Politecnico di Torino

Computational Linear Algebra for Large Scale Problems

Homework 2 - Spectral Clustering

Aurona Gashi s322791
Elisabetta Roviera s328422

21 January 2025

Contents

1 Introduction

In this document, we will explain how we have solved the tasks contained in the second homework of the course, as well as its optional points. The homework is about Spectral Clustering. The codes were implemented in the Matlab language and are provided in the same zip file as this document. The report is accompanied by an instruction file (*Instruction.txt*) that provides a comprehensive guide to programme execution and reproducibility of results. We were given two datasets, namely *'Circle.mat'* and *'Spiral.mat'*, and we created two additional datasets called *'3d_objects.mat'* and *'3d_sphere cube.mat'* to solve the optional exercises. Since we repeated the exercises three times for each dataset, varying the number of k-nearest neighbours, we will show each result, but we will only report the full code parts the first time, then only the lines that had to be changed to solve the other tasks will be shown here. The very next section will give a brief explanation of the topic.

2 Spectral clustering

Spectral Clustering is a powerful technique for grouping data points based on their underlying graph structure. Unlike traditional clustering methods that operate directly on the data points themselves, spectral clustering leverages the relationships between data points as represented by a graph.

The first step that needs to be done is define a similarity function, that, given two points X_i and X_j, computes the similarity s_{ij} between them. Based on this function, we can construct the similarity graph, represented by $G = (V, E)$, where $V = \{v_1, \ldots, v_n\}$ denotes a non-empty set of vertices, and E represents the set of edges, i.e., a set of pairs of vertices. In the similarity graph, each vertex $v_i \in V$ corresponds to a data point X_i. We assume that $s_{ij} = s_{ji}$ and that the edge connecting v_i and v_j is weighted by s_{ij}. As a result, the similarity graph turns out to be undirected. The weighted adjacency matrix is defined as $W_{ij} = s_{ij}$, when $i \neq j$, and $W_{ij} = 0$, when $i = j$. Then, we can compute the *degree matrix* $\boldsymbol{D}$:

$$\mathbf{D} = \operatorname{diag}\left(\sum_{j=1}^{n} W_{ij}\right)_{i=1}^{n}.$$

So, the *Laplacian matrix* can be computed:

$$\mathbf{L} = \mathbf{D} - \mathbf{W}.$$

The core of spectral clustering lies in analyzing the eigenvalues and eigenvectors of the Laplacian matrix. We compute the M smallest eigenvalues and their corresponding eigenvectors, where M is the desired number of clusters. These eigenvectors are then collected into a matrix U, where each row represents a data point in a new coordinate system defined by the eigenvectors. This transformation effectively projects the data points into a lower-dimensional space that highlights the underlying graph structure.

Finally, we apply a standard clustering algorithm, such as k-means, to the rows of the matrix U. This partitions the data points in the lower-dimensional space into M clusters, $C_1, \ldots, C_M$. The original data points are then assigned to the same clusters as their corresponding rows in U. Therefore the clusters $A_1, \ldots, A_M$, where:

$$A_i = \{x_j : y_j \in C_i\}.$$

will be obtained.

3 Dataset: Circle.mat

3.1 Upload the dataset and plot the points

The dataset under consideration contains 900 data points, which are arranged into two columns. These columns represent the two dimensions (x-values and y-values) of the points. It should be noted that each row of the dataset corresponds to a single point.

In order to undertake an initial analysis, the data contained within the file is to be graphically represented in Figure 1. The resulting plot reveals a non-linear distribution of points, with well-defined clusters. This feature suggests the presence of an intrinsic structure in the data. The code for generating the plot is provided below.

```
% Load the file
circle_mat = load('Circle.mat');

% Display the structure of the file
disp(circle_mat);

% Extract the matrix of points
X = circle_mat.X;

%  Plot the points
figure;
scatter(X(:,1), X(:,2), 10, Marker='*');
xlabel('X');
ylabel('Y');
title('Plot of the points in the Circle.mat dataset');
grid on;
```

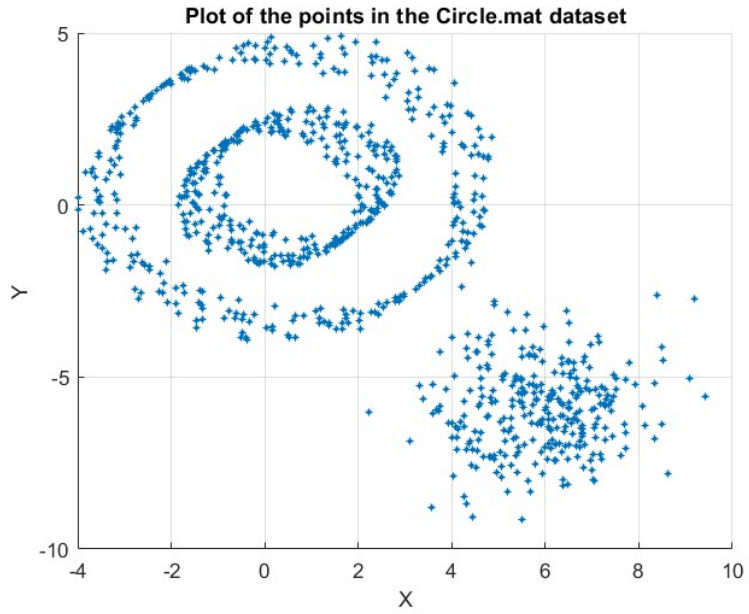


Figure 1: Plot of the points in Circle.mat.

In the following analysis, three distinct cases will be examined, with the number of the k-neighbours of the graph being varied.

3.2 Results for k = 10

3.2.1 Exercise 1

Given a set of data points X and the similarity function:

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

construct the k-nearest neighborhood similarity graph and its adjacency matrix W. Note that the adjacency matrix W has zero values on the diagonal by definition.

Solution:

```
%% 1. Choose k
k_values = [10, 20, 40];
k = 10; % 10, 20, 40

% Construct the k-nearest neighborhood similarity graph and its
    adjacency
% matrix W
W = knn_graph(X, k);

% Visualize the graph using its similarity matrix
figure;
spy(W);
title(['k-NN Graph (k = ', num2str(k), ')']);

% Store W as a sparse matrix
W = sparse(W);

% Visualize the graph G corresponding to the adjacency matrix W
G = graph(W);
figure;
plot(G);
title(['k-nearest neighborhood similarity graph, k = ', num2str(k)]);
```

