Spectral Clustering Analysis of Circle and Spiral Data-sets

Alessia Ciccaglione
#######
Politecnico di Torino
Computational Linear Algebra for Large Scale Problems, A.A. 2024-2025

***Abstract*—This report investigates the application of the spectral clustering technique to two different data-sets: Circle and Spiral.**
The study provides a detailed exploration of this method, evaluates the effectiveness of spectral clustering in identifying clusters and compares its performance with K-means and DBSCAN.
The findings demonstrate the importance of algorithm selection based on data-set characteristics and demonstrate the utility of spectral clustering in revealing hidden patterns in non-linear structures.

I. Introduction

Cluster analysis aims to partition data points into meaningful clusters such that objects within the same group are more similar to each other than to those in other clusters. Traditional methods like K-Means often struggle to identify the presence of non-linear shapes in the data.
Spectral clustering is a graph-based algorithm designed to identify k clusters in the data. The method involves mapping the data to a lower-dimensional space, where the clusters are more distinct and better separated. This separation makes it easier to apply algorithms like k-means, which might not perform well with the original data. The lower-dimensional space is derived from the eigenvectors of a Laplacian matrix, which is where the algorithm's name comes from, since the spectrum of a matrix refers to the set of its eigenvalues.

II. Data-set Description

In this analysis, we examine two data-sets named 'Circle' and 'Spiral' data, as they represent these respective geometric shapes in space. The Circle data-set contains the values of the coordinates x and y of the points, which are distributed in a circular pattern. The data-set includes 900 points, with x and y values ranging approximately from -4 to 9.4 and from -9 to 5, respectively. The Spiral data-set contains also the coordinates of the points and another column corresponding to the index of the correct cluster, which will be utilized in the final section to measure the algorithm's performance. The points follow a spiral pattern and the data-set consists of 312 points, with x and y values ranging both from 3 to 32.

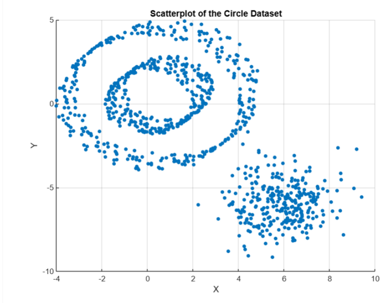


Fig. 1: Scatter plot visualizing the distribution of the **Circle** data-set.

The scatter plots of the Circle and Spiral data-sets *(Figures 1 and 2)* provide an initial visual representation of the distribution of the two data-sets. The Circle data-set represents data points distributed evenly along a circular shape, making it suitable for detecting regular geometric patterns. The Spiral data-set has instead a non-linear structure, consisting of concentric spiral arms.

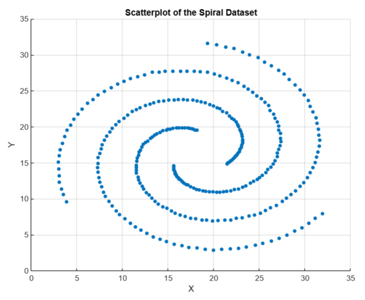


Fig. 2: Scatter plot visualizing the distribution of the **Spiral** data-set.

III. Similarity

To perform clustering, it is essential to define a similarity function that measures the similarity between our points. In this case study, the similarity between two points x_i and

x_j of the data-set is computed using the following Gaussian similarity function:

$$S_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

where:

- x_i and x_j are the coordinates of the two points,
- $\|x_i - x_j\|^2$ represents the squared Euclidean distance between the points,
- σ is the scaling parameter that controls the sensitivity of the similarity to distance. In this analysis, we will consider σ = 1.

This function assigns a high similarity value to points that are close to each other and a low similarity score to points that are farther apart.

We decided to set the similarity between a point and itself to 0:

$$s_{ii} = 0 \quad \forall i$$

In the graph representation of the data-set, nodes are connected by edges based on similarity values. Setting these values equal to zero ensures that there are no self-loops in the graph, which simplifies the graph structure and aligns with the goal of analyzing relationships between distinct points.

The k-nearest neighbor graph is used to define the adjacency relationships, where each point is connected to its k-nearest neighbors based on the similarity values.
In the graph the nodes represent the points and the edges represent the higher similarity. This was done for both data-sets with several values of k, specifically $k = 10, 20, 40$.

The *adjacency matrix* W is a representation of the graph associated with the data-set. Each entry W_{ij} encodes the presence and strength of a connection between two data points x_i and x_j, based on their similarity, indicating whether pairs of vertices are adjacent or not in the graph.

The adjacency matrix is defined as follows:

$$W_{ij} = \begin{cases} s_{ij} & \text{if } x_j \text{ is among the } k\text{-nearest neighbors of } x_i, \\ 0 & \text{otherwise.} \end{cases}$$

The adjacency matrix W is typically sparse, as most data points are not directly connected, especially in large data-sets. Therefore, to save memory, W is converted to a sparse storage format using MATLAB `sparse()` function. Similarly, the matrices L and D, defined in the following section, are also stored in that format.

As indicated, we assume that $s_{ij} = s_{ji}$. Consequently, the similarity graph is undirected, meaning that W must be symmetric.
However, applying the k-nearest neighbor process may result in a non-symmetric matrix.
To address this, we reintroduce symmetry by considering that if the edge (i, j) is relevant, then (j, i) should also be relevant. This is achieved by taking the maximum between W and its transpose, W'.

Our goal is to construct a K-NN graph that effectively separates the shapes, ideally forming distinct cluster. As can be seen in Figure 3 and Figure 4, a value of K=10 achieves this separation effectively. However, for higher values of K, edges are introduced between vertices that likely belong to different clusters, reducing the clarity of the separation.

