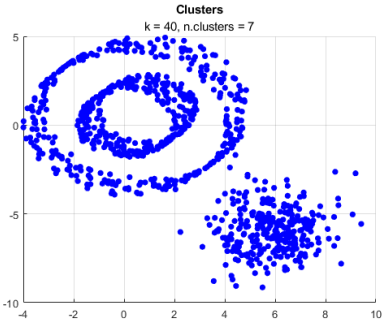
Homework
Spectral Clustering

Gabriele Galilei 302699

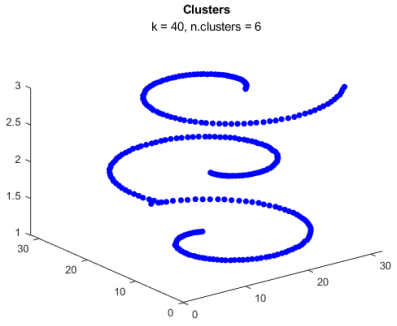
June 8, 2023

Spectral Clustering is an unsupervised learning technique that effectively divides a dataset of points into distinct groups by emphasizing the similarity between points within the same cluster. Unlike algorithms like k-means that rely on the assumption of convex or globular clusters based on euclidean distance, Spectral Clustering excels in the presence of different densities and shapes, such as intertwined spirals. This is because it does not make strong assumptions about the shape of the clusters. Additionally, Spectral Clustering is free from the issues of local minima and the need to restart the algorithm with different initial guesses as it is based on the solution of a sparse linear system.

To evaluate the effectiveness of spectral clustering, I applied it to two different datasets as shown below:



(a) Circle dataset.



(b) Spiral dataset.

For the purpose of this explanation, I will be working with the circle dataset and will provide the results for the spiral dataset at the end.

1 Similarity graph and Adjacency matrix

To begin, calling k the number of nearest neighbors to consider, I constructed a k-nearest neighbor graph in which an undirected edge (i, j) exists only if i is among the k nearest neighbors of j or vice versa. Since the edges in this graph are undirected, the adjacency matrix W is symmetric. Additionally, W is a weighted matrix, where I utilized Gaussian similarity as weight, as required. Specifically, if the edge (i, j) was present, then:

$$W_{i,j} = W_{j,i} = \exp\left(-\frac{||X_i - X_j||^2}{2\sigma^2}\right).$$

The k-nearest neighbors graph can connect points on different scales and can break into several disconnected components if there are high density regions which are reasonably far away from each other. Experimentally this is the best similarity matrix to work with because it result in a sparse matrix and it's usually less vulnerable to unsuitable choices of parameters than the mutual k-nn graph, the ε-neighborhood graph and the fully connected graph.

To compute this similarity matrix, I first calculated the distance matrix in the *DistMatrix* function:

Algorithm 1 Distance Matrix

Require: $X_1, \ldots, X_n \in \mathbb{R}^d$
  for $i = 1$ to n **do**
    for $j = i + 1$ to n **do**
      for $k = 1$ to d **do**
        $D_{i,j} = D_{i,j} + (X_i(d) - X_j(d))^2$
      end for
    end for
  end for
  $D = D + D^T$

Let's observe that I calculated the squared distances because rooting the distances is not required to find neighbors, and it's also computationally expensive, which makes it inconvenient. Moreover, the function only computes the elements in the upper matrix and then copy it in the lower matrix since it is a symmetric matrix.

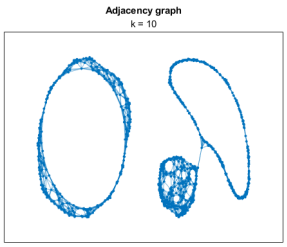
The *WeightMatrix* function calculates the adjacency matrix.

Algorithm 2 Adjacency Matrix

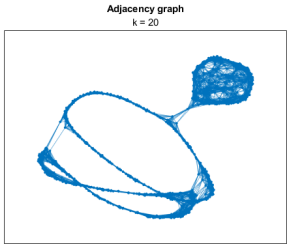
Require: $X_1, \ldots, X_n \in \mathbb{R}^d$
  for $i = 1$ to n **do**
    Consider X_i and find its k nearest neighbors I_i
    for $X_j \in I_i$ **do**
      $W_{i,j} \leftarrow \exp\left(-\frac{||X_i - X_j||^2}{2\sigma^2}\right)$
      if $W(j,i)$ not already been filled **then**
        $W_{j,i} \leftarrow W_{i,j}$
      end if
    end for
  end for

In the actual code, I obtained the $k+1$ nearest neighbors to include the node itself, but this is merely a technical detail.

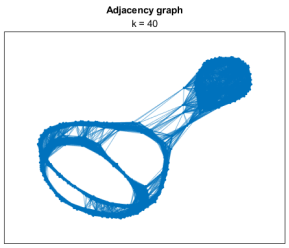
To verify the accuracy of the adjacency matrix, I plotted the resulting graphs for $k = 10, 20, 40$.