The knn_graph function, which constructs the similarity matrix W, is defined as follows. The implementation of the k-nn algorithm for the construction of the W similarity graph required the introduction of a tolerance ($tol = 1e - 20$) in the comparison of distances. Without this tolerance, the symmetry of W was not guaranteed for all tested data sets.

```
function W = knn_graph(X, k)

% Function  that computes the adjacency matrix W based on the knn
% Inputs:
% X : Matrix containing the points as rows, their coordinates as
    columns
% k : number of neighbors that will be used
% Output:
% W: knn adjacency matrix

% Number of points
N = size(X, 1);

%Lower part of the matrix, since it is symmetric
W_lower = (zeros(N));
```

```
% Parameter sigma  for the similarity function
sigma = 1;

% Similarity function
similarity = @(xi, xj, sigma) exp(-sum((xi - xj).^2) / (2 * sigma^2));

% Create row and column indices for the lower triangle (excluding
    diagonal)
[row, col] = find(tril(true(N), -1));

% Use arrayfun to apply the similarity function
W_lower(sub2ind([N, N], row, col)) = arrayfun(@(i, j) similarity(X(i,
    :), X(j, :), sigma), row, col);
% The graph is undirected, so the matrix is symmetric
W = W_lower + W_lower';

% Copy of the matrix W
W_copy = W;

% Computing the k-nearest neighborhood adjacency matrix
for i = 1:N
    % Order the points in descending order wrt the similarity with i
    [~, idx] = sort(W_copy(i, :), 'descend');
    % knn = idx(1:k);

    % Indices of points that are not in the k-nearest neighborhood
    not_knn = idx(k + 1 : end);

    % The entry for these points will be 0
    W(i,not_knn) = 0;
end

% Tolerance that will be used to express if an entry of W is 0
eps = 1e-20;

% Symmetrizing method for the matrix W: if j is in the k-nn of i, then
    also i becomes a
% neighbor of j and viceversa, so now each row could have more than k
    values != 0
% Find indices where either W(i,j) or W(j,i) is greater than eps
idx = (W > eps) | (W' > eps);

% Update W using logical indexing
W(idx) = W_copy(idx);

% Check if it is actually symmetric
dist = norm(W-W');
end
```

Note that the matrix W is certainly symmetric before the *k_nearest neighbours* algorithm is applied to it. Afterwards, it is not guaranteed that it remains symmetric, because for each point we have to select the k points that have the maximum similarity with it, but we cannot be sure that if i is in the *k_nearest neighbourhood* of j, the same is true vice versa. Therefore, we apply an additional step to *"symmetrize"* the final matrix: if a point i is a neighbour of j, then j becomes a neighbour of i and vice versa. In this way, the rows of the similarity matrix could have more than $k = 10$ non-zero entries. The similarity matrix W is in Figure 2, where only the non-zero entries are coloured. The dominant

diagonal structure suggests strong local connectivity.

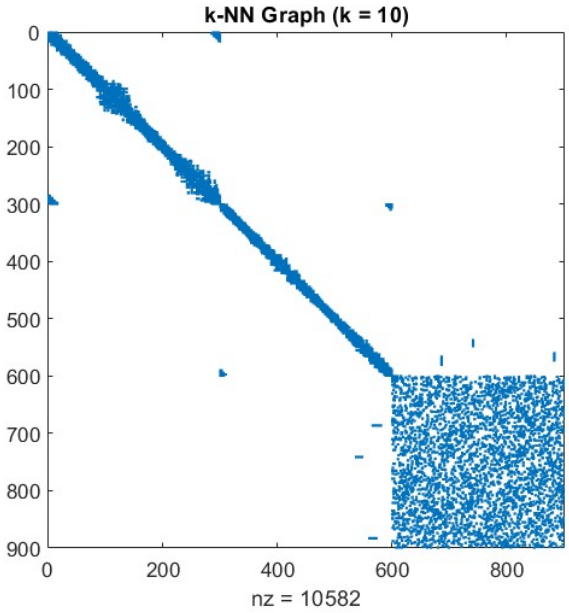


Figure 2: Similarity matrix, k = 10

A visual representation of the graph (Figure 3) can be shown with the Matlab function `graph`. Two connected components are observed, indicating that the data are naturally grouped into two main clusters. The connection between the two clusters is limited, as shown by the few blue dots off the diagonal in Figure 2.

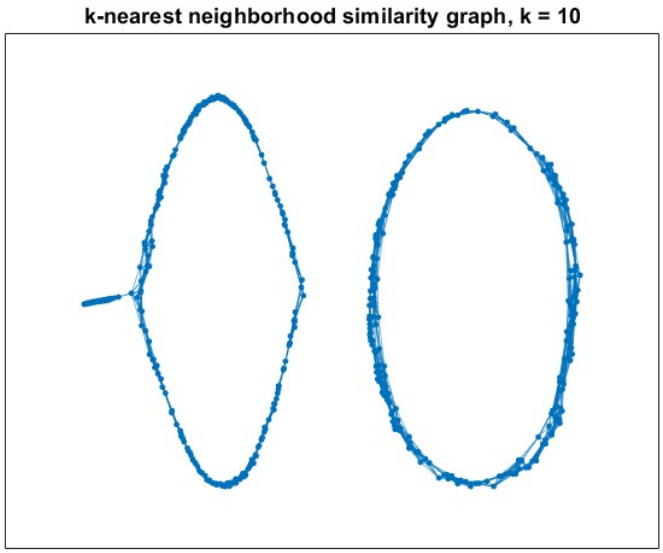


Figure 3: Similarity graph, k = 10

3.2.2 Exercise 2

Construct the degree matrix $\boldsymbol{D}$ and the Laplacian matrix $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$. The sparse format storage for the matrices $\boldsymbol{W}$, $\boldsymbol{D}$ and $\boldsymbol{L}$ is strongly preferable.

Solution:

```
%% 2.
% Number of points
N = size(W, 1);

% Inizialize the degree matrix D
D = zeros(N, N);

% The degree of each point is given by the sum of the elements of each
    row in W
D = diag(sum(W, 2));

% Save D in a sparse format
D = sparse(D);

% Compute the Laplacian matrix L
L = D - W;

if issparse(L) % issparse(L)==1 means that L is stored in a sparse
    format
     disp("The matrix L is stored in a sparse format")
end

% Plot the Laplacian amtrix L
figure;
spy(L);
title(['Matrix L (k = ', num2str(k), ')']);
```

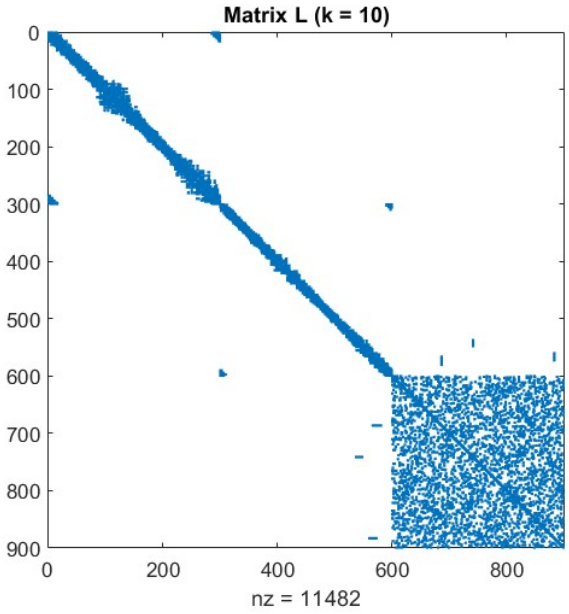


Figure 4: Laplacian matrix, k = 10

From the plots 2 and 4 we can see that the first (about) 600 points represent the circles in the upper left part of the plot 1, since they seem to have almost only the closest points as neighbours, in the way they are ordered in the dataset, while the lower right part is

represented by the points positioned in the lower right part of the plot, since they have connections with "random" neighbours belonging to that part.