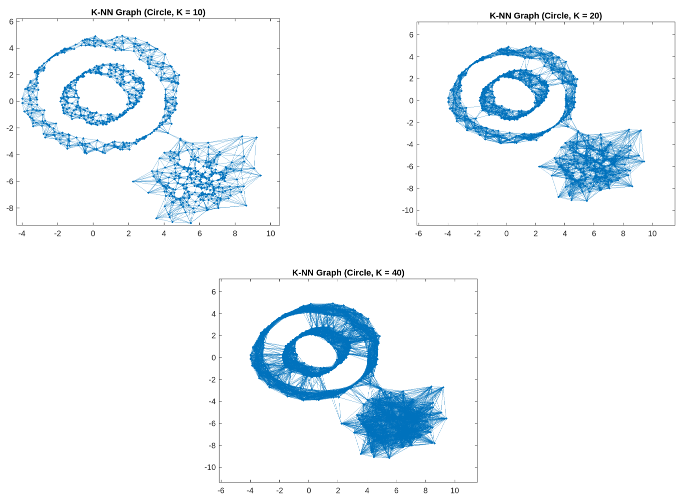


Fig. 3: K-NN graphs plots for **Circle** data-set different k.

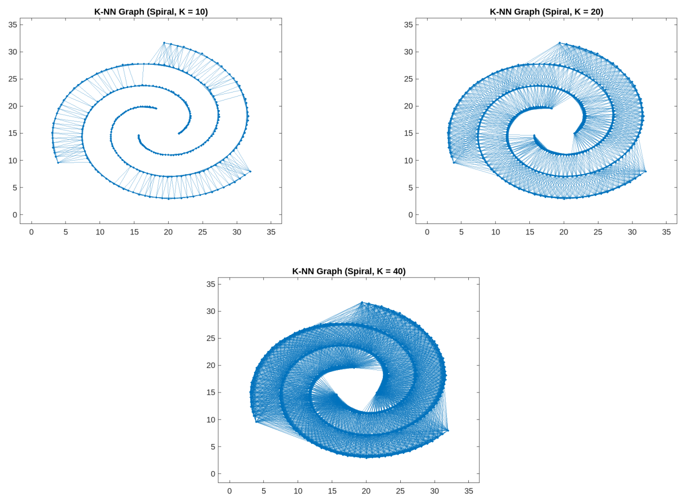


Fig. 4: K-NN graphs plots for **Spiral** data-set with different k.

IV. Degree Matrix and Laplacian Matrix

The adjacency matrix W defines the edges of the graph, which are then used to compute the *degree matrix* D and the *Laplacian matrix* L, upon which spectral clustering is based.

The degree matrix D is a diagonal matrix where each entry represents the sum of the weights of edges connected to a given node. Each entry of the diagonal is defined as:

$$d_i = \sum_j w_{ij}$$

where $W = (w_{ij})$, with $i, j = 1, \ldots, n$, is the adjacency matrix.

The matrix D summarizes the connectivity of each node in the graph. The Laplacian matrix L, instead, reflects the differences between the connectivity of nodes and their direct relationships. It is computed by subtracting the adjacency matrix from the degree matrix D.

$$L = D - W$$

A. *Laplacian matrix graph*

We analyzed the Laplacian graphs of the two data-set as the value k increases. The diagonal blocks in the graph represent the intra-cluster relationships, while the off-diagonal blocks represent the inter-cluster connections, ie. the relationships between points belonging to different clusters.

Circle data-set
For the Circle data-set, in Figure 5, the first two diagonal blocks represent two well-defined clusters, characterized by strong intra-cluster connectivity. As k increases, the diagonal blocks become denser, indicating an higher number of internal connections within each clusters, due to the expanded neighborhood size.

The square block on the diagonal suggests a third cluster with weaker connectivity. Its density increases as k grows, reflecting stronger internal connections. Additionally, as k increases, points near the boundaries of the clusters start forming connections with points in other clusters. This results in the appearance of more connections outside the diagonal blocks. For k=40, in particular, we observe more increasing connections between the first two cluster, which distinction became less definite. This turns out in the appearance of more connections external to the diagonal blocks. This will determine in the final output the formation of only two clusters.

Spiral data-set
For the Spiral data-set, Figure 6,the Laplacian graphs reveal three defined blocks in the diagonal corresponding to the three main clusters in the data-set. Each block grows slightly in size as more connections within each cluster are established due to the larger neighborhood size.

The off-diagonal regions grow larger as k increase, reflecting more frequent inter-cluster connections.

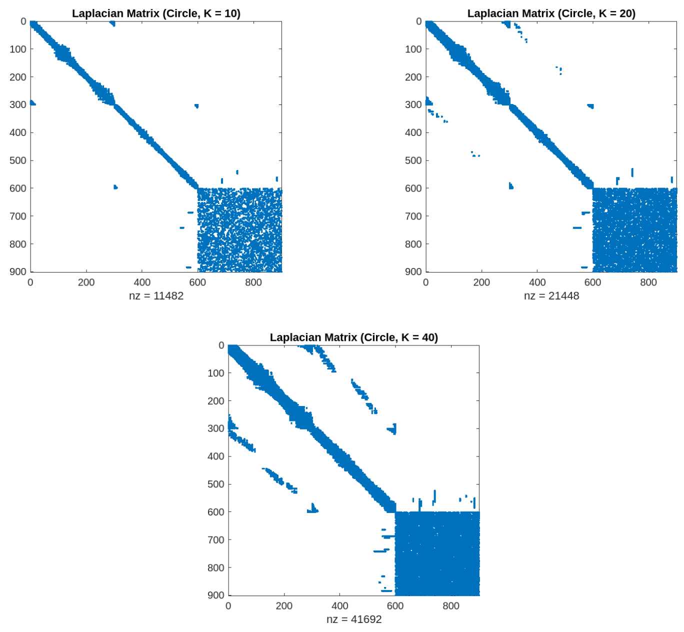


Fig. 5: Laplacian matrix for the **Circle** data-set with different k.

