computation

The following figure shows the run chart of the original time series and of the three moving averages computed in the previous table.

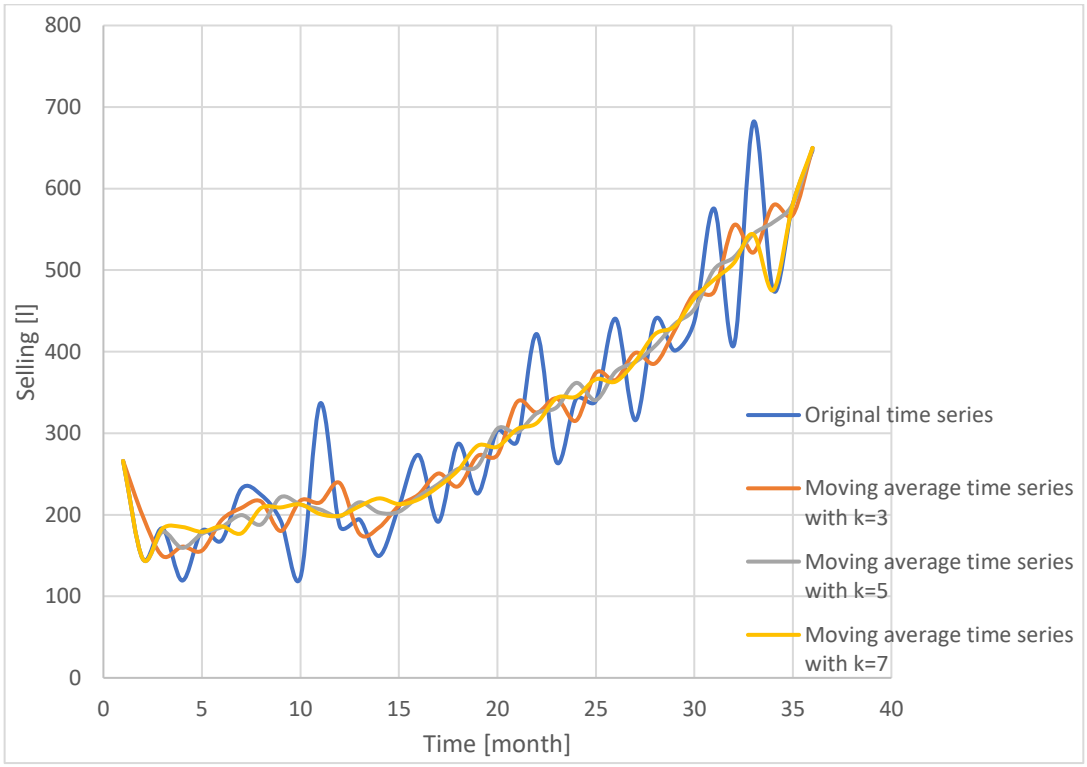


Figure 1.8: Selling of shampoo [liters] and moving averages in three years

As it is possible to observe from figure 1.8, the moving average techniques eliminates a certain amount of oscillations in the time series. Moreover, the higher is the selected length, the smoother is the obtained series (more oscillations are eliminated). It is possible to prove that a moving average of length k does not smooth $\frac{k-1}{2}$ points at the beginning and at the end of the series if k is odd, while does not smooth $\frac{k}{2}$ points at the beginning and at the end of the series if k is even. Typically, moving averages with an even length are used to eliminate the seasonal component of the series, selecting for instance $k = 12$ for monthly data in one year, $k = 4$ for trimestral data in one year, $k = 2$ for semestral data in one year and so on Indeed, the moving average has the property to eliminate the oscillations that have a period equal to the length of the moving average itself [19]. Therefore, by selecting a length equal to the period of the seasonality, it is possible to remove the seasonal component from the original time series, leading to the previously discussed decomposition.

Once that the concept of moving average has been discussed, it is now possible to explain the procedure to decompose a time series in its virtual component. This procedure is the same for additive and multiplicative model, with little differences in the formulas used. Anyway, for the sake of simplicity, it will be presented only for the additive model. However, it is recalled that it is possible to move from a multiplicative model to an additive model by applying the natural logarithm, as equation (1.8) remarks. The main steps to be followed are:

1. Computation of approximate trend-cycle component
   This component is computed for every point of the original series, as a moving average of length 12. In the following, it will be denoted as $MA_{12}(Y_t)$, with $t = 7, 6, \dots, n - 6$ due to the non-smoothing effect at the beginning and end of the series.

2. Computation of the seasonal+irregular component
   This component is given by the sum of the seasonal and irregular components of the original time series. It will be denoted as SI_t and is computed as:

$$SI_t = Y_t - MA_{12}(Y_t) \tag{1.15}$$

3. Estimation of the seasonal component S_t
   This component is obtained by subtracting from SI_t the irregular component, which can be assumed a priori. If no information on the irregular component is available, it is possible to assume that the seasonal component has a constant value, different for every month. Therefore, for a considered month m, this component is computed as the arithmetic mean of the terms SI_t, with $t = m, m + 12, m + 24, \ldots$. For instance, the seasonal component for January is given by the arithmetic mean of all the values of SI_t referring to January. The constant value of the seasonal component in every month is called *seasonality coefficient* for that month. When it is negative, the seasonality creates a contraction of the series with respect to the behaviour it would have without seasonal effects. On the opposite, when it is positive the seasonality amplifies the observed phenomenon.

4. Derivation of seasonally adjusted series
   The seasonally adjusted series is computed as:

$$Y'_t = Y_t - S_t \tag{1.16}$$

   It is recalled that this formula holds just for the additive model.

5. Estimation of the trend-cycle component
   This component is computed with a moving average of length 3 on the seasonally adjusted series.

6. Estimation of the systematic behaviour of the series
   By summing up the estimated trend-cycle component and seasonal component, a new series $\hat{Y}_t$ is obtained. This series contains only the systematic pattern of the original time series.

7. Estimation of the irregular component
   This component is derived by subtracting to the original series its systematic behaviour. In particular:

$$I_t = Y_t - \hat{Y}_t \tag{1.17}$$

Typically, an analysis of this last component is performed to evaluate the goodness of the decomposition. Indeed, if it is valid, the irregular component should not present systematic oscillations and its run chart should be oscillating around the neutral value (0 for the additive model, 1 for the multiplicative model).

Deterministic methods

These estimation techniques suppose that it is possible to represent the seasonally adjusted time series as a function of time. In formal terms:

$$Y_t = f(t) + i_t \tag{1.17}$$

where i_t is a function of time representing the irregular component of the original time series. The definition of $f(t)$ could use any analytic function and is generally derived from the behaviour of the series, observed through its run chart. The most common functions employed are [19]:

- Constant function: $f(t) = K$ (1.18)
- Linear function: $f(t) = At + B$ (1.19)
- Quadratic function: $f(t) = At^2 + Bt + C$ (1.20)
- Exponential function: $f(t) = A * B^t$ (1.21)

Once that a function for the seasonally adjusted series has been selected, the coefficients are estimated with already known techniques. For instance, in case of linear function, the least square method of the linear regression can be employed.

After having estimated the trend-cycle component (seasonally adjusted data), by adding (additive model) or multiplying (multiplicative model) the seasonal component, estimated as in the previous paragraph, the systematic behaviour of the series is obtained. Then, the irregular component is computed with the same expression already presented. Typically, the analytic method is used when there is the need to make predictions on the future development of the series.

Once that the deterministic approach to time series analysis has been examined, it is now possible to move to the modern probabilistic approach. For the sake of simplicity, this latter will be explained briefly and not in details.

1.1.2 Statistical approach to time series analysis

As already discussed, the fundamental hypothesis for this approach is that the values of a time series correspond to the realizations of a random variable Y. More formally:

A time series model for the observed data $\{y_t\}$ *is the specification of the joint distribution (or evenly of the mean and the covariance) of a sequence of random variables* $\{Y_t\}$ *of which* $\{y_t\}$ *is postulated to be a realization.* [21]

A complete probabilistic time series model should specify all the joint distributions between its random variables $\{Y_1, Y_2, Y_3, \dots\}$. However, this specification is rarely used in time series analysis (unless the data are obtained by a well-known mechanism or phenomenon), since it would contain too many parameters to be estimated from the data. Instead, generally only the first and second order moment of the joint distributions are specified. For the sake of clearness, the definition of these two parameters is recalled:

- First order moment: the first order moment of a distribution Y_t is defined as the expectation $E[Y_t]$. The computation of this parameter changes between classical and Bayesian statistics. More precisely, in classical statistic it corresponds to the mean value of the considered distribution, while in Bayesian statistics it corresponds to the mean value of the posterior probability density function, computed through Bayes theorem.

- Second order moment: the second order moment of a distribution Y_t is defined as the expected product $E[Y_t \times Y_{t+h}]$ with $h = 0, 1, 2 \dots$

Generally, the statistical analysis of a time series focuses on properties of the sequence $\{Y_t\}$ that depend on the two previously defined parameters. These properties are called *second order properties*. In the particular case in which all the joint distributions are normal distributions, the second order properties of $\{Y_t\}$ completely determine the joint distributions and therefore provide a complete probabilistic characterization of the sequence [21]. In the following, two examples of simple time series statistical interpretation are discussed.

A. Independent and identically distributed (iid) noise
The iid noise is maybe the simplest model of a time series. In particular, it does not contain any trend or seasonal component and the observations are iid random variables with zero mean. Moreover, two main properties can be defined:

1. $P[Y_1 \leq y_1, \dots, Y_N \leq y_N] = P[Y_1 \leq y_1] \times \dots \times P[Y_N \leq y_N] = F(y_1) \times \dots \times F(y_N)$

(1.22)

indicating with F the cumulative distribution function of each of identically distributed random variables Y_t.

2. $P[Y_{N+h} \leq y | Y_1 = y_1, \dots, Y_N = Y_N] = P[Y_{N+h} \leq y]$, for $h \geq 1$

(1.23)

Equation (1.23) defines the probability that a realization of a future random variable Y_{N+h} of the sequence $\{Y_t\}$ is lower than a certain value y, conditioned to the fact that a realization of all the previous random variables $Y_{1:N}$ has already occurred. This probability is equal to the simple probability that Y_{N+h} is lower than y, thus the knowledge of $\{Y_1, \dots, Y_N\}$ is of no value for predicting the behaviour of Y_{N+h}.

B. Random walk
A random walk time series is obtained by summing up iid random variables. More precisely:

$$Y_t = Y_1 + Y_2 + \cdots + Y_N$$

(1.24)

Depending on the distribution adopted for the random variables, the random walk can be denoted as *Gaussian random walk* (normal distributio1n for the variables) or *simple symmetric random walk* (binary distribution for the variables). For instance, this latter case could represent the location of a pedestrian who starts at position zero at time zero and at each integer time tosses a fair coin, stepping one unit forward each time a head occurs, one unit backward each time a tail occurs. A realization of length 200 of a simple symmetric random walk is shown in figure 1.9 [21]. Note that the result of the toss at time t can be derived as $Y_t - Y_{t-1}$.

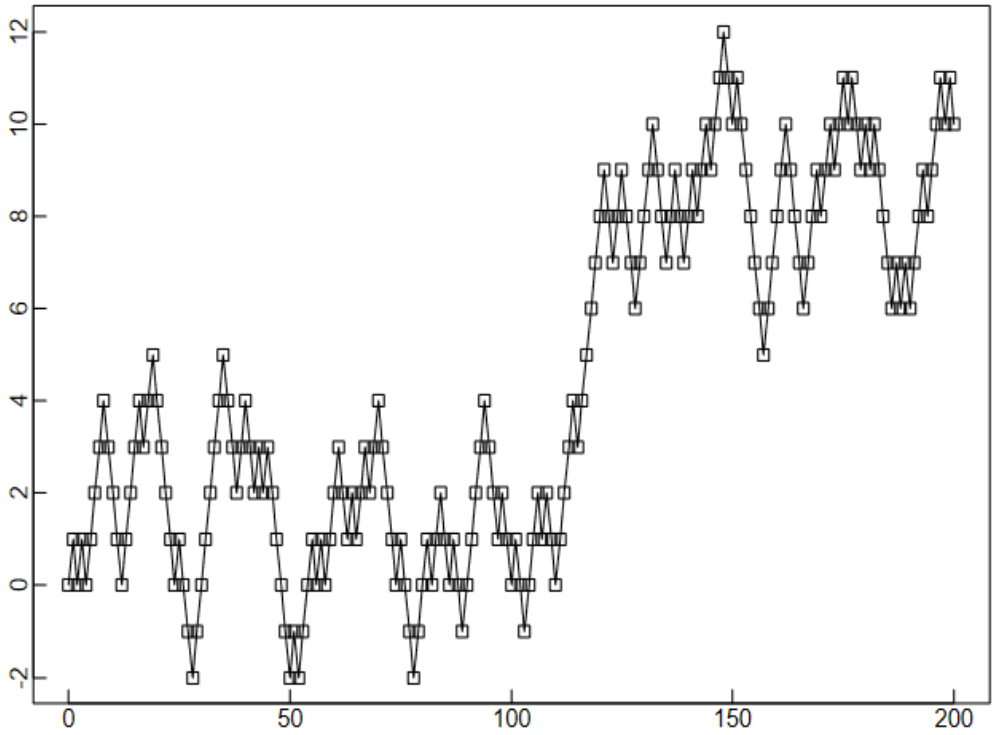


Figure 1.9: Realization of simple random walk of length 200

To conclude this brief discussion about the probabilistic approach to time series, it is important to remark that even in this case the decomposition already presented is valid. Of course, the techniques to estimate the virtual components of the series are different with respect to the deterministic approach. In particular, one method is based again on the moving average, with the important difference that in this case this parameter is a random variable itself. It is possible to prove that, if no seasonal component is present in the series, for a considered length k, the moving average

$MA_k(Y_t)$ represents the best estimate for the trend-cycle component, for $k+1 \leq t \leq n-k$. Then, the irregular component is obtained by subtraction or division of the trend-cycle component from the original time series. Instead, if also a seasonal component of period T exists, the trend component is estimated by selecting $k = T/2$ if T is even or $k = \frac{T-1}{2}$ if T is odd and proceeding with the previously explained method. Then, the seasonal component is estimated by computing the average value of the deviations of the original time series data from the correspondent data in the trend component, and subtracting from this value the sum of all the deviations, divided by the seasonality period. After having determined the seasonal component, the seasonally adjusted data component is computed by subtraction or division of the seasonal component from the original time series. This allows to obtain a time series with no seasonal component, that is exactly the case analyzed before. Therefore, the trend-cycle component can be re-estimated, as well as the irregular component, with the same method already explained.

There is another technique to estimate the virtual components of a time series. This method is based on the definition of new operators and is called *trend elimination by differentiation* [21]. More precisely, if no seasonal component is present, the *lag-1 operator* can be defined as follows:

$$\nabla Y_t = Y_t - Y_{t-1} = (1-B)Y_t \tag{1.25}$$

B is called *backward shift operator* and is defined such that:

$$BY_t = Y_{t-1} \tag{1.26}$$

Powers of the lag-1 operator are introduced in a standard way. For instance:

$$\nabla^2 Y_t = \nabla(\nabla Y_t) = (1-B)(1-B)Y_t = (1-2B+B^2)Y_t = Y_t - 2Y_{t-1} + Y_{t-2} \tag{1.27}$$

The application of this operator supposes that an analytic function representing the trend cycle component has been selected. For instance, if $T_t = At + B$ then $\nabla T_t = B$ while if $T_t = At^2 + Bt + C$ then $\nabla^2 T_t = 2C$. In the same fashion, it can be shown that

every polynomial of degree k can be reduced to a constant by applying the operator ∇^k. This implies that the trend-cycle component is eliminated since, by differentiating, the original time series reduces to the irregular component plus a constant. This suggests the possibility to repeatedly apply the lag-1 operator to the data sequence $\{y_t\}$ until a more or less stationary sequence is found. This sequence corresponds to the irregular component plus a constant, since it presents no apparent trend. Then, the trend-cycle component can be estimated by subtracting or dividing the irregular component from the original sequence. Instead, if also a seasonal component is present in the original time series, the *lag-d* operator is defined as follows:

$$\nabla_d Y_t = Y_t - Y_{t-d} = (1 - B^d)Y_t \tag{1.28}$$

By applying the lag-d operator to the additive model of a time series and selecting d equal to the period of the seasonality, the following result is obtained:

$$\nabla_d(Y_t) = Y_t - Y_{t-d} = \nabla_d(T_t + S_t + I_t) = T_t - T_{t-d} + I_t - I_{t-d} \tag{1.29}$$

Equation (1.29) shows that the selection of d equal to the period of the seasonality eliminates the seasonal component and decomposes the difference $Y_t - Y_{t-d}$ into a trend-cycle component $T_t - T_{t-d}$ and an irregular component $I_t - I_{t-d}$. The first of these two components can be eliminated using the already presented method (application of lag-1 operator) and consequently the irregular component can be obtained. Then, the deterministic behavior of the series is obtained by subtracting or dividing this component from the original time series.

To conclude the analysis of time series it is important to mention the so-called *representation methods*. The representation of a time series Y_t of length N is a model $\overline{Y_t}$ with length $N' < N$ (time dimensionality reduction) or with a reduced number of variables in case of a multivariate time series (dimensionality reduction) such that $\overline{Y_t}$ approximates Y_t [22]. This process is extremely important and is one of the main challenging issues for time series clustering. Indeed, especially in data mining and big-data analysis, time series data are larger than memory size [23]. This increases the required processing power and the time for clustering process increases

exponentially. Thus, it is crucial for time series data to be representative of the observed phenomenon without slowing down algorithm execution time. As for clustering algorithms, also for data representation methods some requirements can be listed [24]:

- Reduce time or variable dimensionality
- Maintain the local and global characteristics of the original time series
- Acceptable computational cost
- Reasonable level of reconstruction from the reduced representation
- Insensitivity to noise or implicit noise handling

Data representations methods can be classified in four main categories:

1. Data adaptive methods: methods in this category try to assign a constant value to an interval of contiguous data of the series. Therefore, the time series is represented by horizontal segments in such a way that the original shape is maintained. Different variations of this technique exist: using equal length segment (*piecewise linear approximation)*, using unequal length segment (*adaptive piecewise linear approximation*), or considering also inclined lines in the representation. The name "data-adaptive" comes from the fact that these methods have changing parameters according to the considered time series data.

2. Non-data adapting methods: use fixed parameters for representing time series data. For instance, applying the discrete Fourier transform to the time series, it is possible to obtain its spectral distribution and energetic coefficient of each harmonic. Typically, most of the information of the time series is contained in the first 10-13 coefficients. To show this, it is possible to check the energetic content of each harmonic and identify the harmonic coefficient for which this parameter decreases significantly. Then, all the harmonics with a lower energetic content are eliminated, and the representative series is retrieved through the inverse Fourier transform.

3. Model based methods: assume that the set of observations of the time series can be described by an underlying model. Once selecting a simpler representative model with respect to the original series, the parameters of these model are computed (real issue of these methods). For instance, hidden *Markov model* technique assumes that the observations are independent but correlated through a probabilistic sequence of unobserved (hidden) variables, related by some parameters, that regulate the observed phenomenon occurrence with respect to time. Then, applying Bayesian statistics methods, these parameters are estimated. This process is typically applied in speech recognition and to generate automatic subtitles in many videos. For instance, in this latter case the hidden variable are the words, and by determining the parameters that relate the sequence of observations (audio wave received as input) and the hidden variables it is possible to assign a variable to a defined observations interval.

4. Data dictated methods: while in the previous categories the new reduced dimensionality is selected by the user, these methods automatically determine the optimal dimensionality reduction rate. For the sake of simplicity, they will not be analysed furthermore.

After having examined time series analysis, it is now possible to move to the next section, regarding distance metrics, an important concept necessary to understand many clustering algorithms, since it is widely used for the computation of the dissimilarity matrix.

1.2 Similarity and dissimilarity measures

The theoretical issue of time series similarity/dissimilarity search has been proposed by Agrawal et al. [25] and subsequently it became one of the main research areas in data mining and clustering, since this latter relies on dissimilarity measures to perform. Moreover, in traditional clustering, the distance between static objects is computed exactly while in time series clustering the degree of similarity or dissimilarity between time series is calculated approximately. Indeed, typically a distance function is exploited to determine the distance measurement between all the points of the multiple time series to be compared, and an estimated distance between time series is then derived. For the sake of simplicity, this thesis will consider just

two univariate time series and will focus on the main methods to estimate the similarity/dissimilarity between them. First of all, it is convenient to formally define the concept of similarity/dissimilarity measure and of time series distance [26].

Similarity measure

Let Y_t be a time series representing a set of data. A function $s: Y_t \times Y_t \to \mathbb{R}$ is called *similarity* on Y_t if it satisfies the following properties, $\forall\, y_i, y_j \in Y_t$:

- Non-negativity: $s(y_i, y_j) \geq 0$

(1.30)

- Symmetry: $s(y_i, y_j) = s(y_j, y_i)$

(1.31)

- If $y_i \neq y_j$ then $s(y_i, y_i) = s(y_j, y_j) > s(y_i, y_j)$

(1.32)

A similarity measure must have a large value for similar objects and zero value for very dissimilar objects. Generally, the similarity value ranges between zero and one, where one indicates the maximum similarity measure.

Dissimilarity measure

A function $d: Y_t \times Y_t \to \mathbb{R}$ is called *dissimilarity* on Y_t if it satisfies the following properties, $\forall\, y_i, y_j \in Y_t$:

- Non-negativity: $d(y_i, y_j) \geq 0$

(1.33)

- Symmetry: $d(y_i, y_j) = d(y_j, y_i)$

(1.34)

- Reflexivity: $d(y_i, y_i) = 0$

(1.35)

Different transformations are possible to obtain the dissimilarity d from the similarity s. The most common are:

- absolute dissimilarity: $d(y_i, y_j) = s(y_i, y_i) - s(y_i, y_j)$

(1.36)

- relative dissimilarity: $d(y_i, y_j) = \frac{s(y_i,y_i)-s(y_i,y_j)}{s(y_i,y_j)}$

(1.37)

- square dissimilarity: $d(y_i, y_j) = \sqrt{s(y_i, y_i) - s(y_i, y_j)}$

(1.38)

Typically, when the dissimilarity measure between two time series has a low value, the distance measure between them has a low value too. This suggests the use of distance metrics, defined below, to determine the degree of similarity between time series, as already done in many papers and works [27].

Distance metric

A function $D: Y_t \times Y_t \to \mathbb{R}$ is called a *distance metric* on Y_t if it satisfies the following properties, $\forall\, y_i, y_j, y_k \in Y_t$:

- Non-negativity: $D(y_i, y_j) \geq 0$

(1.39)

- Symmetry: $D(y_i, y_j) = D(y_j, y_i)$

(1.40)

- Reflexivity: $D(y_i, y_i) = 0$

(1.41)

- Triangle inequality: $D(y_i, y_j) \leq D(y_i, y_k) + D(y_k, y_j)$

(1.42)

If any of these properties is not satisfied, the distance is a measure but not a metric. The key difference between the two is that a metric allows to obtain standardized results since its properties hold always, while a measure can be derived with different techniques and different results can be possible. Although having the characteristics of a metric is desirable, a (dis)similarity measure can be quite effective without being a metric. For the sake of simplicity, in the following the two terms will be used indiscriminately.

The main distance metrics employed in time series analysis and clustering will be presented in the following. Anyway, once a metric has been selected, the distance between the two time series under consideration is typically computed as the sum of the distances between individual points. More formally, if Y_t and $Y_t{}'$ are two time series of length N representing a set of data and y_i, y_j' are generic points of the first and second series respectively:

$$D(Y_t, Y_t') = \sum_{t=1}^{N} D(y_i, y_j'), \qquad \forall i, j \in [1, N] \tag{1.43}$$

Of course, equation (1.43) supposes that the two series under consideration have the same number of points. If this assumption is not true, other measures (typically non-metric) must be adopted, as it will be further explained.

The metrics exploited in clustering algorithms must cope with the problems caused by common features of time-series data such as noise, temporal drift, longitudinal scaling, offset translation, linear drift, discontinuities, and amplitude scaling. Various techniques have been developed for similarity measure, and the method to choose is problem specific. In particular, the choice of a proper distance approach depends on the characteristic of time-series, its length, representation method, and of course on the objective of the clustering process. Typically, the methods to determine similarities between time series can be classified under three main categories

- Similarity in time: distance measures in which the time of occurrence of patterns is crucial. This kind of measuring is computationally costly if applied to raw time series, thus the calculation is performed on transformed time

series. For instance, as already discussed before, a Fourier transform can be applied to the time series to reduce its complexity by eliminating the unnecessary harmonics (low energetic content).

- Similarity in shape: distance measures in which the time of occurrence of patterns is not important. More precisely, clusters of time series with similar patterns are constructed regardless of the time instants. It is possible to see the similarity in time as a special case of similarity in shape.

- Similarity in change: in this approach modelling methods are employed to represent the time series and then a similarity measure is applied on the parameters of the fitted model. For instance, as already discussed, deterministic methods allow to estimate an analytic function that describes the series. Moreover, Hidden Markov Model technique allows to obtain a probability density function, for the considered time series, dependent on some parameters. If this model is applied to two time series, it is possible to exploit similarity in change between their model parameters. The result of using this metric is time-series clusters that have a similar autocorrelation structure. Besides, it is not a suitable metric for short time series [28].

Despite the presented classification, it is also possible to distinguish between *shape level* and *structure level* similarity measure. In particular, shape level similarity measures are exploited to measure similarity in short-length time series by comparing their local patterns. For instance, expression profiles or individual heartbeats can be compared through these techniques. Instead, structure level similarity measures are exploited to measure similarity basing on global/high level structure and are used in long time series data. For instance, the case in analysis of this thesis focuses on a set of load duration/generation curves; each curve is composed by 8760 time instants (one for every 15 minutes of one year) and related electrical power measurement. Therefore, a structure level similarity is searched in order to obtain the dissimilarity matrix necessary for the clustering process. Figure 1.10 resumes the concepts discussed until now.

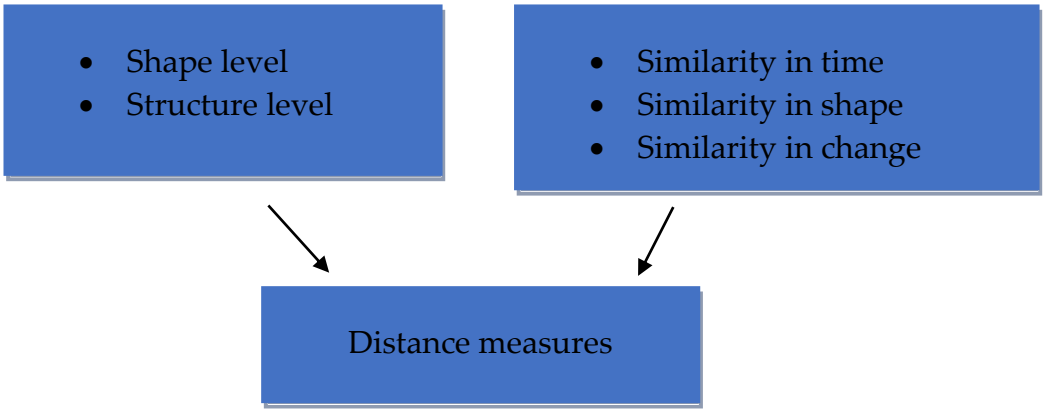


Figure 1.10: Approach to select the correct distance measure

Once that the different similarity classifications have been discussed, it is now possible to present the main distance metric that has been exploited for this thesis, called *dynamic time warping (DTW)*. For an optimal understanding of this metric, it is first necessary to briefly introduce, for two time series, the Euclidean distance and its generalization, Minkowski distance. More formally:

Euclidean distance

Let Y_t and Y_t' be two time series of length N. The Euclidean distance between Y_t and Y_t' is defined as:

$$D_E(Y_t, Y_t') = \sqrt{\sum_{t=1}^{N} (y_t - y'_t)^2} \tag{1.44}$$

This distance is simple and used as benchmark in many works. Moreover, the computational complexity is in the order of $O(n)$, thus its computation is fast. However, it has some disadvantages to be considered. Firstly, it requires that the two time series to be compared have exactly the same length; secondly, it is very weak and sensitive to small shift across the time axis [29]. Indeed, the Euclidean distance between time series is computed as the sum of the Euclidean distances between every couple of points corresponding to the same time instant of the first and second series respectively. Therefore, if one series is time-shifted forward or backward,

every distance of each couple will change, leading to a completely different global result. The generalization of the Euclidean distance is the Minkowski distance, defined as follows:

$$D_{L_p}(Y_t, Y_t') = \sqrt[p]{\sum_{t=1}^{N} (y_t - y'_t)^p} \tag{1.45}$$

where p is called the Minkowski order. Indeed, according to the value of this parameter, the Minkowski distance can be called Manhattan distance ($p = 1$), Euclidean distance ($p = 2$) or Maximum distance ($p = \infty$).

It is important to recall that Euclidean distance, as well as its generalization, defines a similarity in time between the two considered time series. Indeed, as already discussed, the fact that it is computed only for points of a correspondent time instant implies that the time occurrence of a pattern is crucial (definition of similarity in time). Again, this underlines the great sensitivity to time shifts of this distance metric, an open issue that affects this method. To address this issue, a new distance metric is now introduced. This metric is called Dynamic Time Warping (DTW) and is instead based on similarity in shape. The following subsection focuses on DTW, providing all the formal and mathematical details for its full scientific comprehension.

1.2.1 Dynamic time warping

Dynamic time warping is an extremely important algorithm in many areas. It has been introduced in the 60s [30] and extensively explored during the 70s for speech recognition applications [31]. More recently, its role has become essential in many applications such as handwriting and online signature matching [32], gestures recognition [33], data mining and time series clustering (time series databases search) [34], surveillance [35], protein sequence alignment and chemical engineering [36], music and signal processing [37]. As it will be further discussed, DTW allows to address the issue of time shift sensitivity of Euclidean distance and makes also possible to compare two time series with different length. Indeed, since with Euclidean distance method the distance measure is computed for every couple of points corresponding to the same time instants, the two considered series must have the same length. Instead, DTW computes the distance measure for every possible

couple of point and determines the minimum/maximum distance between the two series, thus finding their optimal alignment regardless of their length. In the following, dynamic time warping technique will be analysed in detail from a mathematical and a practical point of view. For the sake of clearness, the two approaches will be discussed separately, with a greater focus on the second one.

Mathematical approach to dynamic time warping

Let $Y = \{Y_1, Y_2, \dots, Y_N\}$ be a set of time series of length T. Note that this assumption is not strict, since as already discussed it is possible to make the hypothesis that the series have different lengths. Anyway, for the sake of simplicity, this case will not be examined in detail. An alignment A of length $|A| = m$ between two time series Y_i, Y_j is defined as the set of m couples, with $T \leq m \leq 2T - 1$, of aligned elements of Y_i to elements of Y_j. More formally:

$$A = \{(a_1(1), a_2(1)), (a_1(2), a_2(2)), \dots, (a_1(m), a_2(m))\} \tag{1.46}$$

a defines a so-called *warping function* that realizes a mapping from time axis of Y_i to time axis of Y_j and the applications a_1 and a_2 obey to the following conditions:

- Boundary condition: $1 = a_1(1) \leq a_1(2) \leq \cdots \leq a_1(m) = T$
  $1 = a_2(1) \leq a_2(2) \leq \cdots \leq a_2(m) = T$ (1.47)

- Monotonicity condition: $a_1(l+1) \leq a_1(l) + 1$ and $a_2(l+1) \leq a_2(l) + 1$
  $(a_1(l+1) - a_1(l)) + (a_2(l+1) - a_2(l)) \geq 1$
  $\forall l \in \{1, \dots, m\}$ (1.48)

Even if the mathematical explanation can appear quite cumbersome, intuitively an alignment A defines a strategy to associate all the elements of the two series. Note that the boundary condition imposes that the first and last point of the two series respectively must be coupled. This is shown by equation (1.47), where $a_1(1), a_2(1)$ are equal to one and $a_1(m), a_2(m)$ are equal to T, and contributes to the low sensitivity of DTW to time shifts. Indeed, if a series is time-shifted forward or

backward, its first and last point will be coupled anyway with the first and last point of the other time series. A slight difference can be present between alignment of the other points in the middle, but anyway the global distance measure does not show a great variation as does instead using Euclidean distance method. It is important to remark that the boundary condition does not impose severe restrictions on the alignment between the other points of the series. For instance, the second point of the first series could be coupled with the third point of the second series or vice versa. Clearly, many different alignments are possible, as represented in figure 1.11, that shows three possible alignments between two series $X_{jt} = \{x_{j1}, x_{j2}, \dots, x_{j7}\}$ and $X_{it} = \{x_{i1}, x_{i2}, \dots, x_{i7}\}$ [26]. For example, the green path in figure 1.11 aligns the two series as:

$$A = \{(x_{i1}, x_{j1}), (x_{i2}, x_{j2}), (x_{i2}, x_{j3}), (x_{i3}, x_{j4}), (x_{i4}, x_{j4}), (x_{i5}, x_{j5}), (x_{i6}, x_{j6}), (x_{i7}, x_{j7})\} \tag{1.49}$$

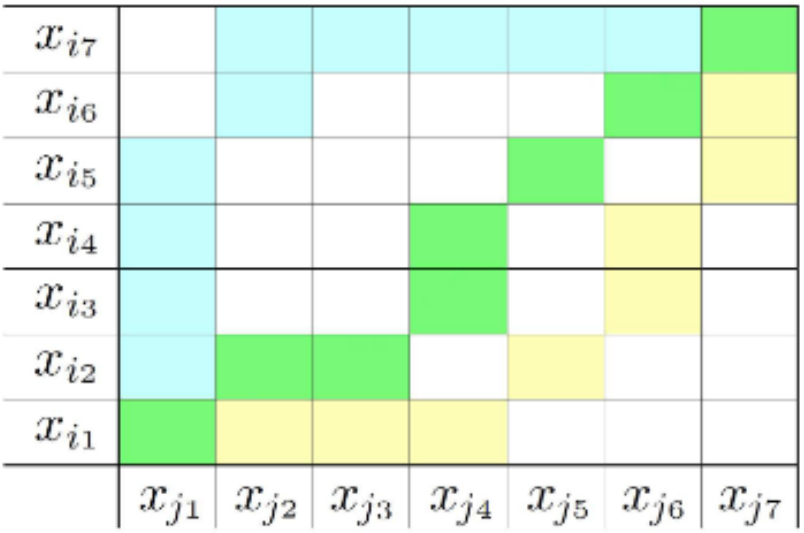


Figure 1.11: Three possible alignments (warping paths) between two time series

In the following, Å will denote all the set of possible alignments between the two considered time series. Dynamic time warping between time series Y_i, Y_j is defined as:

$$DTW(Y_i, Y_j) = min_{A \in Å} \frac{1}{|A|} \sum_{(t',t) \in A} \varphi(y_{it'}, y_{jt}) \tag{1.50}$$

where $\varphi: Y_i \times Y_j \to \mathbb{R}$ is a positive and real value point-distance function (generally the Euclidean distance). As it is possible to note from equation (1.50), DTW is realized by determining the alignment of the two considered time series able to minimize the total distance between them, given by the sum of the point-to-point distances scaled by $\frac{1}{|A|}$. In particular, as it will be clarified in the next section, a $|Y_i| \times |Y_j|$ cost matrix is built up, in which the value at (t, t') is the minimum distance warp path that can be constructed from the two series and the value at $(|Y_i|, |Y_j|)$ is the minimum distance between the two series under the minimum distance warp path Thus, through DTW an optimal alignment is determined with the important property of being almost insensitive to time shifts. Euclidean distance method, instead, consider just one alignment, in which only the points correspondent to the same time instant are aligned.

More formally, the Euclidean alignment between time series Y_i, Y_j is:

$$A_E = \{(a_1(1), a_2(1)), (a_1(2), a_2(2)), \dots, (a_1(T), a_2(T))\} \tag{1.51}$$

where $\forall t = 1, \dots, T: a_1(t) = a_2(t) = t$ and $|A_E| = T$. According to this definition, it is possible to prove that the Euclidean distance between time series Y_i, Y_j is obtained as:

$$D_E(Y_i, Y_j) = \frac{1}{|A_E|} \sum_{k=1}^{|A_E|} \varphi(y_{ia_1(k)}, y_{ja_2(k)}) = \frac{1}{T} \sum_{t=1}^{T} \varphi(y_{it}, y_{jt}) \tag{1.52}$$

This result is coherent with the already proposed definition of Euclidean distance between time series as the sum of the Euclidean distances between their correspondent points. To conclude this brief mathematical analysis of dynamic time warping, it is important to recall that DTW is not a distance metric, since it does not satisfy the triangle inequality (equation (1.42)), and its computational complexity is in the order of $O(n^2)$.