3.2.3 Exercise 3

Compute the number of connected components of the similarity graph.

Solution:

```
%% 3.
% Compute the number of connected components of the graph

% The points with the same number belong to the same connected
   component
bins = conncomp(G);

% Number of connected components
num_components = max(bins);

% Display the result
disp(['Number of connected components: ', num2str(num_components)]);
```

"`Number of connected components:  2`".
We have already seen that the number of connected components is 2, so this just confirms it. Moreover, in the next exercise we will also find this value by looking at the number of eigenvalues of the Laplacian matrix equal to 0, because the number of 0 eigenvalues is equal to the number of connected components of the corresponding graph. So there are many different ways to solve this exercise.

3.2.4 Exercise 4, 5

Compute some small eigenvalues of L and use their values to choose a suitable number of clusters M for the points data-sets. A self-implementation of the numerical method for the eigenvalue approximation is preferable (see optional point 3).

Compute the M eigenvectors $u_1, \ldots, u_M \in \mathbb{R}^N$ that correspond to the M smallest eigenvalues of the Laplacian matrix, and define the matrix $U \in \mathbb{R}^{N \times M}$ with these vectors as columns. A self-implementation of the numerical method for the eigenvalue approximation is preferable (see optional point 3).

(Optional). Implement the Inverse Power Method and the Deflation Method to compute the M smallest eigenvalues of a symmetric matrix. See requests 4 and 5.

Solution:

```
%% 4, 5.
% Compute both M smallest eigenvalues of L and the corresponding
% eigenvectors using the deflation method and the inverse power method

% Set a number M of values to be computed (later it will be changed)
M = 10;

% Inizialize the eigenvalues vector and the eigenvectors matrix
eigvalues = zeros(M, 1);
```

```
eigvects = zeros(N, M);

% Choose the vector v that will be used for the inverse power method
v = 0.5 * ones(N, 1);
v(1:2:N) = -0.5;

% Max iterations in the power method
maxIter = 1000;
% Relative tolerance
relTol = 1e-10;

% A known fact from theory is that L is semi pos def and has at least
    one
% eigenvalue = 0 and that the vector of all ones is a corresponding
    eigenvector
% eigvalues(1) = 0;
 eigvects(:, 1) = ones(N,1)/ norm(ones(N,1));

% Or use the inverse power method to compute them
% [eigvalues(1), eigvects(:, 1)] = invpower_method(L, v(1:end), maxIter
    , relTol);

% Compute the reamining eigenvetors and eigenvalues
[eigvalues, eigvects, residualnorms] = deflation_method(L, v, eigvects,
     eigvalues,M, maxIter, relTol);

% Check how good the approximation is by comparing with eigs function
    of
% Matlab
[mat_eigvects, mat_eigs] = eigs(L, M, 'smallestabs');
norm(eigvalues - diag(mat_eigs))
```

```
ans = 4.1035e-10
```

The `deflation_method` function is as follows.

```
function [eigvalues, eigvects, residualnorms] = deflation_method(A, v,
    eigvects, eigvalues, M, maxIter, relTol)

% Function  that computes the M smallest eigenvalues and eigenvectors
    of a
% symmetrix matrix
% Inputs:
% A: matrix whose values will be computed
% v: starting vector for the inverse power method
% eigvects, eigvalues : contain the smallest eigenvalue and eigenvector
% already computed
% M: number of values to be computed
% maxIter : max number of iterations inside the inverse power method
% relTol : minimum relative tolreance to stop inverse power method
% iterations
% Outputs:
% eigvects, eigvalues : smallest M eigenvalues and corresponding
% eigenvectors
% residualnorms: vector containing the norm of A*x - lambda*x

% Number of points
N = size(v,1);
```

```
% Vector of all zeros except the first entry
e1 = zeros(N,1);
e1(1,1) = 1;

% Construct the j-th Householder matrix
Pj = eye(N,N) - 2*((eigvects(:, 1) + e1)*((eigvects(:, 1) + e1)')) /
   norm(eigvects(:, 1) + e1)^2;
% Construct the similar matrix Bj to A
Bj = Pj * A * Pj;

% inizialize the vector of residual norms
residualnorms = zeros(M,1);
residualnorms(1) = norm(A * eigvects(:,1) - eigvalues(1) * eigvects
   (:,1));

%Inizialize the matrix S, tha will be at each j step: P1*P2*...*Pj_1
S = eye(N);

% Loop to caclulate the eigenvalues and eigenvectors
for j = 2: M
    S = S * Pj;

    % Submatrix containg the remaining eigenvalues
    Aj =  Bj(j:end, j:end);

    % Find the next smallest eigenvalue and the eigenvector of Bj by
       using the inverse power method
    [eigvalues(j), x_bar] = invpower_method(Aj, v(j:end), maxIter,
       relTol);

    % Compute the sumbmatrix for the next matrix Pj
    Pbarj = eye(N -(j - 1),N -(j - 1)) - 2*(x_bar + e1(1: N - (j - 1)))
       *((x_bar + e1(1:N - (j - 1)))') /norm(x_bar + e1(1: N - (j - 1))
       )^2;
    Pj = zeros(N);
    % Position Pjbar below, in the right corner of Pj
    Pj(j : end, j : end) = Pbarj;

    % Fill the remaining elements on the diagonal with 1
    Pj(1:j-1, 1:j-1) = eye(j-1);

    % Construct the new matrix Bj
    Bj = Pj * Bj * Pj;

    % Using the property of symmetric matrices: the eigenvectors of
    % distinct eigenvalues are orthogonal (so we are assuming to have
    % distinct eigenvalues)

    % Matrix containing the eigenvectors computed previously
    X = eigvects(:, 1 :j - 1);

    % Solve the linear system to find the first y elements that along
       with
    % x_bar, will be used the jth eigenvector of A

    D =  X' *S(:,1: j - 1);
    c = - X' * S(:,j : end) * x_bar;
    y = D \ c;
```

```
    % Compute the jth eigenvector and the residual norm
    eigvects(:, j) = S *[y ; x_bar];
    eigvects(:, j) = eigvects(:, j) / norm(eigvects(:, j));
    residualnorms(j) = norm(A * eigvects(:,j) - eigvalues(j) * eigvects
        (:,j));
end
% Check whether it's orthonormal
% eigvects' * eigvects
```