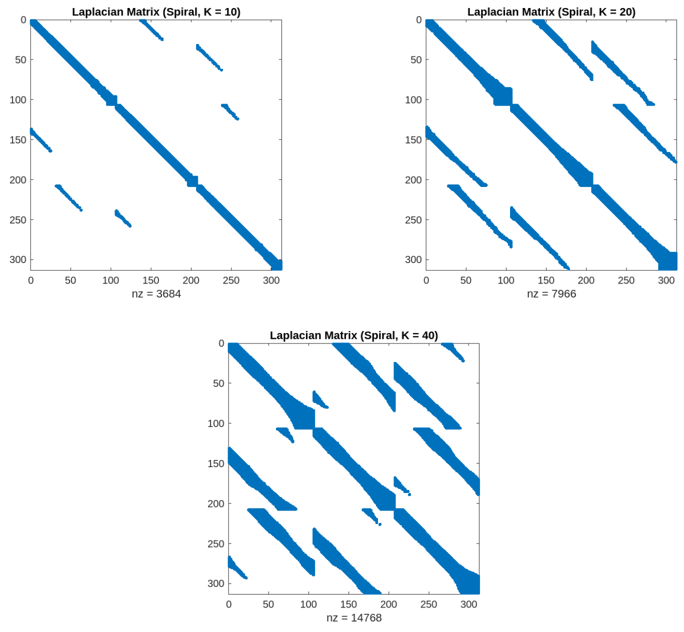


Fig. 6: Laplacian matrix for the **Spiral** data-set with different k.

In both data-sets the dominance of the diagonal blocks suggests that the data-set's structure remains well-defined, preserving the separation between the clusters despite the increased inter-cluster connectivity.
With a smaller k=10, the diagonal blocks are well-defined, reflecting a clear separation between clusters. As k increases, the diagonal blocks expand, reflecting denser intra-cluster connec-

tions. At the same time, off-diagonal connections become more prominent, reflecting increased interactions between points in different clusters. This highlights how larger k values can reduce the clarity of cluster separation.

However, in the Spiral data-set, higher k values lead to more blending between clusters compared to the Circle data-set, where the separation between clusters remains more robust. This difference may be attributed to the simpler geometry of the circular clusters, which allows for clearer boundaries even as k increases, in contrast the intertwined nature of the spiral geometry makes it more susceptible to increased inter-cluster connections.

V. Connected Components

A connected component of an undirected graph is a subgraph in which every pair of nodes is connected by a path and provides insight into the structure of the data-set.
In order to compute that information we use the following result:
Theorem $G = (V, E)$ be a finite graph and let L be its Laplacian matrix. Then L has $\lambda = 0$ as an eigenvalue and its algebraic multiplicity correspond to the number of connected components in G .
The eigenvalue $\lambda = 0$ appears once for each connected component in the graph. This relationship arises because the null space of L corresponds to constant vectors over each connected component.
We calculated the eigenvalues of L with respect to the above property considering the eigenvalues λ that are closest to zero with a tolerance of 10^{-12} to determinate the number of connected components. The procedure was then verified by comparing the results with those obtained from the MATLAB command used to identify the connected components, ensuring consistency and correctness up to that point.
To compute the number of connected components in MATLAB, we used the following command:

```
bins = conncomp(G);
num_components = length(unique(bins));
```

This function computes the connected components by analyzing the graph G and assigning each node to a specific component. The total number of connected components is then obtained by counting the unique values in the output vector.
The consistency between the connected components computed using `conncomp` and those identified by spectral analysis was confirmed.

A. Computation of Eigenvalues

We followed two approaches to compute the eigenvalues of L: using the command `eigs()`, which provides both eigenvalues and eigenvector and applying the Inverse Power method with Deflation.
The graphs and tables related to these methods, applied to both datasets, are provided in the Appendix.

Inverse Power Method with Deflation
The Inverse Power Method is a numerical algorithm designed to find the smallest eigenvalue λ_{min} of a matrix A, as well as its corresponding eigenvector.
The mathematical principle underlying the Inverse Power Method for finding the smallest eigenvalue is as follows:
If A is invertible:

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i \quad \text{so} \quad A^{-1}\mathbf{v}_i = \frac{1}{\lambda_i}\mathbf{v}_i$$

$$A^{-1}\mathbf{v}_i = \mu_i \mathbf{v}_i$$

where $\mu_i = \frac{1}{\lambda_i}$ is the eigenvalue of A^{-1}. Applying the Power method on A^{-1}, we will obtain the largest eigenvalue of A^{-1} μ_{max}, which corresponds to the smallest eigenvalue $\lambda_{\min}$ of A.

$$\lambda_{min} = \frac{1}{\mu_{max}}$$

Observation: This method works only if we assume that the magnitude of the largest eigenvalue λ or μ is strictly greater than the other:

$$|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|$$

Since the Inverse Power Method builds upon the principles of the Power Method, we will now proceed with its mathematical explanation.