(a) k=10



(b) k=20



(c) k=40

2 Degree matrix and Laplacian matrix

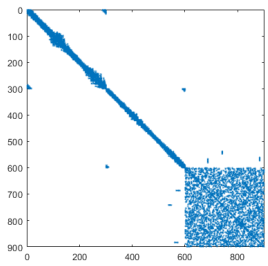
Next, I computed the degree matrix D, which is a diagonal matrix with each diagonal element being the sum of the corresponding row in the adjacency matrix W. This was achieved using the following code in MATLAB:

```
D = sparse(diag(W*ones(n,1)));
```

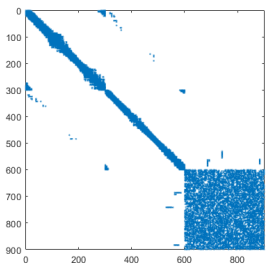
The Laplacian matrix, denoted as L, can be obtained as the difference between the degree matrix D and the adjacency matrix W:

$$L = D - W$$

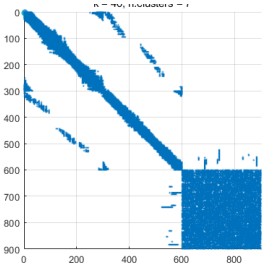
We can plot the sparsity patterns of L for $k = 10, 20, 40$.



(d) k=10



(e) k=20



(f) k=40

3 Connected components

3.a Theoretical basis

Let's analyze the properties of L:

Proposition 3.1. *Let L be a Laplacian matrix.*

1. *For every vector* $f \in \mathbb{R}^n$*:*

$$f'Lf = \frac{1}{2}\sum_{i.j=1}^{n} W_{i,j}(f_i - f_j)^2.$$

2. *The smallest eigenvalue of L is 0 and the corresponding eigenvector is* $\mathbb{1}$*.*

3. *L has n non-negative, real valued eigenvalues* $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$*.*

(no proof)

It's worth noting that the Laplacian matrix is independent of the diagonal elements in the adjacency matrix. This means that self-edges in a graph do not affect the corresponding Laplacian matrix.

Proposition 3.2. *Let G be an undirected graph with non-negative weights and let L be its Laplacian matrix. Then the multiplicity k of the eigenvalue* 0 *of L equals the number of connected components in the graph.*

Proof. Assume $k = 1$ and f eigenvector with eigenvalue 0. Then:

$$0 = f'Lf = \frac{1}{2}\sum_{i,j=1}^{n} W_{i,j}(f_i - f_j)^2.$$

Being this a sum of non-negative terms, it washes out only if every term is null, so only if:

$$0 = W_{i,j}(f_i - f_j)^2, \;\; \forall i, j = 1, \ldots, n$$

so if two nodes are connected ($W_{i,j} > 0$) then it must be $f_i = f_j$. If we are considering a connected graph then this must happen for each possible pair of nodes and so the only possible eigenvector of 0 must be $\mathbb{1}$.

For k connected components we can suppose that nodes are ordered according to the connected component they're in; then the adjacency matrix has a block structure and the same for the Laplacian matrix:

$$L = \begin{bmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_k \end{bmatrix}$$

So for each connected component there is a different Laplacian matrix L_i and each of this matrices has a 0 eigenvalue with multiplicity 1 and the corresponding eigenvalue being $\mathbb{1}$. Since the spectrum of L is given by the union of the spectra of $L_i,\ i = 1, \ldots, k$, then the eigenvalue 0 has multiplicity k. □

3.b Inverse power method

To determine the number of connected components in a graph, we need to compute the first *num* smallest eigenvalues of the Laplacian matrix and count the number of zero eigenvalues. To achieve this, I utilized the inverse power method with Householder deflation as a deflation method.

The inverse power method is initiated to calculate the smallest eigenvalue in terms of magnitude. As the matrix L is symmetric, all of its eigenvalues are real. Since the proposition states that the smallest eigenvalue of L is 0, the method can be applied in this scenario.

Algorithm 3 Inverse power method - calculate the smallest eigenvalue in magnitude

Require: $v_0 = \frac{\mathbb{1}}{||\mathbb{1}||_2}; \lambda = 1; \hat{\lambda} = \frac{1}{\lambda} = 1$
  for $i = 1$ to $numitr$ **do**
    $\hat{\lambda}_{old} \leftarrow \hat{\lambda}$
    Solve $Lv_i = v_{i-1}$ for v_i
    $\hat{\lambda} \leftarrow v_i^T v_{i-1}$
    $v_i \leftarrow \frac{v_i}{||v_i||_2}$
    if $|\hat{\lambda} - \hat{\lambda}_{old}| < tol$ **then**
      break
    end if
  end for

In the context of solving the symmetric sparse linear system $Lv_i = v_{i-1}$, several options were considered: the backslash option provided by Matlab, the Conjugate Gradient (CG) method, the Direct Lanczos method and the Generalized Minimal Residual (GMRES) method. The selection of the solving method can be specified using the flag *solvels*.