Practical approach to dynamic time warping

In this paragraph, a more practical and operative approach to DTW will be presented. In particular, the algorithm for DTW computation will be discussed while the implemented code in Matlab and R Studio will be examined in the third chapter. As already mentioned, in order to compare two time series, a distance function φ must be exploited. Intuitively, φ has a small value when the series are similar and a large value if they are different. It is common to call the distance function *cost function* and the task of optimal alignment of the series becomes the task of arranging and coupling all series points by minimizing the cost function. More precisely, if Y_i and Y_j are two time series to be compared of length N and M respectively, the algorithm starts by computing the distance matrix $C: N \times M$, that contains all the pairwise distances between Y_i and Y_j. Indeed, the (h, k) element of C corresponds to the Euclidean distance between $y_{ih} \in Y_i$ and $y_{jk} \in Y_j$. The distance matrix is commonly called *local cost matrix*. Formally:

$$C: N \times M, \quad c(h,k) = \sqrt{\left(y_{ih} - y_{jk}\right)^2} = \left|y_{ih} - y_{jk}\right|, \quad h \in \{1, \dots, N\}, k \in \{1, \dots, M\} \tag{1.53}$$

Once the local cost matrix has been computed, the algorithm determines the alignment path which minimizes the total local cost, given by the sum of local costs along the path. It is also possible to plot a heatmap of the local cost matrix to highlight that the optimal alignment path runs through its low-cost areas. For instance, figure 1.12 shows an example, taken from [38], of the local cost matrix heatmap. Instead, furthermore in this section, the heatmap related to a specific example case will be examined and discussed in detail.

As it is possible to observe from figure 1.12, the optimal alignment represented by the blue path passes only through the green areas of the heatmap, thus leading to a minimized global cost.

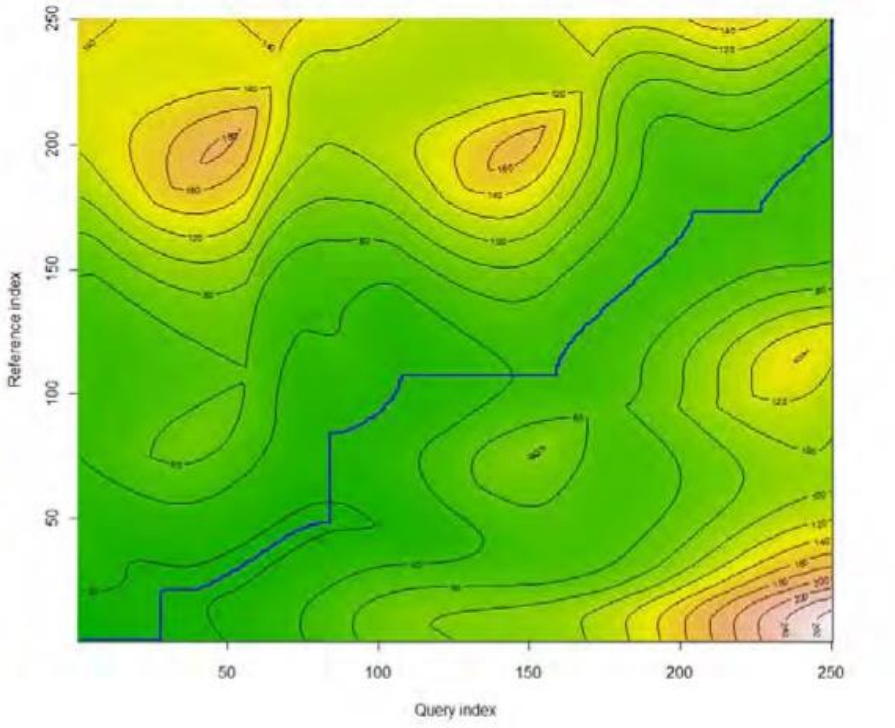


Figure 1.12: example of local cost matrix heatmap and optimal alignment path

The alignment path, also called *warping path* or *warping function*, defines the correspondence of an element $y_{ih} \in Y_i$ to an element $y_{jk} \in Y_j$. More formally, it is a sequence of points $P = (p_1, p_2, \dots, p_L)$ with $p_l = (p_i, p_j) \in [1, \dots, N] \times [1, \dots, M]$ for $l \in [1, \dots, L]$. Of course, there are some conditions and criteria to be fulfilled. These conditions have been already presented in the previous section and will now be recalled, adopting the new notation introduced in this section:

- Boundary condition: $p_1 = (1,1)$ and $p_L = (N, M)$

(1.54)

This condition ensures that the starting and ending points of the warping path must be the first and last points of the aligned series.

- Monotonicity condition: $n_1 \leq n_2 \leq \cdots \leq n_k$ and $m_1 \leq m_2 \leq \cdots \leq m_k$

(1.55)

This condition preserves time ordering of aligned points.

- Step-size condition: $(p_{l+1} - p_l) \in \{(1,1), (1,0), (0,1)\}$

(1.56)

This condition limits the warping path from long jumps (shifts in time) while aligning sequences. Indeed, if for example (2,0) was an allowed jump, a position of the cost matrix could be skipped, since the warping path could move by two positions for each step. This concept will be discussed and examined in detail further in this section.

The cost function associated to a warping path on the local cost matrix is defined as:

$$c_p(Y_i, Y_j) = \sum_{l=1}^{L} c(y_{il}, y_{jl})$$

(1.57)

Equation (1.57) shows that, as already discussed, the global cost related to a warping path is the sum of single local costs of the local cost matrix positions related to the warping path itself.

The warping path which has the minimum global cost is called *optimal warping path* and will be denoted with P^*. To determine the optimal one, every possible warping path between Y_i, Y_j should be tested. Since this could be computationally challenging due to the exponential increase of paths corresponding to a linear increase of the lengths of the two series, DTW employs dynamic programming-based algorithm with computational complexity in the order of $O(NM)$. More precisely, DTW algorithm exploits the DTW distance function, defined as follows:

$$DTW(Y_i, Y_j) = c_{P^*}(Y_i, Y_j) = \min \{c_p(Y_i, Y_j), p \in P^{N \times M}\}$$

(1.58)

where $P^{N \times M}$ denotes the set of all possible warping paths. To determine this distance, the *global cost matrix* or *accumulated cost matrix* D is built up according to the following expressions:

- First row:

$$D(1,h) = \sum_{k=1}^{h} c(y_{i1}, y_{jk}), h \in (1, \dots, M) \tag{1.59}$$

- First column:

$$D(h,1) = \sum_{k=1}^{h} c\left(y_{ik}, y_{j1}\right), h \in (1, \dots, N) \tag{1.60}$$

- All other elements:

$$D(h,k) = \min\left\{D(h-1,k-1) + c\left(y_{ih}, y_{jk}\right), D(h-1,k), D(h,k-1)\right\} + c\left(y_{ih}, y_{jk}\right), \\ h \in (1, \dots, N), k \in (1, \dots, M) \tag{1.61}$$

where c denotes an element of the local cost matrix, already computed.

Once that the global cost matrix has been obtained, the optimal warping path and the global distance are derived. In the following pages, algorithm 1 and algorithm 2 report the main steps to build up the global cost matrix and to find the optimal warping path. Moreover, figure 1.13 shows the so-called *three-way plot* of the alignment between the two considered time series. Indeed, it places one series horizontally in a small lower panel, the other series vertically on a left panel and a larger inner panel holds the warping curve. In this way, matching points can be recovered by tracing indices on the first time series, moving upwards until the warping curve is met, and then moving leftwards to discover the index of the other matched series.

Algorithm 1 Computation of global cost matrix (dtw) [38]

$n = |Y_i|$
$m = |Y_j|$
dtw=matrix($n \times m$)
dtw(1,1)= $c(1,1)$
for $i = 2, i \leq n, i++$ **do**
    dtw$(i, 1)$=dtw$(i-1,1) + c(i, 1)$#first column
end for
for $j = 2, j \leq m, j++$ **do**
    dtw$(1, j)$=dtw$(1, j-1) + c(1, j)$ #first row
end for
for $i = 2, i \leq n, i++$ **do**
    for $j = 2, j \leq m, j++$ **do**
        dtw(i, j)=min{dtw$(i-1, j) + c(i, j)$, dtw$(i, j-1)$, dtw$(i-1, j-1)$}
        $+c(i, j)$
    end for
end for
return dtw

It is recalled that c represents the local cost matrix, which element are the Euclidean distances between all the points of the two series. As it is possible to note, the algorithm computes first the upper left corner element of the global cost matrix, as the Euclidean distance between the first elements of the two time series to be compared. Then, the first row and column are computed through the expressions presented in previous page and reported by equation (1.59) and (1.60). In particular, these equations show that an element of the first row/column is given by the sum of the elements of the local cost matrix in the first row/column until the position of the considered element. This is achieved by summing up to the value of the local cost matrix in the considered position the value of the global cost matrix in the previous position in the row/column under consideration. Then, all the remaining elements of the DTW matrix are computed iteratively according to equation (1.61).

Algorithm 2 Computation of optimal warping path (path) [38]

```
path=new_array(max between n, m)
i=rows(dtw)
j=columns(dtw)
while (i>1) and (j>1) do
    if i==1 then
        i= i − 1
    else if j==1 then
        j= j − 1
    else
        if dtw(i − 1, j)=min{dtw(i − 1, j), dtw(i, j − 1), dtw(i − 1, j − 1)} then
            i = i − 1
        else if dtw(i, j − 1)=min{dtw(i − 1, j), dtw(i, j − 1), dtw(i − 1, j − 1)}
        then
            j= j − 1
        else
            i = i − 1, j = j − 1
        end if
    add to path position (i, j)
    end if
end while
return path
```

As it is possible to note, starting from lower right corner of the global cost matrix (i, j are initialised to the number of rows and columns of the matrix, thus the first considered position is the lower right corner) the three adjacent positions are checked in order to determine the one which has the minimum distance. For instance, if the DTW matrix is 9x9, starting from position (9,9) the examined positions are (8,9), (9,8) and (8,8), that are adjacent to (9,9). The indexes of the new selected position (that have minimum distance from the starting position) are saved in the path array and the procedure is iteratively repeated until the first position of the global cost matrix (upper left corner) is reached. The result is an array containing the positions of the selected element of the global cost matrix, thus containing the indices of the optimal warping path (alignment) between the two considered time series.

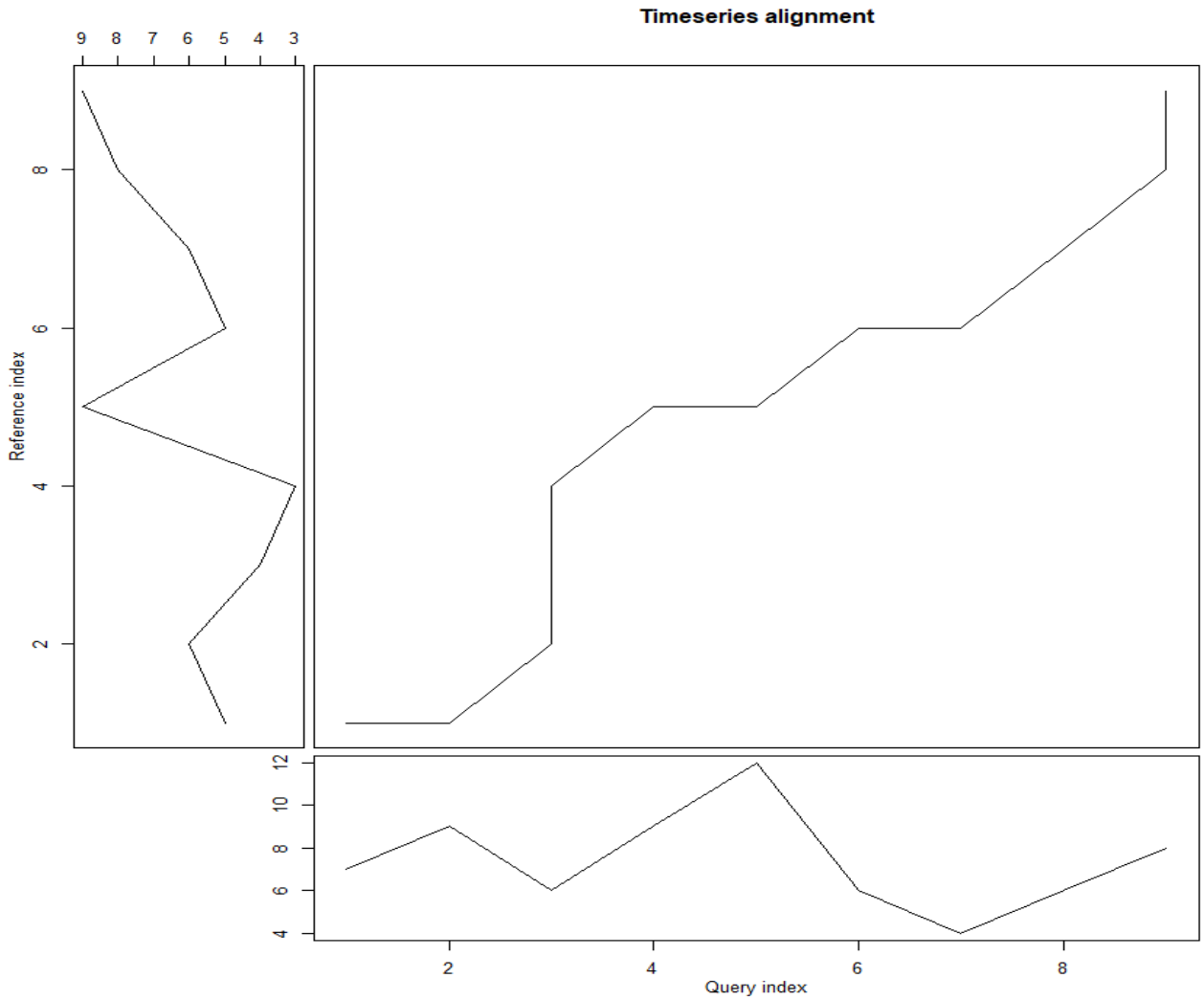


Figure 1.13: three-way plot of time series alignment

Figure 1.13 shows the so-called *three-way plot* of the alignment between the two considered time series. As already discussed, it places one series horizontally in a small lower panel, the other series vertically on a left panel and a larger inner panel holds the warping curve. In this way, matching points can be recovered by tracing indices on the first time series, commonly called *query series*, moving upwards until the warping curve is met, and then moving leftwards to discover the index of the other matched series, commonly called *reference series*. More precisely, the two time series that have been compared are:

- First series: $Y_1 = \{7,9,6,9,12,6,4,6,8\}$
- Second series: $Y_2 = \{5,6,4,3,9,5,6,8,9\}$

Note that both series have length $N = 9$, thus the local and global cost matrixes will be 9×9. For visual purpose, the two considered series are reported in the following figure (Y_1 in blue and Y_2 in red).

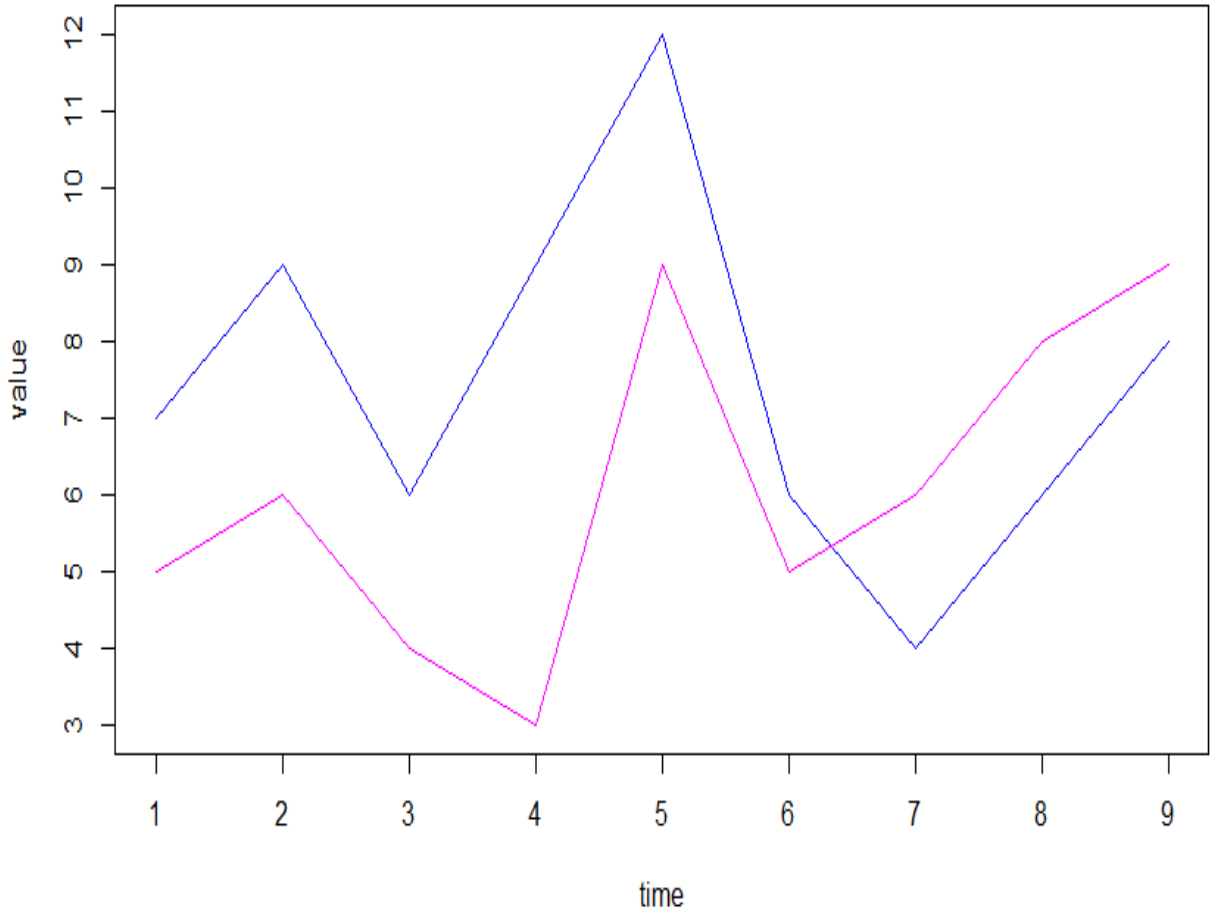


Figure 1.14: two time series under consideration

By applying Algorithm 1, it is possible to compute the global cost matrix. Then, through Algorithm 2, the optimal warping path is defined. Figure 1.15 reports a heatmap of the global cost matrix, with the optimal path shown as a blue line. As it is possible to observe, this path runs only through the low-cost areas of the matrix, as already explained. Moreover, figure 1.16 shows the alignment between the two series through dotted lines connecting their points. For instance, it is possible to observe that the first two points of the red series are aligned with the first point of the blue series. This is confirmed by the three-way plot in figure 1.13, which is made by a horizontal line in the first two positions. On the contrary, when the warping path is

made by vertical lines, more points of the blue series are aligned with one point of the red series.

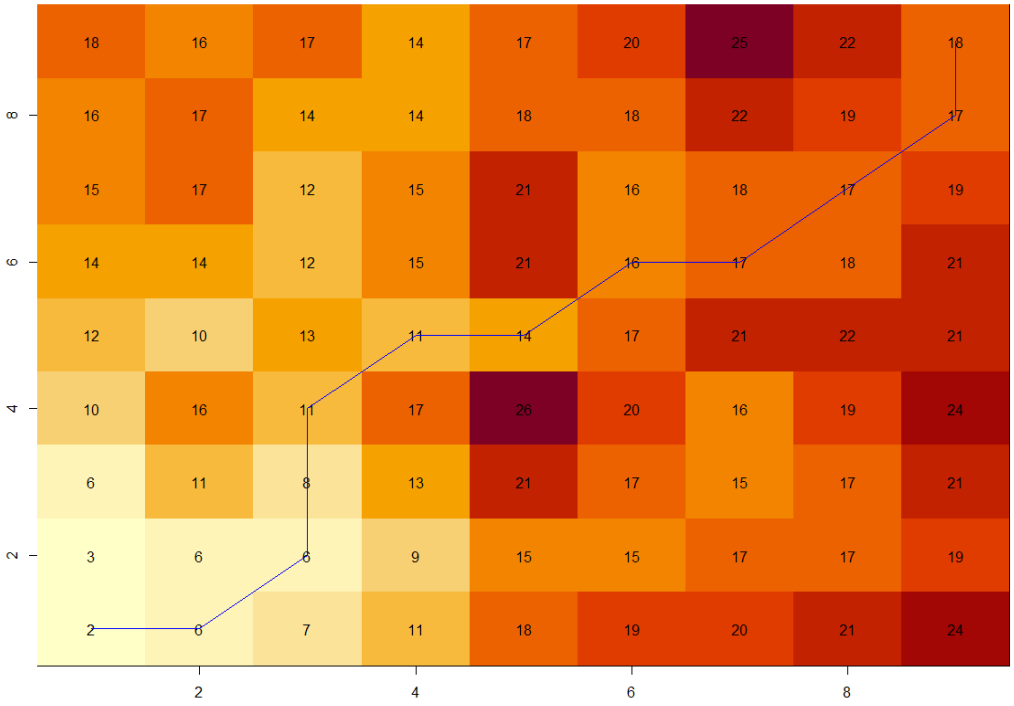


Figure 1.15: global cost matrix heatmap and optimal warping path

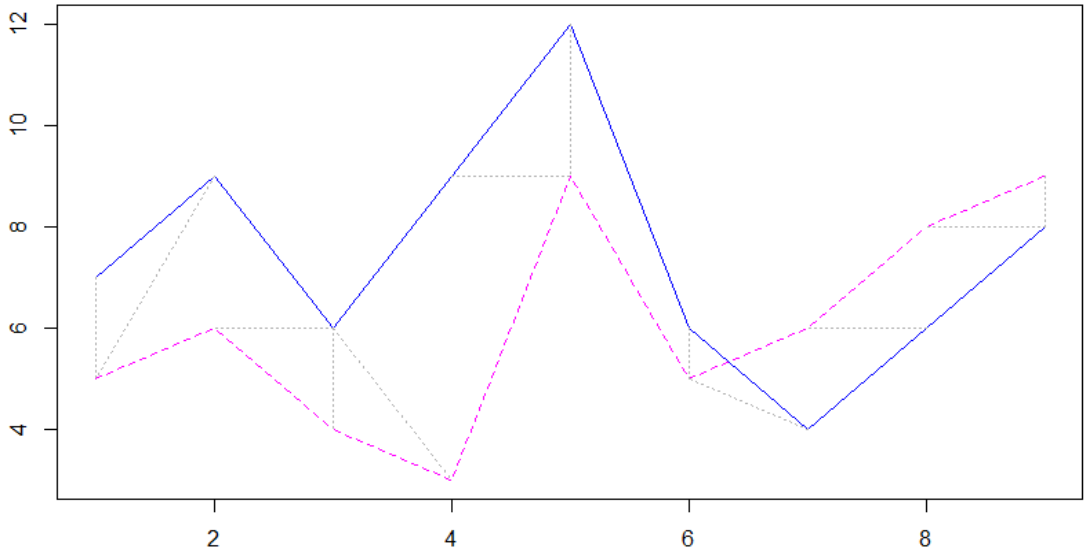


Figure 1.16: optimal alignment between the two considered time series

To conclude the analysis of dynamic time warping, it is now convenient to briefly discuss the effect of the step-size condition on the result. As already explained, when there is no difference between the considered time series, the warping path coincides with a diagonal line. As differences between time series increase, the warping path deviates more from the diagonal line by matching similar time-axis fluctuations. Indeed, if the two series coincide, DTW algorithm surely defines a one-to-one matching of correspondent points, since they have the minimum distance with respect to other points of the series. Thus, the warping path will show a diagonal line, also evident in the three-way plot. Instead, when multiple points of a series are matched to a single point of the other, the warping path becomes horizontal or vertical, deviating from the diagonal behaviour. While dynamic time warping finds the optimal alignment of the two considered time series, sometimes it tends to create an unrealistic correspondence between time series features by aligning very short features of one of them to long features of the other. In order to avoid this phenomenon, the warping path is subject to constraints on each step. These constraints define the possible relations between several consecutive points on the path and are called *step-size conditions* or *step-size functions*. For instance, after moving in horizontal or vertical direction for k consecutive points, the warping path could not be allowed to continue in the same direction before stepping l points in diagonal direction. This situation is shown in the left part of figure 1.17 below [38], in which after k horizontal steps the warping path must proceed in diagonal direction for l steps.

Figure 1.17: example of possible step-size conditions

Instead, the right part of figure 1.17 shows the step-size condition presented in this chapter (equation (1.56)) and used for the optimal warping path computation. This condition imposes that, for a generic position in the cost matrix, the warping path

can move just of one step horizontally, vertically or diagonally, as described in Algorithm 2 explanation. Indeed, according to the right part of figure 1.17, to reach the end point it is possible to move diagonally only from a position adjacent to the end point itself, while it is necessary to move one step horizontally or vertically and one step diagonally from a non-adjacent position to the end point.

Once that a step-size condition has been selected, a weight is assigned to each step of the obtained warping path. Several step patterns have been discussed in literature. A classic paper by Sakoe and Chiba [31] classifies them according to two properties: their symmetry (*symmetric/asymmetric*), and the bounds imposed on the slope expressed through a parameter *P*. Figure 1.18 shows four typical step patterns.

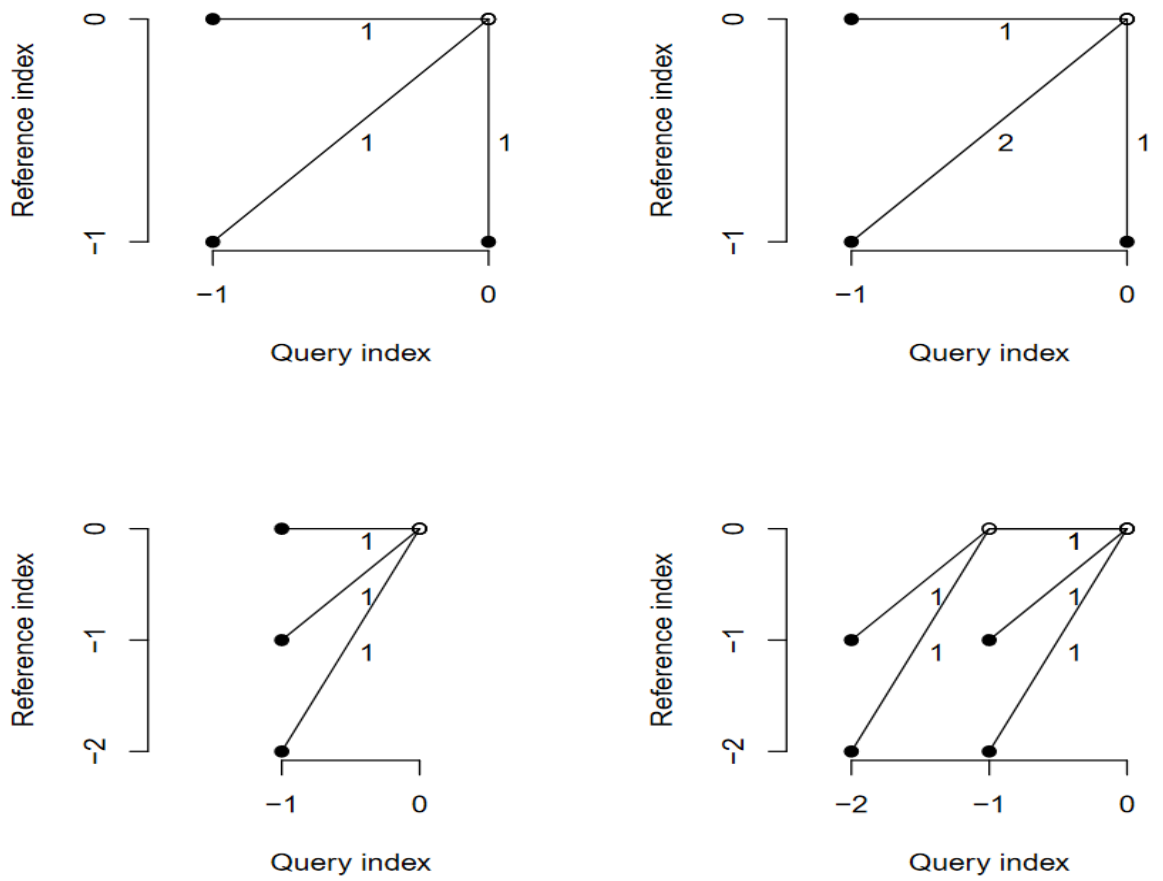


Figure 1.18: four typical step patterns in literature

The step-size condition exploited in Algorithm 2 corresponds to the upper right image in figure 1.18. It is called *symmetric2* step pattern and allows an unlimited number of elements of the query to be matched to a single element of the reference,

and vice-versa, anyway avoiding to skip points since the maximum step is of one position forward (45° slope). The average cost per-step is computed by dividing the cumulative distance by $N + M$, where N is the length of the query time series and M is the length of the reference. Moreover, the cost (weight) assigned to a diagonal step (one-to-one matching) is higher with respect to the weight assigned to horizontal or vertical steps, as shown by numbers on figure 1.18. The left right corner image, instead, shows the so-called *symmetric1* step pattern, similar to symmetric2 but with a same weight assigned to each step. Finally, the lower images show *asymmetric* patterns, where some positions can be skipped since the slope of diagonal lines can be higher of 45°.

To conclude the analysis of dynamic time warping, it is convenient to list its main advantages and disadvantages for a more complete view. On one hand, DTW is capable to handle time shifts, allowing similar shapes to be matched even if they are out of phase along the time axis. Moreover, it can assist clustering of different-length time series and its error rate is lower than using the Euclidean distance. On the other hand, DTW is sensitive to outliers and its complexity is higher than Euclidean distance complexity. To solve this issue, an efficient lower bound approximations of the DTW distance has been proposed [38].

1.3 Time series clustering

Before analysing in detail time series clustering, it is important to briefly discuss the so-called *data preparation methods*, employed to normalize, scale and transform data to achieve time-shift invariance and insensibility to possible offsets. Different possible methods can be exploited; the main are presented in the following.

Z-normalization

This technique transforms the time series to the same time scales by normalizing them. The new scaled element y_i' can be obtained from its related original element y_i as:

$$y_i' = \frac{y_i - \mu}{\sigma} \tag{1.62}$$

In equation (1.62), μ and σ represent respectively the mean and standard deviation of the considered time series of length N, computed as follows:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{1.63}$$

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \mu)^2} \tag{1.64}$$

Figure 1.19 below [18] shows the clustering result of a set of elements before and after z-normalization. The red, blue and yellow dots represent the centroids ("representative element") of the defined clusters, a concept that will be further discussed. As it is possible to observe from the left-side image (before normalization), the red centroid seems closer to the blue one than the yellow one. However, a different scale is adopted for x- and y-axis of the three series, therefore the observed similarity could be false. Indeed, the right-side image (after normalization) shows that red centroid is closer to the yellow one than the blue one. Thus, after z-normalizing data, it is possible to have a more accurate measure of the similarity between clusters.

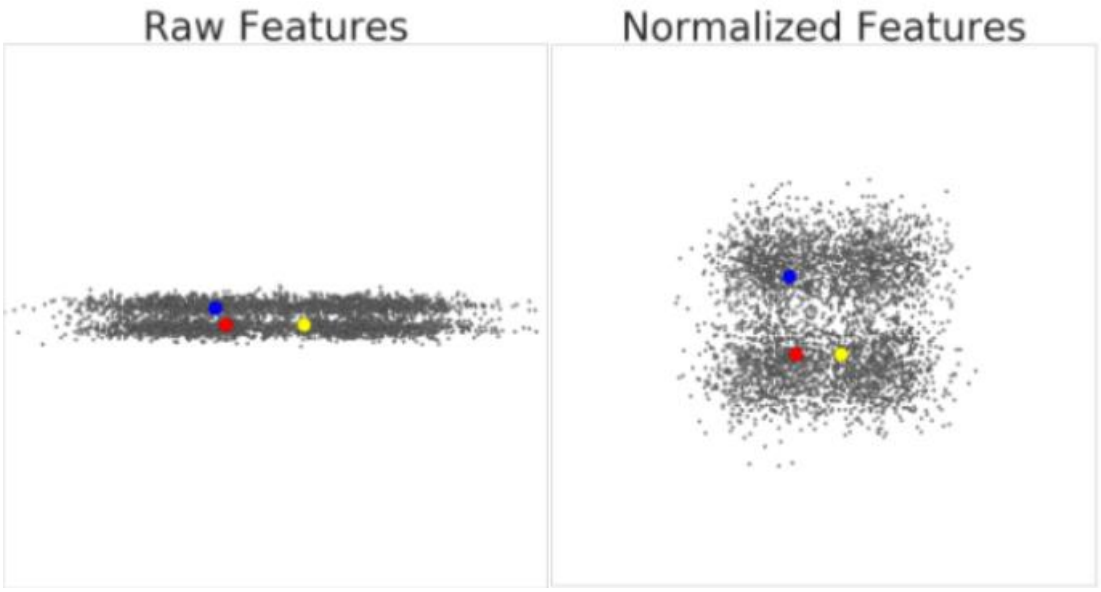


Figure 1.19: clustering results before and after z-normalization

Z-normalization is typically exploited when data have a normal distribution or if the dataset has too few elements to employ other methods. However, it is the simplest and fastest data-preparation method and can be used for preliminary but coherent results.

Log-transform

This method exploits another data transformation, called *log-transform*. When the distribution of the data is non-normal, they are transformed to make them as "normal" as possible and, thus, increase the validity of the associated statistical analyses. The log transformation is, arguably, the most popular among the different types of transformations used to make data approximately conforming to normality. Formally, the new transformed element y_i' can be obtained from its related original element y_i as:

$$y_i' = \ln(y_i) \tag{1.65}$$

For instance, figure 1.20 below [18] shows some data which follows a power-law distribution. For the sake of clearness, it is recalled that the power-law function is defined as:

$$f(y) = ay^{\alpha} + \varepsilon \tag{1.66}$$

where ε is called deviation term and represents uncertainty in observed values. Typically, α ranges between 2 and 3; this implies that the power-law function cannot be a probability density function strictly, since the area below cannot be equal to one. However, it is possible to define a truncated-power-law distribution as follows:

$$f_y(y) = Cy^{-\alpha} + \varepsilon, \quad y \geq y_{min} \tag{1.67}$$

The minimum value y_{min} is needed since the distribution has infinite area approaching zero, while the constant C is a scaling factor to ensure that the total area below the curve is equal to one. Figures 1.20 and 1.21 in the next page show the

results of clustering points of a time series, which follows a power-law distribution, in three clusters, represented by their centroids (blue, red, yellow dots). As it is possible to observe, after a log-transform the distribution of the data is closer to a normal distribution, and the red centroid is closer to blue than yellow, differently from figure 1.20.

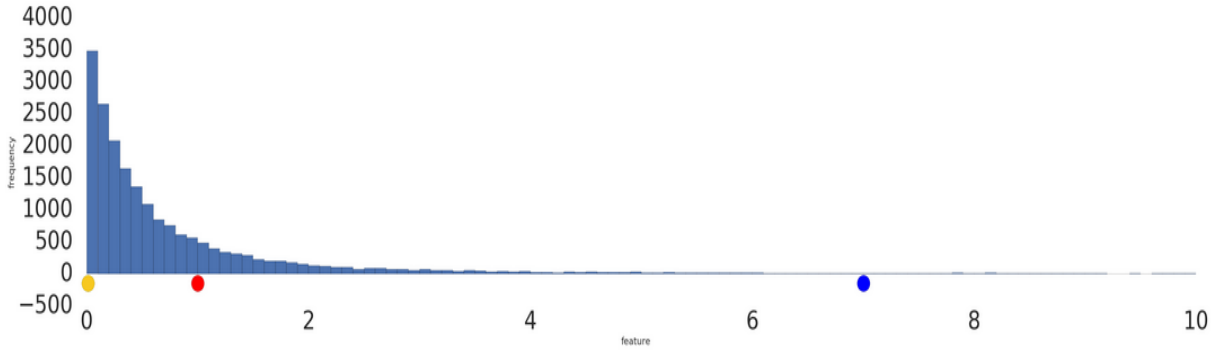


Figure 1.20: power-law distributed data and clusters centroids

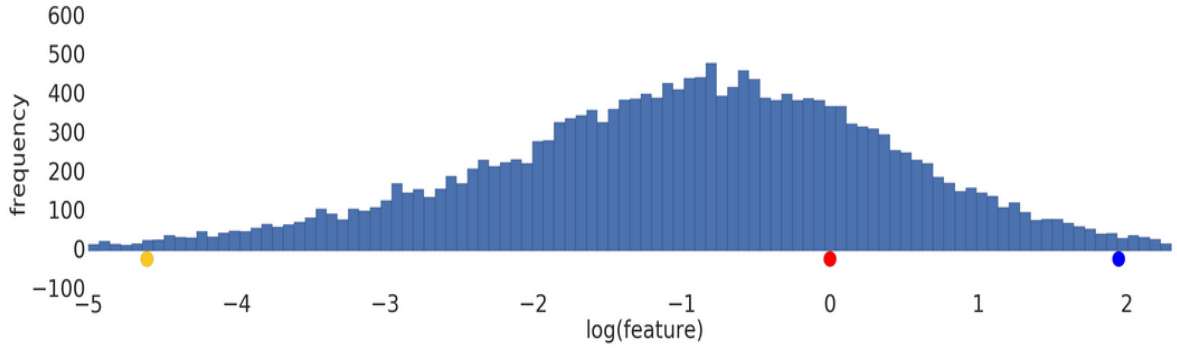


Figure 1.21: data distribution and clusters centroids after log-transformation

Data preparation methods using quantiles

While the previously presented methods typically require a specific data distribution for the time series, this method instead applies to any time series. Indeed, it can make a series statistically identical to a different reference data distribution. Typically, a normal distribution is used as reference so that by quantile-transforming the considered time series, that has an unknown distribution, a new transformed time series, that has a normal distribution, is obtained. The method is based on dividing data in intervals such that each interval contains the same number of elements. The

elements at interval boundaries are called *quantiles*. The main steps to convert data into quantiles are the following:

1. Sort values of series to be transformed and reference series from lowest to highest

2. Compute the mean between the lowest values, the second lowest values, the third lowest values of the two series and so on until the mean of the highest values of the two series is obtained.

