Politecnico di Torino

Computational linear algebra for large scale problems
HW2_Spectral clustering

Bono Giorgio s343572
Camolese Claudio s344788

Contents

1 Introduction

The goal of this project is to get familiarity with the concept of *spectral clustering*.
The spectral clustering is a technique in data clustering which is based on the spectrum of the similarity matrix of the data. It's particularly useful when you have to deal with complex and non-linear structures.

2 Mathematical approach

Given a set of data points X and a *similirarity function* which in this case is described by

$$s_{ij} = exp(-\frac{\|x_i - x_j\|}{2\sigma^2}) \in [0, 1] \tag{1}$$

we build the *similarity graph* $G = (V, E)$, where $V = \{v_1, ..., v_n\}$ denotes a non-empty set of vertices and E denotes the set of edges.
The similarity function used is create in order to manipulate the similarity between two points. If two points are exactly overlapped $s_{ij} = 1$, else, the more distant they are, the closer to zero s_{ij} is. In this project $\sigma = 1$ has been used.

Starting from this result, the *similarity matrix* $W \in \mathbb{R}^{N \times N}$, which represents the weight of the edges in the graph in a matrix form, is created. Since this similarity relation must be computed for each point with respect to all other points, the computational cost of this operation is $O(n^2)$. However, assuming that $s_{ij} = s_{ji}$, the computational cost can be decreased to $O(n^2/2)$ since the matrix W is symmetric. Another property of W, is the fact that the diagonal is always equal to 0, since a point (node) has no connections with itself.

From the matrix W, two new matrices are created.
The degree matrix $D \in \mathbb{R}^{N \times N}$ is a diagonal matrix where each element on the diagonal represents the degree of the corresponding node. The value of the associated degree is computed as:

$$D(i, i) = \sum_{j=1}^{N} W(i, j) \tag{2}$$

this means that each element $D(i, i)$ of the degree matrix is the sum of the weights of the edges connected to node i.

The second matrix is the *Laplacian matrix* $L \in \mathbb{R}^{N \times N}$ which is computed as:

$$L = D - W \tag{3}$$

Let's reflects on some key aspects of the Laplacian matrix.

- L is symmetric. Since W is symmetric and D diagonal.
- L is positive semidefinite. As consequence, all its eigenvalues are ≥ 0. This is due to the fact that given any vector $f \in \mathbb{R}^n$, we have:

$$f'Lf = \frac{1}{2} \sum_{ij=1}^{n} W_{ij}(f_i - f_j)^2 \geq 0 \tag{4}$$

- The sum of the entries in each row of the Laplacian matrix L is zero

$$\sum_{j} L(i, j) = 0 \tag{5}$$

- the smallest eigenvalue of L is $\lambda = 0$ and the corresponding eigenvector is $f = 1_n$.
- The algebraic multiplicity of the zero eigenvalue of the Laplacian matrix corresponds to the number of connected components in the graph. Since at least there is always one eigenvalue equal to zero, the graph is connected. However, if multiple eigenvalues are equal to zero, this indicates that the graph is disconnected, meaning it consists of more than one isolated component (subgraph), each of which corresponds to a distinct cluster

- the value of the eigenvalues represents the strenght of the connection.

The eigenvalues of L are really important because thanks to those, we can understand how many disconnected components we can find in our dataset. But there is more, as for eigenvalues, even the corresponding eigenvectors are really important.

Taking the eigenvectors associated to the most close-to-zero eigenvalues, we can encode the relationships between data points in a lower-dimensional space while maintaining the graph's similarity structure.

Starting from L, we can compute the smallest eigenvalues and the corresponding eigenvectors. Thanks to how many $\lambda \simeq 0$ we find, it be can deduced how many disconnected components there will be.
The corresponding eigenvectors will create a new matrix $U \in \mathbb{R}^{N \times M}$. This matrix is the representation of our initial graph in lower dimensional space. The matrix U is extremely important since, now that the dimensionality has been reduced, we can finally perform the cluster with some well-known algorithms.

The roadmap is the following:

1. compute W
2. compute L
3. find the smallest eigenvalues and the corresponding eigenvectors
4. compute U
5. compute clusters
6. plot results

2.1 Mathematical understanding of the optional part

The projects gives the opportunity to explore more this topic using some other methods.

- use L_{sym} instead of L, known as, *symmetric Laplacian matrix*, is defined as:

$$L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \tag{6}$$

  It has some important features like the fact that eigenvalues lie within the range [0, 2], and the matrix is more normalized compared to L. This leads to better numerical stability and convergence when performing eigenvalue decomposition, especially in large-scale problems.

  The transformation $D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$ enhances the stability of the eigenvector computation. This normalization effectively mitigates the influence of node degrees, ensuring that the degree variations among nodes do not disproportionately affect the spectral decomposition. As a result, the clustering obtained from L_{sym} tends to produce more balanced and meaningful clusters, especially in scenarios where node degree heterogeneity might otherwise dominate the results.

- Implementation of the *Inverse Power method* and the *Deflation Method* to compute the eigenvalues of the matrix.

2.1.1 Computation of eigenpairs

The Inverse Power Method is a powerful technique used to find the smallest eigenvalue and its corresponding eigenvector. However, the limitation of this method is that it cannot find the other eigenvalues associated with the matrix.