Algorithm 1 Power Method

```
Input: Matrix $A$, initial vector $\mathbf{v}^{(0)}$, maximum iterations maxIter, relative tolerance relTol
Output: Largest eigenvalue $\lambda$, eigenvector $\mathbf{v}$
Normalize $\mathbf{v}^{(0)} \leftarrow \frac{\mathbf{v}^{(0)}}{\|\mathbf{v}^{(0)}\|}$
$k \leftarrow 0$ and $\lambda \leftarrow \infty$
while $k \leq$ maxIter do
  Compute $\tilde{\mathbf{v}}^{(k+1)} \leftarrow A\mathbf{v}^{(k)}$
  Compute the Rayleigh quotient for the eigenvalue approximation: $\lambda^{(k+1)} \leftarrow \mathbf{v}^{(k)} \cdot \tilde{\mathbf{v}}^{(k+1)}$
  Compute the new vector $\mathbf{v}^{(k+1)} \leftarrow \frac{\tilde{\mathbf{v}}^{(k+1)}}{\|\tilde{\mathbf{v}}^{(k+1)}\|}$
  $k \leftarrow k + 1$
  if $\left|\lambda^{(k)} - \lambda^{(k+1)}\right| <$ relTol $\left|\lambda^{(k)}\right|$ then
    Stop the loop;
  end if
end while
return $\lambda^{(k)}, v^{(k)}$
```

The Power Method, as outlined in Algorithm 1, is an iterative algorithm that outputs the dominant eigenvalue λ_1 and its corresponding eigenvector $\mathbf{v}_1$ after a certain number of iterations. The method relies on the principle that any vector $\mathbf{x}$ can be expressed as a linear combination of the eigenvectors of the matrix $\mathbf{A}$, since these eigenvectors are linearly independent. Thus:

$$x_0 = \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n$$

where v_i are the eigenvectors of A and α are the corresponding coefficients. When we apply $\mathbf{A}$ to $\mathbf{x_0}$, we obtain:

$$\mathbf{x_1} = \mathbf{A}\mathbf{x_0} = \alpha_1\mathbf{A}\mathbf{v_1} + \alpha_2\mathbf{A}\mathbf{v_2} + \cdots + \alpha_n\mathbf{A}\mathbf{v_n}$$

$$= \alpha_1\lambda_1\mathbf{v_1} + \alpha_2\lambda_2\mathbf{v_2} + \cdots + \alpha_n\lambda_n\mathbf{v_n}$$

By continuing to apply $\mathbf{A}$ iteratively, we can define $\mathbf{x_k} = \mathbf{A}^k\mathbf{x_0}$. Thus, we have:

$$\mathbf{x_k} = \lambda_1^k\left(\alpha_1\mathbf{v_1} + \alpha_2\left(\frac{\lambda_2}{\lambda_1}\right)^k\mathbf{v_2} + \cdots + \alpha_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\mathbf{v_n}\right)$$

As k increases, the terms associated with eigenvalues $\lambda_2, \lambda_3, \dots, \lambda_n$ decay at a rate determined by the ratios $\frac{\lambda_2}{\lambda_1}, \frac{\lambda_3}{\lambda_1}, \dots, \frac{\lambda_n}{\lambda_1}$, assuming $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$. This means that the vector $A^k\mathbf{x_0}$ will be dominated by $\lambda_1^k\mathbf{v}_1$, such that;

$$x^k = A^k\mathbf{x_0} \approx \lambda_1^k\mathbf{v}_1$$

Unfortunately, this means that x^k will be very large if $|\lambda_1| > 1$, or very small if $|\lambda_1| < 1$. For this reason, we normalize the vectors used in that algorithm:

$$\mathbf{x^k} = \frac{\mathbf{A}^k\mathbf{x_0}}{\|\mathbf{A}^k\mathbf{x_0}\|} \approx \left(\frac{\lambda_1}{|\lambda_1|}\right)^k \cdot \frac{\mathbf{v_1}}{\|\mathbf{v_1}\|} \approx \frac{\mathbf{v_1}}{\|\mathbf{v_1}\|}$$

Power iteration allows us to find an approximate eigenvector corresponding to the largest eigenvalue in magnitude. To compute the actual eigenvalue, we can use the Rayleigh quotient. Given $A \in \mathbb{R}^{n\times n}$ and $\mathbf{v} \in \mathbb{R}^n$, the Rayleigh quotient of $\mathbf{v}$ with respect to A is defined as:

$$r_A(\mathbf{v}) = \frac{\mathbf{v}^T A\mathbf{v}}{\|\mathbf{v}\|_2^2}$$

It can be proven that the Rayleigh Quotient of $\mathbf{v}$ with respect to $\mathbf{A}$ converges to the eigenvalue with the largest magnitude.

$$\lim_{k\to\infty} r_A(\mathbf{v}) = r_A^{(k)}(\mathbf{v}) = \frac{(\mathbf{v}^{(k)})^T A\mathbf{v}^{(k)}}{\|\mathbf{v}^{(k)}\|_2^2} = \lambda_1$$

These two statements are used in the algorithm to compute the eigenvector as $\tilde{\mathbf{v}}^{(k+1)} \leftarrow \mathbf{A}\mathbf{v}^{(k)}$ and the eigenvalue as $\lambda^{(k+1)} \leftarrow \frac{\mathbf{v}^{(k)}\cdot\tilde{\mathbf{v}}^{(k+1)}}{\mathbf{v}^{(k)}\cdot\mathbf{v}^{(k)}}$.