3. Replace every value of each series with its correspondent mean (quantile).

By applying this process (with the same reference distribution) to two time series to be compared, after having converted data, the similarity between two elements is inversely proportional to the number of elements between the two, computed as the difference between the correspondent quantiles:

$$sim(y_i, y_j) \approx 1 - |quantile(y_i) - quantile(y_j)| \tag{1.68}$$

Quantiles are typically the best default choice to transform data. However, to obtain reliable indicators of the data distribution, a big dataset is needed. As a rule of thumb, to create n quantiles, at least $10n$ elements are necessary, otherwise other transformation methods are preferred. In this thesis, for the sake of simplicity, z-normalization method has been exploited, since anyway it allows to obtain a low sensitivity to offsets and scaling.

After having applied a data preparation method, a similarity measure must be selected for the clustering algorithm to perform. Indeed, before the algorithm can cluster data, it must be defined how similar pairwise elements are. As already discussed in the previous section, the most employed similarity metrics are commonly the Euclidean distance and the dynamic time warping, which determine similarity between time series in time and shape, respectively. After having prepared the data and selected a similarity measure, the clustering algorithm exploits this latter to cluster data. Eventually, it is important to carefully check the quality of clustering output; this is done by computing some quality parameters that indicates

the goodness of the result, as it will be further discussed. If clustering output is not as good as expected, the algorithm or its parameters are changed and the quality of results is checked again, in an iterative process that stops when the clustering output is accurate enough. Figure 1.22 [18] shows graphically the main steps to perform the clustering process, as discussed above.

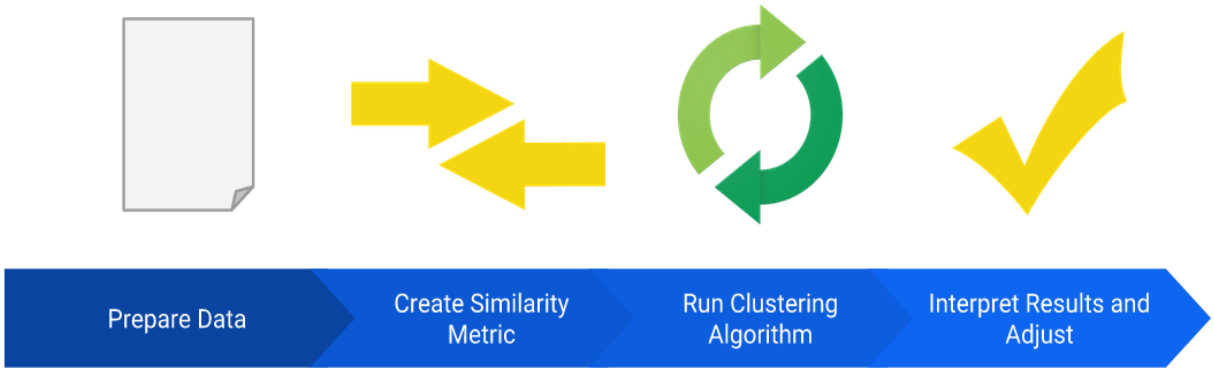


Figure 1.22: main steps to perform clustering on a dataset

Once that data preparation methods have been discussed, it is now possible to analyze in detail the clustering algorithms, defining their classification and working principle. First, it is convenient to formally introduce clustering:

Let $Y = \{Y_1, Y_2, \dots, Y_N\}$ *be a set of* N *time series. Time series clustering is the process of partitioning* Y *into* $C = \{C_1, C_2, \dots, C_k\}$ *according to a similarity measure or criterion.* C_i *is called cluster so that* $Y = \cup_{i=1}^{k} C_i$ *and* $C_i \cap C_j = 0$ *for* $i \neq j$.

The previous definition remarks that clustering algorithms group similar elements of the original set in the same clusters, according to the defined similarity measure. The term element is used on purpose, since two main clustering types are possible: clustering of multiple time series into representative ones (coherent with the provided definition) and clustering of points of a single time series to reduce its time dimensionality. In this latter case, the same definition can be used by defining Y as a set of points of a single time series of length N. Moreover, another distinction can be made for clustering of multiple time series. Indeed, the approach to time series clustering can be classified into:

A. Multiple time series clustering approach

   i. *Whole time series clustering*: clustering a set of time series with respect to their similarity
   ii. *Subsequent time series clustering*: clustering of a set of subsequences extracted from a single time series

B. Single time series clustering approach

   i. *Time point clustering*: clustering of points of a single time series based on a combination of their temporal proximity and similarity. Typically, subsequent time series clustering is preferred to this approach.

This thesis focuses on whole time series clustering, since as already mentioned 365 load/generation curves are clustered into 36 in order to reduce the computational time of many optimization algorithms. Therefore, before introducing the clustering algorithms that have been compared, it is convenient to analyze their possible classifications.

Firstly, there are generally three different approaches to (whole) time series clustering:

- Shape-based approach: the shapes of the (two) series are matched as much as possible, by a non-linear stretching and contracting of the time axes in order to align them. Typically, conventional clustering algorithms are exploited, with a modified similarity measure such as dynamic time warping.

- Feature-based approach: the row time series are converted into feature vectors of lower dimension. Then, a conventional clustering algorithm is applied to these vectors.

- Model-based approach: a parametric model is defined for each time series and then a suitable similarity measure and clustering algorithms is selected and applied to the extracted parameters.

Figure 1.23 [1] in the next page remarks graphically the previously discussed points. As it is possible to observe, shape-based and feature-based approach employ one of the most classical clustering algorithm, called *k-means* or *k-medoids* depending on the selected type of centroid. In this thesis, the *k-medoids* and *hierarchical Ward* algorithms

have been exploited to perform clustering on a set of load/generation curves, thus adopting a shape-based approach. The two algorithms will be discussed in detail in the next chapter.

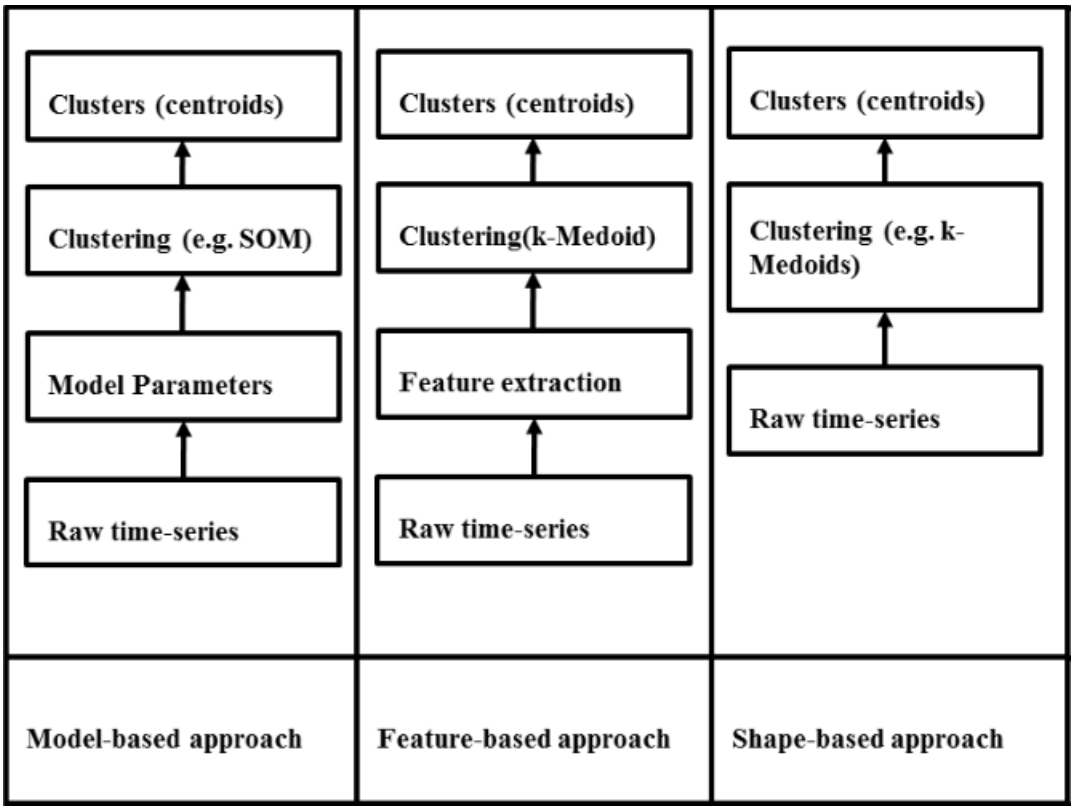


Figure 1.23: approaches to (whole) time series clustering

After having defined the main approaches to time series clustering, it is now possible to analyze the clustering algorithms, starting from their classification. In particular, the main categories in which clustering algorithms can be classified are:

- *Partitional* algorithms: given a dataset of N objects, a partitional algorithms creates k partitions (clusters) of them, with $k \leq N$. Moreover, it is required that each cluster contains at least one object and each object must belong to one and only one cluster. Typically, the number of partitions to be obtained k is an input of the algorithm that begins creating an initial partitioning and applies an iterative re-locating technique which moves objects from one cluster to another to improve the result. The general criterion for an effective clustering

is based on similarity: objects in the same cluster must be similar, close ore correlated while objects in different clusters must be very distant between each other. Once that the portioning process is complete, an optimal representative value for each cluster is defined. As it will be further discussed, in k-means algorithm this value is represented by the cluster mean, while in k-medoids it is represented by one of the objects located near the cluster centre.

- *Hierarchical* algorithms: these algorithms create a hierarchical decomposition of a set of N objects. Moreover, they can be classified in *agglomerative (bottom up)* or *divisive (top-down)*, according to how the decomposition is obtained. More precisely:

    - Agglomerative approach: initially each object forms a separate group. Successively, the closest objects are merged forming groups that are merged again in an iterative process until a desired number of clusters is obtained.
    - Divisive approach: initially all the objects belong to the same cluster. Successively, though an iterative process, this initial group is subdivided into smaller groups, until a desired number of clusters is obtained.

  The main issue with hierarchical algorithms is that they cannot correct errors. Indeed, whenever a step is performed (merging or splitting), there is no possibility to restore it. However, this "rigidity" of the clustering process is useful since it reduces the computational complexity and improves the stability of the method.

- *Density-based* algorithms: these algorithms have been created as an alternative to clustering algorithms exploiting distance measures between objects. The working principle is based on increasing a cluster dimension until the density (number of objects or data-points) in its neighbourhood (represented by a circle of radius r) exceeds a defined threshold. In other words, it is necessary to guarantee that, for each point inside a cluster, the neighbourhood of radius r contains a number of points higher and lower of a minimum and maximum value respectively.

- *Grill-based* algorithms: these algorithms quantize the objects space in a finite number of cells that form a grill structure. Then, all the clustering process is

performed on this quantized space. The main advantage of this approach is the very low time complexity, that does not depend anymore on the number of objects but just on the number of cells in each dimension of the quantized space.

- *Model-based* algorithms: these algorithms are applied to the extracted parameters of the model representing the series. For instance, clusters could be localized from a probability density function that reflects the spatial distribution of the data, by using standard statistical techniques. Typically, these algorithms consider the presence of noise, thus creating robust clusters.

In this thesis, as already discussed, k-medoid algorithm and hierarchical Ward algorithm have been exploited. Therefore, in the following, they will be analysed in detail. Since it would be too long to discuss every category, density-based, grill-based and model-based approaches are left to the reader as a possible deepening. of the subject.
Before examining the two mentioned algorithms, it is necessary to briefly analyse the possible "representative elements" of a cluster, formally called *cluster prototypes* or *cluster centroids*. Given a cluster $C = \{Y_1, Y_2, \dots, Y_N\}$ containing N time series, its prototype $\hat{R}$ minimizes the distance between all time series in the cluster and the prototype itself. More precisely:

$$\hat{R} = \min_R\{D(C,R)\} = \min_R\{\frac{1}{N}\sum_{i=1}^{N} D(Y_i,R)\} \tag{1.69}$$

where D indicated a generic distance measure. Generally, three main cluster prototypes have been defined in literature, as discussed below.

Using averaging prototype

The simplest cluster centroid is represented by the mean of the elements in the cluster. For instance, if this latter contains three time series, the average value between points with a correspondent time instant is computed and a new representative series is obtained (cluster prototype). However, if the series have different length the one-to-one mapping nature of this prototype makes it unable to capture the actual shape of the time series in the cluster, leading to an average

centroid with a distorted shape. The same holds in case of adopting a similarity-in-shape measure like dynamic time warping, since again it is not trivial to implement a pairwise prototype method. Instead, if the cluster contains data points rather than time series (for example to cluster points of a single time series to reduce its dimensionality), the average value of the elements in the cluster represents the optimal cluster prototype. To demonstrate this concept, the following mathematical proof is provided:

Given N points of a dataset assigned to K clusters, the optimal centroid $\widehat{\theta_k}$ of a cluster k minimizes the sum of distances between points in the clusters and the centroid itself. Thus, $\widehat{\theta_k}$ can be determined through the following optimization problem:

$$\begin{aligned} \min_{w,\theta}\{\mathrm{f}(\theta)\} = \ & \min_{w,\theta}\{\sum_{n=1}^{N}\sum_{k=1}^{K} w_{nk}|\theta_k - x_n|^2\} \\ \text{subject to} \quad & w_{nk} \in \{0,1\}\ \forall\ n,k \\ \text{and} \quad & \textstyle\sum_{k=1}^{K} w_{nk} = 1\ \forall\ n \end{aligned} \tag{1.70}$$

In equation (1.70), w_{nk} represents a binary variable equal to one if the nth point of the dataset belongs to the kth cluster and zero otherwise. Since the constraints do not requires difficult computations when derived, it is possible to solve the optimization problem by computing the derivative of the objective function with respect to θ_k and equating it to zero to determine its minimum value as follows:

$$\frac{\partial f(\theta)}{\partial \theta_k} = 2\sum_{n=1}^{N} w_{nk}(\theta_k - x_n) = 0 \tag{1.71}$$

The solution to equation (1.71) gives:

$$\sum_{n=1}^{N} w_{nk}\theta_k = \sum_{n=1}^{N} w_{nk}x_n$$

$$\theta_k \sum_{n=1}^{N} w_{nk} = \sum_{n=1}^{N} w_{nk}x_n$$

$$\widehat{\theta_k} = \frac{\sum_{n=1}^{N} w_{nk} x_n}{\sum_{n=1}^{N} w_{nk}} \tag{1.72}$$

As it is possible to observe from equation (1.72), the optimal cluster centroid $\widehat{\theta_k}$ of the kth cluster is computed as the sum of the points inside the cluster, divided by the number of points of the cluster itself. This is exactly the definition of average value, hence the correctness of the averaging prototype for a set of points is proved.

Using medoid as cluster prototype

In this approach, the centre of a cluster is defined as a time series which minimizes the sum of squared distances to other objects within the cluster. Note that this definition is similar to the one provided in equation (1.70), but the important difference in this case is that the medoid must be one of the time series/points of the cluster. Indeed, given time series in a cluster, the distance of all time series pairs within the cluster is calculated using a distance measure such as Euclidean distance or DTW. Then, the time series in the cluster which has lower sum of square distances is defined as medoid of the cluster itself. If the cluster contains data points rather than time series (for example to cluster points of a single time series to reduce its dimensionality), the same optimization problem already presented can be solved, with the additional constraint that $\vartheta_k \in \{x_1, \dots, x_n\} \forall\, k$. This problem is not trivial at all and is solved by the k-medoid algorithm to determine the cluster centroids. There are several advantages in using medoids as centroids with respect to the average values. First, a medoid-based approach can be used with any similarity measure; secondly, the medoid is comparable to the median. Searching in literature, plenty of articles can be found describing how and why the median is less sensitive to outliers and noise than the average value. Anyway, using medoid as cluster prototype increases the computational time and complexity of the clustering algorithm. For instance, k-means algorithm has a computational complexity in the order of $O(N \times K \times i)$, where i denotes the number of iterations, while k-medoid algorithm has a computational complexity in the order of $O((N-K)^2 \times K \times i)$.

Using local search prototype

In this approach, at first a medoid-based clustering is performed. Then, based on the selected optimal warping path, the average prototype is computed and new warping paths are defined from this prototype. Then, a medoid-based clustering is performed again, in an iterative process that stop when there is no variation from medoid based

and average centroids. This approach can lead to better results then the other, but it increases a lot the computational complexity of the clustering algorithm. Therefore, typically is not used and the medoid prototype is preferred.

The last important topic related to whole time series clustering, before analysing the two mentioned algorithms in detail, consists of evaluating the goodness of the clustering result. The evaluation measures employed in different approaches can be classified in *visualization measures* (visualize the result and analyse it) and *scalar accuracy measures* (a single real number represents the accuracy of the result). These latter numerical measures are further classified into:

- *External validity indexes*: external validity indexes are the measures of the agreement between two clusters, one of which is usually a known/golden cluster (the so-called *ground truth*) and another is derived from the clustering procedure. Ground truth is the ideal clustering that is often built using human experts. In this type of evaluation, ground truth is available, and the index evaluates how well the clustering matches the ground truth.

- *Internal validity indexes:* typically, the goal of a clustering process is to obtain high intra-cluster similarity (objects within a cluster are similar) and low inter-cluster similarity (objects from different clusters are dissimilar). Internal validation indexes compare solutions based on the goodness of fit between each cluster and the data. More precisely, they evaluate clustering results by using only features and information inherent in a dataset and are typically used in case that true solutions (ground truth) are unknown. However, these indexes can only make comparisons between different clusters generated using the same model/metric.

Many different indexes can be listed these two categories. Since it would be too long to analyse them all, they are left to the reader as a deepening of the subject, while the only index which is now presented is an internal validity index called *sum of squared errors*, defined as follows:

$$SSE = \sum_{k=1}^{K} \sum_{Y_i \in C_k} D(Y_i, \widehat{R_k})^2 \tag{1.73}$$

where $\widehat{R_k}$ represents the prototype of the cluster k. As it is possible to observe from equation (1.73), the SSE parameter is obtained by summing up all the distances of every time series to the nearest cluster prototype. Intuitively, a good clustering result is supposed to give a low SSE value.

Figure 1.24 shows the classification of validity indexes (with some examples). Instead, figure 1.25 [1] resumes all the topics discussed until now about whole time series clustering.

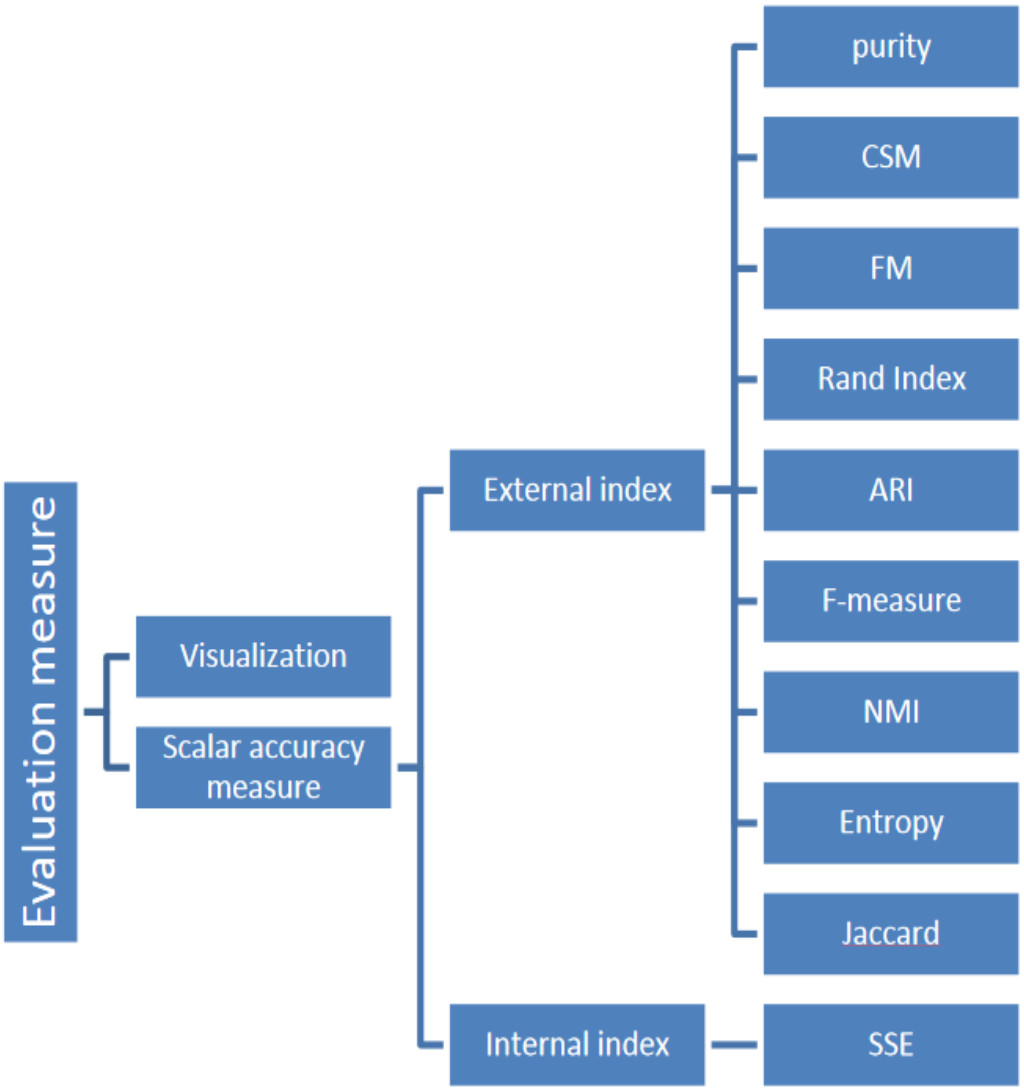


Figure 1.24: evaluation measures classification

Once that the main topics related to whole time series clustering have been discussed, it is now possible to analyse the two mentioned algorithms exploited in this thesis: the *k-medoid* algorithm and the *hierarchical Ward* algorithm. This analysis is examined in the next chapter, while chapter 3 focuses on the practical implementation of these algorithms to an electrical engineering problem.

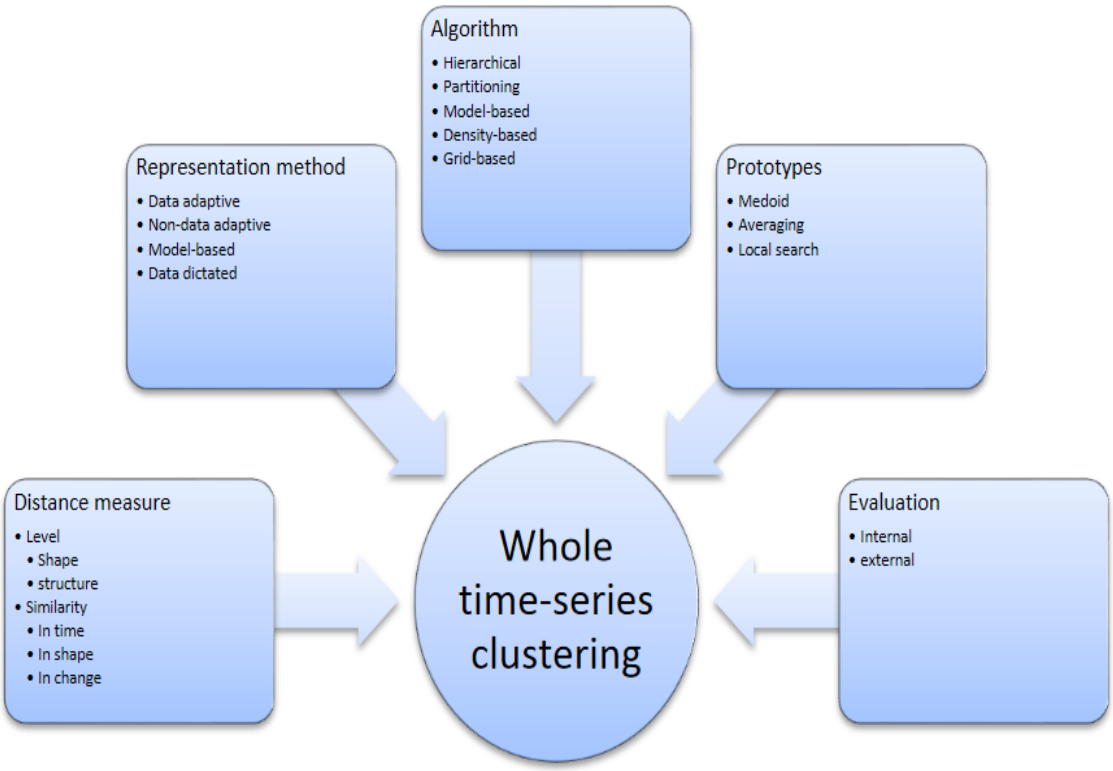


Figure 1.25: overview of the five components of whole time series clustering

2. Clustering algorithms and optimal number of clusters

In this chapter, the two clustering algorithms exploited in this thesis are presented and discussed in detail. Then, their main advantages and disadvantages are listed in order to have a quick and clear comparison between the methods. Eventually, a brief discussion about the optimal number of clusters is conducted, and some parameters are defined.

2.1 K-means clustering algorithm

K-means clustering has been introduced by MacQueen in 1967 [40]. It is a quite simple algorithm with a computational complexity in the order of $O(N \times K \times i)$, indicating with N the number of datapoints, with K the number of clusters and with i the required number of iterations. It is important to remark that K is an input parameter of the algorithm and must be defined by the user. This is one of the main differences with the hierarchical Ward method, in which the optimal number of clusters is automatically selected according to some constraints.

K-means algorithm can be divided into two main stages: the *initialization stage* and the *iteration stage*. In the first stage, K elements are randomly selected from the dataset. These elements represent the initial centroids of the K clusters to be obtained. The initialization stage is followed by the iteration stage, which is divided in two steps: the *assignment step* and the *update step*. In the first step, each element of the dataset is assigned to the cluster whose centroid (average value) is closer to the considered element. In the second step, the cluster centroids are re-calculated according to the recent cluster assignments of the data points. These two steps are iterated until the new centroids after cluster assignment do not vary more than a certain threshold with respect to the previous ones or until the convergence of a certain criterion function is reached. Typically, this function is represented by the sum of the squared errors, defined formally in the next page.

$$SSE = \sum_{k=1}^{K} \sum_{Y_i \in C_k} ||Y_i - \widehat{\theta_k}||^2 \tag{2.1}$$

In equation (2.1), Y_i denotes a time series belonging to cluster C_k and $\widehat{\theta_k}$ denotes the centroid of cluster C_k. As it is possible to observe, the SSE is computed as the sum, for all clusters, of the square of the difference between the element of a cluster and its centroid. The iteration stage of the clustering process is performed until the sum of squared errors becomes lower than a defined threshold. This criterion leads to compact clusters, as much as possible distant between each other.

The above explained algorithm can be summarized in five steps as follows:

1. Initialize the number of clusters K
2. Randomly select K datapoints as clusters centroids
3. Assign each point to the cluster with closer centroid to the considered point
4. Recalculate the centroids based on the recent cluster assignment of data points
5. Repeat steps 3 and 4 until centroids no longer vary or the convergence of a certain criterion function is reached.

K-means clustering is relatively scalable, very efficient in clustering big datasets due to its quite low computational complexity and generally reaches a local optimum. However, it cannot be applied if the average value of the data cannot be defined, for instance in case of presence of categorical variables. Moreover, the fact that the number of clusters is a required input of the algorithm could be seen as a disadvantage. Eventually, this algorithm is sensitive to noise and outliers, since a small number of these data can sensibly affect the average values. Therefore, typically k-means is accompanied by an outlier detection algorithm, a quite young but well-known topic in statistics. Figure 2.1 [42] in the next page shows the algorithm flowchart, while figure 2.2 explains graphically its working principle.

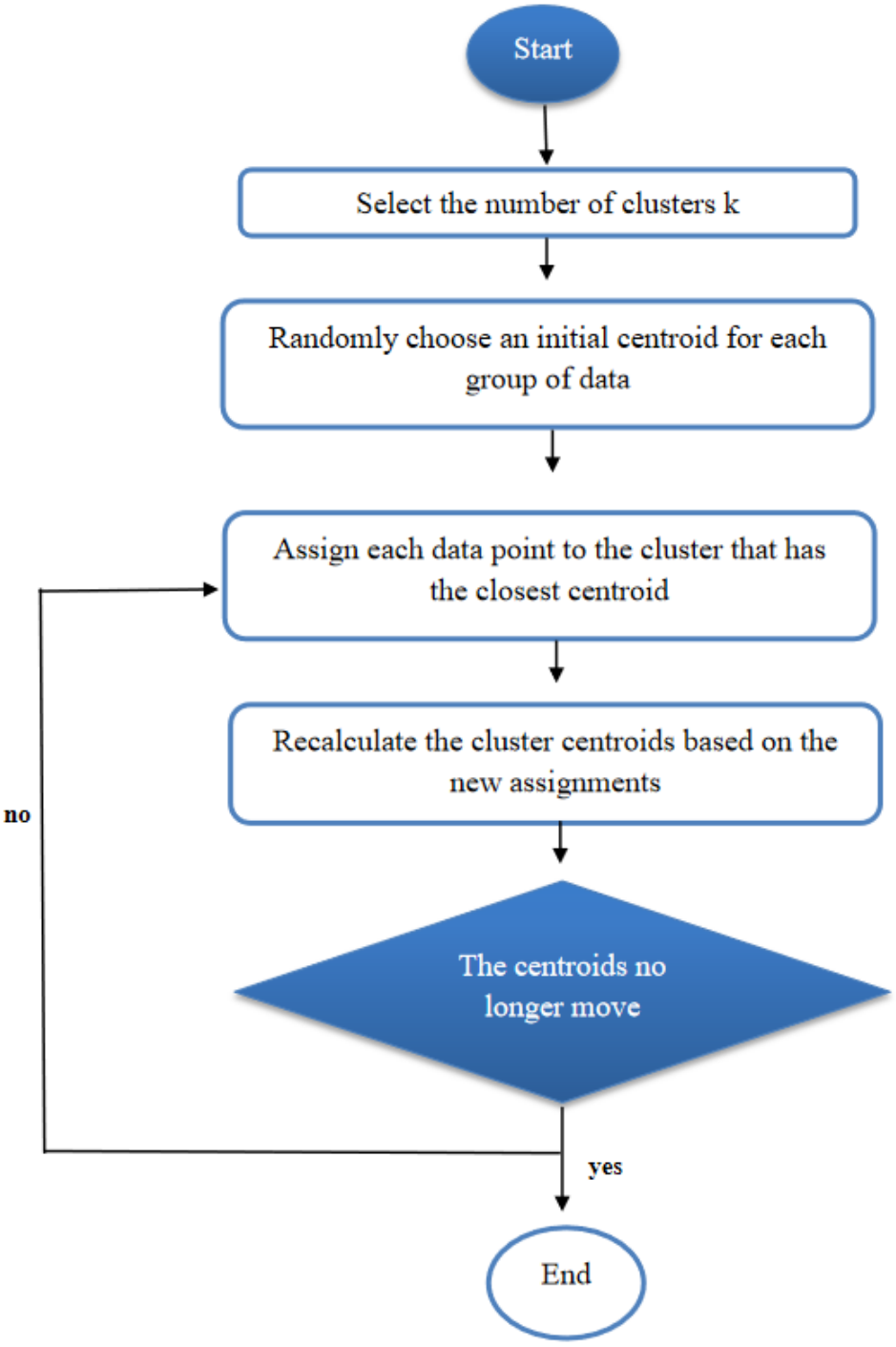


Figure 2.1: k-means algorithm flowchart

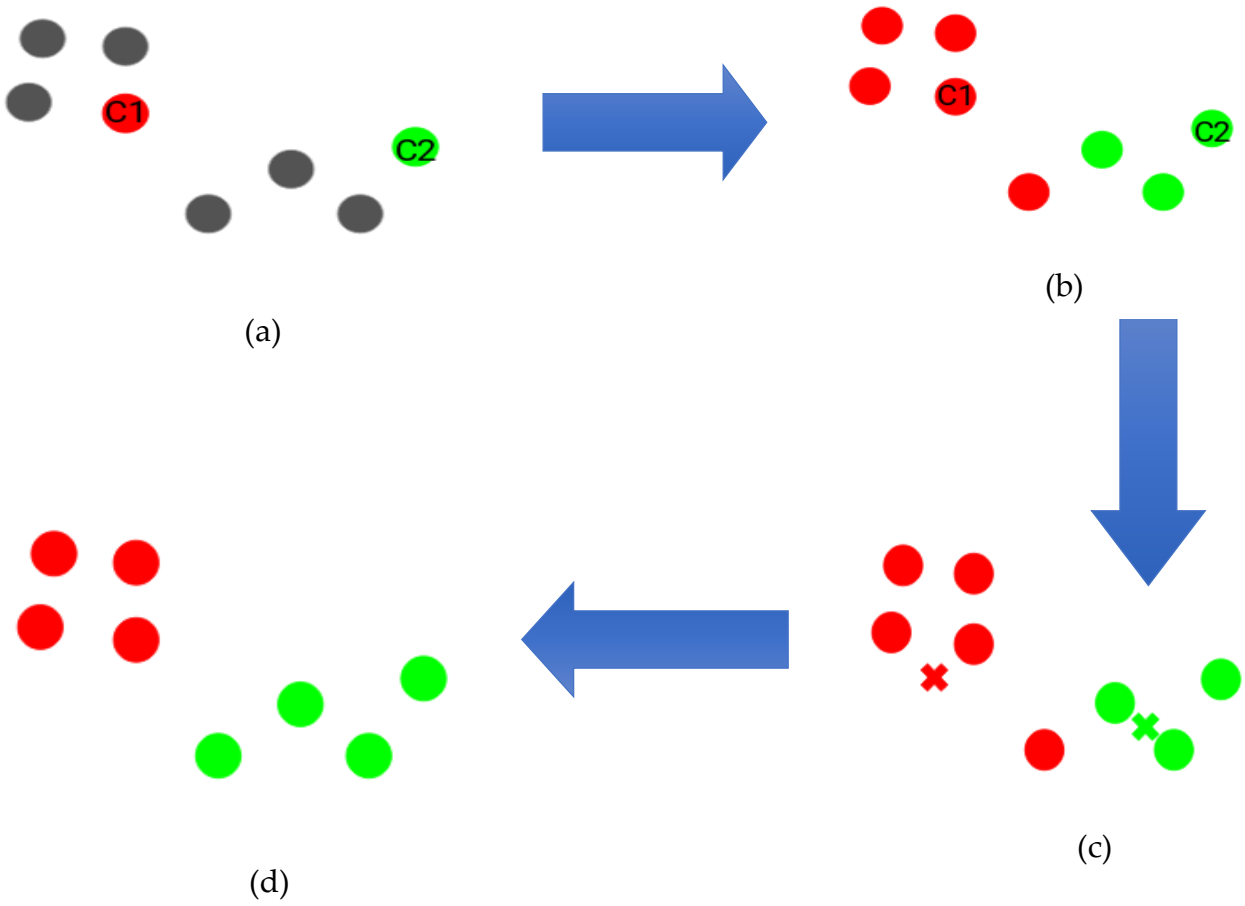


Figure 2.2: illustrative example of k-means clustering algorithm

Figure 2.2 shows graphically the first iteration of k-means algorithm, for a desired number of clusters $K = 2$. Indeed, as it is possible to observe from (a), first two elements of the dataset are randomly selected as centroids C1 and C2 of the two clusters. Then, each element of the dataset is assigned to the cluster whose centroid has the minimum distance with respect to the element itself (b). Therefore, two clusters are formed (represented in red and green color) and their centroid are the initially selected one. Successively, new centroids are computed as the average value of the data within a cluster; these centroids are represented as a red or green cross in (c). Then, data are assigned again to the cluster with minimum data-centroid distance. As it is possible to note from (d), the lower element of the dataset moves from cluster 2 to cluster 1 after the reassignment. This procedure is iteratively repeated until cluster centroids do not move anymore, or until the sum of squared errors is lower than a certain threshold.