The objective, therefore, becomes how to find the subsequent eigenvalues and eigenvectors after having already found the smallest ones. The answer lies in the Deflation Method. This method allows for the removal of the contribution of the most recently found eigenpair from the original matrix, enabling the reapplication of the Inverse Power Method to find the next smallest eigenvalue and its corresponding eigenvector.
The Deflation Method relies on the fact that the matrix obtained by removing the contribution

of the most recently found eigenpair is similar to the original matrix. This is of fundamental importance because two similar matrices share the same eigenvalues, with the same algebraic and geometric multiplicities, so we can reproduce all the eigenvalues of a matrix thanks to this chain of operation.

However, the Deflation Method is not as straightforward to implement. While it is guaranteed that continuing to use this method will reproduce the same eigenvalues, the same cannot be said for the corresponding eigenvectors. This is because each eigenvector is associated with the matrix of that iteration, not with the original matrix. This makes the method more complex to implement.

The fundamental problem is that at each iteration of the method, a submatrix of the original matrix is considered, and consequently, the corresponding eigenvector is of smaller dimension. This reduction in dimensionality complicates the process, as the eigenvectors found at each step are no longer directly associated with the original matrix, but rather with a modified version of it.

The question then becomes how to reconstruct the information lost due to the reduction in the dimensionality of the original matrix. This is made possible by calculating coefficients that recover the lost contribution, allowing the eigenvector to be reconstructed in its original dimension. By doing so, it is possible to map the eigenvector obtained from the reduced matrix back to the full-dimensional space of the original matrix.
The main operations of this method are the following:

$$\begin{cases} P_1 = I_n - 2\dfrac{(x_1 + e_1^n)(x_1 + e_1^n)^T}{\|x_1 + e_1^n\|_2^2}, \\ B_1 = P_1 A P_1 = \begin{pmatrix} \lambda_1 & b_1^T \\ 0 & A_1 \end{pmatrix}, \\ \alpha = -\dfrac{b_1^T x_1^{(n-1)}}{\lambda_1 - \lambda_2}, \\ \begin{pmatrix} (\lambda_1 - \lambda_3)\alpha = -b_1^T P_2 \begin{pmatrix} \beta \\ x_3^{(n-2)} \end{pmatrix} \\ (\lambda_2 - \lambda_3)\beta = -b_2^T x_3^{(n-2)} \end{pmatrix}. \end{cases} \tag{7}$$

3 Code

3.1 main.m

This is the main file of the project. Here all functions are called and the plots are computed. A menu will pop up, allowing the user to select which dataset he wants to upload between three choices (Circle.mat, Spiral.mat, three_spheres_3D.mat) and also if he wants to use L or L_{sym} matrix.

3.2 knn_graph.m

The knn_graph function is used to construct a k-nearest neighbourhood similarity graph and its corresponding adjacency matrix W. This graph is a mathematical representation of data points, where each point is connected to its k-nearest neighbours based on a specified similarity measure.
Inputs: a matrix, number of k nearest neighbours to connect for each data point.
Output: the adjacency matrix W.

3.3 knn_search.m

The knn_search function is used to identify the k-nearest neighbours of a given point within a dataset. It computes the Euclidean distances between the query point and all points in the dataset and returns the indices and distances of the nearest neighbours.
Inputs: a matrix, a point, number of k nearest neighbours to find.
Output: indices and distances.

3.4 LDW.m

The LDW function constructs the degree matrix D and the graph Laplacian matrix L.
Input: W
Output: L,D.

3.5 num_connect_comp.m

The num_connect_comp function computes the number of connected components in a graph based on its graph Laplacian matrix L.
Input: L
Output: number of connected components, a diagonal matrix of eigenvalues, a matrix of eigenvectors

3.6 deflation.m

The deflation function computes a specified number of eigenvalues and their corresponding eigenvectors for a square matrix using the inverse power method combined with a deflation technique.
Inputs: a matrix, number of eigenvalues to compute
Output: a matrix of eigenvectors, a diagonal matrix of eigenvalues.
This is the most critical part of the entire process, not only because finding the correct eigenvectors is essential for accurate dimensionality reduction but also because, from a technical point of view, it was the most challenging to implement.
The equation used are the one in (7). The most difficult part was to be able to performe this method for every new eigenvector that must be computed since it requires the computation of more coefficients at every step.
Another challenge we encountered was optimizing memory usage. This issue arose when we attempted to apply our algorithm to the 3D dataset. The problem was that the algorithm tried to allocate a 20GB matrix in the computer's memory. To address this, instead of creating an initial zero matrix and filling it afterward, we leveraged MATLAB's cell function, which allows for the concatenation of matrices with different dimensions. This approach significantly reduced memory usage, enabling us to perform the computation successfully.

3.7 inverse_power_method.m

The inverse_power_method function computes the smallest eigenvalue and its corresponding eigenvector of a given square matrix. This is achieved through an iterative process that leverages the inverse of A shifted by a small parameter to enhance convergence.
In the code, a shift close to zero is used in order to avoid ill conditioning of computations
Input: a matrix
Output: smallest eigenvalue, eigenvector.

3.8 k_means.m

The k_means function performs clustering on a given dataset using the k-means algorithm with multiple random initializations to improve robustness. It finds the best clustering based on the silhouette score.
Input: a matrix
Output: number of clusters.