Algorithm 2 The Inverse Power Method

```
Input: Matrix $A$, initial vector $\mathbf{v}^{(0)}$, maximum iterations maxIter, relative tolerance relTol
Output: Smallest eigenvalue $\lambda$, eigenvector $\mathbf{v}$
Normalize $\mathbf{v}^{(0)} \leftarrow \frac{\mathbf{v}^{(0)}}{\|\mathbf{v}^{(0)}\|}$
$k \leftarrow 0$ and $\mu \leftarrow 0$
while $k \leq$ maxIter do
  Compute $\tilde{\mathbf{v}}^{(k+1)} \leftarrow A^{-1}\mathbf{v}^{(k)}$
  Compute the Rayleigh quotient for the eigenvalue approximation: $\mu^{(k+1)} \leftarrow \mathbf{v}^{(k)} \cdot \tilde{\mathbf{v}}^{(k+1)}$
  Compute the new vector $\mathbf{v}^{(k+1)} \leftarrow \frac{\tilde{\mathbf{v}}^{(k+1)}}{\|\tilde{\mathbf{v}}^{(k+1)}\|}$
  $k \leftarrow k+1$
  if $\left|\mu^{(k)} - \mu^{(k-1)}\right| <$ relTol $\left|\mu^{(k)}\right|$ then
    Stop the loop;
  end if
end while
return $\lambda^{(k)} = \left|\frac{1}{\mu^k}\right|, v^{(k)}$
```

Both the Inverse Power method and the Power method, as written in Algorithm 2 and Algorithm 1, allows us to determine only the eigenvalue and the eigenvector corresponding to the smallest or largest eigenvalue. However, our goal is to determine a greater number of eigenvalues. To proceed, we have employed the technique of Deflation.

The Deflation is a modification applied to the original matrix such that the eigenvalue λ_2 becomes the one with largest modulus of the modified matrix and therefore, algorithm can be applied to the new matrix to extract the eigenvalue λ_2. Let $P \in \mathbb{R}^{n\times n}$ be an orthogonal matrix such that

$$P = I - 2\frac{(v_1+e_1)(v_1+e_1)^T}{(v_1+e_1)^T(v_1+e_1)}$$

satisfying this property $Pv_1 = -e_1$, where e_1 is the first column of the identity matrix I and v_1 the largest eigenvector of A.

Then the similarity transformation determined by P transforms A into the form:

$$PAP^T = \begin{pmatrix}\lambda_1 & * \\ 0 & B\end{pmatrix}$$

where B is a matrix of order $n-1$ having eigenvalues $\lambda_2, \dots, \lambda_n$ since P is symmetric and orthogonal:

$$PAP^T = P^{-1}AP$$

We use B to compute the eigenvalue λ_2 applying the Power Method and then with the same process we compute the other eigenvalues. However, further steps are required to compute the corresponding eigenvectors, because the matrices A and B have different dimensions, meaning they cannot share the same eigenvectors.

The results confirmed the convergence of the smallest eigenvalues, as illustrated in the graph in Figure 7 and in the tables provided in the appendix. This convergence was particularly evident in cases where there was a significant separation between eigenvalues, especially when the gap between the

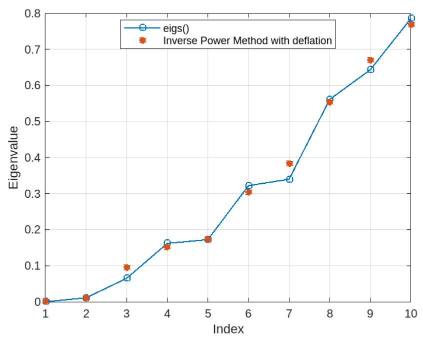


Fig. 7: Eigenvalues: Comparison between eigs() and the Inverse Power Method with Deflation, Circle data with k=20

largest and smallest eigenvalues was larger. Consequently, the eigenvalues obtained from the native function were chosen for further analysis, since the corresponding eigenvectors were also automatically generated.

VI. Number of Clusters

To determine the optimal number of clusters (M) for spectral clustering, we analyzed the eigenvalues of the Laplacian matrix L and identified significant gaps between consecutive eigenvalues.
The eigenvalues and eigenvectors of the Laplacian matrix are defined as:

$$L\mathbf{v} = \lambda\mathbf{v}$$

where:

- L is the Laplacian matrix,
- λ is an eigenvalue,
- $\mathbf{v}$ is the eigenvector corresponding to λ.

The Laplacian matrix is symmetric and positive definite with the smallest eigenvalue equal to 0, such that:

$$0 \leq \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$$

The optimal number of clusters M is determined by finding the largest gap between consecutive eigenvalues.

$$\Delta_i = \lambda_{i+1} - \lambda_i$$

The index i corresponding to the largest gap determines M, the number of clusters.
To visualize this, the eigenvalues are plotted against their indices in an eigenvalue plot, where the x-axis represents the eigenvalue index i, which corresponds to the position of the eigenvalues in ascending order, and the y-axis represents the actual value of the eigenvalues λ_i. Specifically, the analysis focuses on the eigenvalues closest to zero, identifying the point where the reciprocal difference between consecutive eigenvalues increases significantly. This method helps to identify the "elbow" of the curve, the point beyond which adding more clusters does not significantly improve the data representation. The optimal number of clusters M corresponds to this point.

VII. Eigenvectors

A subset of eigenvectors corresponding to the smallest M eigenvalues of the Laplacian matrix L is then selected.
The eigenvector matrix is defined as:

$$U = [\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_M]$$

where M is the number of clusters and $\mathbf{v}_i$ is the i-th eigenvector of L.

In this way each graph node is represented as a row vector in U and the data are projected into an M-dimensional space. This new reference system, represented by the row vectors of U , is related to the strength of the connections among the points, as it considers the eigenvector associated with the smallest eigenvalue. In the original space, data points may have complex and nonlinear relationships that make harder identify distinct clusters. However, when we project the data into the eigenvector space, these connection are transformed, facilitating better cluster separation.
The k-means algorithm is then applied to the row vectors of U, grouping the graph nodes into M clusters.