- **Backslash**: The backslash option, although available in Matlab, yielded unsatisfactory results for this particular problem due to numeric cancellation caused by the ill-conditioning of the sparse matrix L.
- **GMRES**: The Generalized Minimal Residual method (GMRES) is a more general approach applicable to solving general sparse linear systems. However, it is not specifically optimized for symmetric matrices, leading to slower convergence rates. Furthermore, GMRES necessitates the storage of Krylov subspace basis vectors, whose size grows with each iteration, resulting in increased memory usage.
- **Direct Lanczos**: The Direct Lanczos method is an iterative technique used to solve sparse symmetric linear systems. It's based on the observatin that if A is symmetric than the Hessemberg matrix obtained via Arnoldi is tridiagonal. So expliciting the terms, we can construct a far more efficient algorithm in terms of memory usage and computational costs. Despite this being the fastest algorithm to run, its results were not satisfactory, computing higher eigenvalues.

- **CG**: The Conjugate Gradient (CG) method emerged as the most favorable choice considering memory costs and convergence speed. This method is well-suited for solving symmetric positive definite matrices, exploiting their symmetry and positive definiteness to achieve efficient computations and minimize memory requirements.

Taking into account these factors, the CG method was selected as the preferred approach for the subsequent computations, offering the optimal balance between memory efficiency and convergence speed for the given symmetric sparse linear system.

Following the pseudo-code:

Algorithm 4 Conjugate Gradient method - solve iteratively a symmetric sparse linear system.

Require: $A \in \mathbb{R}^{n \times n}$; $x^{(0)} = 0_n$; $r^{(0)} = b - Ax^{(0)}$; $p^{(0)} = r^{(0)}$;
  for $i = 1$ to $numitr$ **do**
    $\alpha^{(i)} \leftarrow \frac{r^{(i-1)T} r^{(i-1)}}{p^{(i-1)T} A p^{(i-1)}}$
    $x^{(i)} \leftarrow x^{(i-1)} + \alpha^{(i)} p^{(i-1)}$
    $r^{(i)} \leftarrow r^{(i-1)} - \alpha^{(i)} A p^{(i-1)}$
    if $r^{(i)T} r^{(i)} <$tol **then**
      break
    end if
    $\beta^{(i)} \leftarrow \frac{r^{(i)T} r^{(i)}}{r^{(i-1)T} r^{(i-1)}}$
    $p^{(i)} \leftarrow r^{(i)} + \beta^{(i)} p^{(i-1)}$
  end for

3.c Deflation

Given a matrix and one of its eigenvectors, householder deflation removes the influence of the corresponding eigenvalue from the matrix.

Algorithm 5 Householder Deflation

Require: A matrix; v eigenvector;

$v \leftarrow \frac{v}{||v||_2}$

$u \leftarrow v + sgn(v(1))e_1$

$P \leftarrow I - 2\frac{uu^T}{||u||_2^2}$

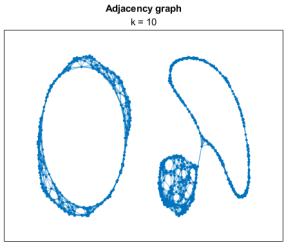
$B \leftarrow PAP^T$

$A \leftarrow B(2:n, 2:n)$

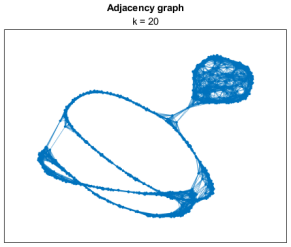
3.d Computational results

The function *kSmallestEigs* iters inverse power and deflation and computes the smallest eigenvalues and corresponding eigenvectors.

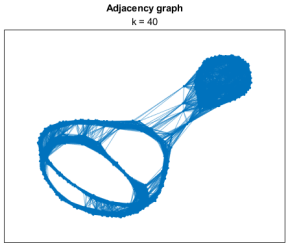
Looking at the circle dataset and using $k = 10, 20, 40$ let's plot the adjacency graph and the corresponding eigenvalues.



(g) k=10



(h) k=20



(i) k=40

k=10	k=20	k=40
0.000000000000000	0.000000000000000	0.000000000000000
0.000000000000000	0.011132309486027	0.048165715699889
0.004821571266605	0.065168295062573	0.702808630783828
0.028585613279939	0.161994708982443	0.779717288408653
0.042474376317906	0.172375616571161	0.906482619120527
0.042886570997170	0.322048843193205	1.376848088215342
0.080554938695825	0.339900325599712	1.480348291026562
0.126366831833811	0.561206417180896	2.400417206376394

We observe that the count of connected components in the adjacency graph aligns with the multiplicity of the eigenvalue 0.