3.9 Instruction

Among all the uploaded files included in the zip, main.m is the one that needs to be opened. Once executed, a menu will prompt you to select the dataset to load and choose between the L or L_{sym} matrix. After the selection, the results will be plotted for each $k = 10, 20, 40$.

4 Analysis of results.m

In this section, will be presented all results obtained with different combinations of datasets, tolerance, k-nearest neighbourhood value $(10, 20, 40)$ and L or L_{sym}. All the dataset presented have some geometrical features which are complicated to be analyzed by a normal cluster algorithm

without the spectral processing.
The figure is divided in four sublots:

1. plot associated to the eigenvalues. The eigenvalues are plotted by index, so in ascending order, with respect to their value.
2. this plot is useful to have an idea of the value of the eigenvalue as percentages of the max eigenvalue plotted
3. plot of the adjacency matrix W. The adjacency matrix encodes the connections between points in the dataset.
4. plot that represents the clustering adopted by the algorithm.

4.1 Preliminary analysis

We can already have an idea of what we can expect from the plot.
We will be able to identify n clusters, where n is the number of eigenvalues close to zero. The "closeness" to zero of the eigenvalue is a condition that can be fine-tuned manually by using the *tol* parameter in the MATLAB code.

Furthermore, the plot of W will be as sparse as the value of k is low. This is because each node will only be connected to its few nearest neighbors. Therefore, there will be many zeros in the matrix.
On the contrary, if k is increased, the similarity matrix will become less sparse, with more connections between the nodes, and thus fewer zeros. This means that each point will be connected to a larger number of nodes, increasing the density of the matrix.

4.2 Circle.mat

In this section we discuss about the Circle.mat dataset. The dataset consists of three point clouds, two of which are nested within each other.

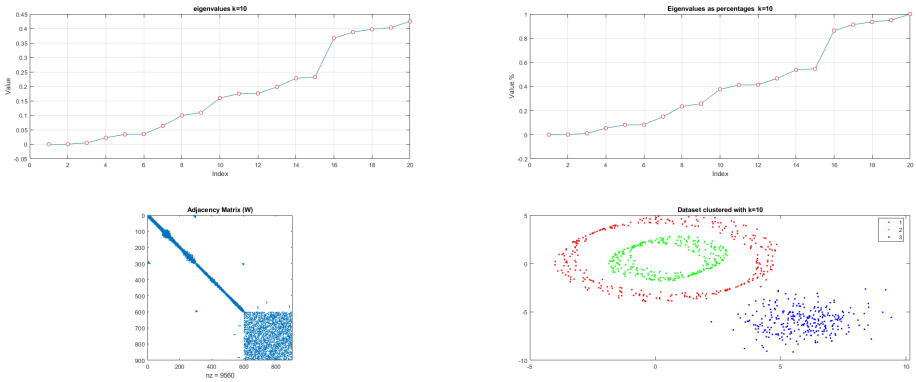

Figure 1: Circles L with k=10

The first two present an ordered geometric figure, while the last one represents a cloud with points that are disorganized and noisy. The points in the cloud are closer to each other, which results in stronger connections in the adjacency matrix, leading to a dense block of non-zero values that form a blue square. As mentioned in the preliminary analysis, as k increases, we find a less sparse adjacency matrix. This happens because, as more points are considered to assign a cluster to an object, connections between more points in the matrix are created, resulting in a less sparse appearance.

Now let's analyse the results with L_{sym}.
The results are nearly the same visually. However, what can be observed is that we can be more stringent when choosing the tolerance values. This is due to the fact that L_{sym} is numerically more robust, which leads to better representation of the connected components. This implies that we can demand more from our algorithm when selecting the tolerance parameter.