VIII. K-means

K-means is used after transforming the data into a new space spanned by the eigenvectors corresponding to the smallest eigenvalues of the Laplacian matrix L. K-means is a clustering algorithm based on the partition of the data points into M clusters by minimizing the within-cluster sum of squares.

$$\sum_{i=1}^{N}\sum_{j=1}^{M} \|x_i - \mu_j\|^2, \quad \text{where } x_i \text{ belong to C}j$$

where x_i is a data point and μ_j is the centroid of cluster j.

The algorithm begins by randomly selecting M centroids to serve as the initial cluster centers. Once the centroids are established, each data point is assigned to the cluster associated with the nearest centroid, using the Euclidean distance as the distance metric. After all data points have been assigned to their respective clusters, the centroids are updated by calculating the mean of all data points assigned to each cluster. This process of assigning data points to clusters and updating the centroids continues iteratively until convergence is reached, which occurs when the centroids stabilize or a predefined maximum number of iterations is completed.

Algorithm 3 K-means Algorithm

```
Randomly initialize the reference vectors: $w_1^{(0)}, \ldots, w_k^{(0)}$;
for $t = 1, \ldots, T$ do
  for each $i = 1, \ldots, k$ do
    Identify $S_i^{(t)} := S \cap V_i^{(t)}$;
  end for
  for each $i = 1, \ldots, k$ do
    Update the reference vectors: $w_i^{(t+1)} = \mu(S_i^{(t)}) = \frac{1}{|S_i^{(t)}|}\sum_{x \in S_i^{(t)}} x$;
  end for
end for=0
```

The K-means algorithm uses the concept of Vector Quantization(VQ) to find the reference vectors w_i, known as centroids in the context of K-means, that minimize the quantization error on the samples. During the process, K-means updates the centroids by assigning points to the nearest centroids and calculating the mean of these points, following an iterative approach similar to that described in VQ. Furthermore, the centroids define the Voronoi regions V_i, which partition the data space into different areas, each associated with the nearest centroid. In this way, each data point belongs to the region of the closest centroid, contributing to establishing the cluster assignments.

The k-means algorithm is executed using a MATLAB function `idx = kmeans(X,k)` passing the transformed data matrix U_M and the desired number of clusters M as inputs. The output is a vector containing cluster indices of each observation.

A. *Analysis of the Results*

Circle data-set

The spectral clustering process on Circle data-set (Figure 16) clearly identified three distinct clusters represented by blue, red, and yellow colors using $k = 20$ and $k = 30$, However, when the number of relevant connections was increased, the two concentric circles in the upper left of the space were considered as a single cluster. Overall, the clusters are well-separated, indicating that the intrinsic structure of the data is effectively captured. With both $k = 20$ and $k = 30$, the algorithm was able to detect the rings and did not treat them as simple convex shapes.

Spiral data-set

Spectral Clustering successfully distinguished the three clusters also in the Spiral data-set (Figure 9). This is evident from the fact that each spiral is associated with a single cluster (blue, red, yellow). There is no noticeable difference between using $k = 10$, $k = 20$, or $k = 40$. The visualization of the centroids in the original space highlights how Spectral Clustering operates in a different reference system. It is evident that the points in the dataset were not assigned to clusters based on their distance to the centroids but rather on their connectivity within the graph.

In contrast, when applied to the original data-set, the performance of k-means is generally low, as we will see in the following section IX.

IX. Comparison with other methods

To evaluate the effectiveness of Spectral Clustering, a comparison with other popular clustering methods such as K-Means and DBSCAN was conducted. K-means was chosen as it is also used within the Spectral Clustering process, making it useful for verifying the added benefits of the spectrum of the similarity matrix. DBSCAN, on the other hand, was selected for its ability to adapt to clusters of arbitrary shapes, which aligns well with the characteristics of the data-sets used in this analysis.

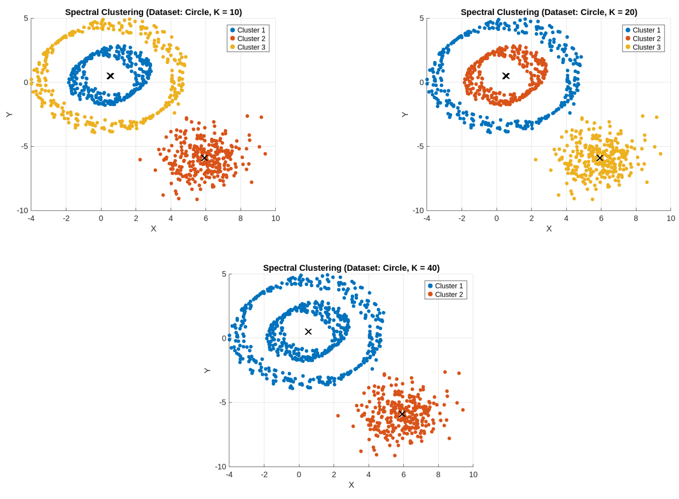


Fig. 8: Spectral Clustering for the **Circle** data-set with different k.