The invpower_method function is as follows.

```
function [eigvalue, eigvect] = invpower_method(A, v, maxIter, relTol)
% Function that computes the smallest eigenvalue and corresponding
% eigenvector of the given matrix
% Inputs:
% A: Symmetric matrix (could also not be symm)
% v: initial vector
% maxIter: max number of iterations
% relTol: relative tolerance for the stopping criteria
% Outputs:
% eigvalue, eigenvect: smallest eigenvalue and corresponding
    eigenvector of the given matrix

% Size of v
v_size = size(A, 1);


% Inizialize values for the method

% Will contain the approximation of the eigenvalue computed ad each
    step
lambdas = zeros(maxIter, 1);
% Inizialize at a very large number
lambdas(1) = 1e8;

% Will contain the approximation of the eigenvector computed ad each
    step
v_vectors = zeros(v_size, maxIter + 1);

% Normalize v
v_vectors(:, 1) = v / norm(v);

% Start the loop
for k = 1:maxIter
    % Solve the linear system to find v_tilde = A^-1*v_vectors(:, k)
    v_tilde = A \ v_vectors(:, k);

    % Compute the new approximation of the eigenvalue
    lambdas(k + 1) = dot(v_vectors(:, k), v_tilde);

    % Normalize the new approximation of the eigenvector
    v_vectors(:, k + 1) = v_tilde / norm(v_tilde);

    % Check if the stopping criteria is met
    if abs(lambdas(k + 1) - lambdas(k)) < relTol * abs(lambdas(k + 1))
        break;
    end
end
```

```
% Save the computed values and vector
eigvalue = 1 / lambdas(k + 1)  ;
eigvect = v_vectors(:, k + 1);

end
```

We have implemented the `deflation_method` together with the `inverse_method` to compute the smallest eigenvalues and corresponding eigenvectors of the Laplacian matrix. From theory we know that L is semi-positive definite and has at least one eigenvalue equal to zero because a graph has at least one connected component. Therefore, in order to reduce the error resulting from solving an ill-conditioned linear system in the inverse power method, due to the high condition number of the matrix $\mathbf{L}$, we initialise the eigenvalues matrix by setting the first value to 0 and the first eigenvector to the normalised all-ones vector, since we know that $L\mathbf{1} = \mathbf{0}$. Following an initial analysis, a warning regarding the ill-conditioning of the matrix was received. Consequently, attempts were made to adopt techniques aimed at preventing the matrix from experiencing such poor conditioning, such as utilising a range of different preconditioners and adjusting the matrix itself. However, the ill-conditioning of the matrix remained unresolved, which ultimately led to the decision to maintain the previous version of the code. Then we continue by applying the two methods to compute the remaining eigenvalues and eigenvectors. We also tried the possibility of calculating the first eigenvalue and eigenvector with the inverse power method, still getting an eigenvalue very close to 0 and a very different eigenvector, but we didn't keep this calculation in order to limit the errors related to the ill-conditioned linear system.

We implemented the `deflation_method` in the case of a symmetric matrix, so to obtain the eigenvectors we took advantage of the property of symmetric matrices that states that the eigenvectors related to different eigenvalues must be orthogonal to each other. Since in this case the 0 eigenvalue appears with a multiplicity of 2, what we're doing is obviously finding an orthonormal subspace for it, so we find two orthonormal eigenvectors related to 0. Also, at each step of the inverse power method, we assume that the smallest eigenvalue in the input matrix has a strictly smaller magnitude compared to the others. If we look at the results and compare them with those obtained using the Matlab function `eigs`, we see that we get a high degree of accuracy. As far as eigenvectors are concerned, we obtained satisfactory results, especially for entries with values of order of magnitude e-4. Whereas for higher-order entrances (e-10), the results deviate, but not significantly, from the results produced by `eigs`; except for the first two, which correspond to the 0 eigenvalue. What we see is that `eigs` finds another pair of eigenvectors, which still forms an orthonormal basis for the 0 eigenspace, while not including the normalised all-ones eigenvector.
To assess the goodness of our results, we computed the norms of the residuals:

$$\|L \cdot \mathbf{v} - \lambda \cdot I\|_2.$$

We see that they are very low, so the approximations can be considered sufficiently accurate.

```
>> residualnorms'
ans =
   1.0e-05 *
```

```
    0.0000    0.0000    0.0003    0.0114    0.0298    0.0334    0.0395
         0.0911    0.1025    0.1534
```

The eigenvalues that we computed are

```
>> eigvalues'
ans =
      0    0.0000    0.0048    0.0286    0.0425    0.0429    0.0806
         0.1264    0.1349    0.2092
```

As expected, the two smallest ones are 0 as they represent the connected components of the graph.

```
 %%
% Now, find the actual number M of eigenvalues that will be used for
    the
% clustering algorithm

% Plot the computed eigenvalues
x = 1:M;

figure;
plot(x, eigvalues, '-o', 'LineWidth', 2);
xlabel('Index of eigenvalues');
ylabel('Eigenvalue magnitude');
title('Plot of the magnitude of the eigenvalues');
grid on;

%%
% k = 10
% The suitable number of eigenvalues is either 2 or 3 since
% eig4 is much larger than eig3

% k = 20, 40
% The suitable number of eigenvalues is 2 since eig3 is much
% larger than eig2

M = 2; %2, 3

% Define the matrix U that will be used for the spectral clustering
U = eigvects(:, 1:M);
```

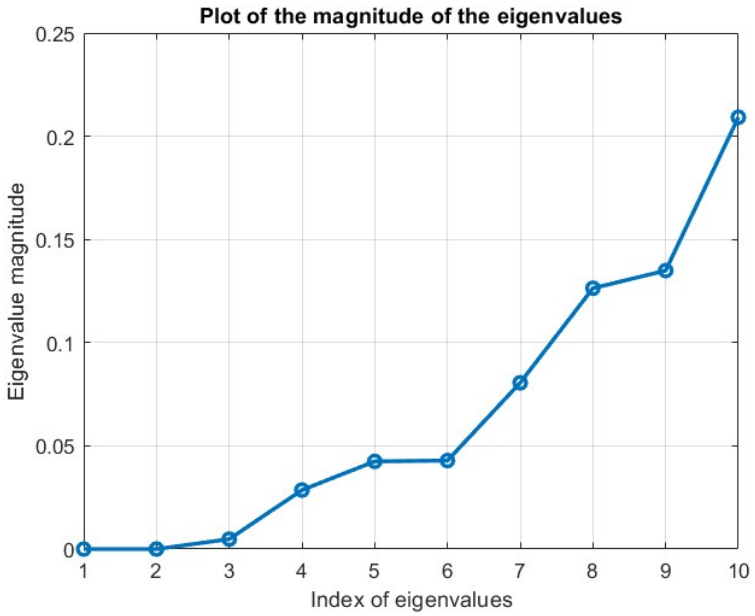


Figure 5: Choice of M, k = 10

Looking at the magnitudes, we can either keep $M = 2$ or $M = 3$. So we tried both cases. We made this choice because the smaller and closer the eigenvalues, the more the corresponding clusters represent well-separated subgraphs of the original graph. Therefore, when $M = 2$, we only consider the connected components of the original graph, but since the third eigenvalue is really close to the first two, this means that it is related to another subgraph that is weakly connected to the others. So the number of eigenvalues $\lambda_i \approx 0$ represents the number of subgraphs that are weakly connected to the others. If we chose $M = 4$, we would add a cluster belonging to a component of a graph that is quite well connected to the first three.