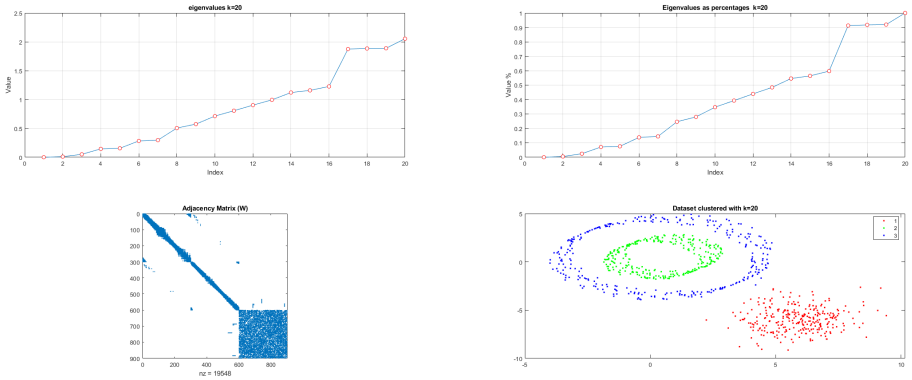

Figure 2: Circles L with k=20

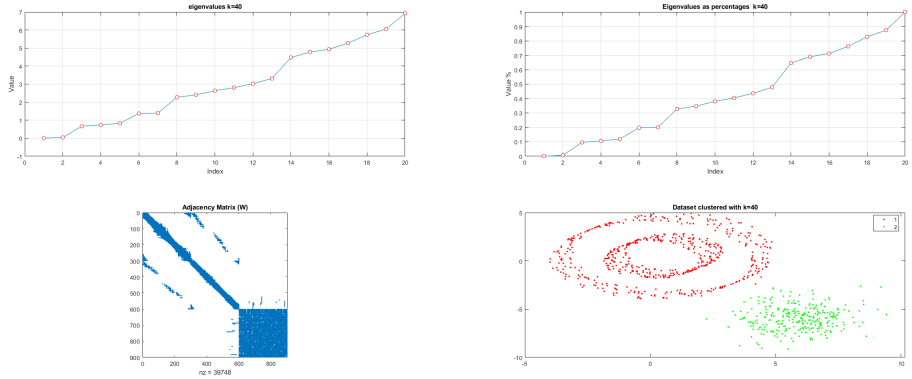

Figure 3: Circles L with k=40

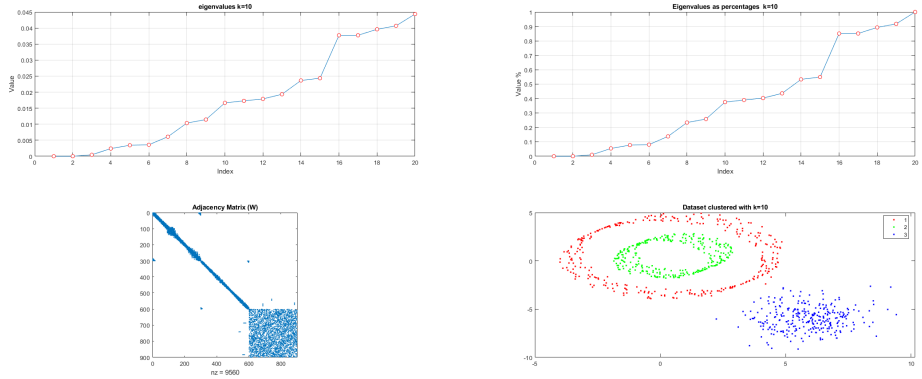

Figure 4: Circles Lsym with k=10

k	10	20	40
L	$1e^{-2}$	$1e^{-1}$	$1e^{-1}$
L_{sym}	$1e^{-3}$	$0.8e^{-2}$	$1e^{-2}$

Table 1: Circle *tol* table

Despite this, the algorithm does not allow for the identification of three clusters unless an ad hoc tolerance value of 0.7 is chosen for $k = 40$. This is because, as seen from the analysis of the eigenvalue trend, the third eigenvalue is far from being close to zero.

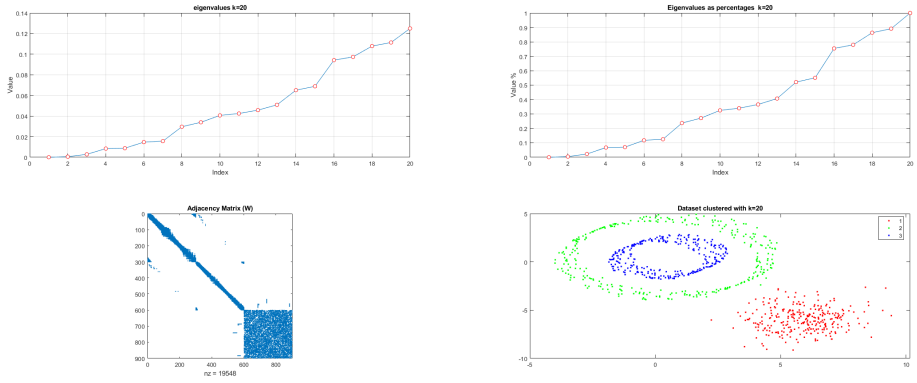

Figure 5: Circles Lsym with k=20

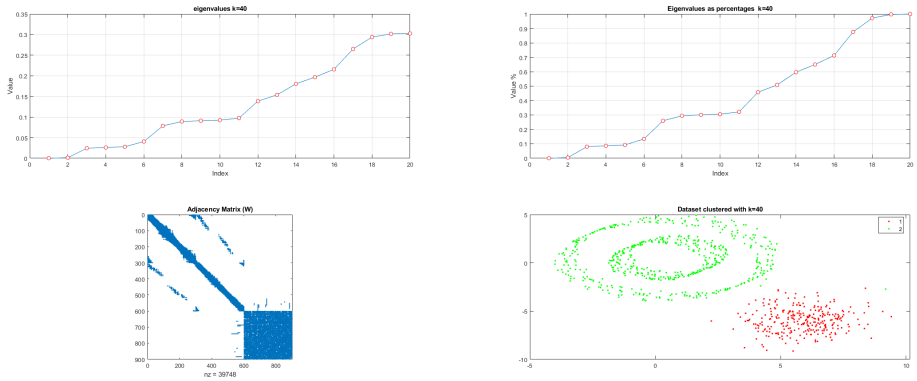

Figure 6: Circles Lsym with k=40

4.3 Spiral.mat

The dataset consists of a three-armed spiral. The figure is therefore organized in an orderly manner, without noise, and follows an ideal pattern.