4 Clustering

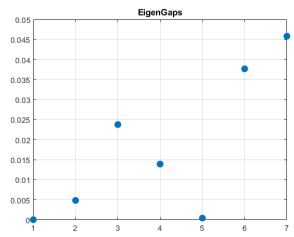
4.a Number of clusters

The eigengap heuristic is a commonly used method for selecting the number of clusters in spectral clustering because it leverages the information contained in the eigenvalues of the Laplacian matrix to determine a natural clustering structure in the data.

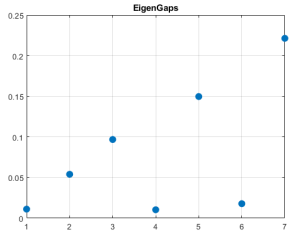
The eigenvectors corresponding to the smallest eigenvalues are used to embed the data into a lower-dimensional space, where it can be more easily clustered.

The eigengap heuristic suggests selecting the number of clusters as the value of k that maximizes the difference between the k-th and $(k+1)$-th smallest eigenvalues of the Laplacian matrix. This heuristic is based on the observation that the eigenvalues of the Laplacian matrix provide a measure of the "smoothness" of the data, and that the largest gaps between eigenvalues correspond to natural clusters in the data. For example the very first big jump corresponds to when the eigenvalue goes from 0 to a positive number and when there are k connected components, this is exactly k: this makes the number of connected components a good choice for the number of clusters.

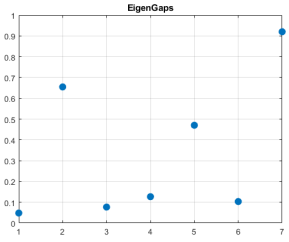
By selecting the number of clusters based on the eigengap heuristic, spectral clustering can effectively identify the underlying structure in the data and produce high-quality clustering results. It does not perform well, instead, when the clusters are noisy or overlapping.



(j) k=10



(k) k=20



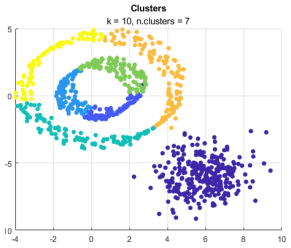
(l) k=40

Thus, it is evident that for values of k equal to 10, 20, and 40, the optimal selection for M is 7, but a better choice in the range of lower values could be 3 for $k = 10$, and 2 for $k = 40$.

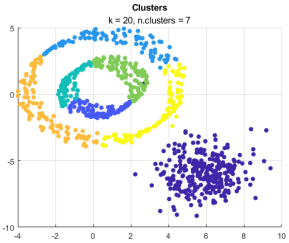
4.b Choice of clustering method

k-means clustering is a simple and efficient clustering method, often used in conjunction with spectral embedding. It works by iteratively assigning data points to the nearest cluster centroid and updating the centroids based on the new assignments, until convergence is reached and a final clustering is obtained.

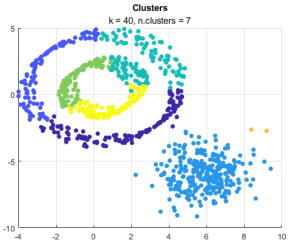
Here the computations of k-means:



(m) k=10

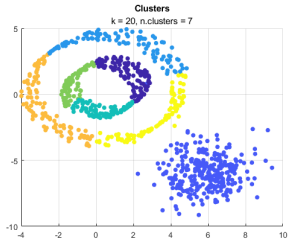


(n) k=20

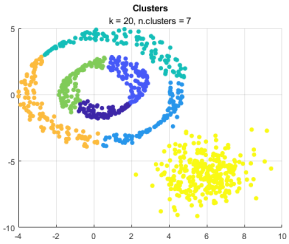


(o) k=40

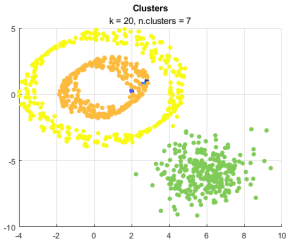
Ultimately, the choice of clustering method depends on the nature of the data and the goals of the analysis. In practice, it is common to try k-means and other clustering methods and compare their performance in terms of clustering quality and computational efficiency. To accomplish this, I implemented k-medoids, Ward hierarchical clustering and hierarchical clustering with euclidean distance for $k = 20$. Following the results:



(p) k-medoids



(q) Ward Hierarchical clustering



(r) Hierarchical clustering with Euclidean distance

Based on the analysis, it is evident that the Euclidean method reaches a lower number of clusters, spotting the exact three clusters that can be identified by eye; both Ward clustering and k-means show comparable results.

5 Normalized Laplacian Matrix

5.a Theoretical basis

The matrix:

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

is called normalized Laplacian matrix. Some of the pros of using this matrix in spectral clustering are:

1. *Robustness to scaling*: The normalized Laplacian matrix is robust to scaling of the data. This means that if the data is rescaled, the resulting clustering will not change. This is an important property because it allows the data to be preprocessed without affecting the clustering results.