3.2.5 Exercise 6, 7, 8

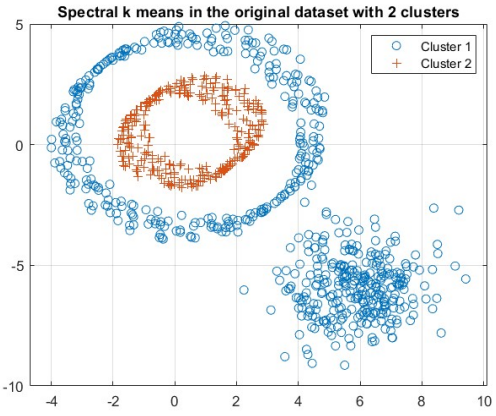
For $i = 1, \ldots, N$, let $\mathbf{y}_i \in \mathbb{R}^M$ be the vector corresponding to the i-th row of $\mathbf{U}$. Cluster the points $\mathbf{y}_i$, for $i = 1, \ldots, N$, in $\mathbb{R}^M$ using the k-means algorithm into clusters $C_1, \ldots, C_M$.

Assign the original points in $\mathbf{X}$ to the same clusters as their corresponding rows in $\mathbf{U}$ and construct the clusters $A_1, \ldots, A_M$, with $A_i = \{\mathbf{x}_j : \mathbf{y}_j \in C_i\}$.

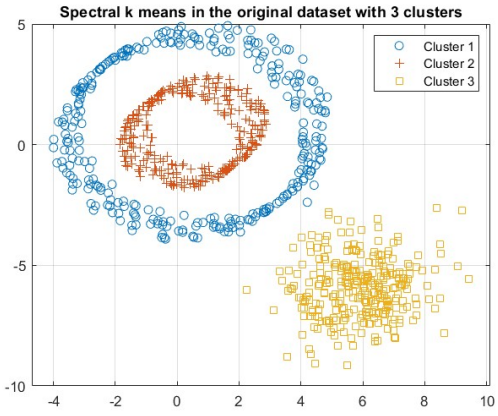
Plot the clusters of points $\mathbf{X}$ with different colors.
Solution:

```
%% 6, 7, 8.
% Spectral clustering; k-means

% Clusterize using k-means and obtain the indices (and the centroids)
    inside the clusters of each point
[idx, C] = kmeans(U, M);

% Assing the original data to the corresponding clusters
A = cell(M, 1);
```

```
for i = 1:N
    % Find the cluster of y_i
    cluster_idx = idx(i);

    % Assing it to x_i
    A{cluster_idx} = [A{cluster_idx}; X(i, :)];
end

% Plot of the clusterized data in the original space
X_spect_clust = zeros(N, 3);
X_spect_clust(:, 1: 2) = X;
X_spect_clust(:, 3) = idx;

figure;
markers = ['o', '+', 's'];
gscatter(X_spect_clust(:,1), X_spect_clust(:,2), X_spect_clust(:, 3),
    [], markers, [], 5);
title(['Spectral k-means in the original dataset with ', num2str(M), '
    clusters']);
legend('Cluster 1', 'Cluster 2', 'Cluster 3');
grid on
```



(a) Spectral k-means, M = 2 clusters



(b) Spectral k-means, M = 3 clusters

Here, with 2 clusters (Figure 6a), we can see that the inner circle forms a separate cluster. With 3 clusters (Figure 6b), the separation between the main figures is clear and correct.

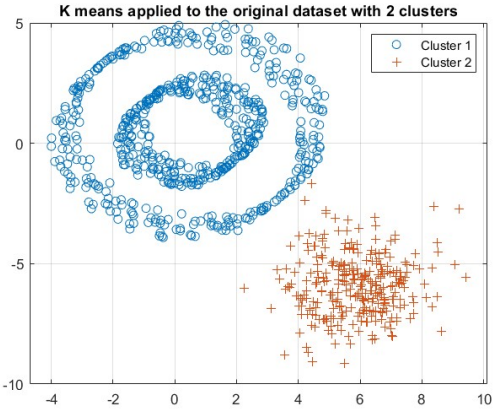
3.2.6 Exercise 9

Compute the clusters for the same set of points with other clustering methods (k-means,...) and compare the results.

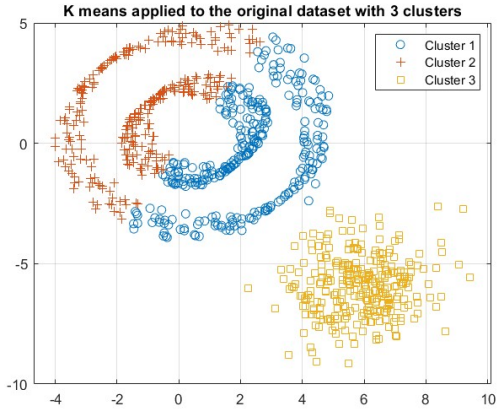
Solution: We applied the k- means clustering and the DBSCAN clustering to the original data X. When using DBSCAN, the outliers are included together in the cluster '-1'. We first take a look at the k-means results:

```
%% 9.a k-means TO THE ORIGINAL DATA
k_value = 2;  %2 , 3

% Clusterize the original data
[idx_k, C_k] = kmeans(X, k_value);
```

```
% Add the index to X_kmeans
X_kmeans = zeros(N, 3);
X_kmeans(:, 1: 2) = X;
X_kmeans(:, 3) = idx_k;

figure;
markers = ['o', '+', 's'];
gscatter(X_kmeans(:,1), X_kmeans(:,2), X_kmeans(:, 3), [], markers, [],
    5);
title(['k-means applied to the original dataset with ', num2str(k_value
   ), ' clusters']);
legend('Cluster 1', 'Cluster 2', 'Cluster 3');
grid on;
```