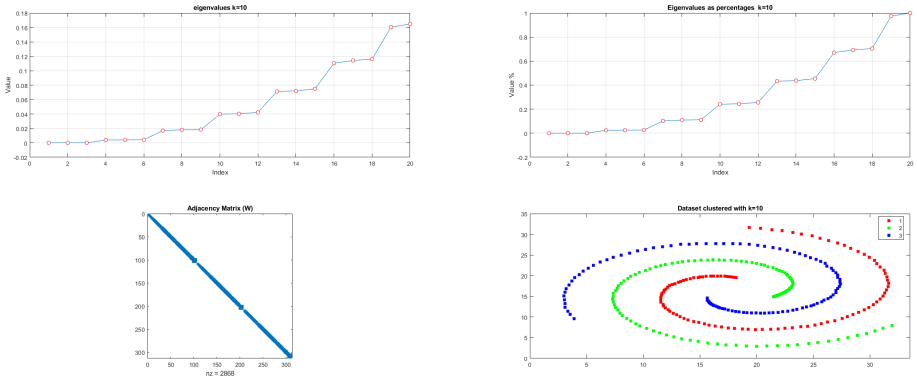

Figure 7: Spiral L with k=10

This is reflected in the adjacency graph, which is particularly sparse, resulting in a clear division of the clusters. Naturally, as k increases, the graph becomes less sparse, but the division remains clear due to the nature of the dataset.

In this case, with all the combinations, the clusters are correctly recognized even with a fairly stringent tolerance level. As previously mentioned, when using L_{sym}, it is possible to demand more in terms of tolerance.