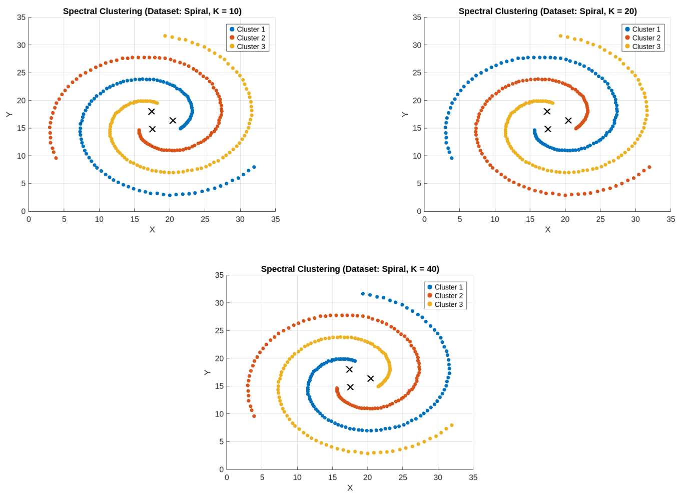


Fig. 9: Spectral Clustering for the **Spiral** data-set with different k.

A. *DBSCAN*

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based clustering algorithm designed to discover clusters and noise within data. The DBSCAN clustering was implemented using the `dbscan()` function provided by MATLAB, specifying the values of epsilon, which defines the neighborhood search radius, and minPts, the minimum number of neighbors required for a point to be considered part of a cluster. Since there is a relationship between MinPts and Eps, these two values are determined by fixing MinPts and calculating Eps as a consequence. The value of Eps is determined plotting the distances of each point to its k-th nearest neighbor, with k = MinPts, and selecting the distance at the elbow of the curve, where is observed a change. By fixing MinPts = 5 for both data-sets and applying this procedure, as shown in the Figure 10, we obtained that Eps = 0.5 for the Circle data-set and Eps = 2 for the Spiral data-set. DBSCAN proved effective in clustering data-sets with non-linear structures, such as spirals and circles, where traditional methods like K-Means often fail. In the Spiral data-set, it successfully identified three distinct clusters while isolating a single outlier, demonstrating its ability to handle arbitrary cluster shapes. Similarly in the Circle data-set DBSCAN was able to highlight noisy points as outliers, but it appeared to be more sensitivity to variations in density, producing four clusters instead of the expected three.

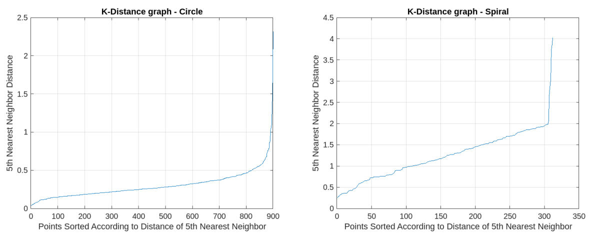


Fig. 10: Sorted distance plots of every point of **Circle** and **Spiral** Data to their 5th nearest neighbor to compute **eps** with MinPts=5

B. *K-Means*

K-Means is a centroid-based clustering algorithm widely used for partitioning data-sets into k distinct clusters.
K-Means directly operates on the original data, unlike Spectral Clustering which uses the eigenvector space.
The K-Means clustering was implemented using the `kmeans()` function in MATLAB, specifying the number of clusters (k) for each data-set.
Since K-Means requires a pre-defined k, the optimal number of clusters was determined by employing the Elbow Method. This technique analyzes the Sum of squared error (SSE) as a function of k, identifying the point where the SSE curve exhibits a significant "elbow," indicating diminishing returns for increasing k. This analysis led to the identification of $k = 4$ for the Circle data-set and $k = 3$ for the Spiral data-set. We tried to determine the optimal number of clusters k also using the Silhouette Score, but it identified excessively high values, especially for the circle dataset. This led us to adopt this alternative approach.

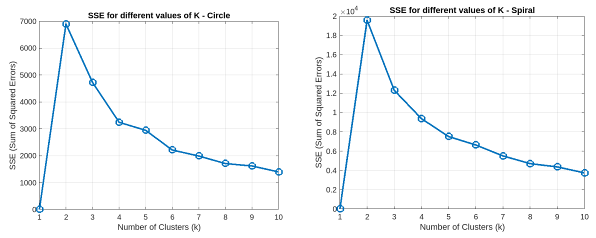
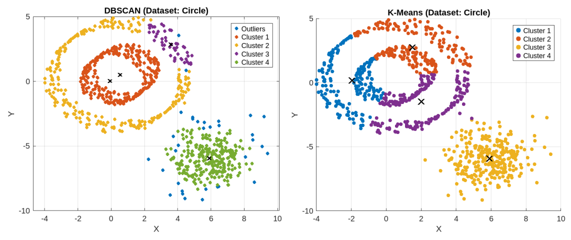


Fig. 11: SSE of **Circle** and **Spiral** Data to identify **k**

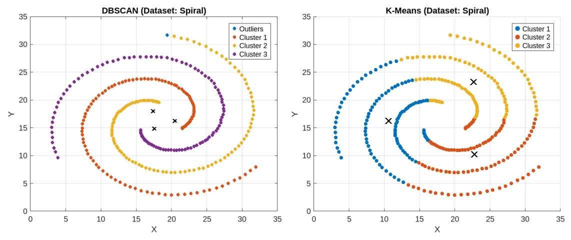
K-Means failed in determining the appropriate clusters due to its assumption of globular shapes and reliance on Euclidean distance, which is not suitable for non-linear structures like spirals or concentric circles, like we have in this case.

In both the data-sets, the algorithm was unable to correctly identify the shapes and instead focused primarily on clustering based on spatial proximity.



(a) DBSCAN (b) K-Means

Fig. 12: **Circle** data-set



(a) DBSCAN (b) K-Means

Fig. 13: **Spiral** data-set

Comparing the obtained results we observe that K-means performs poorly on both datasets, focusing on spatial proximity and failing to account for geometric complexities, while DBSCAN is capable to handle non-linear structures well but struggles with variable densities, leading to suboptimal clustering in the Circle dataset.