2. *Spectral gap*: The normalized Laplacian matrix has a spectral gap, which means that the eigenvalues of the matrix are well-separated. This is important because it allows for easy identification of the number of clusters in the data. The number of clusters is equal to the number of eigenvalues that are greater than a certain threshold.

3. *Consistency*: The normalized Laplacian matrix is consistent, which means that as the number of data points increases, the clustering results become more accurate. This is important because it ensures that the clustering results are reliable, even for large datasets.

4. *Symmetry*: The normalized Laplacian matrix is again symmetric, which means that it can be diagonalized using an orthogonal matrix. This is important because it allows for efficient computation of the eigenvalues and eigenvectors of the matrix.

These properties make spectral clustering with a normalized Laplacian matrix an effective and reliable technique for clustering data.

5.b Relation between eigenvalues of L and L_{sym}

Let's now focus on the relations between eigenvalues of the regular and normalized Laplacian matrix; since L is positive semi-definite and symmetric, L is diagonalizable:

$$L = PAP^{-1}$$

where P has on the columns the eigenvectors of L and A is a diagonal matrix where the diagonal entries are the eigenvalues of L; since L is symmetric and positive semi-definite, then its eigenvectors forms an orthogonal basis and so $P^{-1} = P^T$:

$$L = PAP^T = \begin{bmatrix} | & | & & | \\ w_1 & w_2 & \cdots & w_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix} \begin{bmatrix} — & w_1 & — \\ — & w_2 & — \\ & \vdots & \\ — & w_n & — \end{bmatrix}$$

where I called w_i and λ_i the eigenvectors and eigenvalues of L. We can then look at the eigenvalues $\hat{\lambda}_i$ and eigenvectors v_i of L_{sym}:

$$\begin{aligned} D^{-\frac{1}{2}} L D^{-\frac{1}{2}} v &= \hat{\lambda} v \\ D^{-\frac{1}{2}} P A P^T D^{-\frac{1}{2}} v &= \hat{\lambda} v \\ (A P^T D^{-\frac{1}{2}}) v &= \hat{\lambda} (P^T D^{\frac{1}{2}}) v \end{aligned}$$

Then we can compute the two matrices in the brackets:

$$\begin{aligned} A P^T D^{-\frac{1}{2}} &= \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \begin{bmatrix} — & w_1 & — \\ & \vdots & \\ — & w_n & — \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{D_1}} & & \\ & \ddots & \\ & & \frac{1}{\sqrt{D_n}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{w_1(1)}{\sqrt{D_1}} & \frac{w_1(2)}{\sqrt{D_2}} & \cdots \\ \frac{w_2(1)}{\sqrt{D_1}} & \frac{w_2(2)}{\sqrt{D_2}} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \\ P^T D^{\frac{1}{2}} &= \begin{bmatrix} — & w_1 & — \\ & \vdots & \\ — & w_n & — \end{bmatrix} \begin{bmatrix} \sqrt{D_1} & & \\ & \ddots & \\ & & \sqrt{D_n} \end{bmatrix} \\ &= \begin{bmatrix} w_1(1)\sqrt{D_1} & w_1(2)\sqrt{D_2} & \cdots \\ w_2(1)\sqrt{D_1} & w_2(2)\sqrt{D_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \end{aligned}$$

where D_i is the i-th diagonal element of D.

So since:

$$(A P^T D^{-\frac{1}{2}} - \hat{\lambda} P^T D^{\frac{1}{2}}) v = 0$$

and v is an eigenvector for L_{sym} thus being different from the null vector, then the matrix $M := AP^T D^{-\frac{1}{2}} - \hat{\lambda} P^T D^{\frac{1}{2}}$ must have a $rank(M) < n$. Being $M_{i,j} = (\frac{\lambda_i}{\sqrt{D_j}} - \hat{\lambda}\sqrt{D_j}) w_i(j)$, I fixed the eigenvalue $\hat{\lambda}_k$ and I imposed that a linear combination of the rows of this matrix must be 0_n:

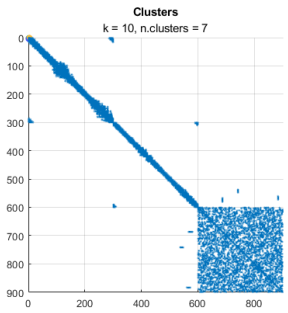
$$\begin{aligned} \forall j = 1, \dots, n: \quad & \sum_{i=1}^{n} \frac{\alpha_{i,k}}{\sqrt{D_j}} (\lambda_i - \hat{\lambda}_k D_j) w_i(j) = 0 \\ & \sum_{i=1}^{n} \alpha_{i,k} \lambda_i w_i(j) = \sum_{i=1}^{n} \alpha_{i,k} \hat{\lambda}_k D_j w_i(j) \\ & \hat{\lambda}_k = \frac{\sum_{i=1}^{n} \alpha_{i,k} \lambda_i w_i(j)}{D_j \sum_{i=1}^{n} \alpha_{i,k} w_i(j)}. \end{aligned}$$

This proves that there is a linear relationship between eigenvalues of L and L_{sym}.