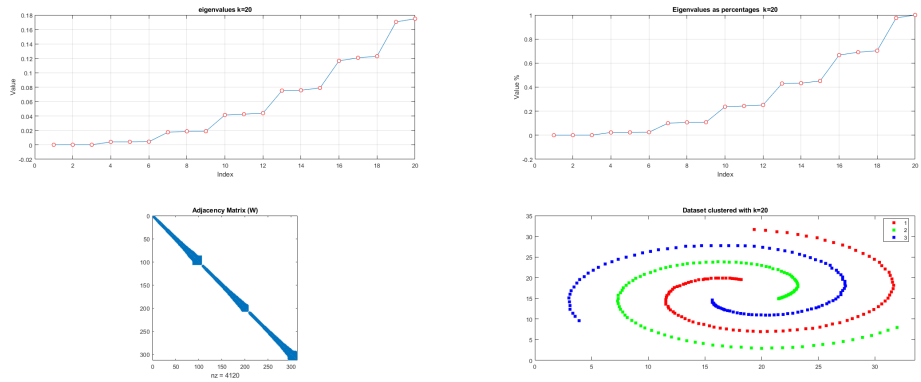

Figure 8: Spiral L with k=20

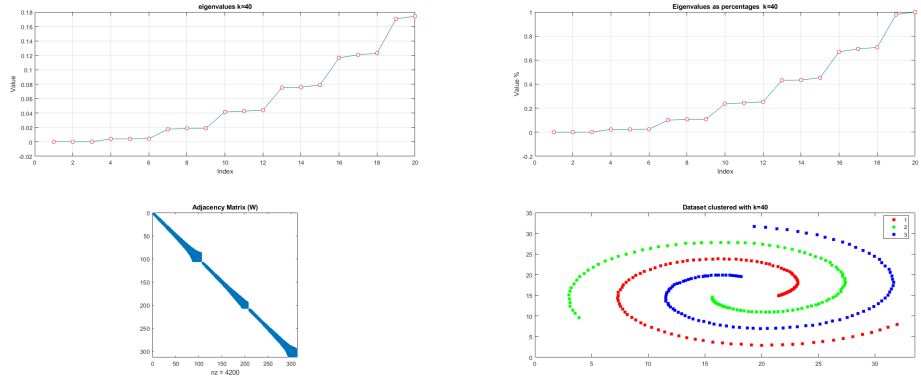

Figure 9: Spiral L with k=40

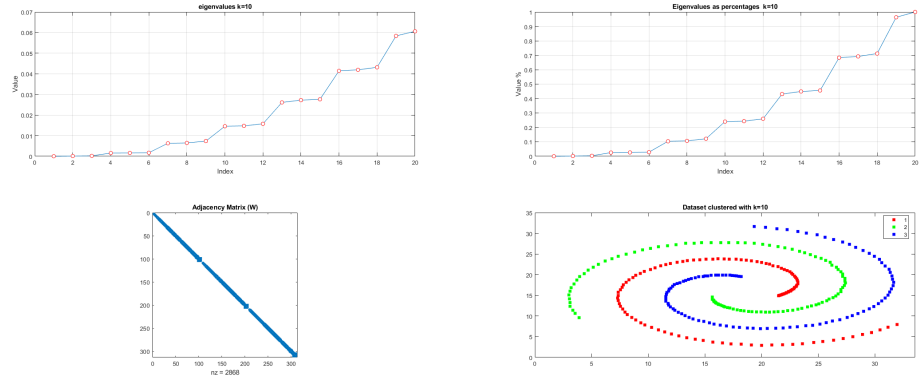

Figure 10: Spiral Lsym with k=10

k	10	20	40
L	$1e^{-3}$	$1e^{-3}$	$1e^{-3}$
L_{sym}	$1e^{-3}$	$1e^{-3}$	$1e^{-3}$

Table 2: Spiral *tol* table

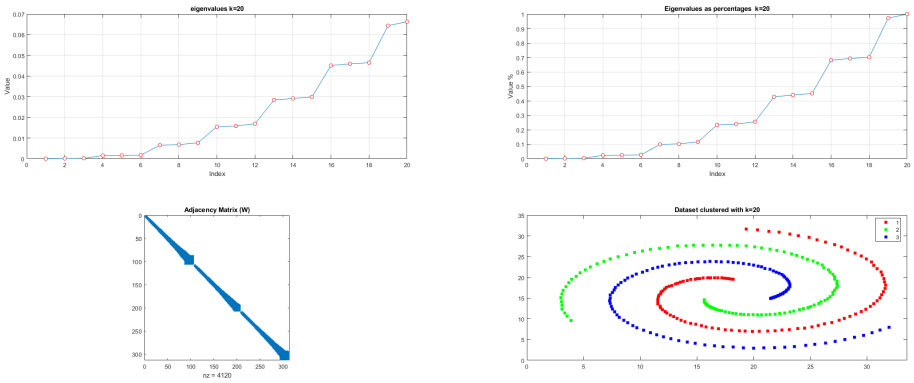

Figure 11: Spiral Lsym with k=20

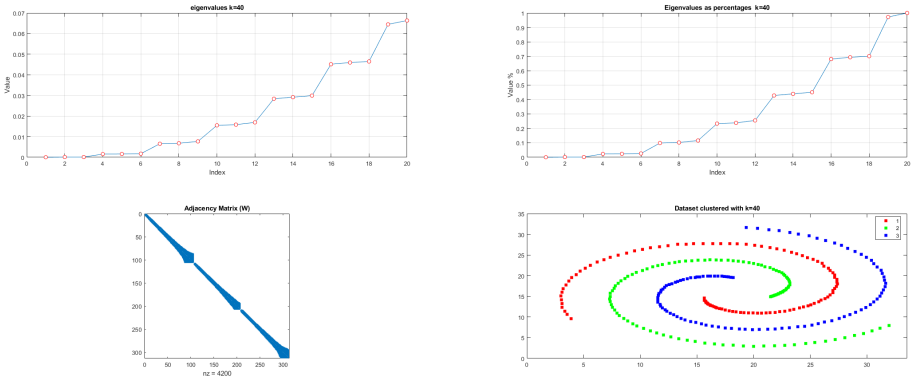

Figure 12: Spiral Lsym with k=40

4.4 3D dataset

This is a 3D dataset of our own creation to test the data in a higher-dimensional geometric space. The dataset consists of three spheres, one inside the other.