X. Conclusions

The final results obtained from spectral clustering on Spiral data-set are satisfactory when compared to the given labels as we can see analyzing Figure 14 and Figure 9.

In conclusion, our analysis confirms that Spectral Clustering is highly effective for datasets with non-linear structures, performing accurately and significantly better than traditional methods, especially for these two nonlinear data-sets. However, in terms of memory usage, Spectral Clustering can be more demanding since it requires the computation and storage of the similarity, degree and Laplacian matrices, as well as the computation of eigenvalues and eigenvectors. By using $k = 20$, we reduce the memory required for storing L and W, while still achieving better results, making this a good approach and an optimal trade-off between performance and computational requirements.

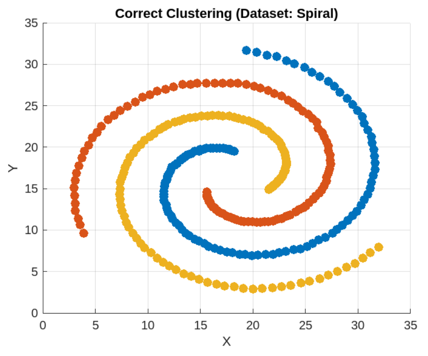


Fig. 14: Correct clustering of the **Spiral** data-set.

APPENDIX

A. Eigenvalue Convergence

eigs()	inv-def()
0.0000	0.0000
0.0000	0.0000
0.0048	0.0028
0.0286	0.0113
0.0425	0.0134
0.0429	0.0146
0.0806	0.0159
0.1264	0.0189
0.1349	0.0303
0.2092	0.0722

(a) k=10

eigs()	inv-def()
0.0000	0.0000
0.0111	0.0111
0.0652	0.0969
0.1620	0.1581
0.1724	0.1738
0.3220	0.2961
0.3399	0.4006
0.5612	0.6584
0.6434	0.6628
0.7856	0.7895

(b) k=20

eigs()	inv-def()
0.0000	0.0000
0.0482	0.0423
0.7028	0.4539
0.7797	0.7031
0.9065	0.8251
1.3768	1.0688
1.4803	1.2064
2.4004	1.3877
2.5472	2.2980
2.7842	2.4498

(c) k=40

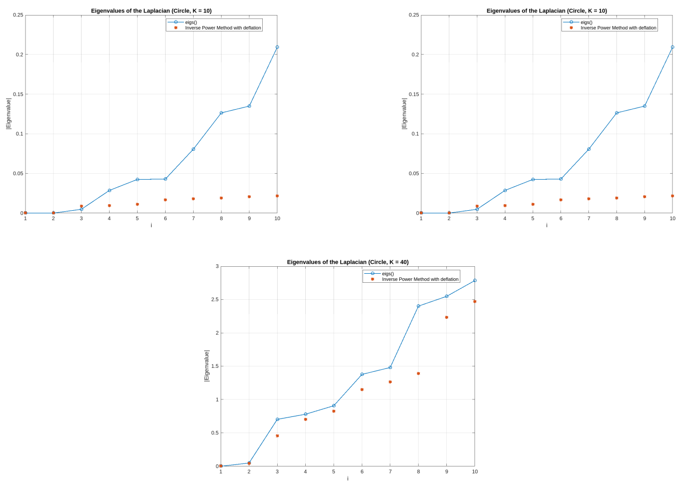


Fig. 15: Eigenvalues Comparison between eigs() and the Inverse Power Method with Deflation - **Circle** data-set

eigs()	inv-def()
0.0000	0.0000
0.0002	0.0002
0.0003	0.0003
0.0041	0.0043
0.0044	0.0043
0.0046	0.0045
0.0172	0.0175
0.0184	0.0182
0.0189	0.0187
0.0404	0.0407

(a) k=10

eigs()	inv-def()
0.0000	0.0000
0.0018	0.0010
0.0020	0.0019
0.0048	0.0042
0.0054	0.0061
0.0056	0.0064
0.0184	0.0105
0.0196	0.0194
0.0203	0.0199
0.0426	0.0229

(b) k=20

eigs()	inv-def()
0.0000	0.0000
0.0023	0.0023
0.0025	0.0025
0.0049	0.0048
0.0062	0.0062
0.0067	0.0066
0.0188	0.0187
0.0199	0.0198
0.0205	0.0203
0.0429	0.0399

(c) k=40

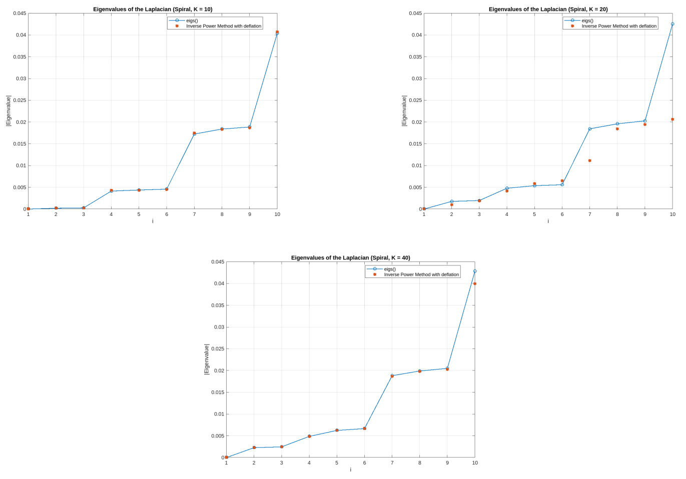


Fig. 16: Eigenvalues Comparison between eigs() and the Inverse Power Method with Deflation - **Spiral** data-set