(a) k-means, M = 2 clusters

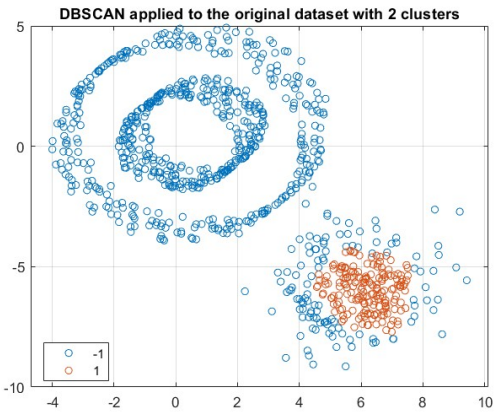


(b) k-means, M = 3 clusters

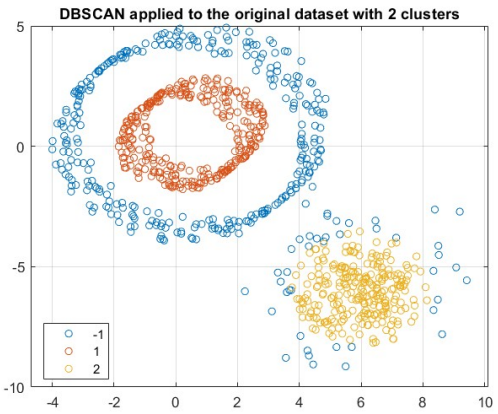
When using 2 clusters (Figure 7a), the clustering is done differently than before. When there are 3 clusters (Figure 7b), we can see that the k-means does not handle the division well, as it only takes into account the distance between the points without any kind of similarity, thus giving the wrong division and not dividing the circles in the right way.

The code and results for the DBSCAN algorithm are as follows.

```
%% 9.b DBSCAN TO THE ORIGINAL DATA
% Apply the DBSCAN algorithm
% Neighborhood radius
epsilon = 0.5;
% Minimum points for a cluster
minPts = 10;
idx_d = dbscan(X, epsilon, minPts);

X_dbscan = zeros(N, 3);
X_dbscan(:, 1: 2) = X;
X_dbscan(:, 3) = idx_d;

figure;
gscatter(X_dbscan(:,1), X_dbscan(:,2), X_dbscan(:, 3), [], 'o', 5);
title(['DBSCAN applied to the original dataset with ', num2str(M), '
    clusters']);
grid on;
```



(a) DBSCAN 1



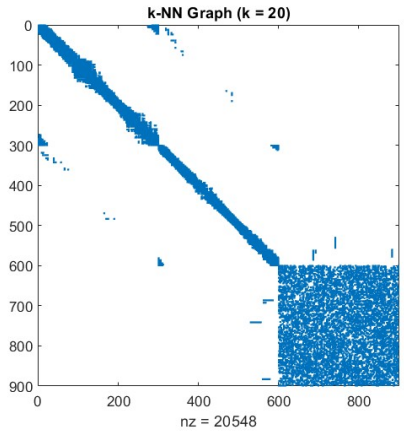
(b) DBSCAN 2

Figures 8a and 8b show the results of applying the DBSCAN algorithm to the original dataset with the aim of identifying two and three clusters respectively. The analysis shows that by varying the DBSCAN parameters, the separation of the clusters is not accurate. This behaviour is consistent with the inherent characteristics of the algorithm, which struggles to deal with datasets with varying densities or complex shapes. In both cases DBSCAN fails to capture the spiral structure of the dataset. This highlights DBSCAN's limited ability to handle data sets with this particular geometry. Hence, the spectral k-means method proves to be the most effective clustering strategy for this particular dataset, as it is able to faithfully represent its intrinsic structure.

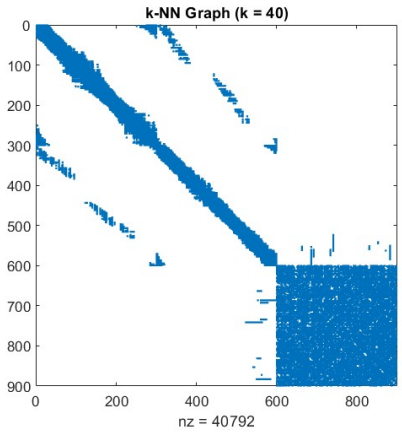
3.3 Results for k = 20, 40

We are going to put together the results in relation to $k = 20$ and $k = 40$, because there are not that many differences between them.

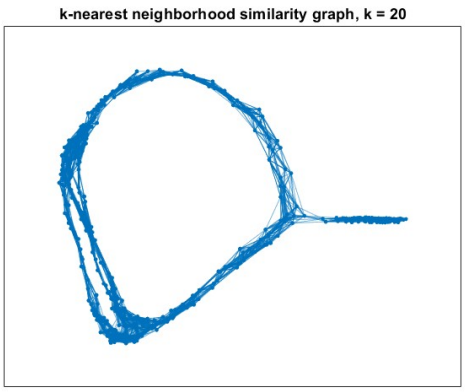
3.3.1 Exercise 1



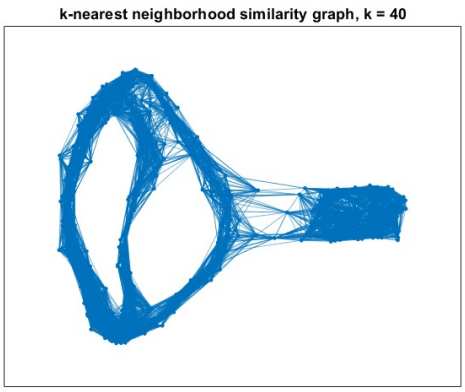
(a) Similarity matrix, k = 20



(b) Similarity matrix, k = 40



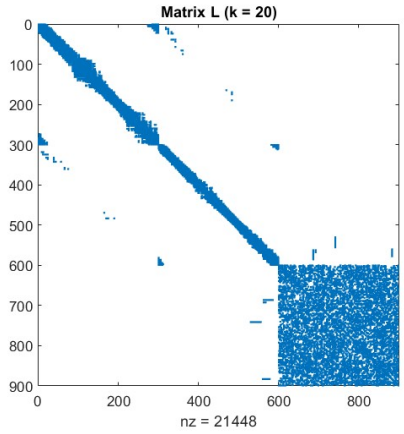
(a) Similarity graph, k = 20



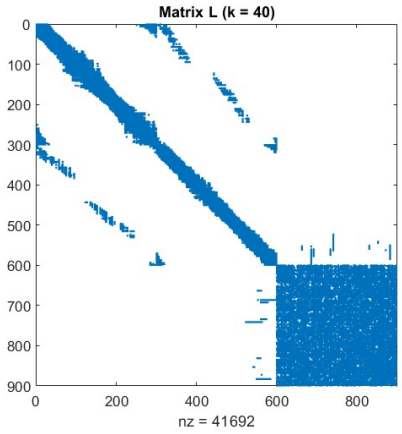
(b) Similarity graph, k = 40

The division is quite similar to the case we had with $k = 10$, except that the connections within the objects are stronger, while especially with $k = 40$ there are more defined cross connections between the circles and the larger circle and the remaining cloud of points. That's why we now get a single connected component.