There are several variants of the k-means algorithm, differing from the original one in the selection of the *K* initial means, in the computation of the similarity measure or in the strategy for the computation of the cluster mean. An interesting approach, which typically gives good results, consists in applying first a hierarchical agglomerative algorithm to determine the optimal number of clusters and an initial data classification and then improving it by applying the iteration stage of the k-means algorithm. Moreover, another variant is the *k-modes* algorithm, that extends the clustering method also to categorical data, by replacing the cluster mean with its mode, adopting new similarity measures that consider categorical data and employing frequency-based methods to update the modes. K-means and k-modes algorithms can be integrated to cluster data with mixed values (categorical, numerical, binary); the resulting algorithm is called *k-prototypes*.

As already discussed, k-means is a quite scalable clustering algorithm. However, to conclude the analysis, it is convenient to discuss how the scalability can be improved. A common approach to address this problem is to identify three classes in the data region:

- Discardable data: data with a certain belonging to a cluster
- Compressible data: data that cannot be discarded but belong to a reduced sub-cluster. A data structure called *clustering feature* is used to summarize the objects that have been discarded or compressed.
- Main memory data: data that cannot be discarded nor compressed and must be kept in the main memory.

To improve the scalability, the iteration stage of the algorithm considers only the features related to main memory and compressible data. In this way, good results are obtained for small and large datasets and a very important amount of data can be processed. Another approach to obtain the same improvement consists in creating micro-clusters by clustering near objects in the dataset and, successively, applying k-means algorithm to the micro-clusters.

2.2 K-medoids clustering algorithm

K-medoids algorithm has been introduced to address the issues of k-means. Indeed, it is less sensitive to noise and outliers, resulting in a more robust algorithm [41]. More precisely, instead of using the mean as the centroid of a cluster, k-medoids selects an actual point in the cluster to represent it. Typically, the medoid is the most centrally located object of the cluster, as it will be further defined. This choice for the computation of the centroids allows to greatly reduce the sensitivity of the algorithm to outliers, since their presence affect significantly only the mean and not the medoid of the cluster, as it does not modify crucially the data distribution. However, the computational complexity of k-medoids is higher than k-means complexity, since determining the value of the median is more computationally expensive than determining the value of the mean. More precisely, if k-means algorithm has a computational complexity in the order of $O(N \times K \times i)$, k-medoids algorithm has a computational complexity in the order of $O((N-K)^2 \times K \times i)$. To reduce the complexity, typically the criterion function that is used to stop the iterative stage is not the sum of squared errors, but instead the *sum of absolute errors*, defined as follows:

$$SAE = \sum_{k=1}^{K} \sum_{Y_i \in C_k} ||Y_i - \widehat{\theta_k}|| \tag{2.2}$$

As it is possible to note from equation (2.2), the difference between an element of a cluster and its centroid is no more squared, thus reducing the necessary number of operations and the computational time. The working principle of the k-medoids algorithm is similar to the one of k-means algorithm, with several differences that will now be examined. It is possible to summarize the main performed steps as follows:

1. Initialize the number of clusters K
2. Randomly select K datapoints as clusters centroids
3. Assign each data point to its nearest cluster by minimizing the sum of dissimilarities between each point and its medoid.

4. Recalculate the medoids of each cluster by determining the object that decreases its average dissimilarity coefficient (cost function).

5. If there is no change in the medoids the algorithm stops. If medoids still change then steps 3 and 4 must be repeated until medoids does not vary anymore or until the convergence of a criterion function is reached.

The base strategy of the k-medoids algorithm is to partition the N object into K clusters. As for k-means, K is an input parameter and must be defined a priori by the user. As the previously presented steps highlight, K objects are first randomly selected as medoids of the clusters. Then, each remaining object is inserted in the cluster related to the most similar medoid to the object itself. The similarity is typically computed through Euclidean distance or dynamic time warping measures. Successively, the algorithm iteratively substitutes medoids o_j with all the non-medoids objects o_{random}, stopping when the clustering quality cannot be improved anymore. This quality is estimated through a cost function that measures the average dissimilarity of an object and the medoid of its cluster. More precisely, to determine if a non-medoid object o_{random} is a good substitute of the current medoid o_j it is necessary to examine the four following cases, for each of the other non-medoid objects p:

1. p currently belongs to the cluster related to medoid o_j. If o_j is substituted by o_{random} and p is closer to another medoid $o_i, i \neq j$, then p is assigned to the cluster related to medoid o_i.

2. p currently belongs to the cluster related to medoid o_j. If o_j is substituted by o_{random} and p is closer to o_{random}, then p is assigned to the cluster related to medoid o_{random}.

3. p currently belongs to the cluster related to medoid $o_i, i \neq j$. If o_j is substituted by o_{random} and p is still closer to o_i, then p is still assigned to the cluster related to medoid o_i (no changes).

4. p currently belongs to the cluster related to medoid $o_i, i \neq j$. If o_j is substituted by o_{random} and p is closer to o_{random}, then p is assigned to the cluster related to medoid o_{random}.

Every time that a reassignment of data occurs, the cost function varies from its previous value; this variation can be positive or negative. The sum of all the variations of the cost functions obtained after the reassignment of all the non-medoids objects according to the previous cases is called *total swapping cost*. If the total swapping cost is negative, o_j is substituted with o_{random} since the total error is reduced. Instead, if the total swapping cost is positive, the current medoid o_j is considered acceptable and is not substituted. The examined procedure is repeated until medoids do not vary anymore or until the sum of absolute errors is lower than a defined threshold. Figure 2.3 below [42] shows the flowchart of k-medoid algorithm, while figure 2.4 in the next page shows graphically the four cases presented in page 75.

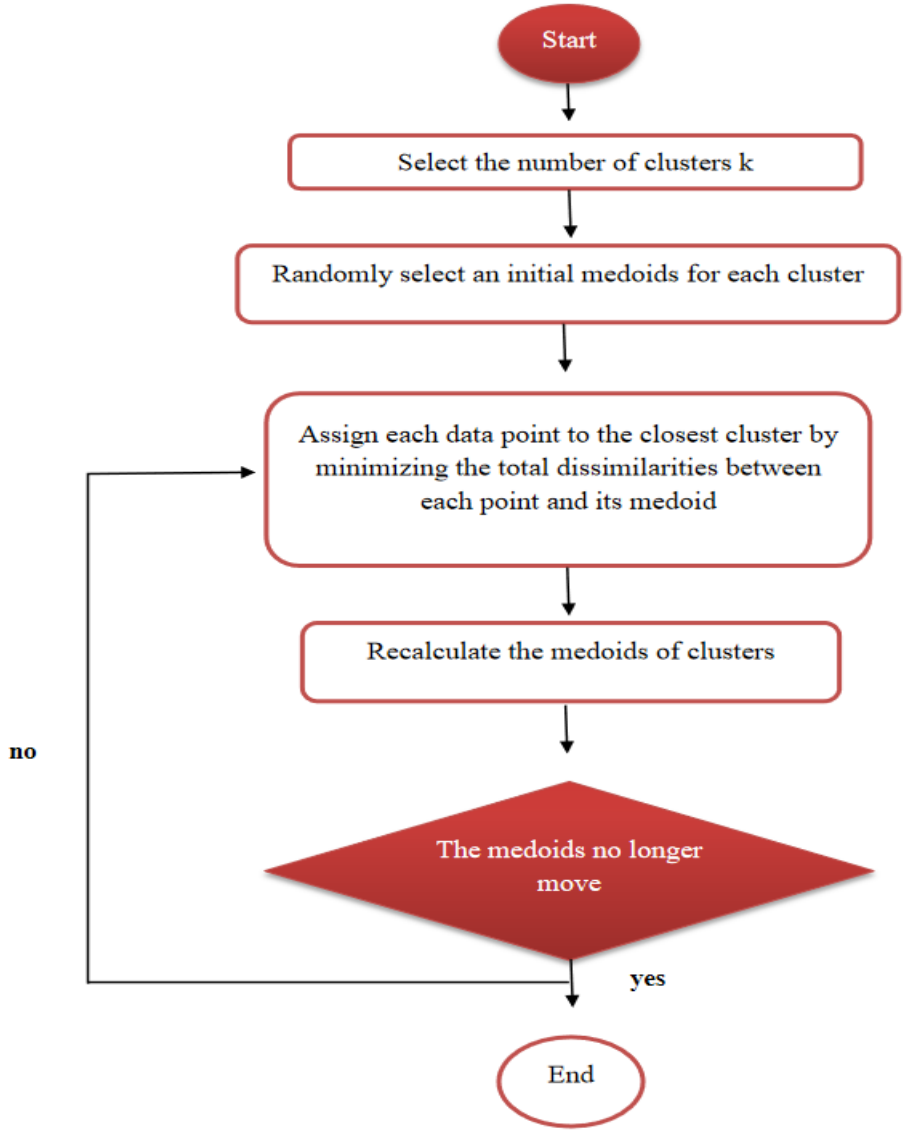


Figure 2.3: k-medoids algorithm flowchart

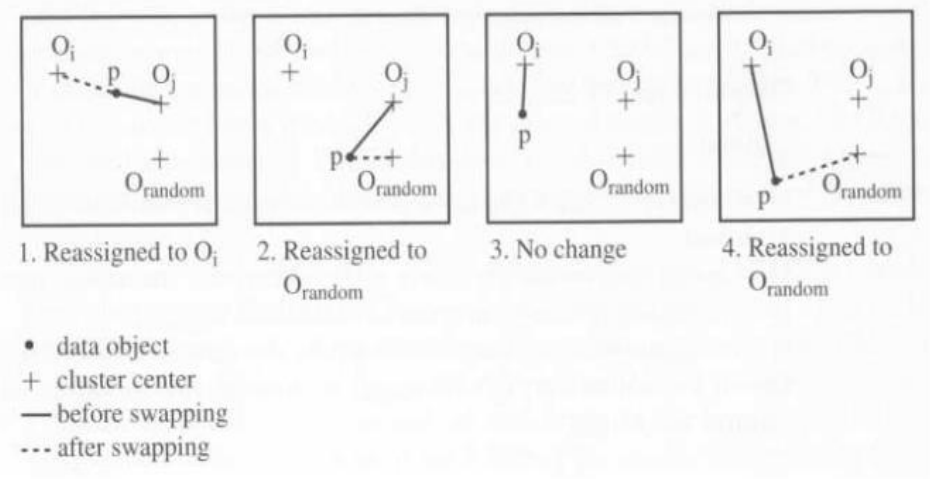


Figure 2.4: the four possible cases of the cost function for the k-medoids algorithm

As it is possible to note looking at figure 2.4, the four cases are enumerated in the exact same way in which have been presented (page 75). In particular, case 1 represents the reassignment of p to the clusters related to the medoid o_i. Indeed, before substituting o_j with o_{random} p was closer to o_j, while after the swapping p is closer to o_i, thus is assigned to its related cluster and the cost function varies. Instead, in case 2 after the swapping of o_j with o_{random}, p is closer to o_{random} and therefore is assigned to its related cluster and the cost function varies. Case 3 represents a situation in which no changes occur on the assignment of point p. Indeed, before and after the swapping of o_j with o_{random}, p is closer to o_i so its belonging to the cluster represented by medoid o_i is unchanged and the cost function does not vary. Eventually, in case 4 after the swapping p is closer to o_{random}, thus is assigned to its related cluster and the cost function varies.

To conclude the analysis of k-medoids algorithm, as done for k-means, a method to improve the scalability is now discussed. This method is based on sampling, and the algorithm that performs it is called *CLARA (Clustering LARge Applications).* The concept behind CLARA is the following: instead of considering the entire dataset, a small portion (sample) is selected and considered representative of all data., and the medoid is computed through a traditional k-medoids algorithm only on this sample. If more than one samples are collected randomly, they should represent the whole dataset quite well and the determined medoids should be similar to the ones that would have been obtained by applying a traditional k-medoids algorithm on the complete data. Therefore, CLARA algorithm creates different samples of the dataset, applies a traditional k-medoids algorithm on the various samples and selects the best clustering result as output. This allows to perform a medoid-based partitional

clustering process also on very big datasets, thus improving the scalability of the method. However, this increases the computational complexity, that becomes in the order of $O(Ks^2 + K(N - K))$, denoting with s the sample dimension, with K the desired number of clusters and with N the total number of objects.

2.3 Ward hierarchical algorithm

Before introducing the main steps of this algorithm, it is convenient to examine a little bit more than in the previous chapter the hierarchical clustering algorithms. The hierarchical approach to clustering works by grouping the objects into the so-called *cluster trees*. Moreover, as already discussed in chapter 1, hierarchical clustering algorithms can be classified into:

- Agglomerative algorithms: bottom-up strategy that starts by considering each object as a separate cluster and then merges these atomic clusters until some determined conditions are satisfied. The various algorithms inside this category differ just for the computation of the similarity measure between clusters (inter-cluster similarity).

- Divisive algorithms: top-down strategy operating in the inverse way of agglomerative algorithms. It starts by assigning all the objects to the same cluster and then divides this cluster into smaller and smaller portions, until some determined conditions are satisfied.

The conditions to be satisfied to stop the algorithm can be various; for instance, a desired number of clusters can be required. However, one of the main advantages of hierarchical clustering algorithms is that the necessary number of clusters can be automatically obtained by imposing different conditions, related to the distance between clusters or to some quality parameters, to stop the algorithm. For instance, a minimum/maximum distance between clusters in the divisive/agglomerative approach respectively or a threshold for a quality parameter can be defined as stopping conditions. Figure 2.5 in the next page shows one of the most important graphical representations for hierarchical clustering algorithm: the so-called *cluster tree* or, more commonly, *dendrogram*.

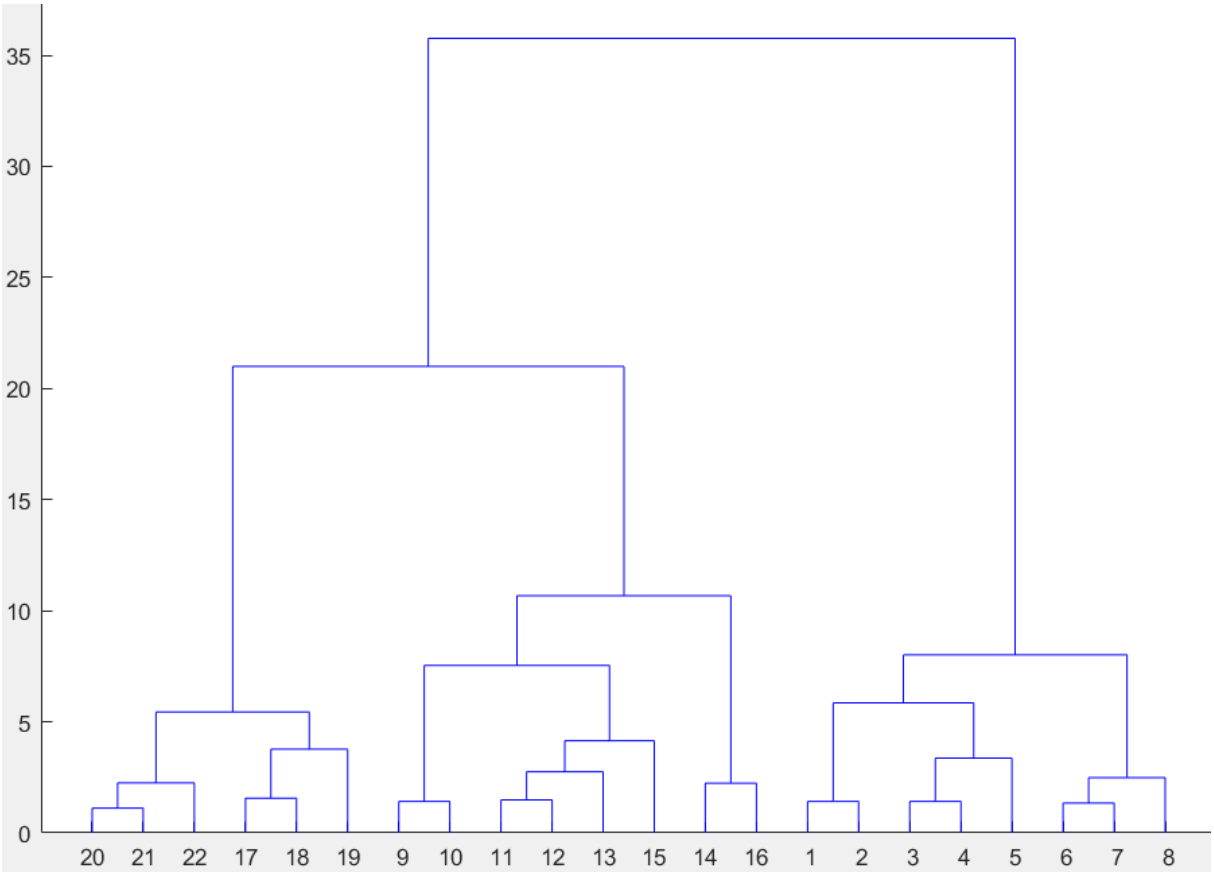


Figure 2.5: example of cluster dendrogram

As it is possible to observe from figure 2.5, the dendrogram is related to an agglomerative clustering algorithm of 22 objects. Indeed, the different levels of the graph represent the merging process performed by the algorithm, starting from separated clusters composed by a single object (bottom level or level 0) and (in this case) stopping when the desired number of clusters has been obtained (top level). For instance, let's analyze the transition from level 0 to level 1 and from level 1 to level 2: figure 2.5 provides a simple way to understand which clusters have been merged and why. Indeed, starting from 22 separated atomic clusters, the algorithm merges together clusters 20 and 21, clusters 17 and 18, cluster 9 and 10 and so on. The objects order in the level 0 of dendrogram reflects the fact that the clusters that are merged are the ones with lower inter-cluster distance. For instance, cluster 14 is closer to cluster 16 than cluster 15, thus objects 14 and 16 are merged in the first transition. Successively, in the second merging step, the cluster formed by objects 20 and 21 is merged with the cluster formed by object 22, as it is possible to see from level 1 of the dendrogram. The process iteratively proceeds until a desired number of clusters is reached or until some conditions are satisfied. Finally, the y-axis of the dendrogram reports the distances between two clusters that have been merged. This distance can

be computed in different ways; the most typical are shown below. For the sake of clearness, it is recalled that p and p' are two generic objects belonging to different clusters C_i and C_j, $mean_{C_i}$ is the mean value of the cluster C_i and N_i denotes its number of objects.

- Minimum distance (single linkage): $d_{min}(C_i, C_j) = \min_{p \in C_i, p' \in C_j}\{|p - p'|\}$

(2.3)

- Maximum distance (complete linkage): $d_{max}(C_i, C_j) = \max_{p \in C_i, p' \in C_j}\{|p - p'|\}$

(2.4)

- Mean distance (centroid linkage): $d_{mean}(C_i, C_j) = |mean_{C_i} - mean_{C_j}|$

(2.5)

- Average distance (average linkage): $d_{average}(C_i, C_j) = \frac{1}{N_i N_j}\sum_{p \in C_i}\sum_{p' \in C_j}|p - p'|$

(2.6)

Actually, Ward's hierarchical algorithm exploits another distance function, called *Ward distance* [43] and defined as:

$$d_{Ward}(C_i, C_j) = \frac{N_i N_j}{N_i + N_j}\sum_{p \in C_i}\sum_{p' \in C_j}|p - p'|^2$$

(2.7)

As it is possible to note from equation (2.7), Ward distance is a variant of the average distance, where the scaling coefficient is modified and the distance between objects is squared rather than linear. Moreover, there is another simple way to compute this distance, based on the sum of squared errors presented in equation (1.73):

$$d_{Ward}(C_i, C_j) = SSE(C_i \cup C_j) - [SSE(C_i) + SSE(C_j)]$$

(2.8)

In equation (2.8), $C_i \cup C_j$ represents the combined cluster obtained by merging clusters C_i and C_j. This strategy to compute the distance is fast and simple and can be implemented iteratively with optimal results.

Hierarchical clustering algorithms have many advantages such as the automatic definition of the necessary number of clusters and the capability to threat noisy data. However, several issues can be defined. Indeed, these methods are crucially based on the definition of the merging/splitting points. This selection is critical since once that a group of objects is merged or split, the algorithm will operate on the new clusters at the next iteration, without any possibility to come back and restore the previous objects. Therefore, since hierarchical clustering algorithms never cancel what have been done and never reassign objects from a cluster to another, if the merging/splitting decision is wrong or bad low-quality clusters are obtained. Moreover, these methods are not very scalable, since any decision for merging/splitting objects requires an evaluation of a quite high number of objects. Thus, typically in big datasets partitional clustering techniques are preferred., or the hierarchical clustering method is integrated with other strategies to improve the quality of the clustering result.

To conclude the analysis of hierarchical clustering algorithms, it is now convenient to introduce formally Ward's hierarchical algorithm. More precisely, the main steps to be performed are:

1. Set the initial number of clusters n equal to the total number of objects N.
2. Determine the centroid of each cluster as the mean value of the cluster itself.
3. Compute the distance between each pair of clusters according to Ward's method.
4. Merge the two closest clusters.
5. Update $n = n - 1$
6. If a predefined condition is satisfied (for instance, the number of clusters is equal to a desired value) go to step 7, otherwise go to step 2.
7. Determine the centroid of each cluster as the cluster medoid.

The previous steps can be graphically summarized by the following example:

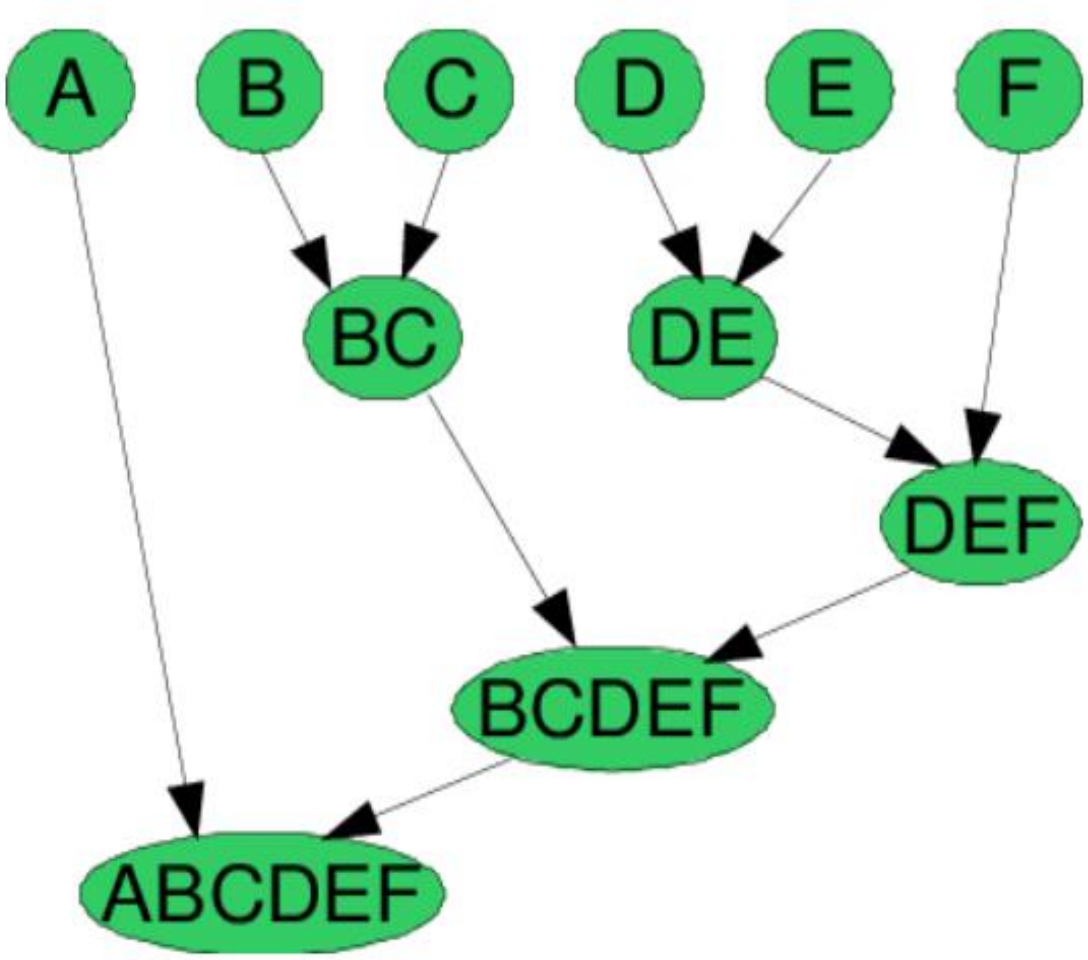


Figure 2.6: agglomerative hierarchical clustering example

As it is possible to observe from figure 2.6, in the initial step the distance between atomic clusters formed by a single object is computed. Then, similar clusters are merged to form a new cluster. For instance, in the considered example, B,C and D,E are similar clusters thus they are merged and only four clusters remain: A, BC, DE, F. Successively, the distance between these four clusters is computed again and similar clusters are merged. Figure 2.6 shows that DE and F are merged thus only three clusters remain: A, BC, DEF. This process iteratively proceeds until a single cluster containing all the objects is obtained, as in figure 2.6, or until a certain condition is satisfied. As already discussed, this process can be visualized by a dendrogram or by a Euler-Venn diagram. Both these graphs are shown in the next page; the dendrogram does not show any value on the y-axis since no real data have been used for this example.

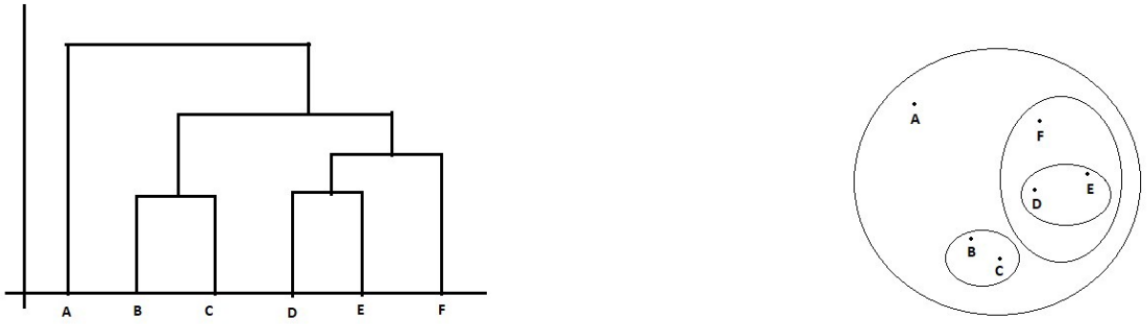


Figure 2.7: dendrogram and Euler-Venn diagram of example reported in figure 2.6

For the sake of clearness, it is recalled that the arrangement of the clades (branches of the dendrogram) is based on how similar (or dissimilar) clusters are. Clades with same or close height are similar to each other; whereas clades with different heights are dissimilar from each other. This implies that the higher is the value of the difference in height between clades, the more dissimilar will be the related clusters.

Ward's hierarchical method has a computational complexity in the order of $O(N^2 i)$. As it is possible to note, the parameter K doesn't appear in this expression since typically the number of clusters is not an input of the algorithm, but instead is an output related to satisfying certain conditions. Some examples will be discussed in detail in the next chapter. As already discussed, the complexity of hierarchical algorithms is higher than the one of partitional algorithms. This implies that, even if higher-quality clusters are obtained, these methods have a lower scalability and can require an important amount of time when applied to big datasets.

2.4 Optimal number of clusters and clustering evaluation

This last section focuses on determining the optimal number of clusters. This is a very important task, especially when there is no prior knowledge on the data. Many different methods have been proposed in literature; the most commonly employed are the so-called *elbow method* [44], *average silhouette method* [45] and *gap statistical method* [46]. In the next pages, these methods will be analyzed briefly. Successively, some parameters to indicate the quality of the clustering result will be discussed. These parameters will be employed in the next chapter to examine the goodness of the proposed approach to an electrical engineering problem.

The elbow method is a very simple strategy that is based on an iterative process. In particular, the sum of squared errors is used as an indicator to detect the optimal number of clusters. Indeed, a clustering algorithm (typically partitional, to reduce the computational complexity) is iteratively performed considering a required number of clusters ranging from 1 to +infinite, and for every clustering the sum of squared errors is computed. Then, a graph of the SSE vs the number of clusters is plotted. This graph will show a strong increase in the first part, since increasing the number of clusters will add intra-cluster variance, and then after an elbow the increase is strongly reduced, meaning that increasing the number of clusters does not affect any more significantly the sum of squared errors. Therefore, the elbow point of the graph is selected as optimal and the related number of clusters K is the optimal number of clusters.

The average silhouette method is an alternative strategy to the elbow method and follows a similar process. The average silhouette S is a parameter defined as the mean of the average point silhouette S_i, computed for every object i in the clustered dataset. This parameter characterizes the overall quality of the partition in clusters, thus it is used also for clustering evaluation, as it will be discussed further. More precisely, it is defined as follows:

$$S = mean(S_i), \qquad S_i = \frac{d_{intra,i} - d_{inter,i}}{\max\{d_{intra,i}, d_{inter,i}\}} \tag{2.9}$$

In equation (2.9), $d_{intra,i}$ represents the average distance between object i and other data points in the same cluster, while $d_{inter,i}$ represents the average distance between object i and all the data points in the nearest neighboring cluster. The silhouette index is computed for every object in the dataset after the clustering process, and its value is a good indicator of the clustering quality. Indeed, the accuracy of the result is high when S_i has a value close to one, while is low when S_i has a value close to zero. Therefore, a clustering algorithm considering a required number of clusters ranging from one to +infinite is iteratively performed on the dataset and for every clustering result the average silhouette parameter is computed. The model in which this parameter has the highest value (close to one) is selected as optimal and the related number of clusters K is the optimal number of clusters.

The gap statistic method is based on comparing the total within-cluster variation with its expectation for a different number of clusters. Formally, it can be defined as:

$$\widehat{K} = \min_K \{K | G(K) \geq G(K+1) - s'_{K+1}\} \tag{2.10}$$

Since equation (2.10) can result quite difficult to understand, it is convenient to explain it better. The optimal number of clusters $\widehat{K}$ is obtained through a minimization problem that considers the result of clustering algorithm performing with different values of K, ranging from one to +infinite. Moreover, for a given number of clusters K, two distributions are considered: the dataset distribution $W_{k,data}$ and a reference uniform distribution $W_{k,unif}$ composed of 20 points and used as a reference cluster. The different terms appearing in equation (2.10) are computed as:

$$\text{Gap function: } G(K) = \ln\left(W_{k,unif}\right) - \ln\left(W_{k,data}\right) \tag{2.11}$$

$$s_{K+1} = s_K \sqrt{1 + \frac{1}{20}} \tag{2.12}$$

$$s_K = std.dev.of\ \ln\left(W_{k,unif}\right) \tag{2.13}$$

As equations (2.13) and (2.12) shows, the initial value of s_K is the standard deviation of a uniform distribution of 20 points. Then, increasing the number of clusters, this deviation is increased by $\sqrt{1 + \frac{1}{20}}$; this means that is continuously higher for each performed step. The basic idea behind the Gap Statistics technique is to choose the number of K for which the biggest variation in within-cluster distance occurred, based on the overall behavior of uniformly drawn samples. However, it could happen that only a very slight reduction in within-cluster distance occurs. For this reason, s_{K+1} acts as a threshold, to sort out too small changes and to remove the sampling noise from the data. Only if the change is so big that the threshold s_{K+1} plays no role anymore, the optimal value of K will be selected. In practice, the optimal number of clusters is selected such that, increasing K, the gap function starts

decreasing and the maximum variation of within cluster distance is obtained. Figure 2.8 below shows an example of gap function behavior for different K.

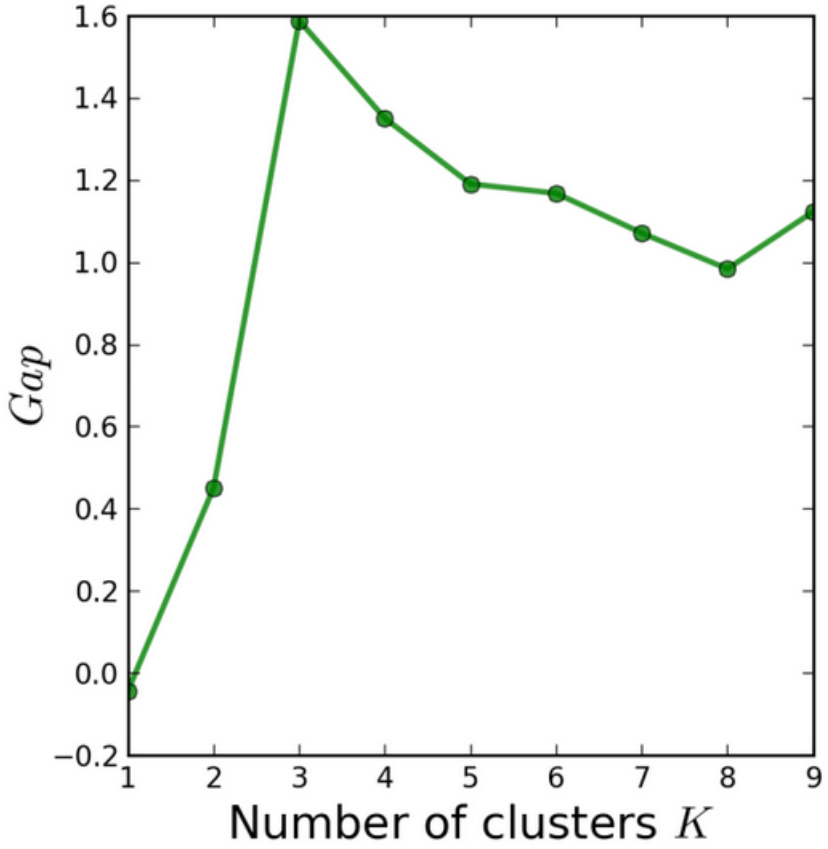


Figure 2.8: example of gap function behavior for different K

As it is possible to observe from figure 2.8, the optimal number of clusters for the example case is K=3. Indeed, the gap function increases for $1 \leq K \leq 3$ and starts decreasing for $K \geq 3$. Thus, the minimum value of K for which equation (2.10) holds is exactly 3.

After having analyzed the techniques to determine the optimal number of clusters, to conclude this chapter, it is convenient to briefly examine the parameters for the clustering evaluation. In particular, two main clustering quality indexes can be defined: the *Dunn index* [47] and the *Davies-Bouldin index* [48].

The Dunn index, introduced by Dunn in 1974 [47], determines the quality of the clustering result by detecting if clusters are well-separated between each other. More precisely, it is defined as follows:

$$Q_D = \min\{separation\}/\max\{diameter\} \tag{2.14}$$

In equation (2.14), $separation$ refers to the inter-cluster distance, while $diameter$ refers to intra-cluster distance. Thus, the Dunn quality index is determined as the ratio between the minimum inter-cluster distance (distance between data points in different clusters) and the maximum intra-cluster distance (distance between data points in the same cluster). The value of this parameter is high if clusters are well-separated, meaning that the clustering process has been efficient. Typically, Dunn index is used to estimate the quality of partitional clustering algorithms such as k-means or k-medoids, already explained in the previous sections.

The Davies-Bouldin index, introduced by Davies and Bouldin in 1979 [48], is a sort of inverse of the Dunn index, anyway computed with different formulas. More precisely, it is defined as the average value of the ratios of within-cluster distances and between clusters distances and is computed as follows:

$$Q_{DB} = \frac{1}{K}\sum_{k=1}^{K} B_i \tag{2.15}$$

In equation (2.15), K represents the total number of clusters while B_i is a parameter related to the data point y_i, defined as:

$$B_i = \max\left\{\frac{var(y_i) + var(y_j)}{|\widehat{y_i} - \widehat{y_j}|}\right\}, \forall j \neq i \in \{1, \dots, N\} \tag{2.16}$$

For the sake of clearness, it is recalled that:

$$var(y_i) = \frac{1}{N_i - 1}\sum_{i=1}^{N_i}(y_i - \widehat{y_i})^2 \tag{2.17}$$

where N_i is the number of data points in cluster C_i and $\hat{y}_i$ is the centroid related to the cluster C_i, to which y_i belongs.

As it is possible to note from equation (2.15) and (2.16), the Davies-Bouldin index is computed as the average value on all clusters of the parameter B_i, which is defined as the maximum of the sum between the intra-cluster distance of cluster C_i and the intra-cluster distance of all the other clusters C_j, divided by the distance between their centroids $\hat{y}_i$ and $\hat{y}_j$. (average inter-cluster distance) The lower is the value of the Davies-Bouldin index, the higher is the quality of the clustering result. Indeed, if clusters are well-separated, the inter-cluster distance, represented by the distance between centroids (denominator) is high, while the intra-cluster distance, represented by the variance (numerator), is low. Figure 2.9 below shows an example of computation of this quality index, denoted as $\bar{R}$ instead than Q_{DB} [48].