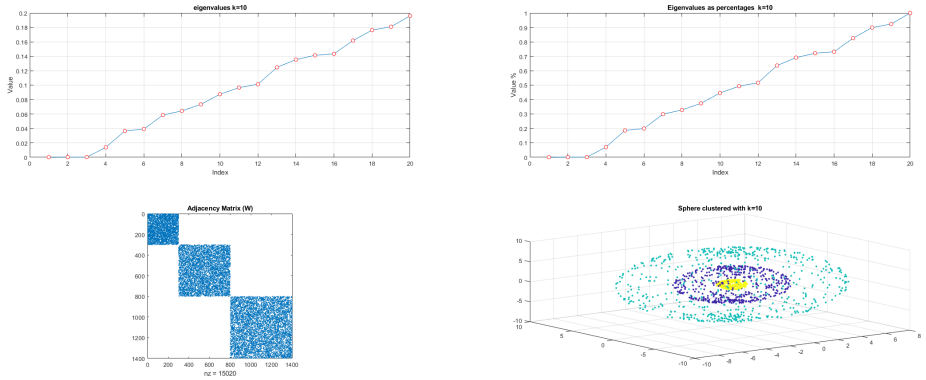

Figure 13: 3D L with k=10

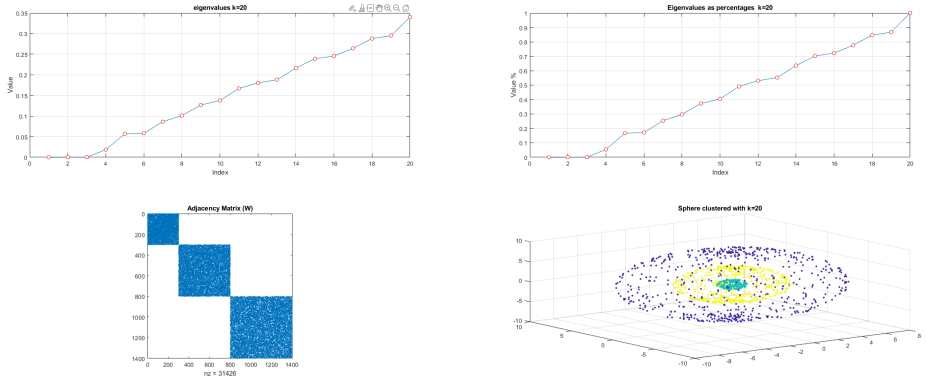

Figure 14: 3D L with k=20

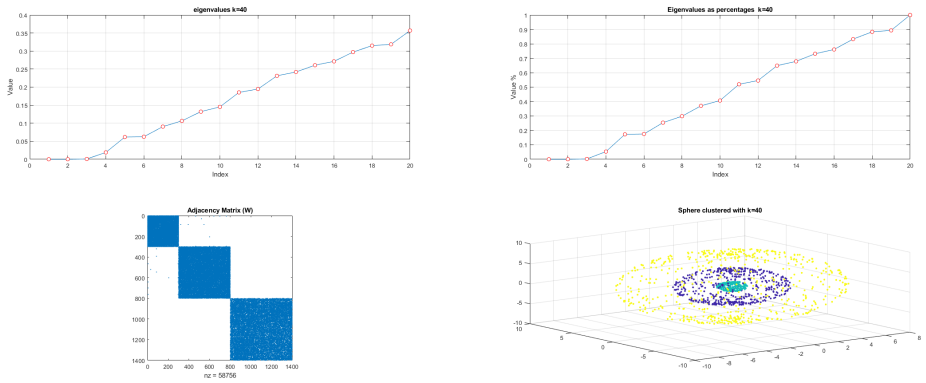

Figure 15: 3D L with k=40

As can be seen from the cluster plot, the algorithm works correctly. The adjacency matrix appears less sparse compared to (1), which is due to both the three-dimensionality, the presence of more points, and the proximity of the spheres.
Other considerations are the fact that each square block in the matrix W corresponds to one of the spherical shells in the dataset.
The size of the blocks increases as you move down the diagonal, reflecting the fact that the outer

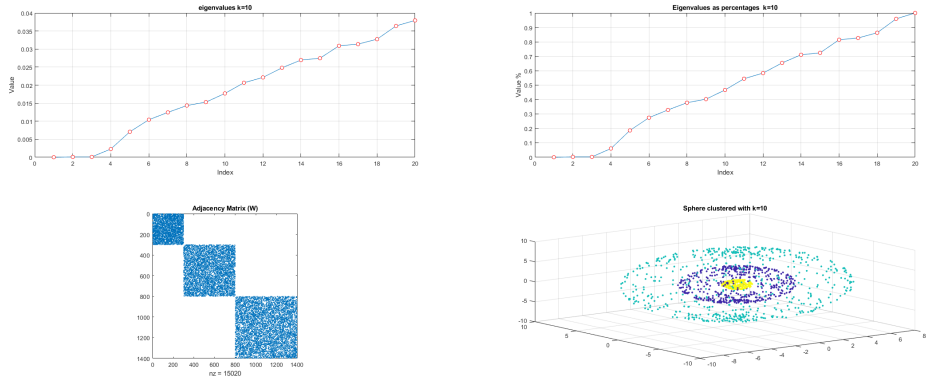

Figure 16: 3D Lsym with k=10

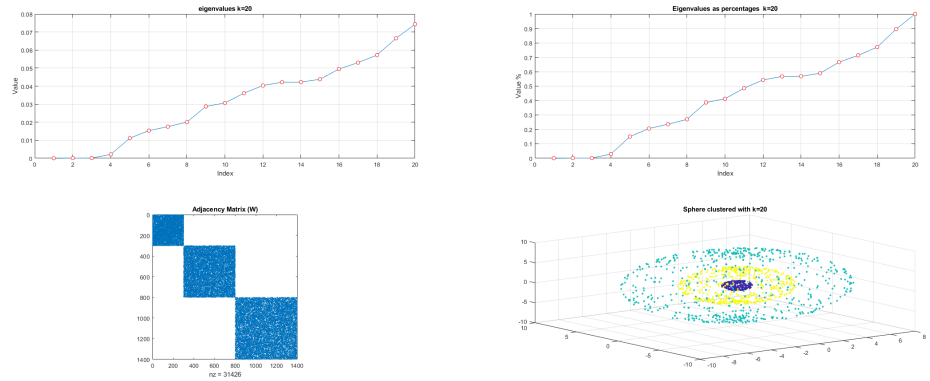

Figure 17: 3D Lsym with k=20

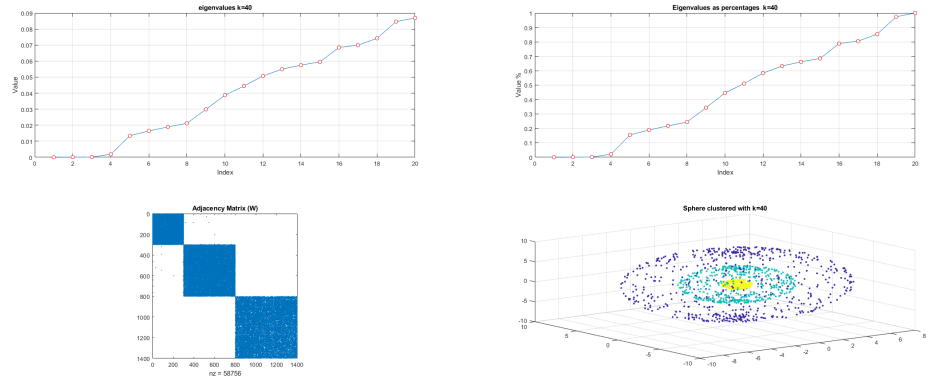

Figure 18: 3D Lsym with k=40

spherical shells contain more points. The smallest block at the top corresponds to the innermost shell.
This well-defined block structure in the matrix W reflects the fact that points within the same spherical shell are closer to each other than points in different shells, thus creating natural clusters in the data.

k	10	20	40
L	$1e^{-3}$	$1e^{-3}$	$1e^{-2}$
L_{sym}	$1e^{-3}$	$0.8e^{-3}$	$1e^{-3}$

Table 3: 3D *tol* table

Are valid all the considerations done since now about L_{sym}.