3.3.2 Exercise 2



(a) Laplacian matrix, k = 20

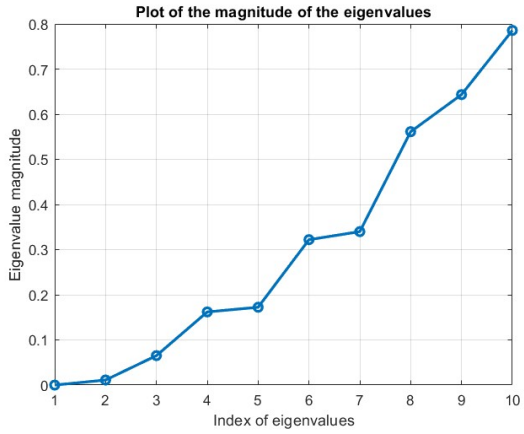


(b) Laplacian matrix, k = 40

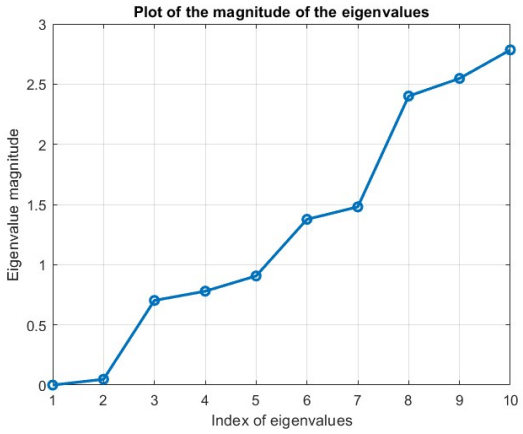
3.3.3 Exercise 3

The result is: "`Number of connected components: 1`".
In both scenarios analysed, the application of the Connected Component Identification algorithm unambiguously determined the presence of a single connected component, indicating that all points in the dataset considered are mutually accessible within the graph structure.

3.3.4 Exercise 4, 5



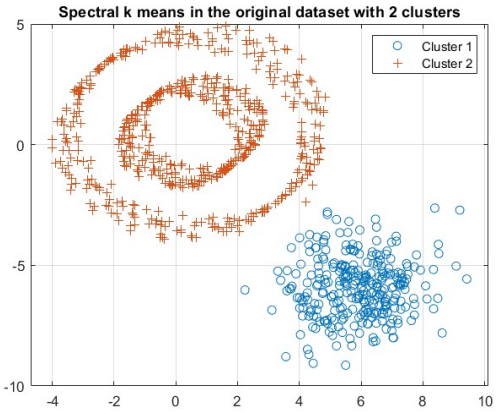
(a) Choice of M, k = 20



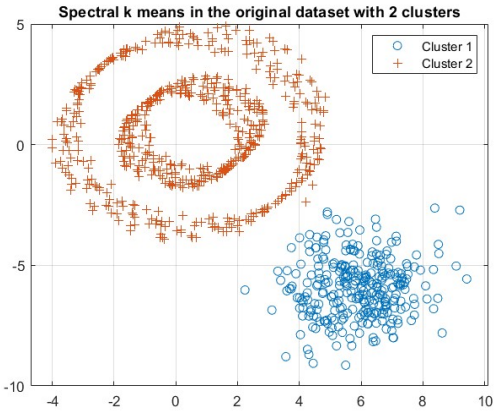
(b) Choice of M, k = 40

Since there is a significant difference between the second and third eigenvalue, we chose $M = 2$.

3.3.5 Exercise 6, 7, 8



(a) Spectral k-means, k = 20 neighbors



(b) Spectral k-means, k = 40 neighbors

In both cases, the same partitioning of the data is observed: the two concentric rings are grouped into one cluster, while the isolated group of points forms the second cluster. This result is different from that obtained with $k = 10$ and $M = 2$, where the two rings were separated. However, the partitioning obtained with $k = 20$ and $k = 40$ agrees with that produced by the k-means algorithm applied directly to the original data with $k = 2$ clusters, suggesting that for sufficiently high values of k, spectral clustering tends to converge to a solution similar to that of standard k-means, losing the ability to capture more complex structures present in the data set.

3.3.6 Exercise 9

The partition obtained is consistent with previous observations and depends solely on the configuration of the input data, as it is not affected by the variation of the parameter k.

4 Dataset: Spiral.mat

The dataset under consideration consists of $N = 312$ observations, each described by three attributes. The first two attributes represent the two-dimensional coordinates of the points in space, while the third attribute encodes the membership of each point in the reference cluster, thus providing the class label.

4.1 Upload the dataset and plot the points

```
%%
% Load the file
spiral_mat = load('Spiral.mat');

% Display the structure of the file
% disp(spiral_mat);

% Extract the matrix of points
% X_tot contains the right labels of the clusters as third column
X_tot = spiral_mat.X;
X = X_tot(:, 1:2);

%%  Plot the points
figure;
scatter(X(:,1), X(:,2), 10, Marker='*');
xlabel('X');
ylabel('Y');
title('Plot of the points in the Spiral.mat dataset');
grid on;
```

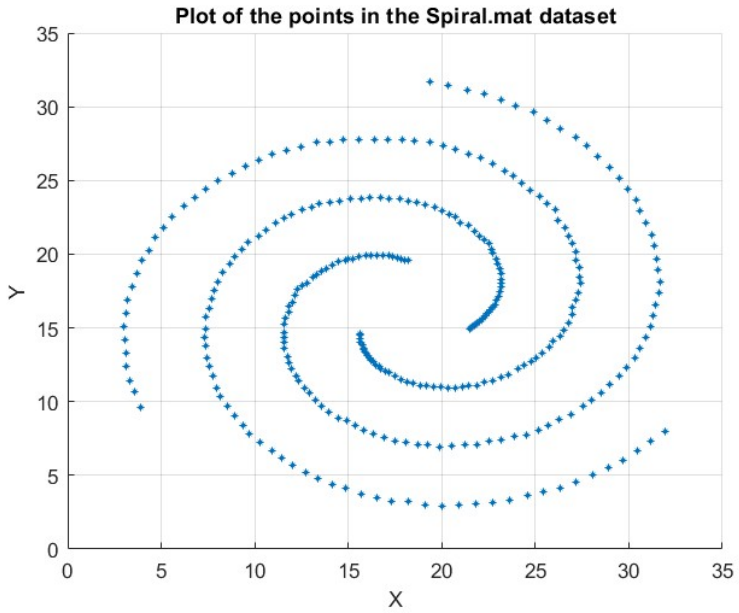


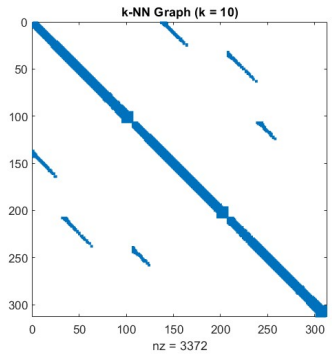
Figure 14: Plot of the points in Spiral.mat.