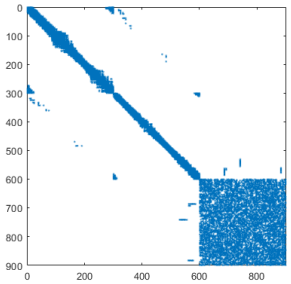
5.c Computational results

The following computations are the results of using the normalized Laplacian matrix:

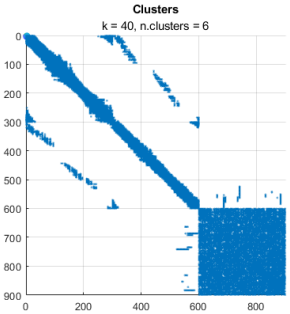
Sparsity patterns:



(s) k=10



(t) k=20

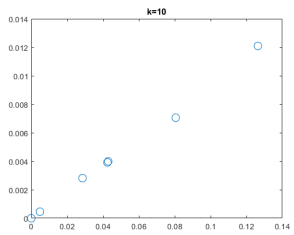


(u) k=40

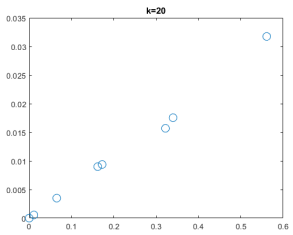
Eigenvalues:

k=10	k=20	k=40
0.000000000000000	0.000000000000000	0.000000000000000
0.000000000000000	0.000559548092543	0.001418919691899
0.000456190951577	0.003497413290786	0.025457301242011
0.002813660222829	0.009002307173512	0.027140676726954
0.003915628625542	0.009377246205805	0.030228351830862
0.003992971965608	0.015698242109621	0.042715854571498
0.007055614014802	0.017548434023604	0.083755849995226
0.012095357632533	0.031741777974625	0.091867468282118

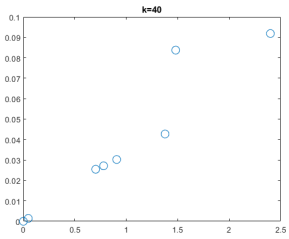
Comparison between eigenvalues with and without normalization:



(v) k=10



(w) k=20



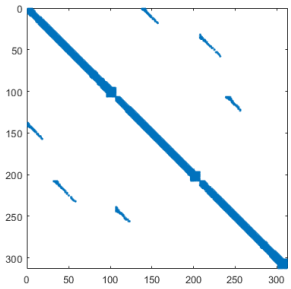
(x) k=40

So it seems that there is a linear relation between the eigenvalue before and after normalization, where the eigenvalues after normalization are much smaller.

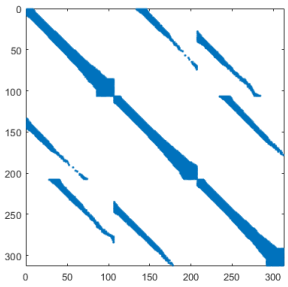
6 Spiral Dataset

Here I report the same computations for the spiral 3D dataset.

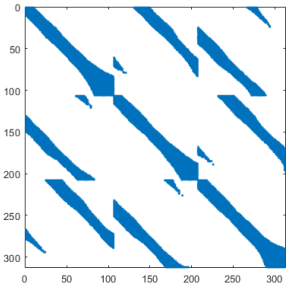
Sparsity pattern of the Laplacian matrix:



(y) k=10

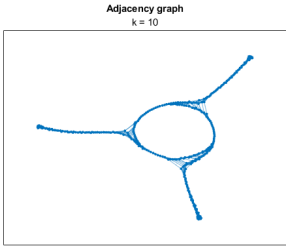


(z) k=20

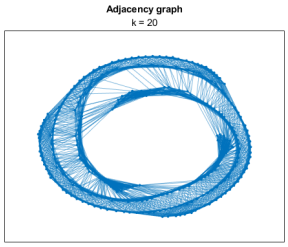


(aa) k=40

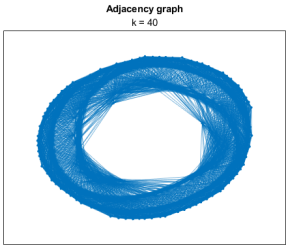
Adjacency graph:



(ab) k=10



(ac) k=20

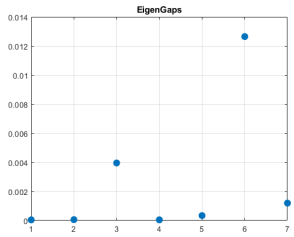


(ad) k=40

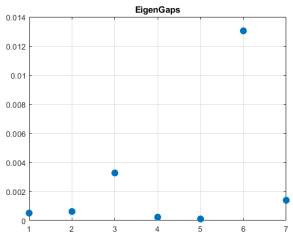
Eigenvalues:

k=10	k=20	k=40
0.000000000000000	0.000000000000000	0.000000000000000
0.000046098897213	0.000512354677666	0.000827669824789
0.000107698458959	0.001132187712059	0.001579284593435
0.004064429605757	0.004400856760607	0.004454455683574
0.004111628667764	0.004633755104461	0.004888099853938
0.004451774790485	0.004741311945941	0.005336226123243
0.017097751499587	0.017779237165630	0.017988656111637
0.018296636469800	0.019171319596946	0.019305591163640

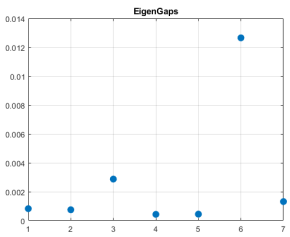
Eigengaps:



(ae) k=10

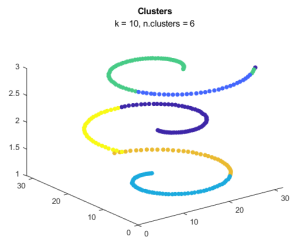


(af) k=20

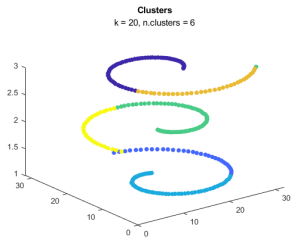


(ag) k=40

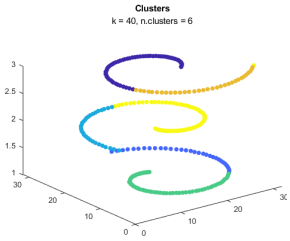
k-means:



(ah) k=10



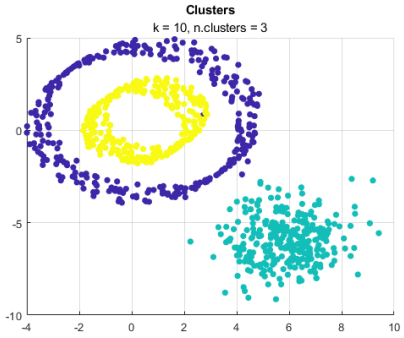
(ai) k=20

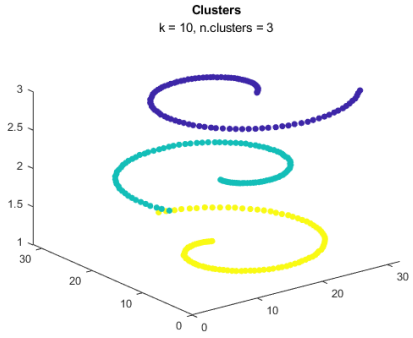


(aj) k=40

7 Interesting results

We can notice how, in both the spiral and the circle dataset, three clusters can be spotted: so I try to set the maximum number of clusters to 5; the results are accurate:





The plots above are obtained using the un-normalized Laplacian matrix, number of neighbors 10, and maximum number of clusters 5.

8 Landmines Dataset

The last dataset is a bitted map of landmine galleries.

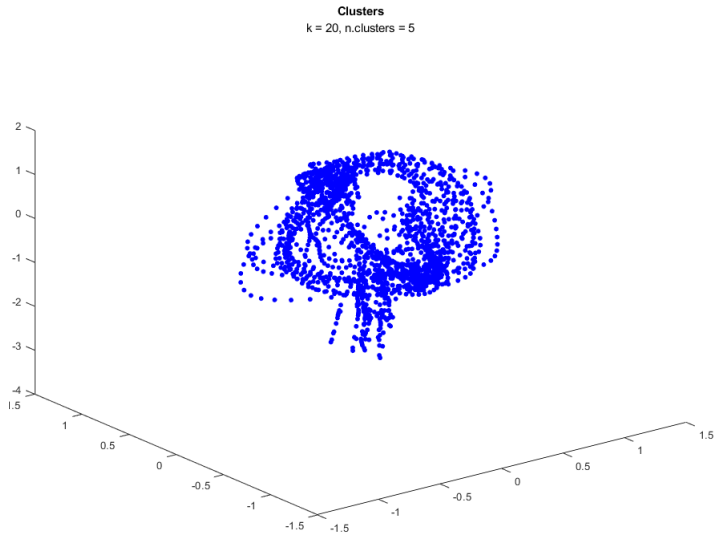
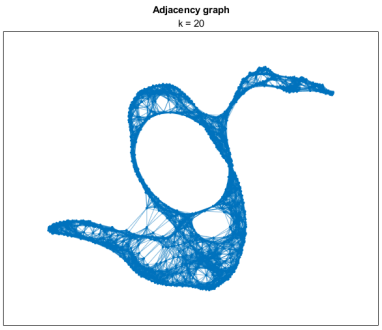
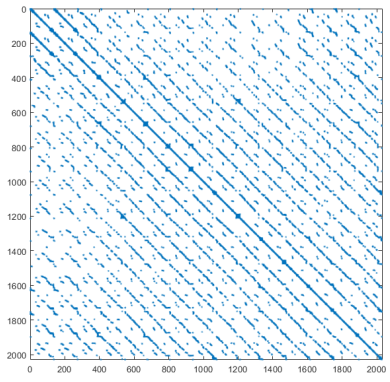


Figure 1: Landmines Dataset.