4.5 Other clusters method

In this section, the results of other clustering methods are presented. Two methods have been considered: Dbscan and Agglomerative Hierarchical Clustering.

`DBSCAN (Density-Based Spatial Clustering of Applications with Noise)` is a density-based clustering algorithm designed to discover groups of points that are densely packed and separated by areas of low density. Unlike algorithms such as K-means, which require a predefined number of clusters, DBSCAN is based on two main parameters: the *maximum distance between two points* (ϵ) to consider them as part of the same cluster, and the *minimum number of points* (minPts) required to form a dense cluster. Points that do not belong to any cluster and do not satisfy the density criteria are labeled as *noise*.

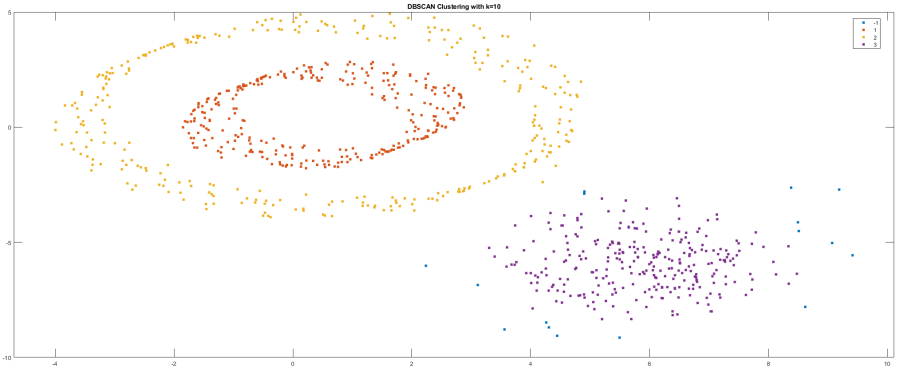

Figure 19: Cicles: DBSCAN with k=10

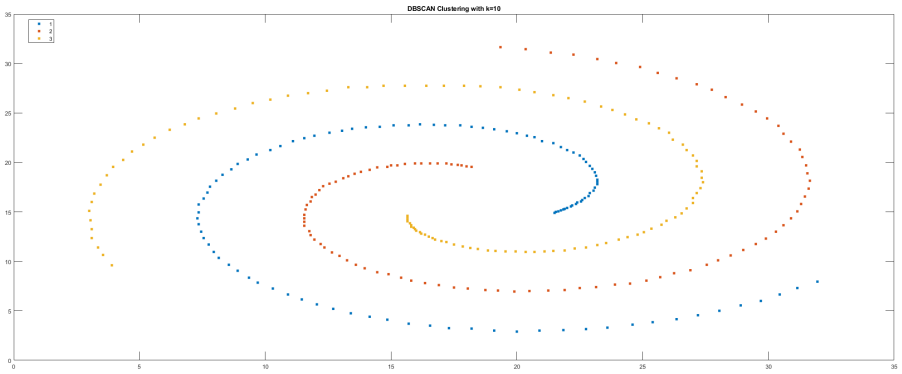

Figure 20: Spiral: DBSCAN with k=10

As we can see in (20) DBSCAN can effectively detect clusters in the three-armed spiral because it identifies regions of high point density and follows the spiral's structure, handling non-linear shapes well.

DBSCAN is particularly useful for identifying arbitrarily shaped clusters and handling noise in the data, but it can be sensitive to the chosen parameters (ϵ and minPts).

DBSCAN excels at identifying geometric shapes and non-linear structures in data because it is a density-based algorithm. It clusters points based on regions of high density separated by regions of low density, without assuming any particular shape for the clusters. This makes it particularly effective for datasets like spirals or concentric circles, where clusters are defined by density rather than distance or hierarchy.

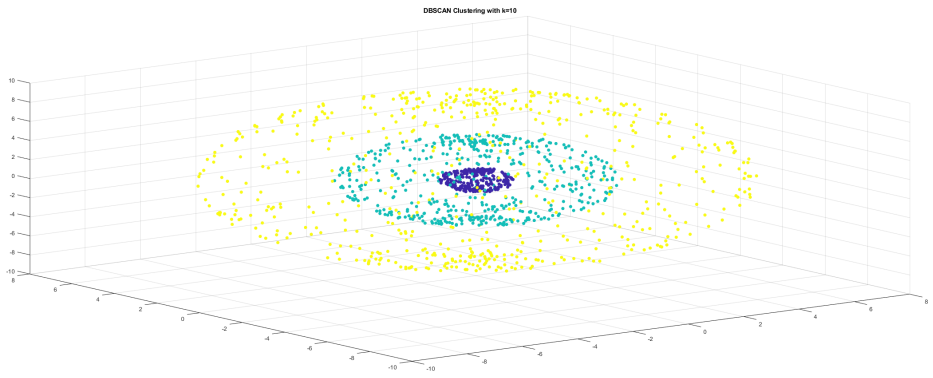

Figure 21: 3D: DBSCAN with k=10