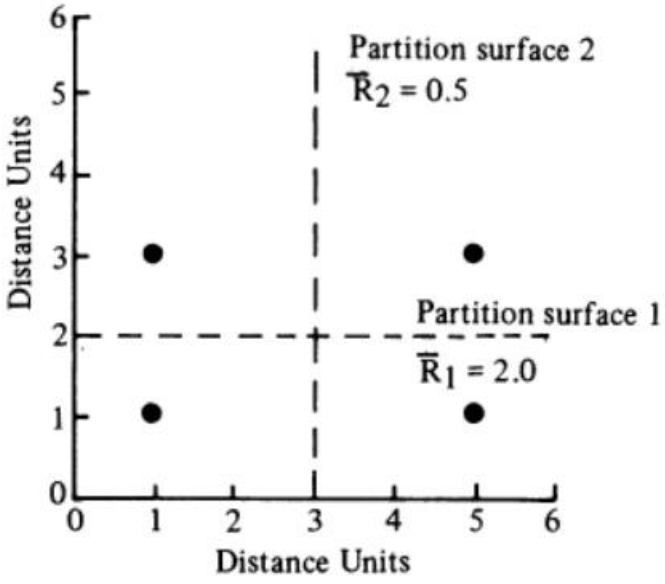


Figure 2.9: comparison of partitions of a 4-points dataset and use of Davies-Bouldin index to compare their quality

The 4-points data set represented in figure 2.9 has been partitioned through the k-means algorithm. As it is possible to observe, two partitions are obtained, depending on the selected initial centroids. More precisely, if points (1,1) and (1,3) are chosen as initial centroids, the algorithm produces the partition indicated by surface 1, with clusters centers at (3,1) and (3,3). Instead, if points (1,1) and (5,1) are chosen as initial centroids, the algorithm produces the partition indicated by surface 2, with clusters centers at (1,2) and (5,2). Computing the Davis-Bouldin index for the

two partitions, it is possible to obtain: $\overline{R_1} = 2.0$ and $\overline{R_2} = 0.5$. Since $\overline{R_2} < \overline{R_1}$, the second partition has a higher quality, and its clusters are better separated. It is important to remark that a data set must be partitioned into at least two clusters with different centroid for the Davis-Bouldin index to have meaning. This is a mathematical necessity since the distance measure in the denominator of Q_{DB} must be non-zero for it to be defined Moreover, the use of this index becomes limited if clusters containing a single object are allowed, since such clusters have zero variance. If Q_{DB} is used as a parameter to estimate the clustering quality, these two limitations should be always kept in mind.

Once having analyzed the main parameters to select the optimal number of clusters and to detect the quality of clustering result, it is now possible to move to the practical implementation of clustering algorithms to an electrical engineering problem, examined in the next chapter that also provides, as well as Appendix A, the adopted codes in Matlab and R-Studio.

3. Clustering algorithms implementation: an Electrical Engineering problem.

Before analyzing the practical implementation of the clustering algorithms described in the previous chapter, it is convenient to examine the considered dataset. More precisely, firstly 365 load curves have been grouped in 36 clusters and some quality parameters have been computed to determine the goodness of the result. The same procedure can be applied to a set of generation curves without any difficulty. The centroids of the obtained clusters are extremely important curves, called *typical load/generation days* since they represent the typical behavior of the energy consumption/generation curves belonging to a certain cluster Therefore, instead of considering 365 curves as input for a generic algorithm, it is possible to exploit just the 36 determined typical days curves to obtain coherent and consistent results. Finally, the last part of this chapter focuses on the time-point clustering approach (clustering of points of a single time series based on a combination of their temporal proximity and similarity), introducing a new issue that has never been examined until now and that will be further analysed in the conclusion: the chronological ordering of clustered objects.

The considered dataset of load duration curves can be downloaded from the ENTSO-E power statistics website [49]. Once on the website, many sections are available; the set of curves under analysis can be found in "Monthly Hourly Load Values", that reports the aggregated monthly electrical power consumption of different countries on an hourly basis, from 2015 to 2019. The 365 selected load curves correspond to the electrical consumption of Germany in 2015 (one curve per day, with 24 points per curve).

For the sake of clearness, three curves of the dataset are shown in the next page:

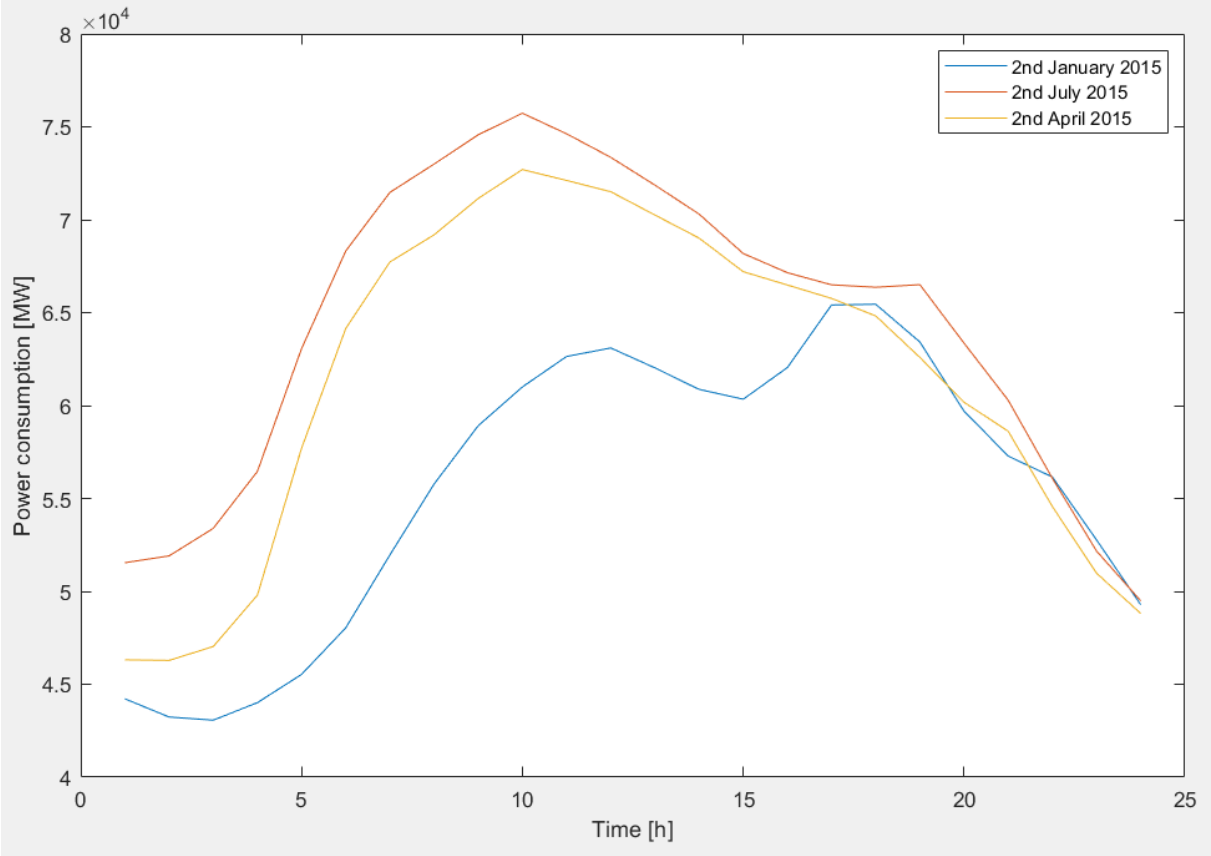


Figure 3.1: examples of the considered load duration curves

Figure 3.1 shows three days of different months. For instance, the blue curve represents the aggregated electrical power consumption of Germany on the 2nd January 2015, the yellow curve represents the aggregated power consumption of Germany on the 2nd April 2015 and the red curve represents the aggregated power consumption of Germany on 2nd July 2015. As it is possible to observe, in July the electrical energy consumption is higher than in January or April due to the usage of air conditioning systems. This is generally true for all summer months with respect to winter months, in which the power consumption is lower.

Once having examined the considered dataset, as discussed in chapter 1, the first task to be performed is a data-preparation method in order to make the clustering process less sensitive to scaling, time shifting and offsets. Several data-preparation techniques have been examined in the previous chapters. For the sake of simplicity, the z-normalization method has been selected for this practical implementation, since it allows to obtain anyway a clustering model with low sensitivity. For the sake of

clearness, the following figure shows the three load curves reported in figure 3.1 after having z-normalized the data.

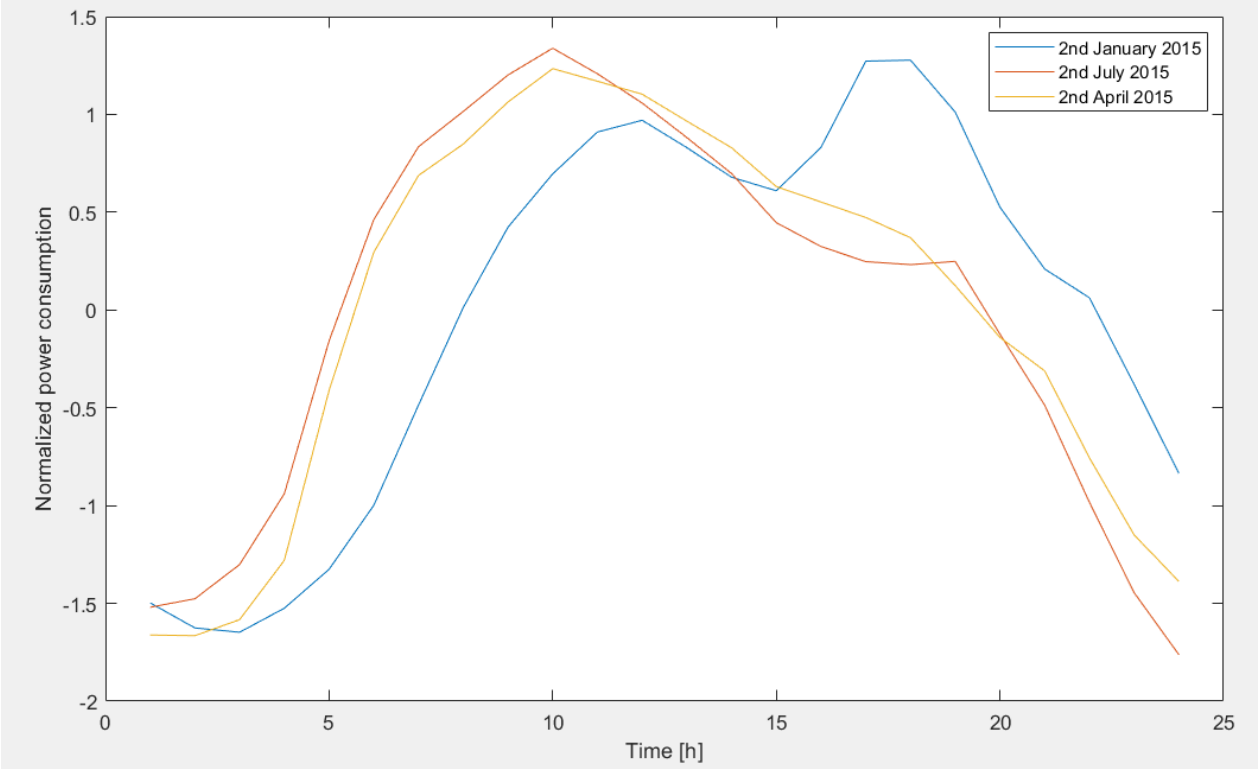


Figure 3.2: curves of figure 3.1 after z-normalization

As it is possible to note from figure 3.1 above, the power consumption values now are adimensional, since the division by the standard deviation eliminates the measurement unit. Moreover, data points now range around zero, anyway maintaining the same shape for each curve with respect to its original one.

After having normalized the dataset, it is possible to perform the clustering process to determine the typical load days. Three main clustering algorithms have been employed, as already defined in the previous chapter: k-means, k-medoids and Ward's hierarchical algorithm. Moreover, two softwares have been considered: Matlab and R-Studio. For the sake of clearness, the analysis of the implemented code and result will be conducted in separate sections.

3.1 Matlab clustering implementation

The implemented code for clustering implementation in Matlab is reported in Appendix A. The algorithms needed to perform clustering can be found in the "Statistical and Machine Learning Toolbox". As it is possible to note by looking at the code in Appendix A, the functions that implement the clustering process are:

```
[idx,c,sumd,d]=kmeans(data_norm,n_clust,'Distance',
'sqeuclidean', 'replicates',5);
```

(3.1)

```
[idx,c,sumd,d]=kmedoids(data_norm,n_clust,'Distance',
@dtwf, 'replicates',5);
```

(3.2)

These functions produce four outputs:

- idx: vector containing the cluster indexes for each of the load duration curves. For instance, if idx is equal to N in a certain position X, the Xth curve of the dataset belongs to the Nth cluster.
- c: matrix containing the centroids of the obtained clusters. Since 365 load curves with 24 points per curve are clustered to 36 curves, c will be a 36x24 matrix in which each row represents a typical day, i.e., the centroid of the related cluster.
- sumd: vector containing the sum of within-cluster series-to-centroids distances.
- d: matrix containing distances between each time series of the dataset and every centroid. Therefore, in the considered case d will be a 365x36 matrix in which the element i, j represents the distance between the load curve i of the dataset and the centroid of the cluster j.

Instead, the required input to perform are:

- data_norm: matrix containing the 365 power consumption curves. Since data points are collected hourly, data_norm is a 365x24 matrix.

- n_clust: the required number of clusters. As already discussed in the previous chapter, this is a user-defined parameter for k-means and k-medoids algorithm.

- 'Distance': definition of the similarity measure to be used. For example purpose, the Euclidean distance has been adopted for the k-means algorithm, while dynamic time warping has been adopted for k-medoids algorithm.

- 'replicates': a non-mandatory parameter which defines the number of times the algorithm must be repeated. For instance, if 'replicates' is equal to 5 as in the examined case, the software performs the clustering process five times and select the optimal result as output.

It is important to remark that Matlab does not allow to compute DTW distance between more than two curves at a time and anyway the output is always scalar. Therefore, to exploit dynamic time warping advantages, it is necessary to write a simple function that extends DTW computation also to a matrix containing a set of time series (in this case power consumption curves). This function has been called `dtwf` and the related code is reported below:

```
function dist = dtwf(x,y)
m2 = size(y,1);
dist = zeros(m2,1);
for i=1:m2
    dist(i) = dtw(x,y(i,:));
end
end
```

(3.3)

As it is possible to note from code (3.3), the `dtwf` function is based on a simple for-cycle that repeats dynamic time warping computation. Indeed, x is $1 \times N$ vector containing a single time series (one of the load curves in the dataset) while y is a $m2 \times N$ matrix containing multiple time series (all the other load curves in the dataset). The output dist is a $m2 \times 1$ vector of distances, in which dist(k) represents the distance between time series x and the kth time series in y. As an example.

figure 3.3 below shows the result of dynamic time warping between April and July power consumption curves reported in figure 3.2.

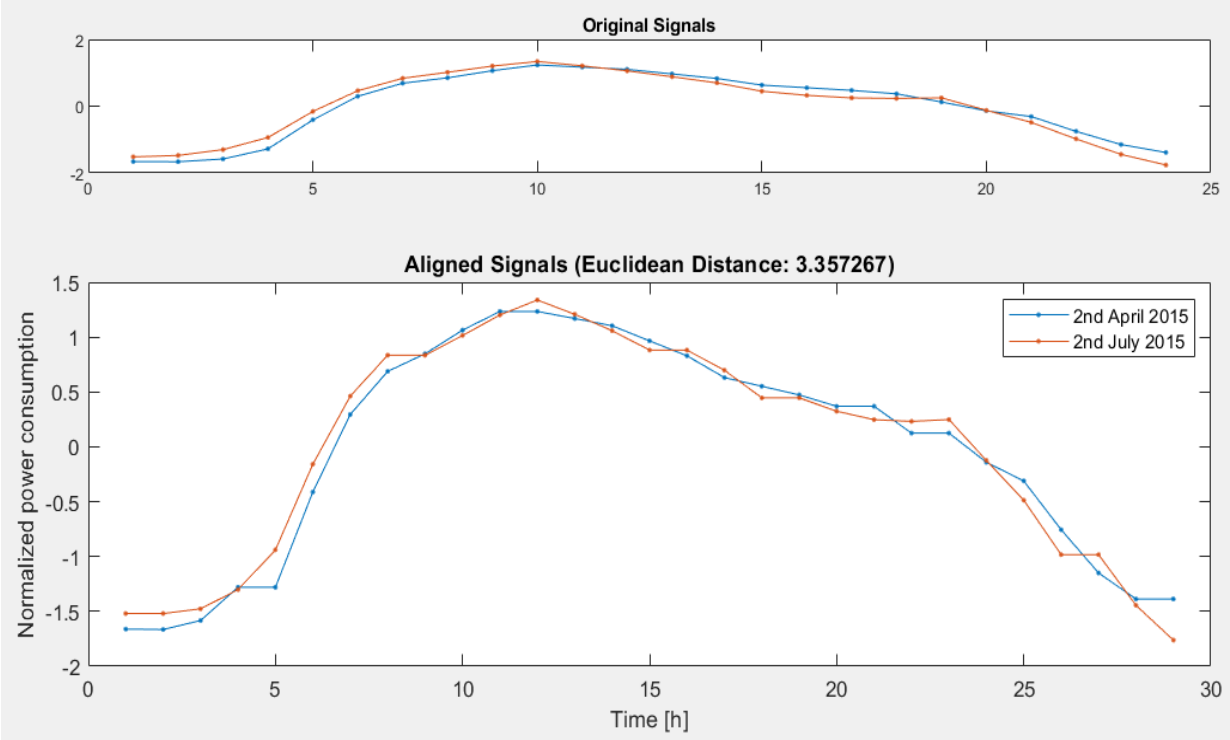


Figure 3.3: result of DTW between two of the curves reported in figure 3.2

As already discussed in detail, dynamic time warping determines the optimal alignment between the two considered time series, shown in the bottom image in figure 3.3, thus computing their minimum total distance, equal to 3.36 in this case. Though the dtw function presented in code (3.3), this procedure is repeated for all the possible pairs of time series in the considered dataset, extending DTW computation to a matrix of time series and therefore allowing to perform k-medoids clustering in Matlab without problems.

After having examined the functions needed to implement partitional clustering in Matlab, it now possible to analyze the results. In particular, by clustering through k-means algorithm and k-medoids algorithms the set of curves, 36 clusters are obtained. The figures reported in the next pages report some results of the clustering process, showing the curves belonging to a cluster and their related cluster centroid, represented by a line with a bigger width.

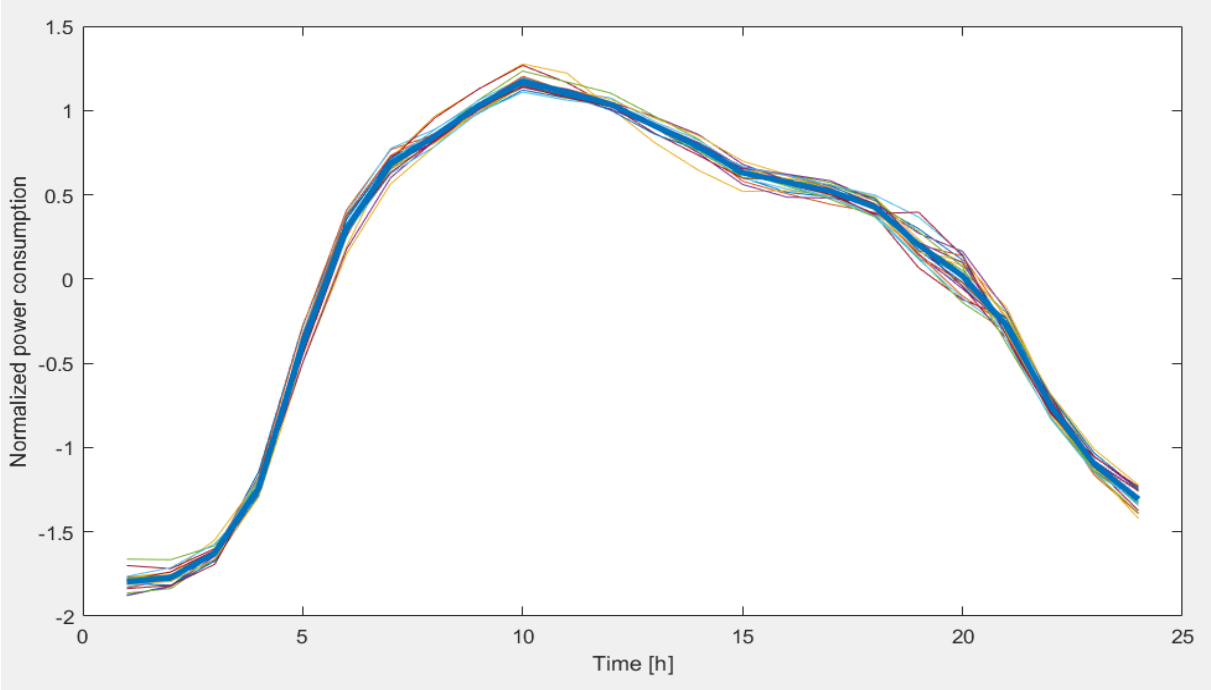


Figure 3.4: cluster1 obtained by k-means

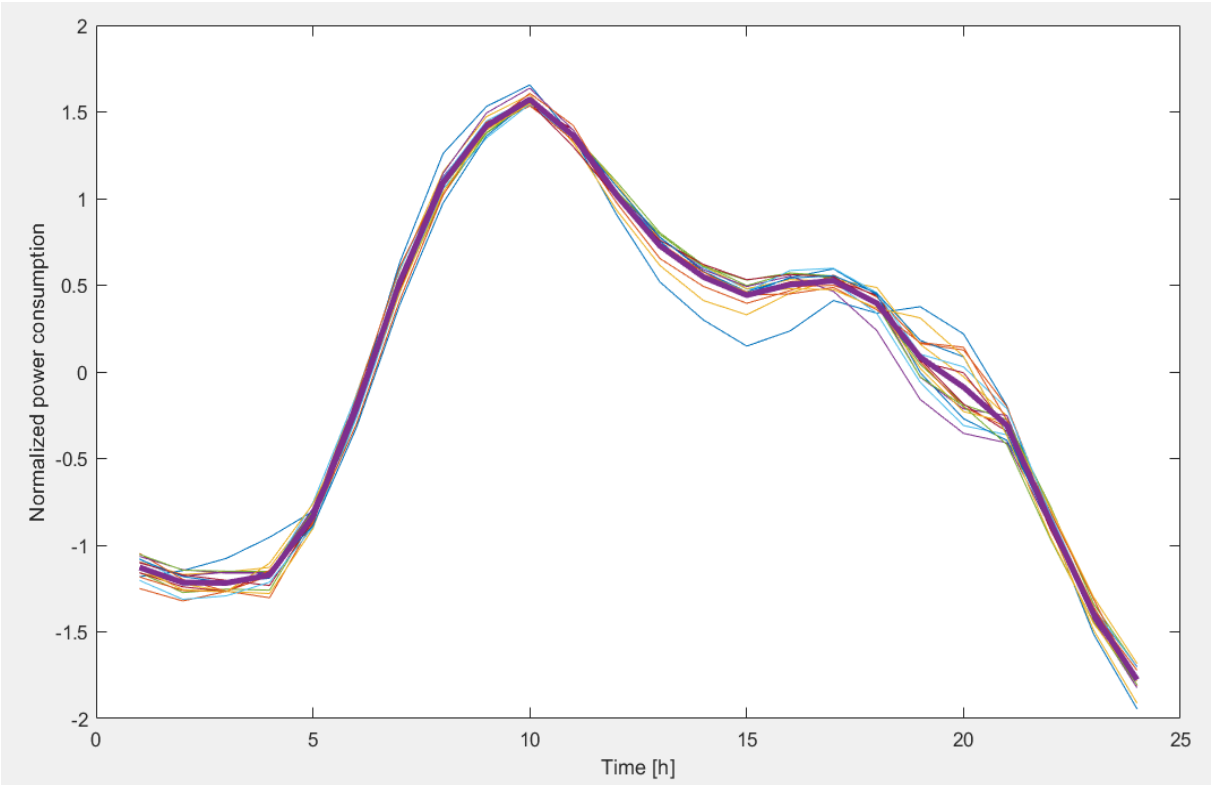


Figure 3.5: cluster2 obtained by k-means

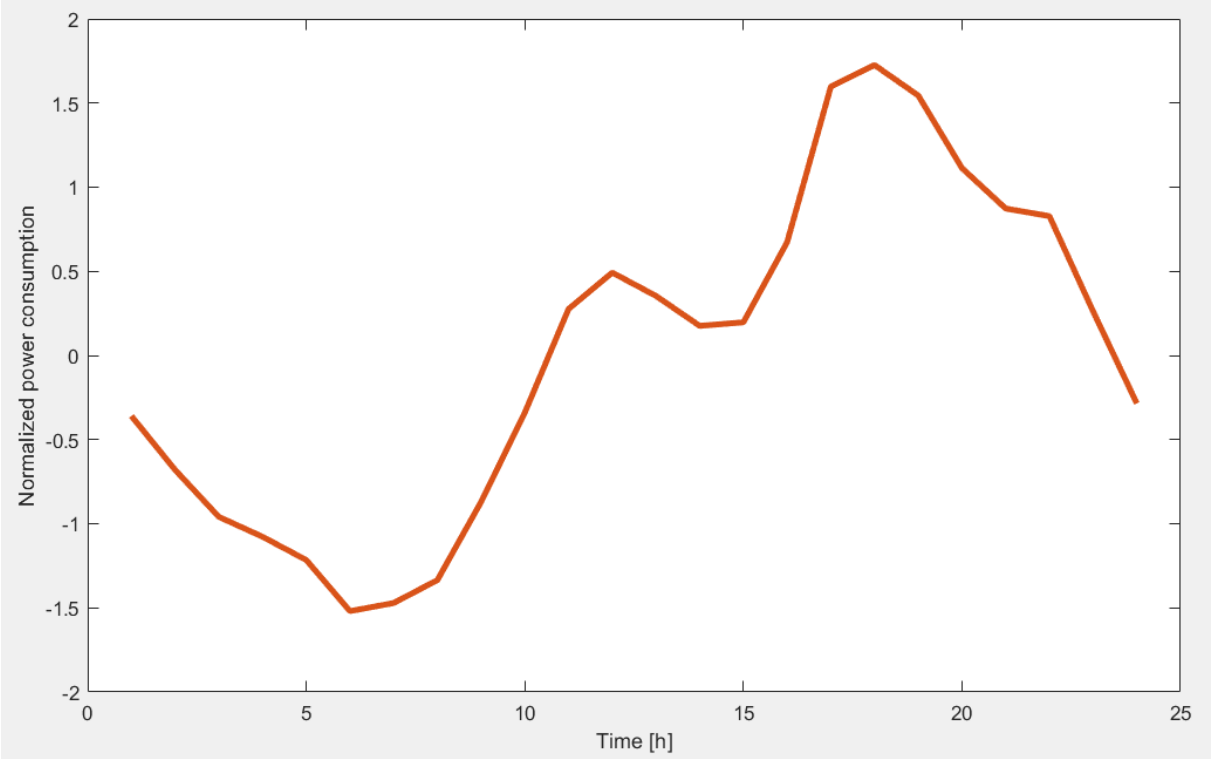


Figure 3.6: cluster21 obtained by k-means

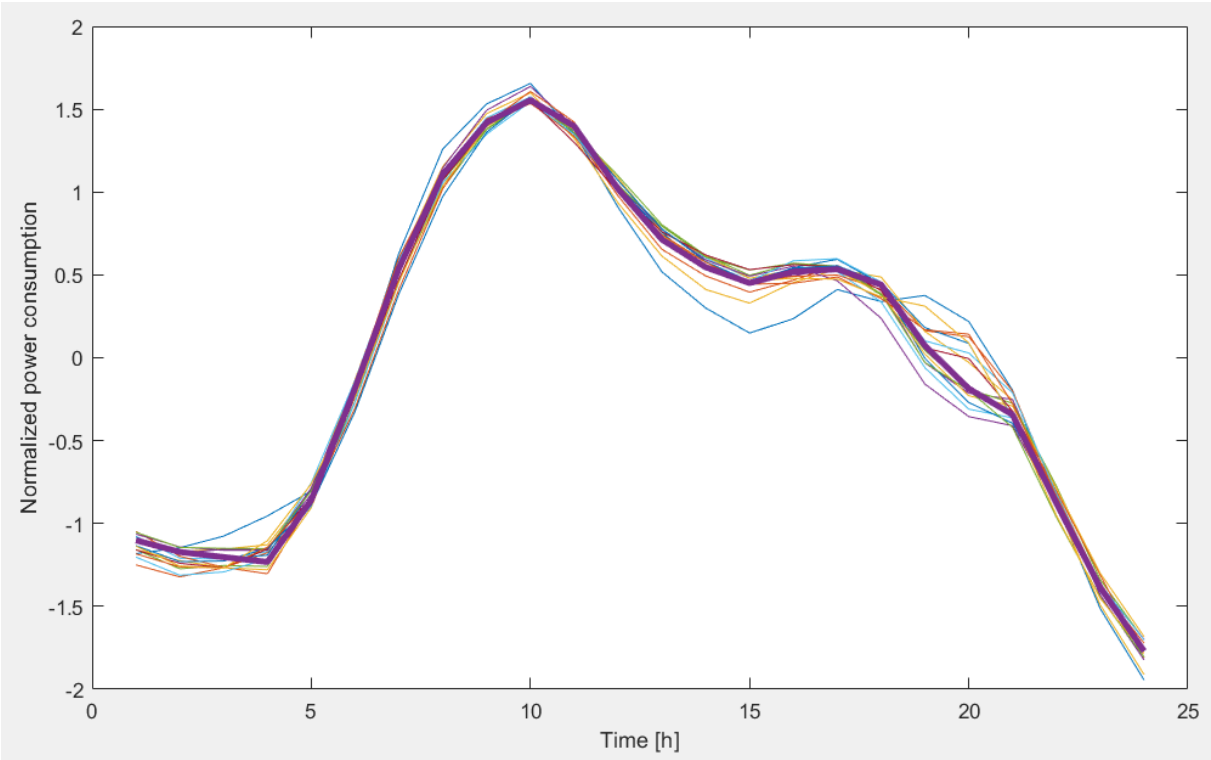


Figure 3.7: cluster1 obtained by k-medoids

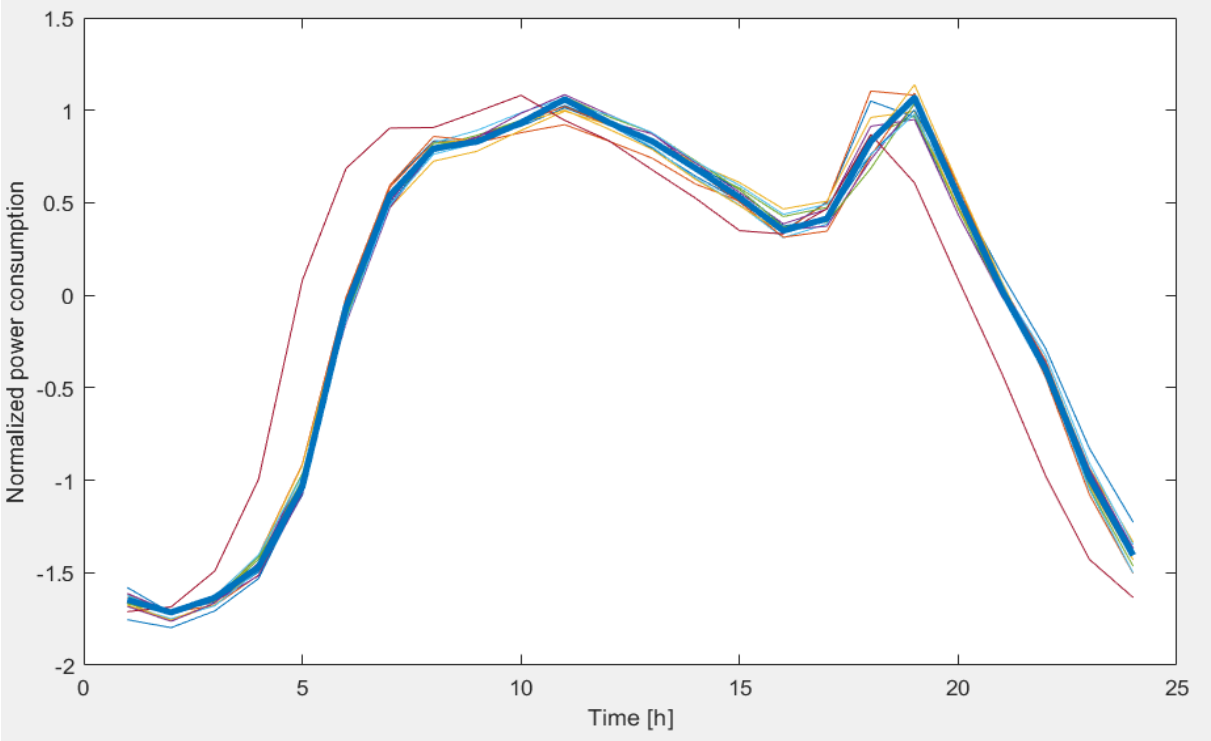


Figure 3.8: cluster2 obtained by k-medoids

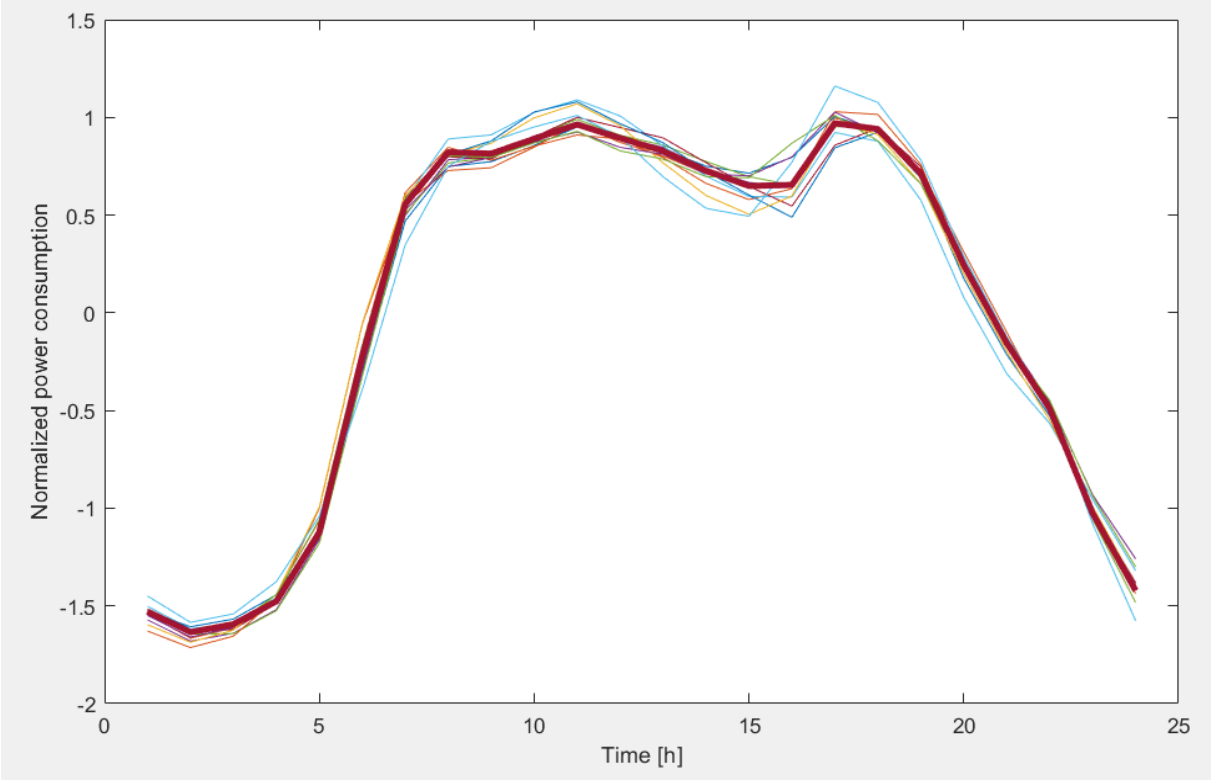


Figure 3.9: cluster21 obtained by k-medoids

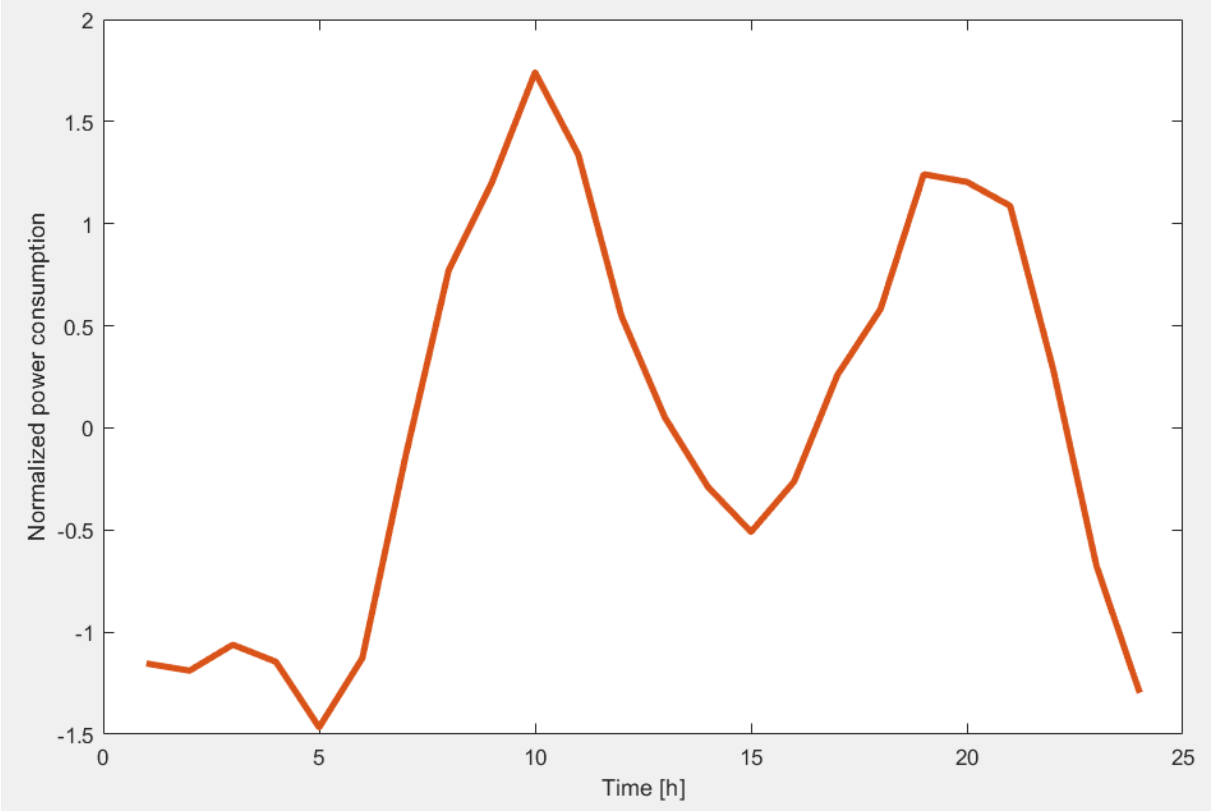


Figure 3.10: cluster35 obtained by k-medoids

As it is possible to observe from the previous figures, applying different clustering algorithms gives different results. Indeed, by comparing figure 3.4 with 3.7, figure 3.5 with 3.8 and figure 3.6 with 3.9 it is clear that a certain cluster contains different curves depending on the clustering method. Moreover, figure 3.6 and 3.10 shows that it is possible to obtain a cluster composed by a single curve. The centroids of the determined clusters are called *typical days* or *representative days* of electrical consumption. It is important to remark that:

- In the k-means algorithms, the typical days that are obtained are not actual days of electrical consumption in Germany, since they are computed as the average values of the curves belonging to a cluster. Instead, in the k-medoids algorithms, the obtained typical days are actual days of electrical consumption. This allows to define the most important days of the year in term of representation. Exploiting these days as input for different algorithm instead of the complete 365 curves leads to coherent and consistent results in a fraction of the original computational time.