It consists of intertwined lines that represent different underground galleries in a coalmine. Ideally, spectral clustering is able to distinguish the galleries, or at list blocks of them.



(a) Adjacency graph.



(b) Sparsity pattern of the Laplacian matrix.

It's evident how the adjacency graph is not banded, so the computational costs rise.

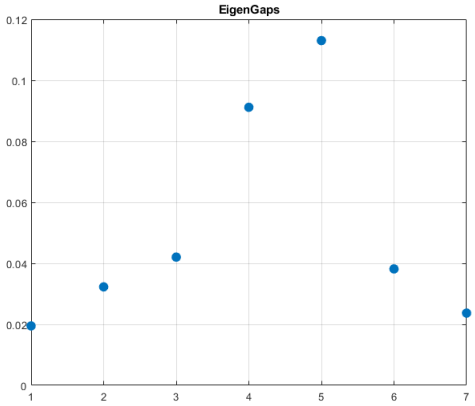


Figure 2: Eigengaps.

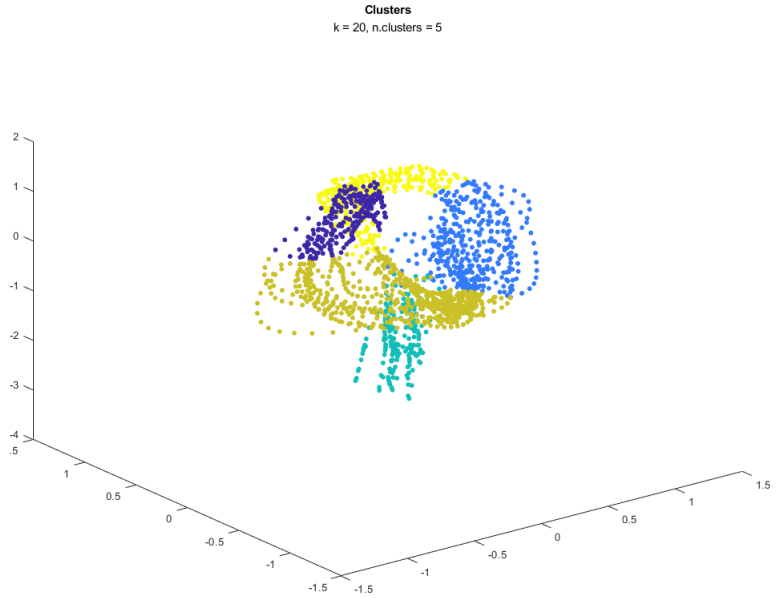


Figure 3: k-means.

Although the current results may not be satisfactory, they could be improved by reducing the number of neighbors.

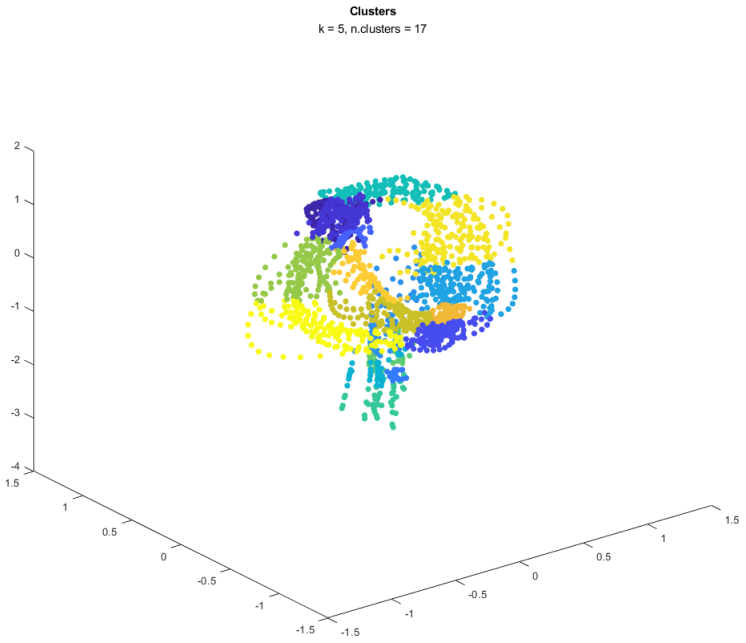


Figure 4: k-means.

The clustering is better but it could be better with $k = 2$ and using k-medoids.