- A surely important electrical consumption day is represented by the peak load day, that must be included as a typical day in the model in order not to undersize or underestimate components and plants. Therefore, it is necessary to verify if this day has been selected as medoid by the clustering algorithm and, if not, it must be manually included.

Since different results are obtained by performing different algorithms, to determine the optimal solution some quality indexes must be computed. As already discussed, k-means and k-medoids exploit different error parameters. More precisely, k-means algorithm typically relies on the sum of squared errors while k-medoids algorithm typically relies on the sum of absolute errors. Therefore, these two parameters will not be computed since it makes no sense to compare them. Instead, two different indexes will be analysed in order to determine the optimal number of clusters and the quality of clustering result. These indexes have a different nature: the first is the percentage mean error computed on the load duration curves (the real one and another one built up with clustered data), while the second is the already presented Dunn index. Before discussing the obtained results, it is convenient to analyse the Matlab implementation of these two indexes.

The load duration curve is an extremely important curve used to make predictions and planning forecasts. It is built up by ordering in descending order the values of the power consumption, and then plotting them on the y-axis of a graph, while the x-axis reports the time instants all over the year. Figure 3.11 shows the load duration curve related to the case under analysis.

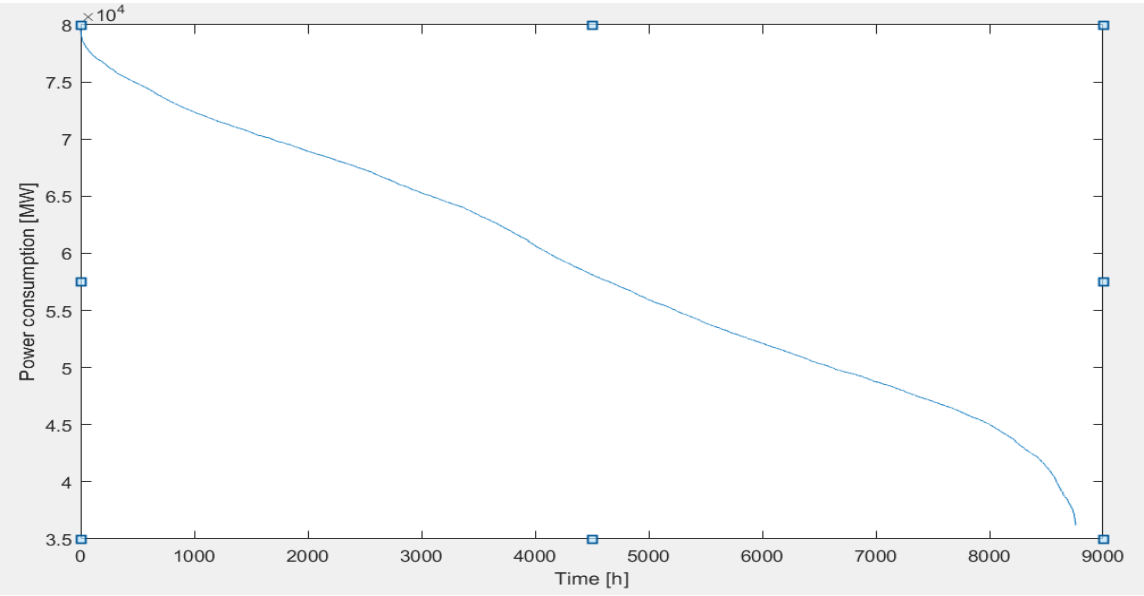


Figure 3.11: load duration curve related to the considered dataset

As it is possible to note from figure 3.11, the x-axis (time axis) goes from zero to 8760, i.e., the number of hours in one year. Instead, as already defined, the y-axis reports the load values in descending order. Note that the real data have been used (and not the normalized ones) since the load duration curve has a physical meaning that is not evident if normalized data are considered. Indeed, a point (t, P) on the curve defines that for t hours along the year the power consumption has been higher than P. For instance, entering in the graph with $t = 2000\ h$ and moving upwards until the curve is met allows to read on the y-axis a power value $P \approx 70000\ MW$. This implies that for 2000 hours in one year the power consumption has been higher than 7000 MW. These considerations are extremely important in order to make predictions about the necessary generation capacity for the next years in terms of both energy and power requirements. After the clustering process, instead than the original 365 curves, only 36 curves are obtained, corresponding to the centroids of the 36 determined clusters through k-means or k-medoids. Therefore, instead than 8760 hourly values (365x24), only 864 (36x24) are available. However, to build up a load duration curve using clustered data, all the hours in a year must be considered Thus, a weight corresponding to the number of curves inside the cluster has been defined for every cluster. Then, the centroid curve of every cluster has been replicated a number of times equal to its weight, so that at the end of the process 365 curves are available. For instance, if the first determined cluster contains 10 curves, its centroid is replicated ten times in a new vector used to sort in descending order the power consumption values in order to build up the reconstructed load duration curve from clustered data. In the following, the implemented Matlab code to perform the explained process is reported.

```
w=zeros(1,n_clust);
for i=1:n_clust
     a=find(idx==i);
     l=length(a);
     w(1,i)=l;
end
dur_data_rec=zeros(8760,1);
temp=zeros(1,24);
temp=c(1,:);
dur_data_rec(1:w(1)*24,1)=(repmat(temp, 1, w(1)))';
jj=w(1)*24+1;
for i=2:n_clust
     temp=c(i,:);
     dur_data_rec(jj:jj+w(i)*24-1,1)=(repmat(temp, 1,
     w(i)))';
```

```
    jj=jj+w(i)*24;
end
dur_curve_rec=sort(dur_data_rec, 'descend');
```

(3.4)

As it is possible to note from code (3.4), the weight vector w contains in each position the number of curves inside the cluster related to that position. Then, a vector temp containing the centroid of a considered cluster is repeated for a number of times equal to the related value of the vector w though the Matlab function "repmat" and the result is stored in the vector dur_data_rec. Finally, by sorting in descending order this last vector, the vector dur_curve_rec, used to plot the reconstructed load duration curve, is obtained. Figure 3.12 below shows the real load duration curve and the reconstructed one from clustered data in 36 clusters:

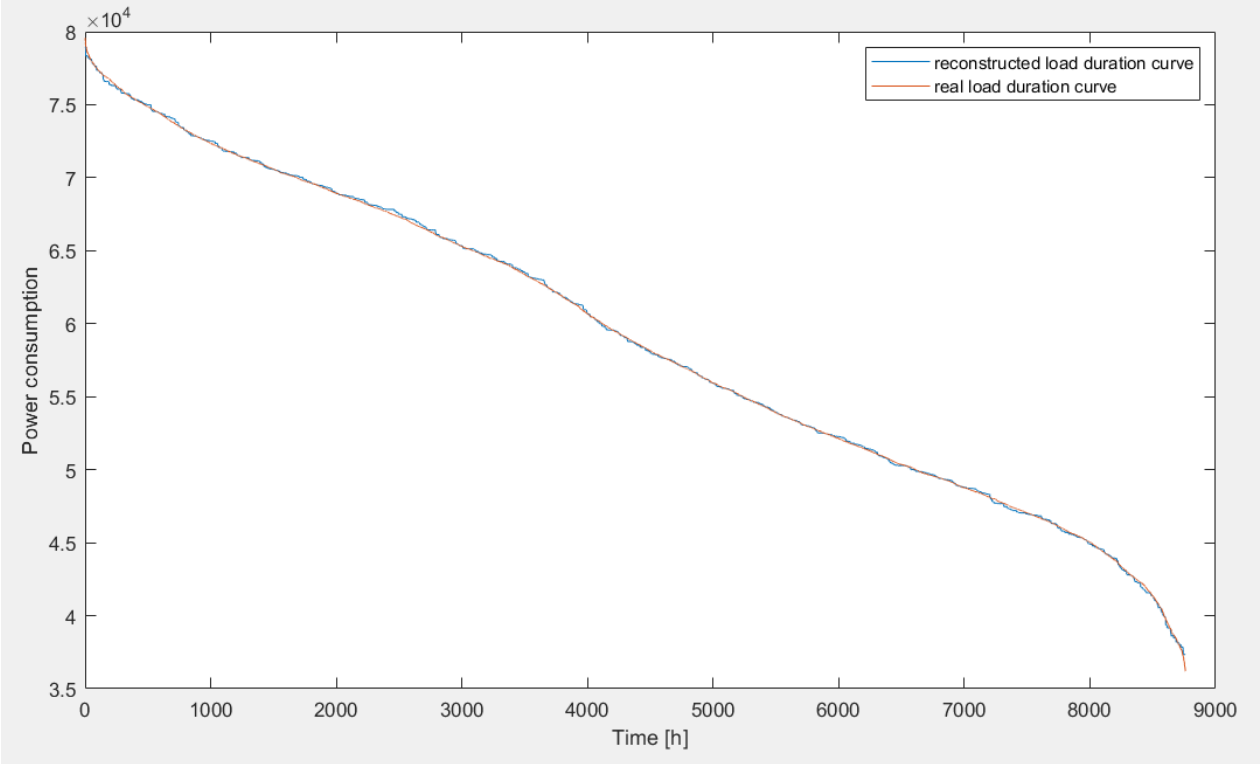


Figure 3.12: real and reconstructed load duration curves for K=36

Figure 3.12 shows that the reconstructed load duration curve is made by a series of steps of different length, corresponding to the fact that a certain power consumption value has been repeated for a certain number of times, depending on the number of

repetitions of the centroid to which that value belongs. Moreover, the reconstructed curve is sometimes higher, sometimes lower than the real load duration curve, as the focus in figure 3.13 shows.

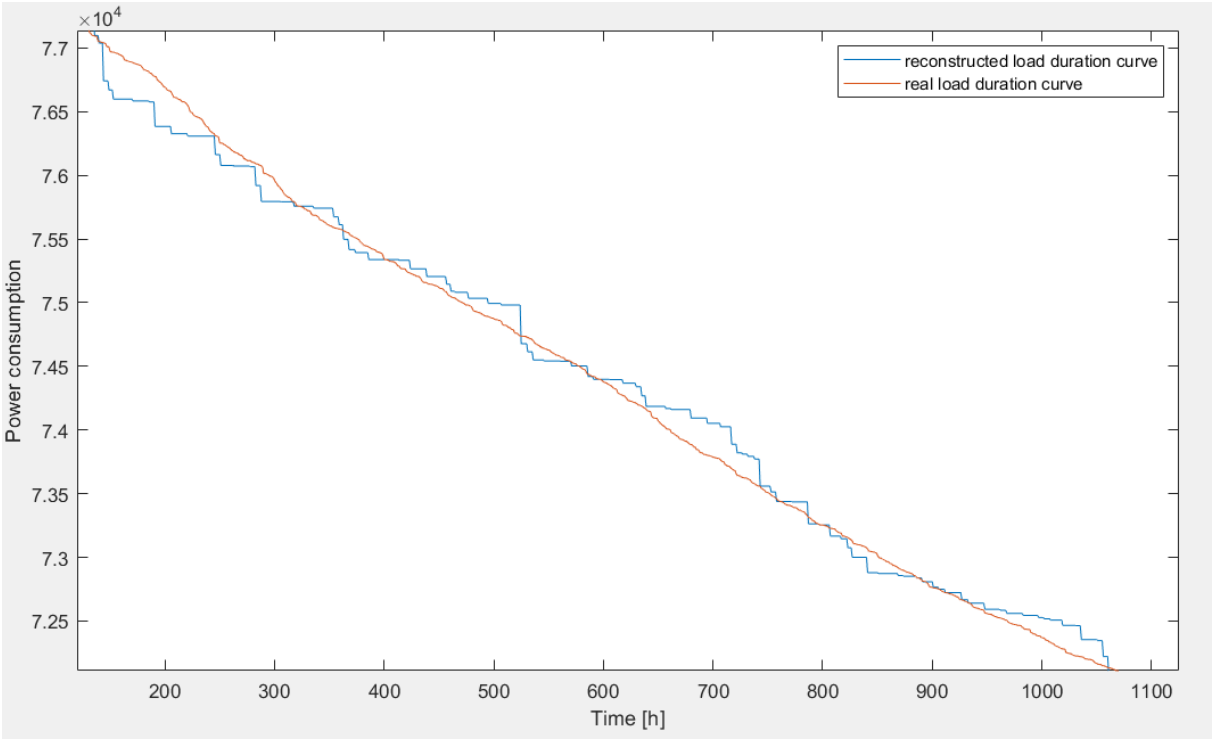


Figure 3.13: focus on figure 3.12

It is possible to consider the error between these two curves as a reliable indicator of the clustering quality. More precisely, the mean percentage error can be computed as:

$$MPE = \frac{1}{N}\sum_{t=1}^{N}\frac{|dur_{rec,t} - dur_{real,t}|}{dur_{real,t}} \times 100 \tag{3.5}$$

For a number of clusters $K = 36$, the computation of the mean percentage error gives $MPE = 0.17\%$. Therefore, the real load duration curve is well approximated by the reconstructed load duration curve using only the clustered data. If the number of clusters is set to an extremely low value $K = 2$, the computation of the mean percentage error gives $MPE \approx 2\%$. The MPE has increased by more then ten times

with respect to considering 36 clusters, thus reducing the accuracy of the clustering model and the goodness of fit of the typical days. This can be shown also graphically, by plotting on the same graph the real and reconstructed load duration curve with only 2 clusters, as figure 3.14 reports.

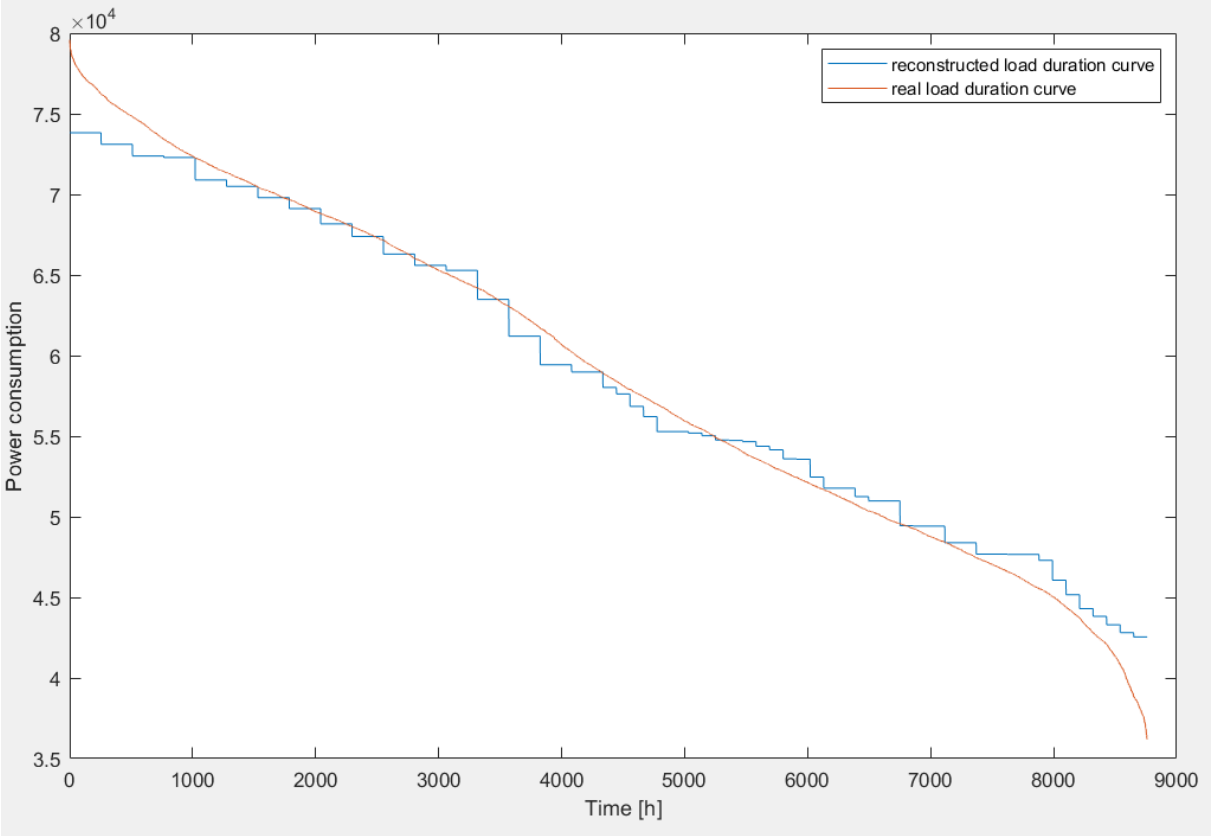


Figure 3.14: real and reconstructed load duration curves for K=2

It is also possible to compute the area below the two curves in order to determine another percentage approximation index. For the sake of clearness, it is recalled that this area represents the total energy required by the load along the year. Once the two areas have been evaluated, the percentage area error can be defined as:

$$PAE = \frac{A_{real} - A_{rec}}{A_{real}} * 100 \tag{3.6}$$

To compute the area below a curve, the Matlab function "trapz" can be exploited. This function numerically approximates the integral of the curve by using the trapezoids method. Again, A_{rec} has been evaluated for $K = 36$ and $K = 2$. In the first

case, $PAE = 0.05\%$ while in the second case $PAE = 0.08\%$. This implies that the model with a higher number of clusters is better, as the computation of the MPE also shows.

The previously discussed results can be obtained also by computing the Dunn index, presented in the previous chapter, after clustering data with a number of clusters ranging from 1 to 40. More precisely, once a value for K has been defined, the clustering algorithm is performed and the Dunn index is computed. Then, to determine the optimal number of clusters and the quality of the clustering result, the model with the highest value of the Dunn index is selected as optimal. The implementation of this index on Matlab is not trivial at all, and its computation can be performed through the following code, developed in 2010 by Julian Ramos [50]:

```
function DI=indexDN(data,labels,distance)
if ~exist('distance','var') || isempty(distance)
     distance = 'euclidean';
end
i= length(unique(labels));
distM = squareform(pdist(data,distance));
ind = labels;
denominator=[];
for i2=1:i
     indi=find(ind==i2);
     indj=find(ind~=i2);
     x=indi;
     y=indj;
     temp=distM(x,y);
     denominator=[denominator;temp(:)];
end
num=min(min(denominator));
neg_obs=zeros(size(distM,1),size(distM,2));
for ix=1:i
     indxs=find(ind==ix);
     neg_obs(indxs,indxs)=1;
end
dem=neg_obs.*distM;
dem=max(max(dem));
DI=num/dem;
end
```

(3.7)

Looking at code (3.7), it is possible to note that it implements the formula presented in equation (2.14). Indeed, the numerator of the Dunn index ratio is computed as the minimum inter-cluster distance, determined as the minimum value of a vector "denominator" which contains distances between every couple of curves non-belonging to the same cluster. Instead, the denominator of the Dunn index ratio is computed as the maximum intra-cluster distance, determined as the maximum value of a vector "dem" which contains distance between every couple of curves belonging to the same cluster. As already mentioned, the clustering algorithms has been performed with a variable number of clusters ranging from 1 to 40 and for every clustering output the Dunn index has been computed. The result can be observed in figure 3.15 below:

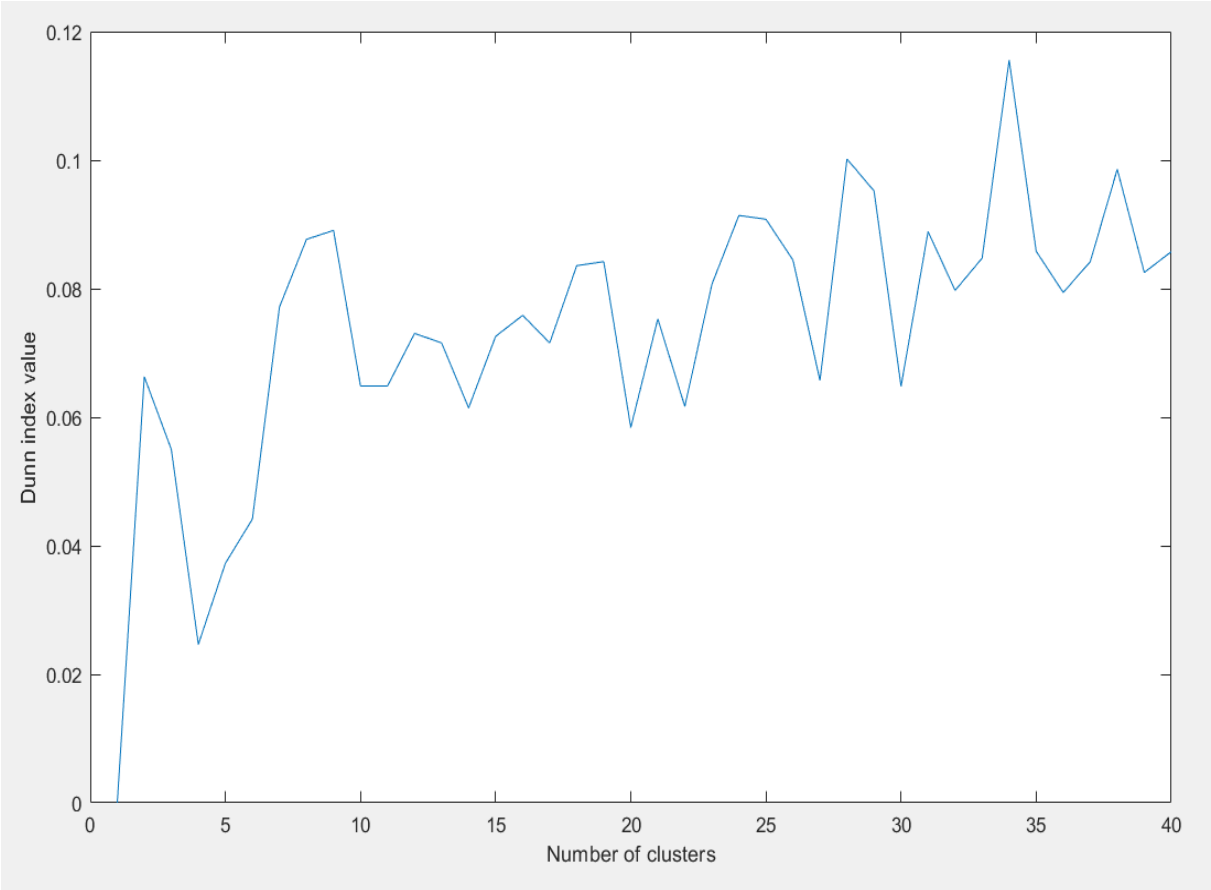


Figure 3.15: Dunn index values for different number of clusters

For the sake of clearness, it is recalled that since the Dunn index is defined as the ratio between the inter-clusters and intra-cluster distances, the higher is its value the better is the clustering result. Therefore, figure 3.15 shows that the optimal number of

clusters is $K = 34$, since the Dunn index reaches its highest peak for this number of clusters. The performed analysis considering 36 clusters is of course valid, since with $K = 36$ the Dunn index has still a high value that reflects the goodness of fit of the clustering model. Thus, the 36 obtained curves (representative load days) can be employed as input for many different algorithms in order to reduce the computational time and complexity anyway reaching coherent and consistent results.

3.2 R-studio clustering implementation

The implemented code for clustering implementation in R-studio is reported in Appendix B. While in the previous section the partitional clustering algorithms have been exploited, now the Ward hierarchical algorithm has been considered. Before analyzing the obtained results, it is convenient to show also how the dynamic time warping distance has been determined. Indeed, R-studio includes a library for the computation of dtw distance measure and its use is very simple. However, for the sake of clearness, the code related to this library is reported and examined in the following:

```
#compute the distance matrix bewtween elements of two time series of length 9
dtw(a1, a2)$stepPattern; #shows equations that are used
dtw_matrix = matrix(rep(c(0),81), nrow=9, ncol=9, byrow = TRUE);
dtw_matrix[1,1] = sqrt(a1[1]^2 - a2[1]^2); #first element, euclidean distance
#first column
for (i in 2:9){
        dtw_matrix[i,1] = sqrt((a1[i] - a2[1])^2) + dtw_matrix[i-1,1];
}
#first row
for (j in 2:9){
        dtw_matrix[1,j] = sqrt((a1[1] - a2[j])^2) + dtw_matrix[1,j-1];
}
```

```
#rest of the matrix
for (i in 2:9){
        for (j in 2:9){
                dtw_matrix[i,j] = sqrt((a1[i] - a2[j])^2) + min(dtw_matrix[i,j-1], dtw_matrix[i-1,j], dtw_matrix[i-1,j-1]+sqrt((a1[i] - a2[j])^2));
        }
}
#find the optimal allignement bewtween time series-->minimum global distance path on matrix
#if d(i−1,j−1)=d(i,j−1) we choose d(i,j−1) and if d(i−1,j−1)=d(i−1,j) we choose d(i−1,j−1)
path = c(9,9); # starting with furthest place in matrix (lower right corner)
i = 9;
j = 9;
while(i>1 & j>1){
        if (j == 1) {
                j = j - 1;
        } else if (i == 1) {
                i = i - 1;
        } else if (dtw_matrix[i,j-1] == min(dtw_matrix[i-1, j-1], dtw_matrix[i-1, j], dtw_matrix[i, j-1])){
                j = j - 1;
        } else if (dtw_matrix[i-1,j-1] == min(dtw_matrix[i-1, j-1], dtw_matrix[i-1, j], dtw_matrix[i, j-1])){
                i = i - 1;
                j = j - 1;
        } else {
                i = i - 1;
        }
        path = rbind(path, c(i,j));
}
```

```
path = rbind(path, c(1,1));
plot(dtw(a1,a2));
points(path[,1], path[,2], type="l");
plot(dtw(a1,a2, k=TRUE), type="three");
```

(3.8)

As it is possible to note from code (3.8), the implemented procedure is exactly the one described previosuly: first, the local cost matrix is computed by determining initially its first row and column and then computing the remaining elements using the step-pattern equation. Then, starting from lower right corner of the global cost matrix (i, j are initialised to the number of rows and columns of the matrix, thus the first considered position is the lower right corner), the three adjacent positions are checked in order to determine the one which has the minimum distance. For instance, if the DTW matrix is 9x9, starting from position (9,9) the examined positions are (8,9), (9,8) and (8,8), that are adjacent to (9,9). The indexes of the new selected position (that have minimum distance from the starting position) are saved in the "path" array and the procedure is iteratively repeated until the first position of the global cost matrix (upper left corner) is reached (while cycle is stopped). The result is an array containing the positions of the selected element of the global cost matrix, thus containing the indices of the optimal warping path (alignment) between the two considered time series. As in the previous section, the following figures show the obtained optimal alignment between two time series (the same load curves reported in figure 3.3) by means of the three-way plot.

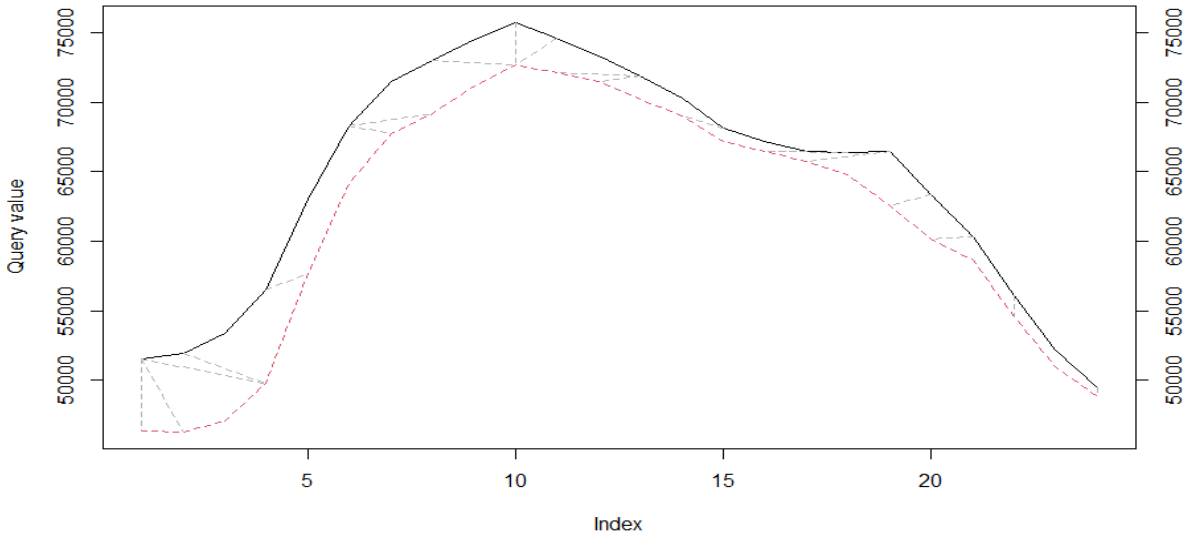


Figure 3.16: optimal alignment between time series of figure 3.3

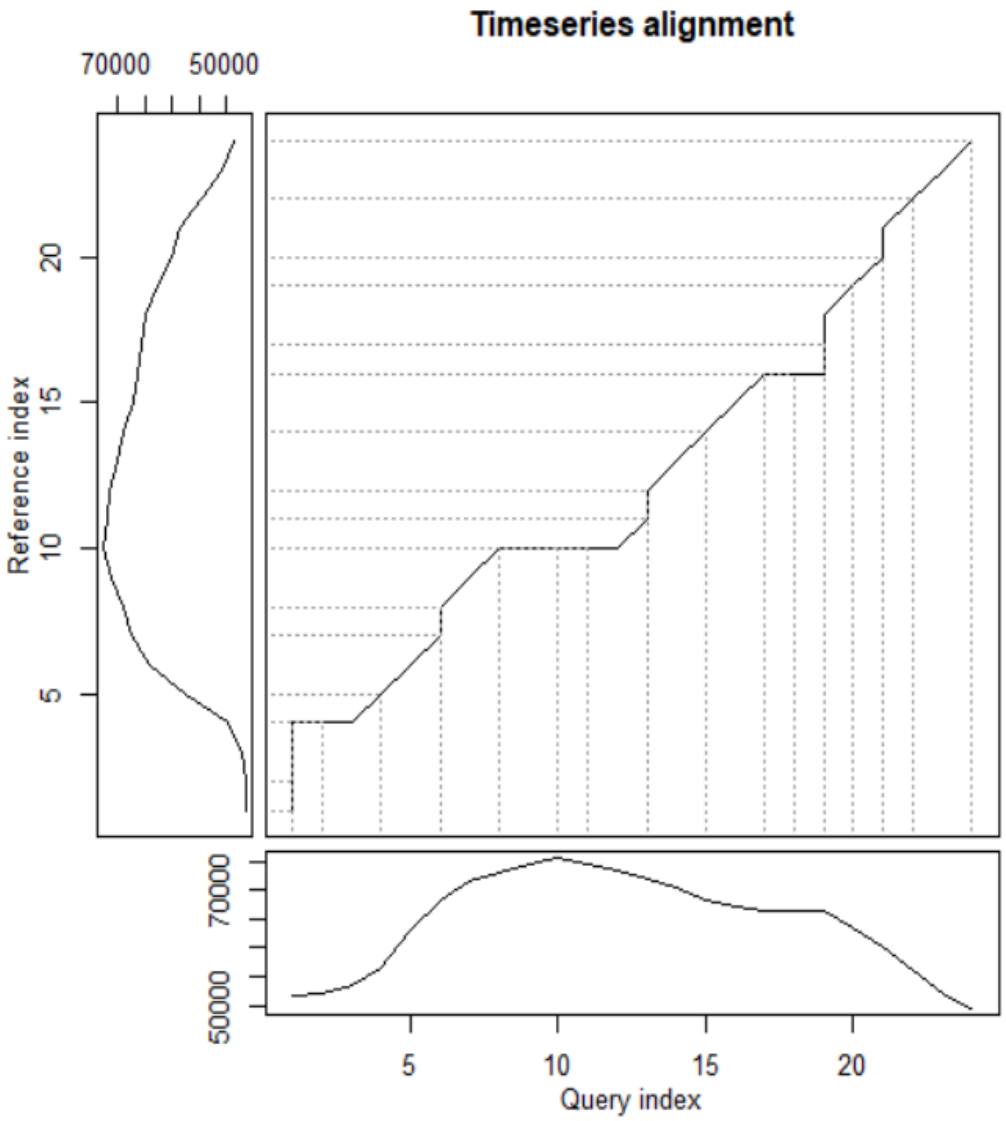


Figure 3.17: three-way plot of the optimal alignment reported in figure 3.16

It is not convenient for graphical purpose to report the local or global cost matrix, since it has 365 rows and columns. Anyway, a portion of it limited to 20 rows and columns is represented in figure 3.18. Even if it is not possible to precisely read the distance values in each position of the matrix, its meaning is clear. Indeed, as it is possible to observe, the optimal warping path (represented in blue) is the first portion of the alignment reported in figure 3.17. As already mentioned many times, this path passes only through the low-cost areas of the heatmap, denoted by a lighter color, thus leading to a minimized global cost.

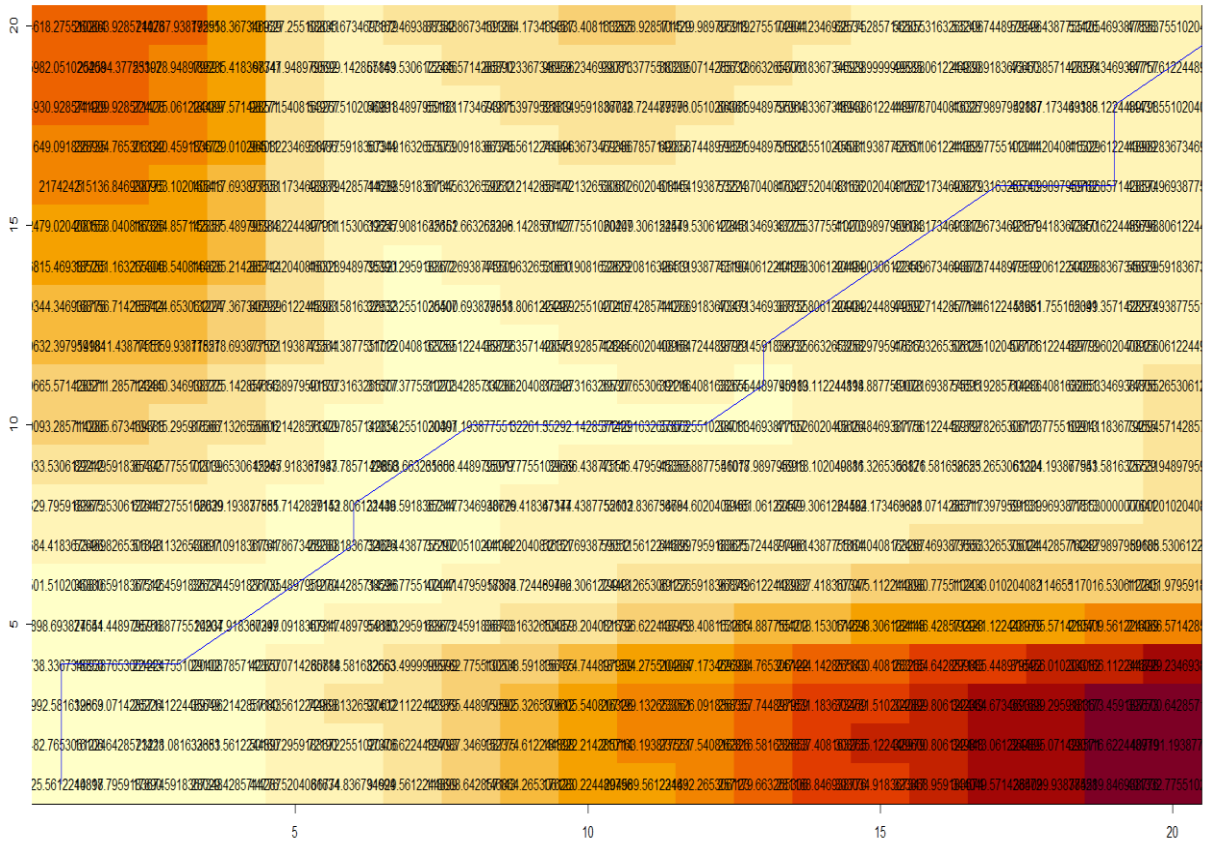


Figure 3.18: heatmap of the global cost matrix

Before discussing the clustering results, it is convenient to analyze the function that implements the clustering process. In particular, the R-studio code is:

```
cluster_dtw_h <-list()
for (i in 2:36)
{
    cluster_dtw_h[[i]] <- tsclust(df_list_z, type = "h", k = i,  distance = "dtw", control = hierarchical_control(method = "complete"), preproc = NULL, args = tsclust_args(dist = list(window.size = 5L)))
}
```

(3.9)

As it is possible to note from code (3.9), the function exploited to cluster the dataset is denoted as “tsclust”. More precisely, it produces the same outputs as Matlab (vector

of clusters belonging indexes, centroid matrix, distance matrix and sum) while the required inputs are listed in the next page.

- df_list_z: input dataset containing the normalized load curves. The "tsclust" function works with datasets and not matrix, therefore it is necessary to convert the 365x24 input matrix to a data frame (basic R-studio structure) through the command:

  data_df=as.data.frame(t(data_mat));

  (3.10)

- type: desired category of clustering algorithms. Possible types are "p" (partitional clustering) or "h" (hierarchical clustering).

- k: desired number of clusters.

- distance: selected similarity measure. Typically, dynamic time warping is employed (distance="dtw").

- control: appropriate list of control parameters.

- preproc: function to pre-process the data. Since a z-normalization technique has been already applied to the dataset, the value of this parameter is set to NULL.

- args: appropriate list of arguments for pre-processing, distance and centroid functions. The definition of the window size has the same meaning than in Matlab: selecting 5L as window size implies that the clustering algorithm is repeated five times and the optimal result is selected and stored.

Moreover, code (3.9) shows that with R-studio it is possible to perform automatically the clustering process in a for-loop, creating a list of clustering results for different values of K. This process has been performed considering a number of clusters ranging from 1 to 36 and then computing some quality indexes as done in the previous section to determine the best model and the optimal number of clusters. Some of the obtained clusters for $K = 2$ and for $K = 36$ are reported in the next pages.

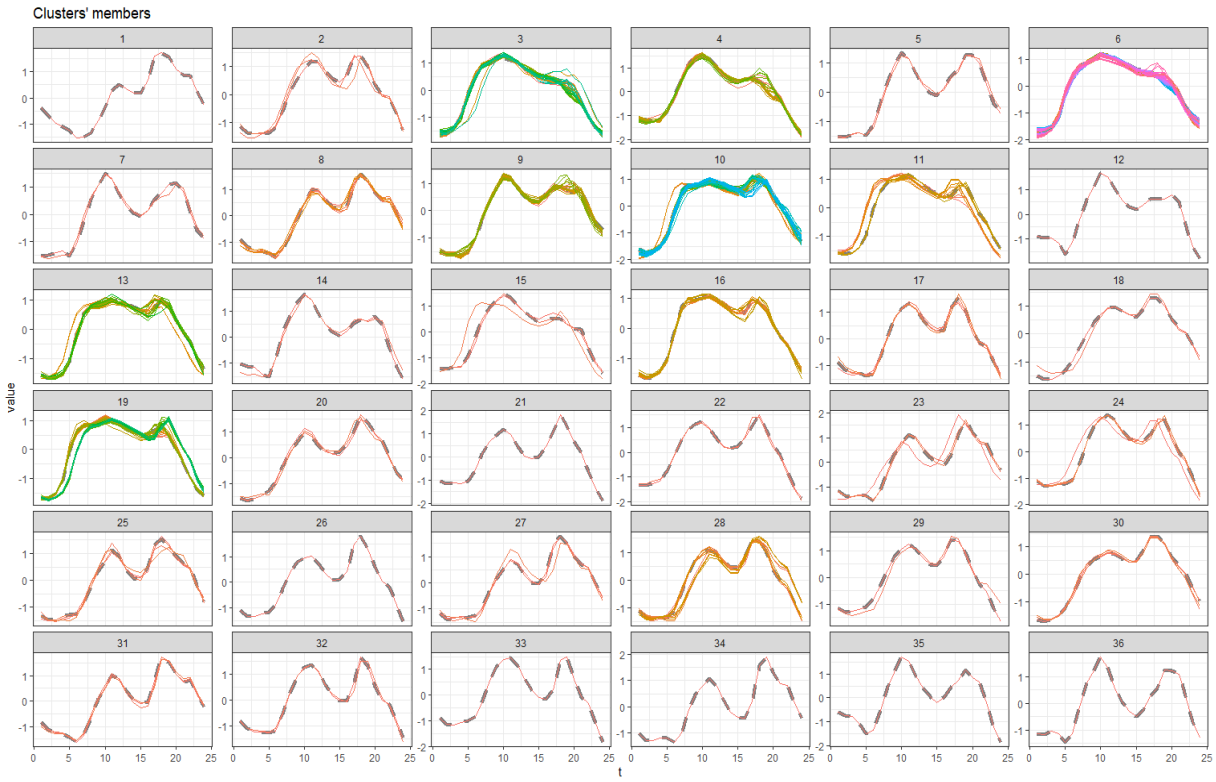


Figure 3.19: overview of the obtained clusters for K=36

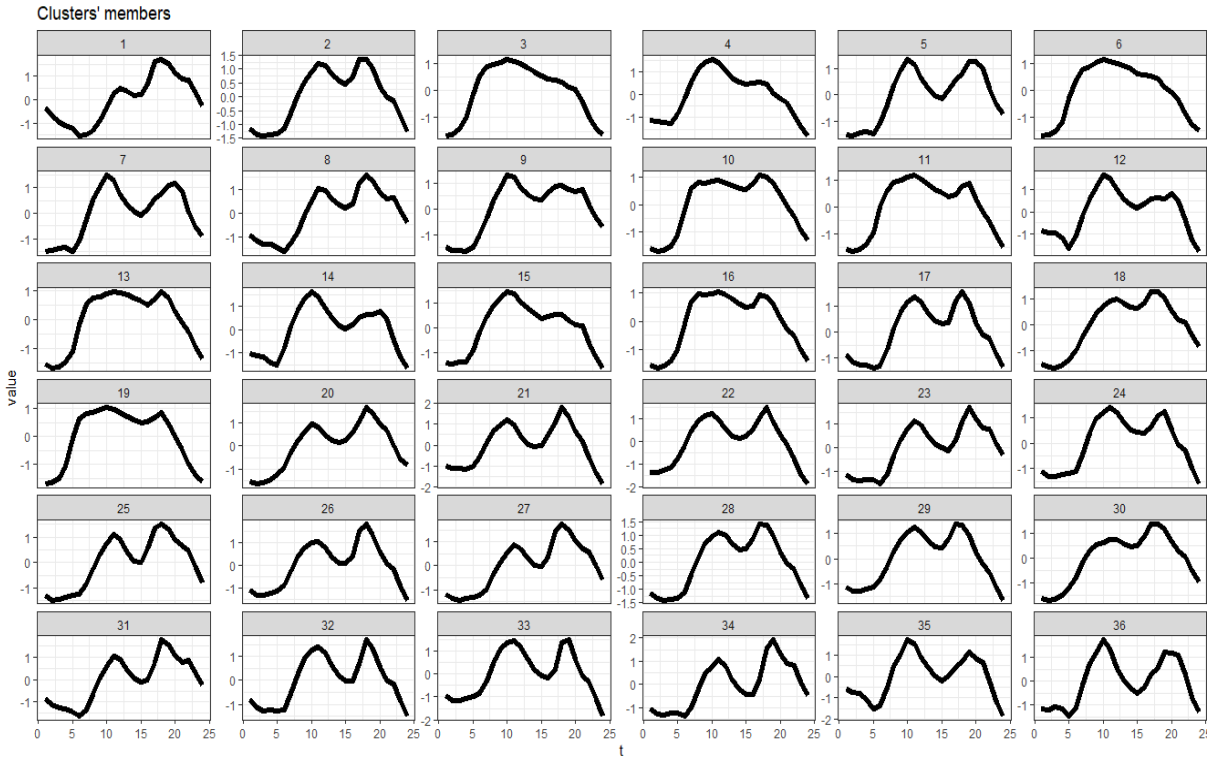


Figure 3.20: centroids (typical days) of the 36 obtained clusters

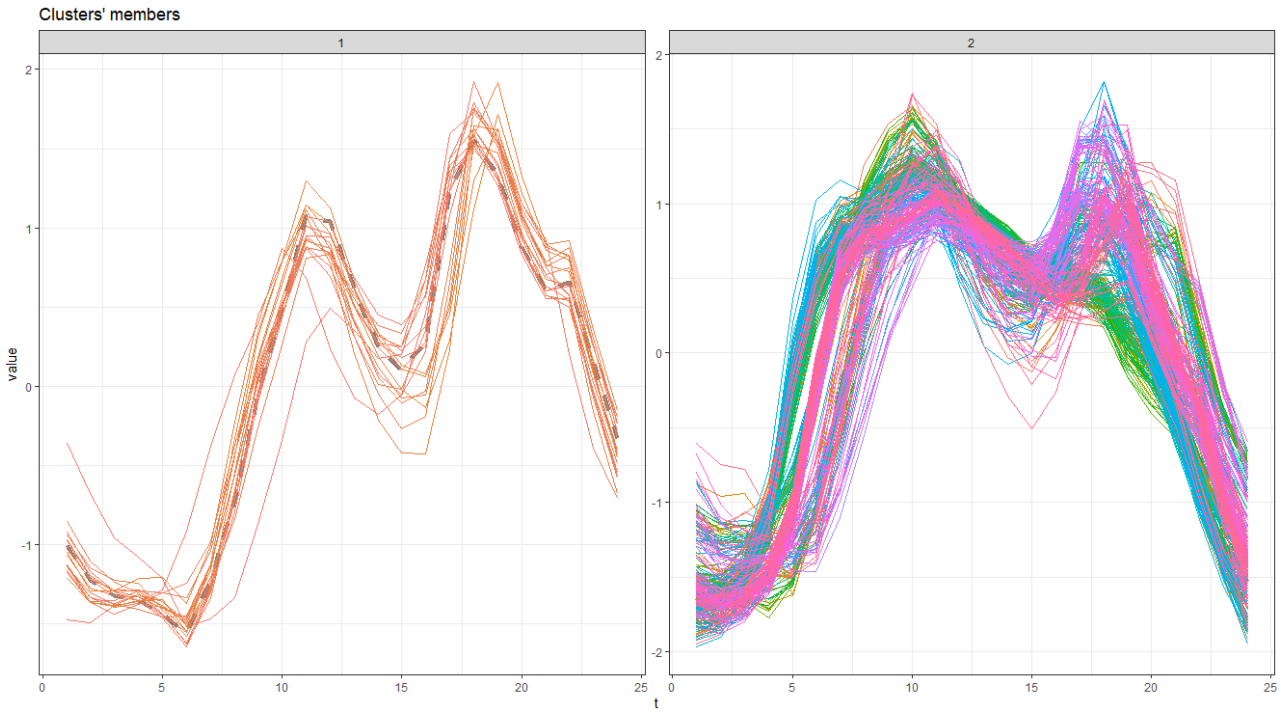


Figure 3.21: overview of the obtained clusters for K=2

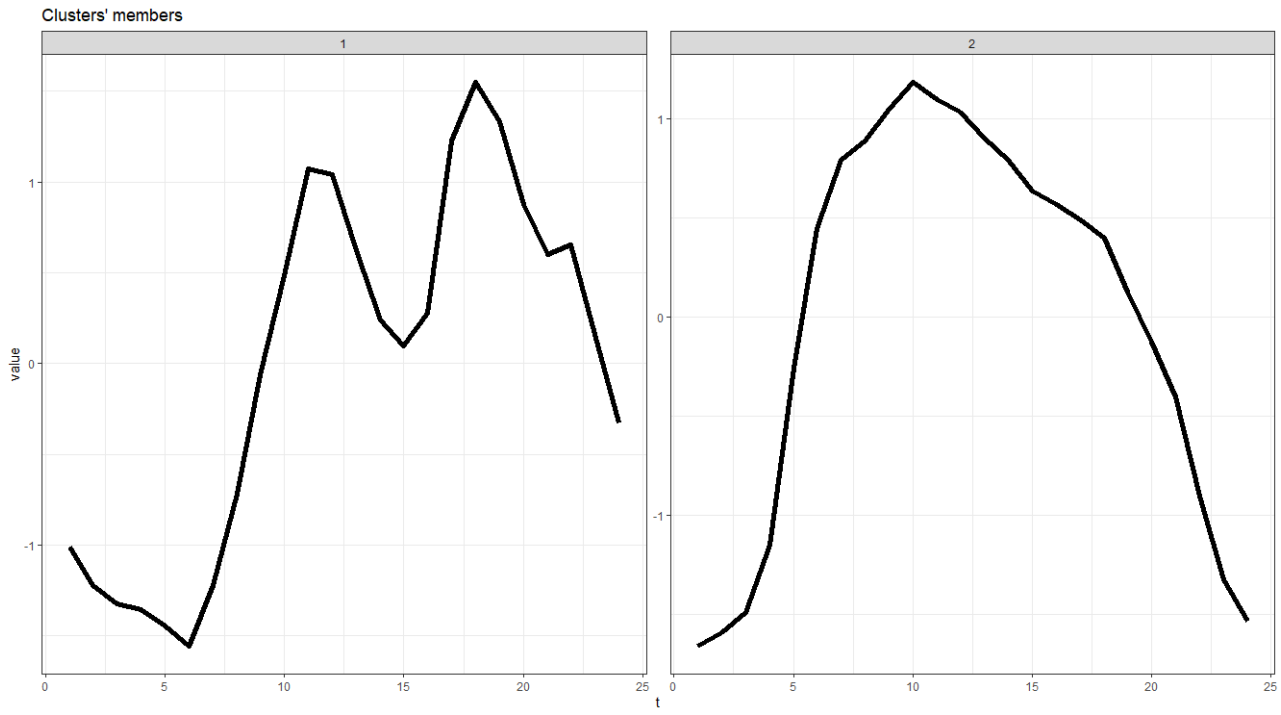


Figure 3.22: centroids of the 2 obtained clusters

As it is possible to note from figures 3.22 and 3.20, clustering the load curves in a too low number of clusters (K=2) results in a bad definition of the typical load days. Indeed, even if cluster1 centroid catches well the shape of its cluster participants, cluster2 centroid misses an important load peak around 19 p.m., thus it does not represent well the behavior of the load during the days belonging to cluster2 itself. Instead, clustering the dataset in 36 clusters (K=36) allows to obtain very representative typical load days. Indeed, for each obtained cluster, the centroid resembles the shape of the cluster participants. This implies that using the typical load days as input of whichever algorithm leads to obtain coherent result with using the entire dataset, of course significantly reducing the computational time. The following figure shows the cluster dendrogram for K=36:

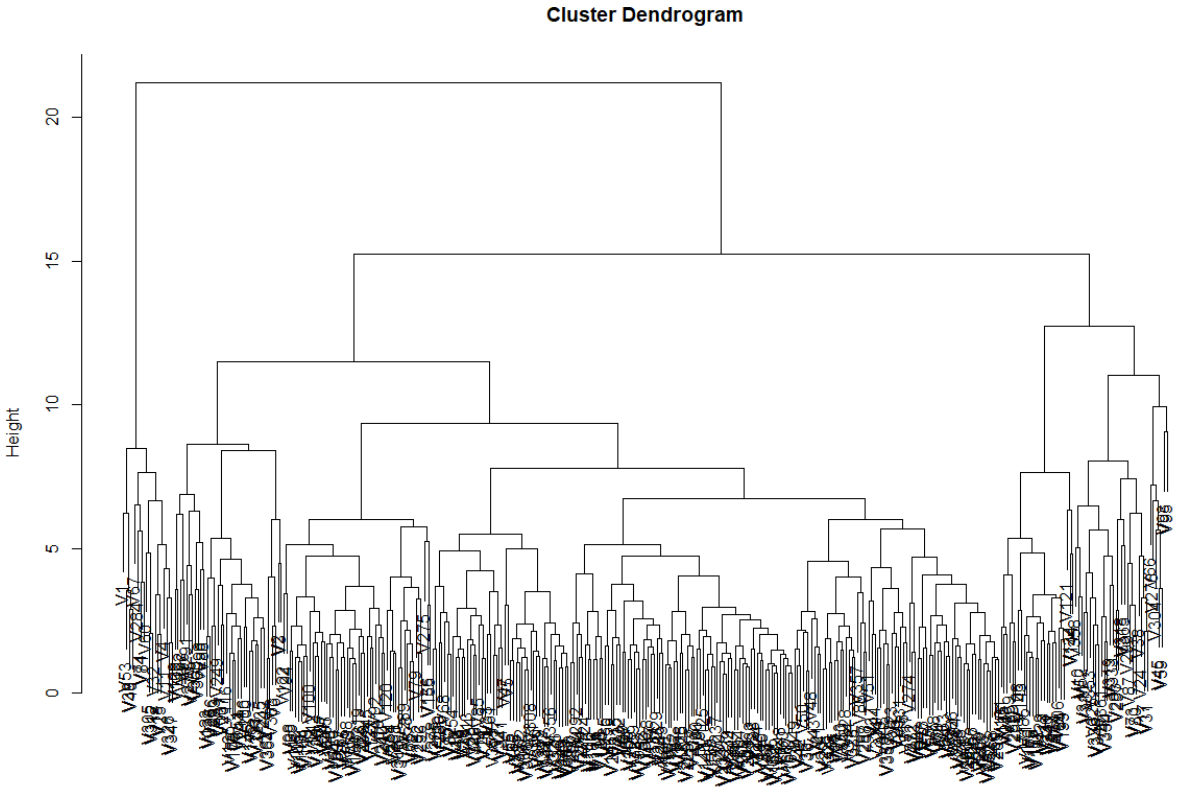


Figure 3.23: cluster dendrogram for K=36

Even if it is not possible to precisely read which load curves have been clustered together from the dendrogram, its meaning is clear. Indeed, the y-axis reports the distances between time series and the different branches show which clusters have been merged in every step until a single cluster containing all the objects is obtained. Of course, the procedure stops when the selected number of clusters is reached.

Typically, exploiting hierarchical algorithms the quality of clustering results is higher and the optimal number of clusters is lower than using partitional algorithms. This can be proved by computing quality indexes as done for Matlab implementation. In particular, the following figures show the Dunn index, the Davies Bouldin index and the gap statistic behavior for different numbers of clusters:

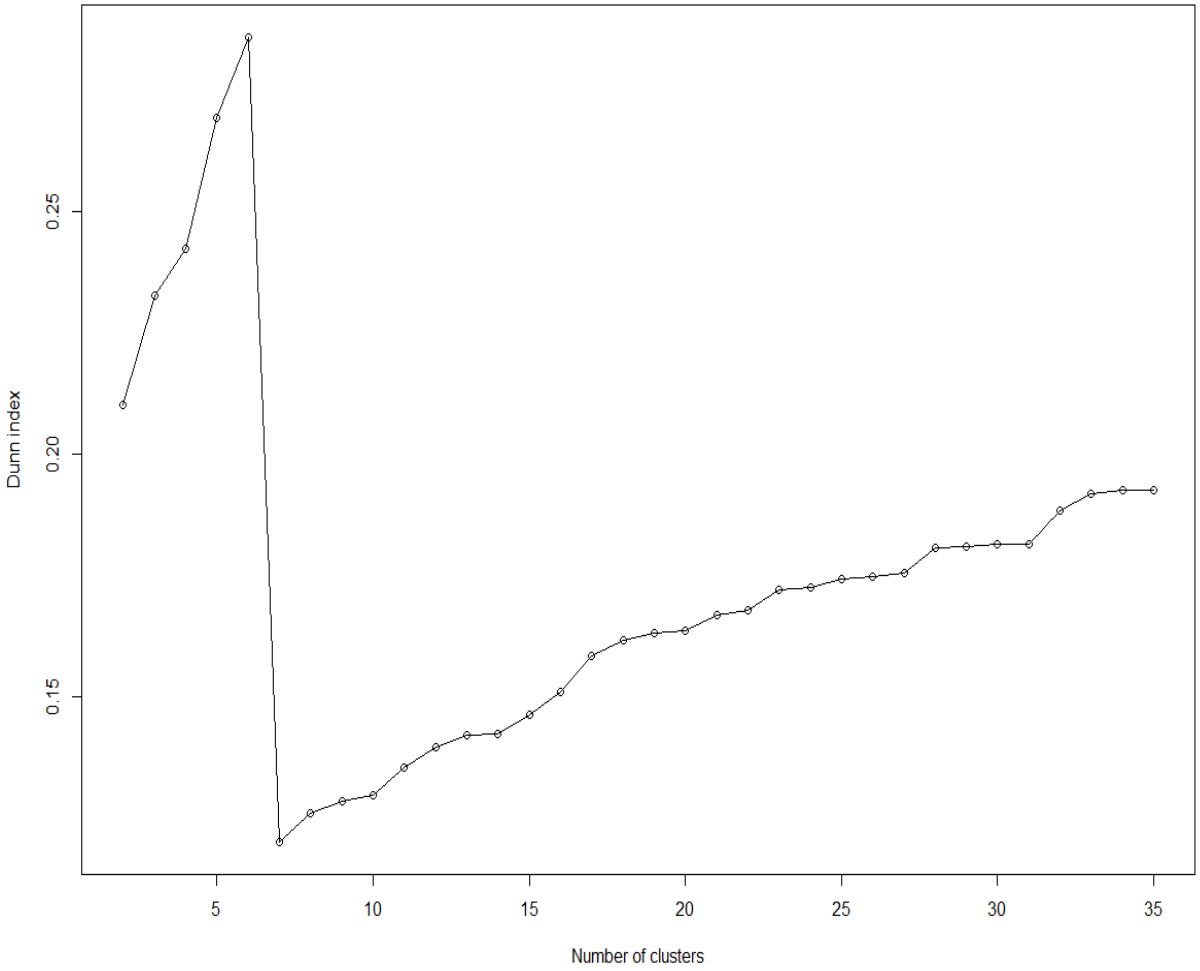


Figure 3.24: Dunn index for different numbers of clusters

As it is possible to note from figure 3.24, the highest value of the Dunn index is obtained for k=6. However, this is in contrast with the already used criterion of matching as much as possible the real load duration curve with a reconstructed one from clustered data. Making several attempts, it is possible to determine a good compromise between the Dunn index value and the error on the load duration curves with a number of clusters $K \geq 31$, coherently with the result obtained in the previous section.

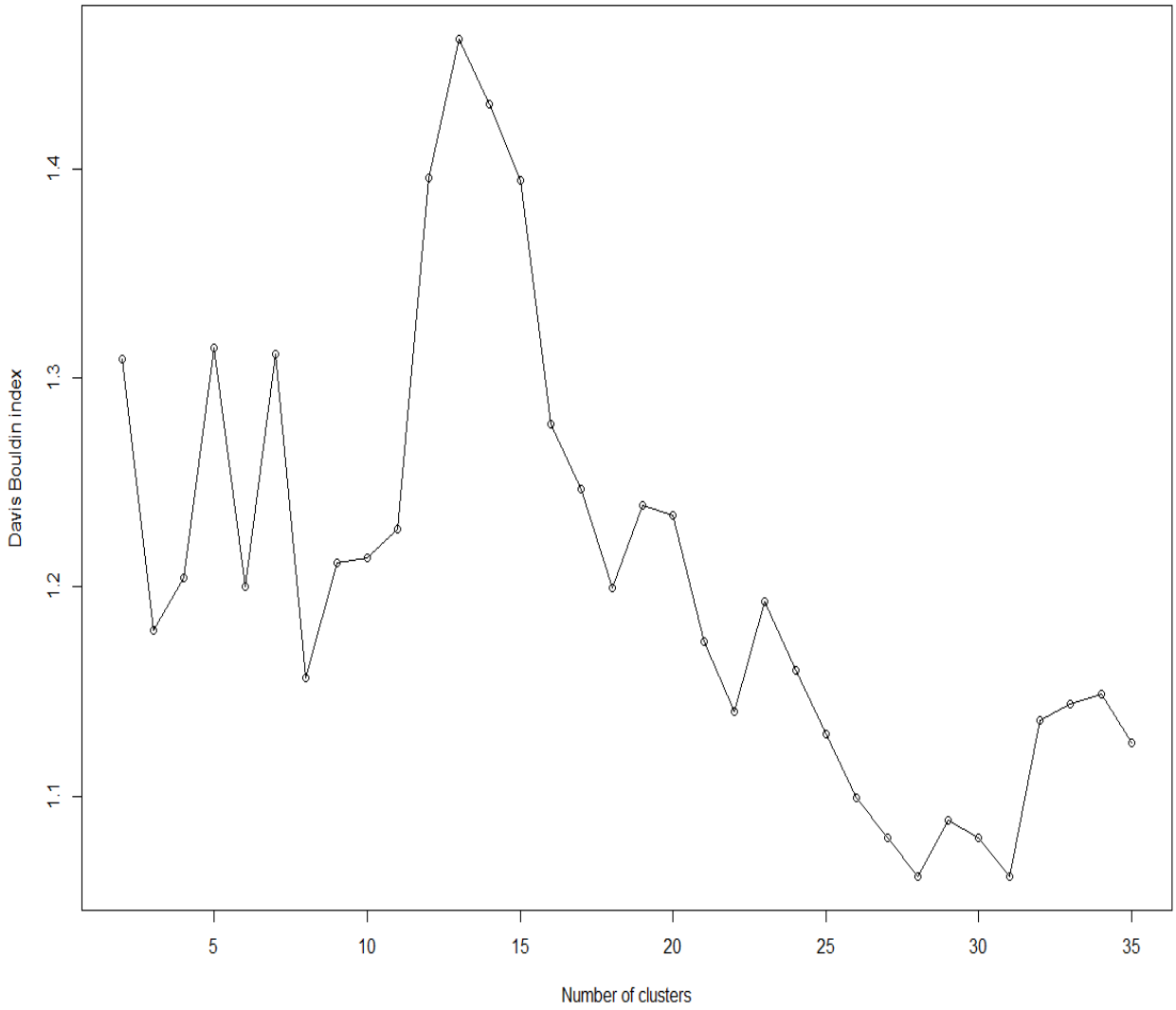


Figure 3.25: Davies-Bouldin index for different numbers of clusters

According to the Davies-Bouldin index, the optimal number of clusters is 31. Indeed, it is recalled that the best clustering model is the one that minimizes the Davies-Bouldin index and maximizes the Dunn index. This value is coherent with the tradeoff discussed in the previous page (it is possible to determine a good compromise between the Dunn index value and the error on the load duration curves with a number of clusters $K \geq 31$) and allows to obtain a very precise reconstructed load duration curve.

Eventually, figure 3.26 in the next page shows the gap statistic behavior for different number of clusters. As it is possible to note, the optimal value of clusters suggested by this parameter is K=10. Indeed, for $K \geq 10$ the gap statistic does not exhibit a great variation, while it increases rapidly until 10 clusters are considered. Of course,

increasing the number of clusters increases the gap statistic value and the matching degree between the load duration curves (advantages) but the computational effort increases too and the values of other quality indexes can be reduced (disadvantages). Again, after several attempts, it is possible to determine a good compromise between the gap statistic value and the error on the load duration curves with a number of clusters $K \geq 21$,

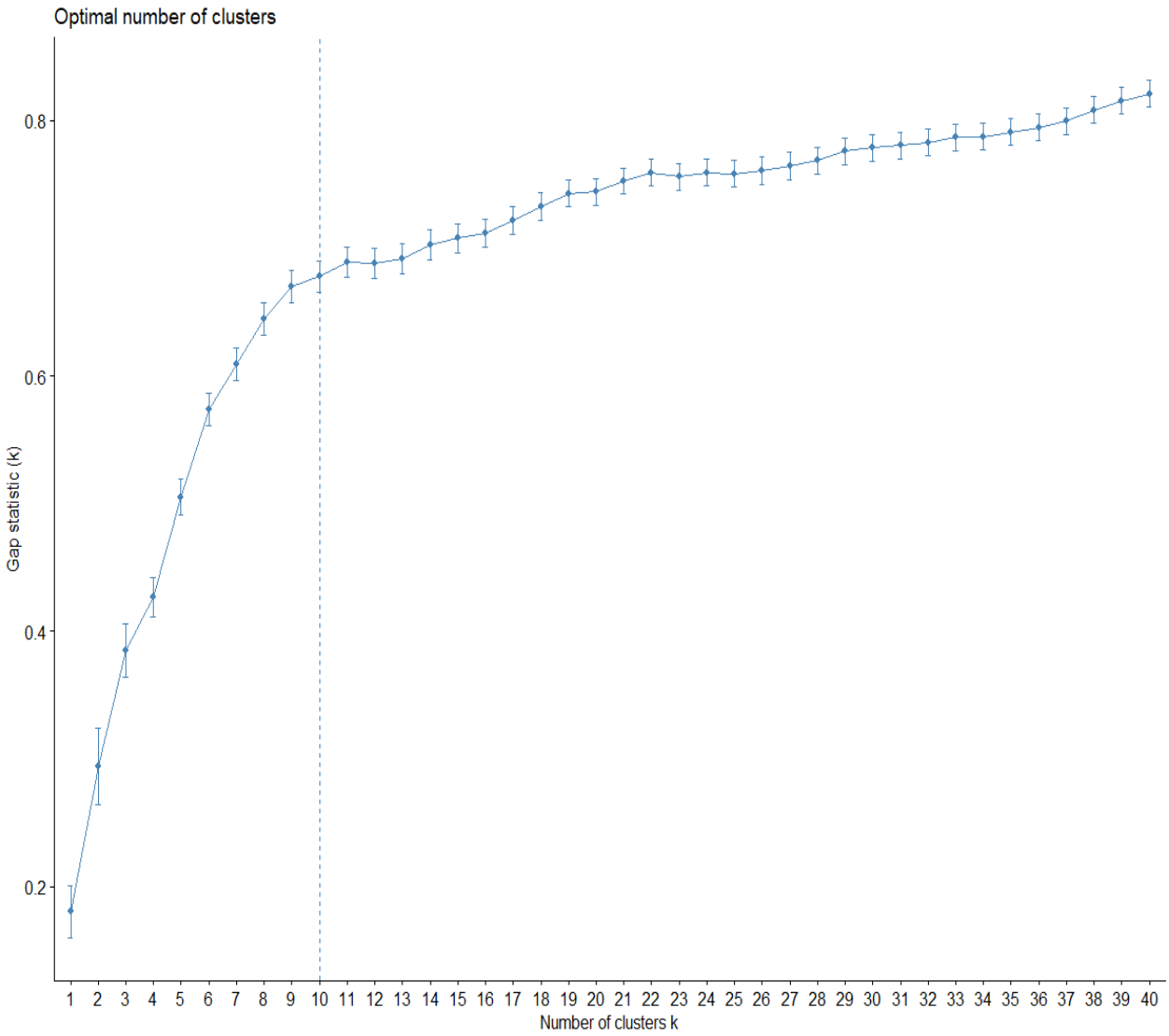


Figure 3.26: gap statistic value for different number of clusters

After having examined the obtained results with Matlab and R-Studio, it is convenient to make a brief comparison between them. First, as it is possible to note from the computation of the quality indexes, there is never a purely correct answer to a clustering problem. Different parameters provide different results and a compromise between them must be always found, without focusing too much on one of them but rather searching for the optimal tradeoff solution. For instance, with R-Studio a minimum number of clusters $K = 31$ has been determined as a good compromise between the Dunn index value and the error on the load duration curves. The same considerations can be made after the computation of the Davis-Bouldin index and the gap-statistic index. This result is coherent with the one obtained with Matlab. Indeed, as shown by the Dunn index plot (figure 3.15), the optimal number of clusters found with this software is $K = 35$. Moreover, the computation of the quality indexes has shown that, with a hierarchical clustering algorithm, the necessary number of clusters is lower than the one with a partitional clustering algorithm. However, the need to correctly represent the load duration curve increases this number up to the already determined value.

The previous comparison and the different results show that, with both softwares, the centroids of the obtained clusters (typical load days) are able to well-represent the behavior of the load along the year, leading to a small error on the load duration curve and on its total area. To properly remark this conclusion, it is possible to check if the minimum/maximum load ramps are still present after having clustered the load curves. For this purpose, the following code can be exploited:

```
ramp=zeros(365,24);
for ii=1:365
     for jj=2:24
          ramp(ii,jj)=data(ii,jj)-data(ii,jj-1);
     end
end
ramp(1,1)=data(365,24)-data(1,1);
for ii=2:24
     ramp(ii,1)=data(ii,1)-data(ii-1,24);
end
ramp_rec=zeros(36,24);
for ii=1:36
     for jj=2:24
          ramp_rec(ii,jj)=c(ii,jj)-c(ii,jj-1);
     end
end
ramp_rec(1,1)=c(36,24)-c(1,1);
```

```
for ii=2:24
     ramp_rec(ii,1)=c(ii,1)-c(ii-1,24);
end
max_ramp=max(max(ramp));
max_ramp_rec=max(max(ramp_rec));
min_ramp=min(min(ramp));
min_ramp_rec=min(min(ramp_rec));
```

(3.11)

The previous Matlab code computes a matrix "ramp", which reports the load ramp values for every hour of the day. More precisely, the position i,j of this matrix contains the load ramp between hours j and $j-1$, on the day i. Then, using the 36 obtained typical days, another matrix "ramp_rec" is computed with the same logic. Successively, the maximum positive and negative values are found for both matrixes, to determine if the maximum load ramps are preserved in the clustered model. By running code (3.11), the following result is obtained:

	Maximum positive ramp [MW]	Maximum negative ramp [MW]
Original dataset	1.031×10^4	-0.612×10^4
Clustered dataset	1.056×10^4	-1.811×10^4

Table 3.1: Maximum positive and negative ramps for original and clustered dataset

As it is possible to observe from table 3.1, similar values of the maximum ramps are obtained after the clustering process. More precisely, it is possible to compute the percentage of error as:

$$E_{ramp,\%} = \frac{|ramp_{original} - ramp_{clustered}|}{|ramp_{original}|} * 100$$

(3.12)

The computation of the previous error index using the values from table 3.1 gives $E_{ramp\ pos,\%} = 2.45\%$ and $E_{ramp\ neg,\%} = 66\%$. While the error on the maximum positive ramp is very low, the error on the maximum negative ramp is quite high. Anyway, the maximum negative ramp after clustering is higher (more negative) than the one before clustering. This can be a conservative hypothesis (assuming that the load varies more than its actual variation) not to undersize machines and components and to make worst-case scenario economical evaluations. On the other

side, if this error must be reduced, it is necessary to include constraints in the clustering algorithm (for instance: impose a maximum negative ramp rate). This approach is not easy at all since it need clustering algorithms which consider the chronological occurrence of data points, as it will be explained in the next section.

The previous comparison shows that there is never a purely correct answer to a clustering problem. Different parameters provide different results and a compromise between them must be always found, without focusing too much on one of them but rather searching for the optimal tradeoff solution. Moreover, the clustering algorithms need a starting condition to perform and some crucial decisions are taken during operations. The facts that the starting condition is typically random and that the merging/splitting decisions are irreversible and can be affected by errors highlight the importance of interpreting the results. Due to this, data scientists are currently studying and implementing new clustering techniques, addressing all the issues that are still present in the clustering process. In this perspective, one of these issues will be examined in the next section, and two solutions under development will be briefly discussed.

3.3 Future developments

To conclude this thesis, one last aspect about time series clustering must be analyzed. More precisely, this aspect is related to the chronological ordering of clustered time series. Indeed, in most capacity expansion models the representative days or weeks are chosen using classical clustering techniques such as K-means or hierarchical clustering [51]. This approach works well only if the considered load/generation profile has a repetitive daily pattern along the year or if it exhibits a certain seasonality. However, with the penetration of renewable energy sources in the generation area, this is no more true and the extremely dynamic behavior of generation cannot be properly captured by representative days. Moreover, in parallel with the growth of fluctuating renewable power generation, the increasing role of energy storage in power systems also questions the use of representative days in capacity expansion models [52]. For instance, the use of inter-day ESS is crucial in Europe when the weather conditions reduce the wind and PV production to a minimum. Anyway, adopting representative days it is possible to keep track of the energy level of only the intra-day ESS, while for the inter-day ESS this is not possible, since it is not sure that the transition from a typical day to another one represents well the actual transition from one day to another. Due to all these reasons, many researchers are studying a strategy to cluster the time periods of a capacity expansion model while keeping, as much as possible, the chronological information of the time-

dependent parameters throughout the whole planning horizon. In this way, the resulting capacity expansion model can capture the longer dynamics of renewable power generation and properly model the energy conservation constraints of interday storage. In this conclusion, two main methods for clustering data preserving their chronological features are illustrated. The first one is related to whole time series clustering and determines clusters in which the belonging time series are adjacent in time, avoiding to obtain clusters containing extremely time-separated load curves, even if with similar shape. For instance, in this way the load curves of January cannot be clustered with load curves of July, thus the longer dynamics of the load can be analyzed though the typical days and intraday storage can be included in the model. Instead, the second proposed method exploits an optimization problem to chronologically order the already selected representative days. In the following, for the sake of simplicity, the two strategies will be briefly discussed while their implementation is left as a possible deepening.

The first proposed method has been presented by Salvador Pineda and Juan M. Morales in [53]. The adopted approach is based on the classical Ward hierarchical algorithm, with an important variation in the fourth task. Indeed, the main steps to be performed are:

1. Set the initial number of clusters n equal to the total number of objects N.
2. Determine the centroid of each cluster as the mean value of the cluster itself.
3. Compute the distance between each pair of clusters according to Ward's method.
4. Merge the two closest *adjacent* clusters.
5. Update $n = n - 1$
6. If a predefined condition is satisfied (for instance, the number of clusters is equal to a desired value) go to step 7, otherwise go to step 2.
7. Determine the centroid of each cluster as the cluster medoid.

As it is possible to note, now only adjacent clusters can be merged. Two clusters C_i, C_j are defined adjacent if C_i contains a time instant that is chronologically consecutive to

a time instant in C_j. In this way, only the load curves of consecutive days will be merged, thus the resulting cluster preserves the chronological feature of clustered data and allows to capture the long-time dynamic of the load.

The second proposed method has been presented by Bram van der Heijde et al. in [54]. After having selected the representative days, the adopted approach exploits an optimization problem to chronologically order them in such a way to preserve the time features of the original dataset. This problem is formulated as a mixed integer quadratic problem (MIQP) as follows:

$$\min_w \{\sum_{s\in S}\sum_{d\in D}\sum_{t\in T}(f_s(d,t)-g_s(d,t))^2\} \tag{3.13}$$

s.t.:

$$\sum_{d\in D} w(d,r) = n(r) - 1, \forall r \in R \tag{3.14}$$

$$w(d, r = d) = 1, if\ d \in R \tag{3.15}$$

$$\sum_{r\in R} w(d,r) = 1\,, \forall d \in D \tag{3.16}$$

$$\sum_{r\in R} w(d,r) * f_s(r,t) = g_s(d,t), \forall d \in D, \forall t \in T \tag{3.17}$$

In the previous equations, d is the considered day of the year belonging to $D = [0,1,\dots,364]$ while t is the time-step during the day belonging to $T = [0,\dots,n_{steps}-1]$. In case of hourly values $T = [0,1,\dots,24]$. Moreover, r is one of the days belonging to the determined set of representative days R while $w(d,r)$ is a binary variable equal to 1 if the typical day r is selected for representing day d and

equal to 0 otherwise. Eventually, $n(r)$ denotes the number of repetitions (weight) of the representative day r (number of participants in the cluster which centroid is r), s is the index of all time series in set S with related value $f_s(d,t)$ and $g_s(d,t)$ is the reconstructed year, obtained by substituting to a day d its representative day r. Moreover, different constraints are defined: equation (3.15) ensures that the typical days are selected to represent themselves in the reconstructed year. For instance, if day 3 of the year is selected as typical day, it must represent day 3 too in the reconstructed year. This implies that the total number of repetitions of the representative day r in the rest of the year is $n(r) - 1$, as shown by equation (3.14). Moreover, equation (3.16) indicates that to every day of the year must be assigned exactly one representative days and, finally, equation (3.17) assign the time series data $f_s(r,t)$ of the representative day r to the reconstructed time series $g_s(d,t)$. Figures 3.27 and 3.28 show graphically the above-described process:

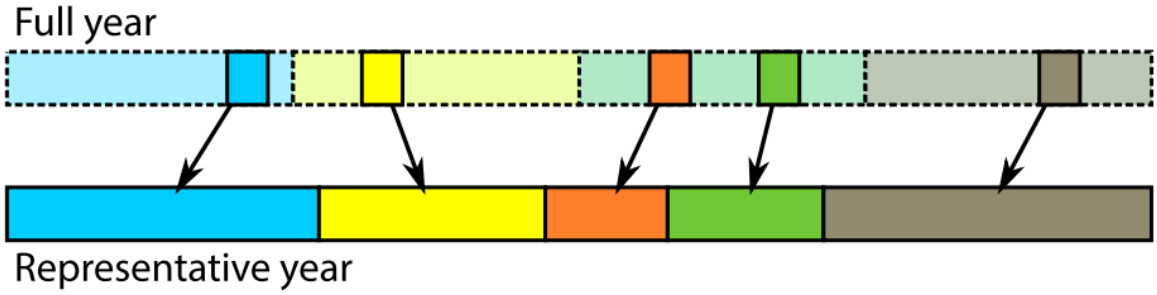


Figure 3.27: objective of the second proposed approach

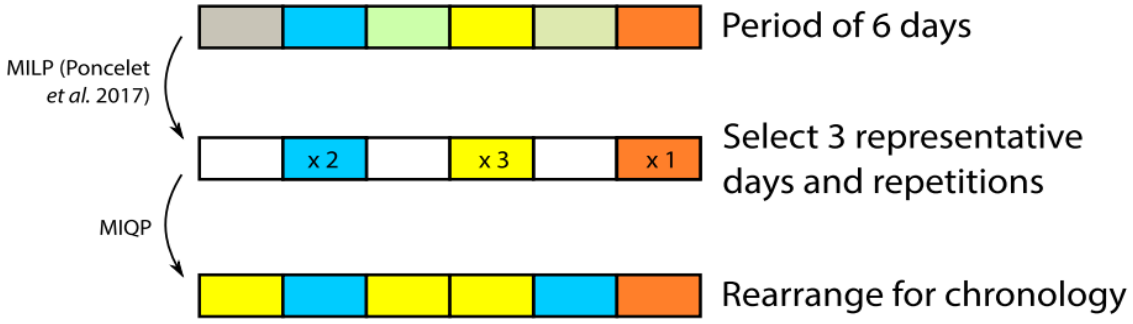


Figure 3.28: working principle of the second proposed approach

Figure 3.28 shows an example of a "year" consisting of 6 different days. The year must be represented by 3 representative days. The first step is to select which days are optimal to represent the full dataset and how many times they must be repeated to end up with the same total duration. This can be done though whichever clustering algorithm, as already discussed in the previous chapter. Successively, the proposed MIQP orders the days in a new chronology that minimizes the error with the original data sets. In this way, the resulting reconstructed year preserves the chronological feature of original data and allows to capture the long-time dynamic of the load.

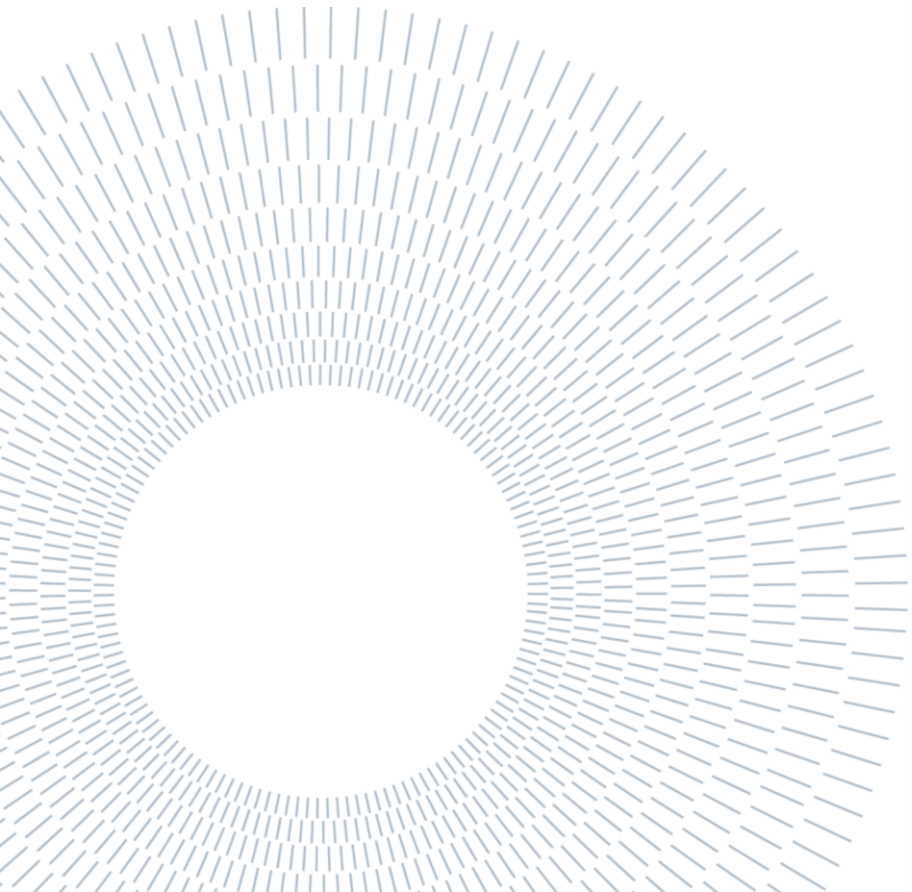
4. Conclusion

The purpose of this thesis was to highlight the extreme importance that clustering has in everyday life. Indeed, the introduction has reported several examples and applications of the clustering process in many different research areas and subjects. Moreover, chapter 1 has provided the theoretical knowledge and support necessary to fully understand what clustering is and how it is implemented, deepening all the aspects related to its input parameters such as the selected similarity measure or the concept of time series. Chapter 2, instead, has focused on the three clustering algorithms employed in this thesis (k-means, k-medoid, Ward hierarchical algorithm), explaining their working principle, their features and their main advantages and disadvantages. For the sake of clearness, these last are listed in a summarizing table at the end of this conclusion. Moreover, chapter 2 has examined some indexes to determine the optimal number of clusters and to check the quality of the clustering results. Chapter 3, instead, has discussed the practical implementation of the clustering algorithms in Matlab and R-studio, applying the clustering process to an electrical engineering problem to determine the so-called representative load days. These days are extremely important to reduce the computational time of many algorithms which exploit the load curves to make predictions or to size electrical components. Indeed, by considering only the representative days instead of the original curves, coherent results can be achieved in a fraction of the required computational time. This shows that clustering is essential also in electrical engineering or more generally in all the scientific areas in which big datasets are used as input of subject-specific algorithms. The goodness of the clustering model has been estimated through the computation of the presented quality indexes, and an optimal number of clusters has been found. Eventually, a comparison between the obtained results has been examined, showing that even if they are coherent between different softwares, there is never a purely correct answer to a clustering problem. Different parameters provide different results and a compromise between them must be always found, without focusing too much on one of them but rather searching for the optimal tradeoff solution.

Again and for the last time, time series clustering has gained an extreme importance in the last years, due to its capability to reduce the computational time of many algorithms or machine-learning models. Nowadays, it is part of many research areas that try to improve it by adding specific features, such as the chronological order preservation or even more. This thesis has provided a solid theoretical basis for understanding and implementing the main time-series clustering algorithms, applying them to a real-life electrical engineering problem by determining the representative load days, useful in many planning, sizing and development algorithms. To conclude, as already mentioned, the following table reports the main advantages and disadvantages of the three employed clustering algorithms, to have a quick comparison between them.

	K-means	**K-medoids**	**Ward hierarchical**
Advantages	Low computational complexity Easy and simple to implement Quite scalable Efficient in clustering big datasets	Low complexity (higher than k-means) Quite scalable Less sensitive to noise and outliers	No need to define a priori the number of clusters Capability to treat noisy data
Disadvantages	Cannot be applied if categorical variables are present Need to select a priori the number of clusters Sensitive to noise and outliers	Need to select a priori the number of clusters Sensitive to noise Need a scalability improvement for clustering big datasets	Crucially based on the definition of merging/splitting point Not very scalable due to its computational complexity

Table 4.1: Advantages and disadvantages of the employed clustering algorithms



Bibliography

[1] Punj G, Stewart DW. *Cluster Analysis in Marketing Research: Review and Suggestions for Application.* Journal of Marketing Research. 1983;20(2):134-148. doi:10.1177/002224378302000204, 1983.

[2] M Aitken, Philip Brown, Christine Buckland, H.Y. Izan, Terry Walter. *Price clustering on the Australian Stock Exchange.* Pacific-Basin Finance Journal, vol. 4, pp. 297-314, 1996.

[3] J. Oyelade, I. Isewon, F Oladipupo, et al. *Clustering Algorithms: Their Application to Gene Expression Data.* Bioinform Biol Insights, vol. 10, pp. 237-253, 2016.

[4] P. Berkhin. *A Survey of Clustering Data Mining Techniques.* In: Kogan J., Nicholas C., Teboulle M. (eds) Grouping Multidimensional Data. Springer, Berlin, Heidelberg, 2016.

[5] S. R. A. F. Yazdi. *Clustering of large time series datasets using a multistep approach*. University of Malaya, 2013.

[6] Goddard, Steve & Weitl-Harms, Sherri & Reichenbach, Stephen & Tadesse, Tsegaye & Waltman, William. *Geospatial Decision Support for Drought Risk Management*. Communications of the ACM. 46. 10.1145/602421.602442, 2001.

[7] D. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511574931, 1997.

[8] Bar-Joseph, Gerber, Gifford, Jaakkola, & Simon et al. *Transcriptional Regulatory Networks in Saccharomyces cerevisiae.* In: Science, vol. 298, pp. 799-804, 2002.

[9] Mori, Takaki & Uehara, Kuniaki. *Extraction of primitive motion and discovery of association rules from motion data*. 200 - 206. 10.1109/ROMAN.2001.981902, 2001.

[10] R. Honda, S. Wang, T. Kikuchi, et al. *Mining of Moving Objects from Time-Series Images and its Application to Satellite Weather Imagery*. Journal of Intelligent Information Systems 19, pp. 79–93, 2002.

[11] Vuori, Vuokko et al. *Influence of erroneous learning samples on adaptation in on-line handwriting recognition.* Pattern Recognit. 35, pp. 915-925, 2002.

[12] V. Niennattrakul, C. Ratanamahatana. *Clustering Multimedia Data Using Time Series*. Hybrid Information Technology, International Conference vol. 1, pp. 372-379. 10.1109/ICHIT.2006.253514, 2006.

[13] M. Gavrilov, D. Anguelov, P. Indyk, R. Motwani. *Mining the stock market: Which measure is best.* Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000.

[14] V.Kavitha, M. Punithavalli. *Clustering Time Series Data Stream – A Literature Survey*. International Journal of Computer Science and Information Security, vol.8, 2010.

[15]. R.Sangeeta, S. Geeta. *Recent Techniques of Clustering of Time Series Data: A Survey.* International Journal of Computer Applications, vol. 52. 1-9. 10.5120/8282-1278, 2012.

[16] Sipser, Michael. *Introduction to the Theory of Computation. Course Technology Inc.* ISBN 0-619-21764-2, 2006.

[17] D. Arthur, S. Vassilvitskii. *K-means++: The Advantages of Careful Seeding.* In: Proceedings of the Eighteenth Annual ACM-SIAM Symposiumon Discrete Algorithms, 2007.

[18] URL https://developers.google.com/machine-learning/clustering/overview.

[19] A. Marliani. *Serie storiche*, chapter 3: *metodi di scomposizione*, Università di Firenze, 2014.

[20] Makridakis, Wheelwright, Hyndman. *Forecasting Methods and Applications*, John Wiley & Sons Inc, 1997.

[21] P.J. Brockwell, R. A. Davis. *Introduction to Time Series and Forecasting*. Springer-Verlag, New York, 1996.

[22] Agronomischer, Zeitreihen. *Time Series Clustering in the Field of Agronomy*. Technische Universitat Darmstadt; 2013

[23] B. Derya. *Data Mining: Methods, Applications and Systems*. IntechOpen, ISBN 978-1-83968-319-0, 2021.

[24] P. Esling, C. Agon. *Time-series data mining*. ACM Computing Surveys (CSUR), 2012.

[25] R. Agrawal., C. Faloutsos, Swami. *Efficient Similarity Search in Sequence Databases.* 4th International Conference, 1993.

[26] S. Soheily-Khah. *Generalized k-means based clustering for temporal data under time warp*. Artificial Intelligence course. Universite Grenoble Alpes, 2016.

[27] E. Keogh., S. Kasetty. *On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration*. Data Mining and Knowledge Discovery, 2003.

[28] K. Kalpakis, D. Gada, V. Puttagunta. *Distance measures for effective clustering of ARIMA time-series.* Proceedings 2001 IEEE International Conference on Data Mining; 2001.

[29] Keogh, Eamonn & Ratanamahatana, Chotirat. *Exact indexing of dynamic time warping.* Knowledge and Information Systems. 7, 2005.

[30] R. Bellman, R. Kalaba. *On adaptive control processes*. Automatic Control, IRE Transactions on, vol. 4, no. 2, 1959.

[31] H. Sakoe, S. Chiba. *Dynamic programming algorithm optimization for spoken word recognition.* Acoustics, Speech and Signal Processing, IEEE Transactions on, vol 26, no. 1, 1978.

[32] A. Efrat, Q. Fan, S. Venkatasubramanian. *Curve matching, time-warping, and light fields: New algorithms for computing similarity between curves*. J. Math. Imaging Vis., vol. 27, no. 3, 2007.

[33] A. Corradini. *Dynamic time warping for off-line recognition of a small gesture vocabulary.* RATFG-RTS '01: Proceedings of the IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems. IEEE Computer Society, 2001.

[34] V. Niennattrakul, C. A. Ratanamahatana. *On clustering multimedia time series data using k-means and dynamic time warping* Multimedia and Ubiquitous Engineering, 2007.

[35] Z. Zhang, K. Huang, T. Tan. *Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes*. ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition. IEEE Computer Society, 2006.

[36] J. Vial, H. Nocairi, P. Sassiat, S. Mallipatu, G. Cognon, D. Thiebaut, B. Teillet, D. Rutledge. *Combination of dynamic time warping and multivariate analysis for the comparison of comprehensive two-dimensional gas chromatograms application to plant extracts*. Journal of Chromatography A, 2008.

[37] S. Salvador, P. Chan. *Toward accurate dynamic time warping in linear time and space.* Intell. Data Anal., vol. 11, no. 5, 2007.

[38] P. Senin. *Dynamic Time Warping Algorithm Review.* Information and Computer Science department, University of Hawaii at Manoa, 2008.

[39] S.W. Kim, Sanghyun Park,W. Chu. *An index-based approach for similarity search supporting time warping in large sequence databases*. Proceedings 17th International Conference on Data Engineering, 2001.

[40] J. B. MacQueen. *Some methods for classification and analysis of multivariate observations.* In: L. M. Le Cam & J. Neyman (Eds.), Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, Vol. 1, University of California Press, 1967.

[41] X. Jin., J. Han. K-*Means Clustering.* In: Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA, 2011.

[42] Elbiomy, Alaa & Mousa, Salwa & Abo-Hussien, Amina. *A comparison study of k-means and k-medoids Algorithms With an application on COVID-19 Data as an example*. 2022.

[43] J. H. Ward. *Hierarchical Grouping to Optimize an Objective Function*. Journal of the American Statistical Association, 1963.

[44] R. L. Thorndike. *Who Belongs in the Family?* Psychometrika, 1953.

[45] P. Rousseeuw. *Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis.* Comput. Appl. Math. 20, 53-65. Journal of Computational and Applied Mathematics. 20. 53-65. 10.1016/0377-0427(87)90125-7, 1987.

[46] R. Tibshirani, G. Walther, T. Hastie. *Estimating the number of clusters in a dataset via the gap statistic.* Department of Health Research and Policy and Department of Statistics, Stanford University, Stanford, CA 94305, USA, 2001.

[47] J. C. Dunn. *Well-Separated Clusters and Optimal Fuzzy Partitions.* Journal of Cybernetics, 4:1, 95-104, DOI: 10.1080/01969727408546059, 1974.

[48] D. L. Davies, D. W. Bouldin. *A Cluster Separation Measure.* In: IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-1, no. 2, pp. 224-227, DOI: 10.1109/TPAMI.1979.4766909, 1979.

[49] URL https://www.entsoe.eu/data/power-stats .

[50] URL https://it.mathworks.com/matlabcentral/fileexchange/42199-modified-generalized-dunn-s-index

[51] M. S. ElNozahy, M. M. A. Salama, R. Seethapathy, *A probabilistic load modelling approach using clustering algorithms.* In Proc. IEEE Power Energy Soc. General Meeting, 2013, pp. 1–5.

[52] M. Beaudin, H. Zareipour, A. Schellenberglabe, W. Rosehart, *Energy storage for mitigating the variability of renewable electricity sources: An updated review.* Energy Sustain. Develop., vol. 14, no. 4, pp. 302–314, 2010.

[53] S. Pineda, J. M. Morales. *Chronological Time-Period Clustering For Optimal Capacity Expansion Planning With Storage.* In IEEE Transactions on Power Systems, vol. 33, no. 6,, pp. 7162-7170, 2018.

[54] B. van der Heijde, A. Vandermeulen, R. Salenbien, L. Helsen. *Representative days selection for district energy system optimisation: a solar heating system with seasonal storage.* In Applied Energy, vol. 248, pp. 79-94, 2019.

A. Appendix A

K-means Matlab implementation

```
%z-normalization to be invariant to offsets and scaling
for i=1:365
     for j=1:24
          data_norm(i,j)=(data(i,j)-
          mean(data(i,:)))/std(data(i,:));
     end
end
n_clust=36; %desired number of clusters
%clustering
[idx,c,sumd,d]=kmeans(data_norm,n_clust,'Distance',
'Euclidean','replicates',5);
%clusters plot
for i=1:n_clust
     a=find(idx==i);
     figure
     plot(data_norm(a(1,1),:));
     hold on
     for j=2:length(a)
          plot(data_norm(a(j,1),:));
     end
     plot(c(i,:), 'LineWidth', 3);
     hold off
end
%comparison of load duration curves (real vs
reconstructed)
%real ldc
dur_data_real=reshape(data_norm, [], 1);
dur_curve_real=sort(dur_data_real, 'descend');
plot(dur_curve_real);
%dur_data=reshape(data, [], 1);
%dur_curve_non_norm=sort(dur_data, 'descend');
```

```
%weight vector for every typical day-->weight=number of
days in cluster
w=zeros(1,n_clust);
for i=1:n_clust
     a=find(idx==i);
     l=length(a);
     w(1,i)=l;
end
dur_data_rec=zeros(8760,1);
temp=zeros(1,24);
temp=c(1,:);
dur_data_rec(1:w(1)*24,1)=(repmat(temp, 1, w(1)))';
jj=w(1)*24+1;
for i=2:n_clust
     temp=c(i,:);
     dur_data_rec(jj:jj+w(i)*24-1,1)=(repmat(temp, 1,
     w(i)))';
jj=jj+w(i)*24;
end
dur_curve_rec=sort(dur_data_rec, 'descend');
plot(dur_curve_rec);
hold on
plot(dur_curve_real);
hold off
edc=(1-mean(((dur_curve_real-
dur_curve_rec)./dur_curve_real).^2))*100; %percentage mean
square error
```

K-medoids Matlab implementation

```
%z-normalization to be invariant to offsets and scaling
for i=1:365
     for j=1:24
          data_norm(i,j)=(data(i,j)-
          mean(data(i,:)))/std(data(i,:));
     end
end
n_clust=36; %desired number of clusters
%clustering
[idx,c,sumd,d]=kmedoid(data_norm,n_clust,'Distance',@dtwf,
'replicates',5);
%clusters plot
```

```
for i=1:n_clust
     a=find(idx==i);
     figure
     plot(data_norm(a(1,1),:));
     hold on
     for j=2:length(a)
          plot(data_norm(a(j,1),:));
     end
     plot(c(i,:), 'LineWidth', 3);
     hold off
end
%comparison of load duration curves (real vs
reconstructed)
%real ldc
dur_data_real=reshape(data_norm, [], 1);
dur_curve_real=sort(dur_data_real, 'descend');
plot(dur_curve_real);
%dur_data=reshape(data, [], 1);
%dur_curve_non_norm=sort(dur_data, 'descend');
%weight vector for every typical day-->weight=number of
days in cluster
w=zeros(1,n_clust);
for i=1:n_clust
     a=find(idx==i);
     l=length(a);
     w(1,i)=l;
end
dur_data_rec=zeros(8760,1);
temp=zeros(1,24);
temp=c(1,:);
dur_data_rec(1:w(1)*24,1)=(repmat(temp, 1, w(1)))';
jj=w(1)*24+1;
for i=2:n_clust
     temp=c(i,:);
     dur_data_rec(jj:jj+w(i)*24-1,1)=(repmat(temp, 1,
     w(i)))';
jj=jj+w(i)*24;
end
dur_curve_rec=sort(dur_data_rec, 'descend');
plot(dur_curve_rec);
hold on
plot(dur_curve_real);
hold off
```

```
edc=(1-mean(((dur_curve_real-
dur_curve_rec)./dur_curve_real).^2))*100; %percentage mean
square error
```

B. Appendix B

Ward hierarchical algorithm R-Studio implementation

```
library(xlsx);
library(factoextra);
library(cluster);
library(tidyverse);
library(dendextend);
library(TSclust);
library(dtwclust);
library(dtw);
library(R.matlab);
data_mat=readMat('C:/Users/Simone/Desktop/data_mat.mat');
data_mat=matrix(unlist(data_mat$data), nrow=365);
data_df=as.data.frame(t(data_mat));
data_long=gather(data_df[c(1:24), c(1:365)]); #create a column vector containing all the observations (24 observations, 365 variables)
data_long$time = rep(1:24,365); #add time to data_long -> from 1 to 24, 365 times
#plot of data
data_long %>%
  ggplot(aes(x= time, y= value, color= key)) +
  geom_line( size=0.2) +
  ggtitle("Control chart sequences") +
  facet_wrap(~ key , scales = 'free_x', nrow= 2);
```

```
df_list <- as.list(utils::unstack(data_long, value ~ key)); #create a list of time series
df_list_z <- dtwclust::zscore(df_list); #z-normalization

#make from 2 to 36 clusters with HIERARCHICAL METHOD
cluster_dtw_h <-list()
for (i in 2:36)
{
  cluster_dtw_h[[i]] <- tsclust(df_list_z, type = "h", k = i,  distance = "dtw", control = hierarchical_control(method = "complete"), seed = 390, preproc = NULL, args = tsclust_args(dist = list(window.size = 5L)))
}
#example with 36 clusters
plot(cluster_dtw_h[[36]]); #dendrogram
plot(cluster_dtw_h[[36]], type = "sc"); #plot cluster partecipants
lines(cluster_dtw_h[[36]], type = "centroid"); #centroids

#make 2 to 36 clusters with PARTITIONAL METHOD-->K-MEDOID
cluster_dtw_p <-list()
for (i in 2:36)
{
  cluster_dtw_p[[i]] <- tsclust(df_list_z, type = "p", k = i,  distance = "dtw", control = partitional_control(nrep=1L), seed = 390, preproc = NULL, args = tsclust_args(dist = list(window.size = 5L)))
}
#example with 36 clusters
#plot(cluster_dtw_p[[36]]);
plot(cluster_dtw_p[[36]]); #plot cluster partecipants and centroids
```

List of Figures

Figure 0.1: Commonly used time complexity functions in algorithm analysis 4

Figure 1.1: The total Italian load curve is an example of run chart of a single variable time series 14

Figure 1.2: additive time series 17

Figure 1.3: non-additive increasing time series 17

Figure 1.4: non-additive decreasing time series 17

Figure 1.5: original time series (actual) and prediction (predicted) 19

Figure 1.6: seasonal component of the time series 19

Figure 1.7: seasonally adjusted time series (no seasonal component) 20

Figure 1.8: Selling of shampoo [liters] and moving averages in three years 23

Figure 1.9: Realization of simple random walk of length 200 29

Figure 1.10: Approach to select the correct distance measure 38

Figure 1.11: Three possible alignments (warping paths) between two time series 41

Figure 1.12: example of local cost matrix heatmap and optimal alignment path 44

Figure 1.13: three-way plot of time series alignment 49

Figure 1.14: two time series under consideration 50

Figure 1.15: global cost matrix heatmap and optimal warping path 51

Figure 1.16: optimal alignment between the two considered time series 51

Figure 1.17: example of possible step-size conditions 52

Figure 1.18: four typical step patterns in literature 53

Figure 1.19: clustering results before and after z-normalization 55

Figure 1.20: power-law distributed data and clusters centroids 57
Figure 1.21: data distribution and clusters centroids after log-transformation 57
Figure 1.22: main steps to perform clustering on a dataset..... 59
Figure 1.23: approaches to (whole) time series clustering..... 61
Figure 1.24: evaluation measures classification 67
Figure 1.25: overview of the five components of whole time series clustering 68
Figure 2.1: k-means algorithm flowchart 71
Figure 2.2: illustrative example of k-means clustering algorithm..... 72
Figure 2.3: k-medoids algorithm flowchart 76
Figure 2.4: the four possible cases of the cost function for the k-medoids algorithm . 77
Figure 2.5: example of cluster dendrogram..... 79
Figure 2.6: agglomerative hierarchical clustering example..... 82
Figure 2.7: dendrogram and Euler-Venn diagram of example reported in figure 2.6 83
Figure 2.8: example of gap function behavior for different K 86
Figure 2.9: comparison of partitions of a 4-points dataset and use of Davies-Bouldin index to compare their quality 88
Figure 3.1: examples of the considered load duration curves 92
Figure 3.2: curves of figure 3.1 after z-normalization 93
Figure 3.3: result of DTW between two of the curves reported in figure 3.2 96
Figure 3.4: cluster1 obtained by k-means 97
Figure 3.5: cluster2 obtained by k-means 97
Figure 3.6: cluster21 obtained by k-means 98
Figure 3.7: cluster1 obtained by k-medoids..... 98
Figure 3.8: cluster2 obtained by k-medoids..... 99
Figure 3.9: cluster21 obtained by k-medoids..... 99
Figure 3.10: cluster35 obtained by k-medoids..... 100
Figure 3.11: load duration curve related to the considered dataset..... 101
Figure 3.12: real and reconstructed load duration curves for K=36..... 103
Figure 3.13: focus on figure 3.12..... 104

Figure 3.14: real and reconstructed load duration curves for K=2 105
Figure 3.15: Dunn index values for different number of clusters 107
Figure 3.16: optimal alignment between time series of figure 3.3 110
Figure 3.17: three-way plot of the optimal alignment reported in figure 3.16 111
Figure 3.18: heatmap of the global cost matrix 112
Figure 3.19: overview of the obtained clusters for K=36 114
Figure 3.20: centroids (typical days) of the 36 obtained clusters 114
Figure 3.21: overview of the obtained clusters for K=2 115
Figure 3.22: centroids of the 2 obtained clusters 115
Figure 3.23: cluster dendrogram for K=36 116
Figure 3.24: Dunn index for different numbers of clusters 117
Figure 3.25: Davies-Bouldin index for different numbers of clusters 118
Figure 3.26: gap statistic value for different number of clusters 119
Figure 3.27: objective of the second proposed approach 125
Figure 3.28: working principle of the second proposed approach 125

List of Tables

Table 0.1: Some applications of time series clustering in different domains 7

Table 1.1: Shampoo selling of a company and moving averages computation 23

Table 3.1: Maximum positive and negative ramps for original and clustered dataset 121

Table 4.1: Advantages and disadvantages of the employed clustering algorithms .. 128

2. List of Symbols

Variable	Description	SI unit
y	number of years	-
n	number of input data	-
$O()$	big O notation	-
DTW	dynamic time warping	-
T_t	trend component of a time series at time t	-
C_t	cyclical component of a time series at time t	-
S_t	seasonal component of a time series at time t	-
I_t	irregular component of a time series at time t	-
T'_t	trend-cycle component of a time series at time t	-
Y'_t	seasonally adjusted time series at time t	-
MA	moving average	-
m	considered month	-

$\boldsymbol{Y}$	random variable	-
iid	independent and identically distributed	-
$\boldsymbol{\nabla}$	lag-1 operator	-
$\boldsymbol{\nabla_d}$	lag-d operator	-
$\boldsymbol{s(y_i, y_j)}$	similarity measure between y_i, y_j	-
$\boldsymbol{d(y_i, y_j)}$	dissimilarity measure between y_i, y_j	-
$\boldsymbol{D(y_i, y_j)}$	distance measure between y_i, y_j	-
$\boldsymbol{D_E(Y_t, Y_t')}$	Euclidean distance between Y_t, Y_t'	-
$\boldsymbol{D_{L_p}(Y_t, Y_t')}$	Minkowski distance between Y_t, Y_t'	-
$\boldsymbol{DTW(Y_i, Y_j)}$	dynamic time warping between Y_i, Y_j	-
C	local cost matrix	-
$\boldsymbol{c_p}$	cost function related to a warping path	-
$\boldsymbol{y_i'}$	normalized y_i value	-
$\boldsymbol{\mu}$	mean value	-
$\boldsymbol{\sigma}$	standard deviation	-
$\boldsymbol{\widehat{R}}$	cluster prototype (whole time series)	-
$\boldsymbol{\widehat{\theta_k}}$	cluster centroid (time point)	-
$\boldsymbol{w_{nk}}$	binary variable	-

SSE	sum of squared errors	-
SAE	sum of absolute errors	-
o_j	medoid object	-
o_{random}	non-medoid object (candidate)	-
p	non-medoid object	-
$d_{min}(C_i, C_j)$	minimum distance between clusters C_i, C_j	-
$d_{max}(C_i, C_j)$	maximum distance between clusters C_i, C_j	-
$d_{mean}(C_i, C_j)$	mean distance between clusters C_i, C_j	-
$d_{average}(C_i, C_j)$	average distance between clusters C_i, C_j	-
$d_{Ward}(C_i, C_j)$	Ward distance between clusters C_i, C_j	-
S_i	average point silhouette index	-
$G(K)$	gap function	-
Q_D	Dunn index	-
Q_{DB}	David-Bouldin index	-
MPE	mean percentage error	-
PAE	percentage area error	-
$E_{ramp,\%}$	percentage error on load ramps	-

3. Acknowledgements

I would like to thank my advisor Alberto Berizzi and co-advisor Marina Petrelli for this challenge, that has given me the opportunity to make a journey in the data-science world, a very fascinating research area that I'm glad to have deepened.

Most importantly, I would like to thank my parents Attilia Ottoboni and Alberto Pivari for the love and support, also economical, that they always gave to me, without any back expectation. I love you, thanks for all. Finally, I would also like to thank my girlfriend Mariama Darboe for the emotional support and understanding during the hard periods