4.2 Results for k = 10, 20, 40

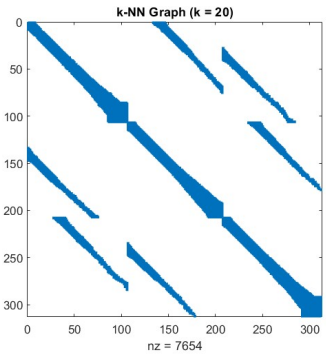
Given the substantial agreement of the results obtained with the spectral clustering k-means as the parameter k varies, we will proceed with a joint discussion of the results, generalising the observations.

4.2.1 Exercise 1

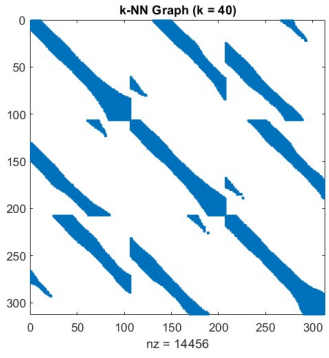
The figures below show the similarity matrix and the graphical representation of the k-NN graph for different values of k. It is evident that there are three distinct components in the dataset, corresponding to the three spirals. Within each spiral, points show strong connections with their closest neighbours, as evidenced by the diagonal block structure in the similarity matrices. Inter-spiral connections are also observed, the intensity of which increases proportionally with increasing k. This behaviour is expected, since a greater value of k implies the consideration of a greater number of neighbours for each point, thus increasing the probability of creating connections between different, even more distant spirals. In particular, for $k = 40$, the structure of the spirals is almost completely lost in the graph representation, due to the large number of inter-cluster connections.



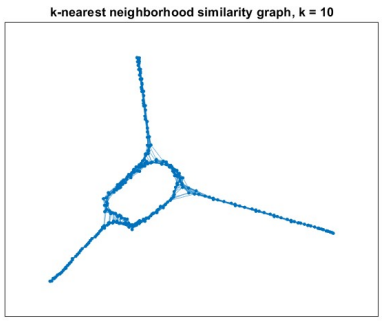
(a) Similarity matrix, k = 10



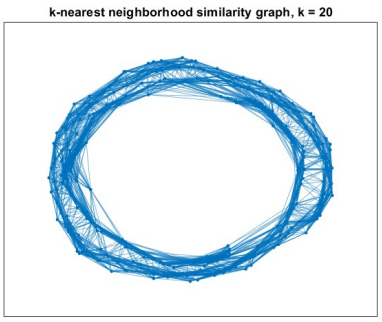
(b) Similarity matrix, k = 20



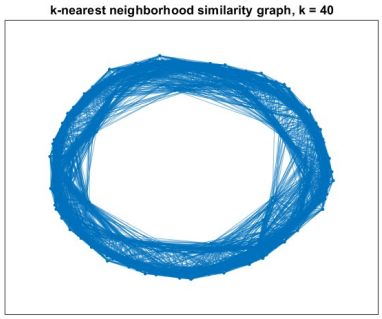
(c) Similarity matrix, k = 40



(a) Similarity graph, k = 10



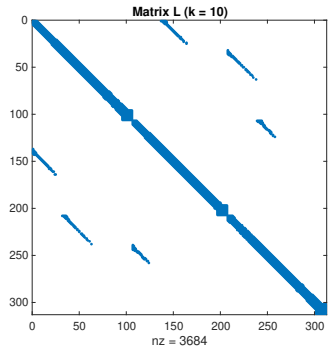
(b) Similarity graph, k = 20



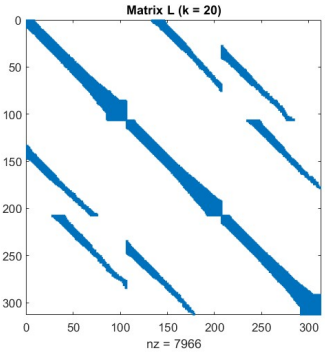
(c) Similarity graph, k = 40

4.2.2 Exercise 2

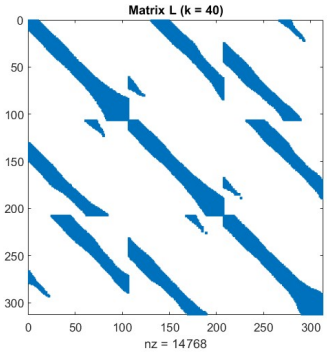
The diagonal block structure indicates the presence of clusters. As k increases, blocks become more connected and off-diagonal connections increase, suggesting greater interconnections between clusters and a potential loss of their distinctiveness.



(a) Laplacian matrix, k = 10



(b) Laplacian matrix, k = 20



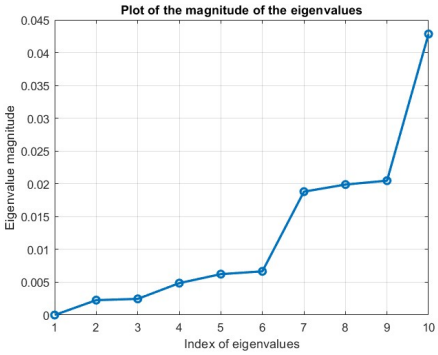
(c) Laplacian matrix, k = 40

4.2.3 Exercise 3

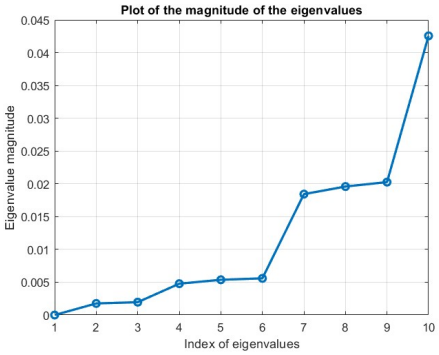
For every k the result is: "`Number of connected components: 1`".
This indicates that the inter-spiral connections are sufficiently robust to prevent the spirals from being distinguished into separate connected components.

4.2.4 Exercise 4, 5

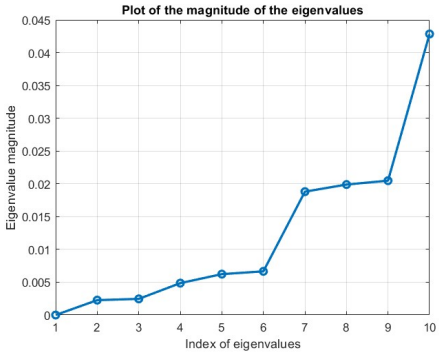
As can be seen from the graphs below, the first three eigenvalues (the first of which is zero) have values close to each other, while the fourth eigenvalue shows a significant distance to the previous ones. This observation motivates the choice of $M = 3$ as the number of eigenvalues to be considered for spectral clustering, since it suggests the presence of three distinct clusters in the dataset.



(a) Choice of M, k = 10

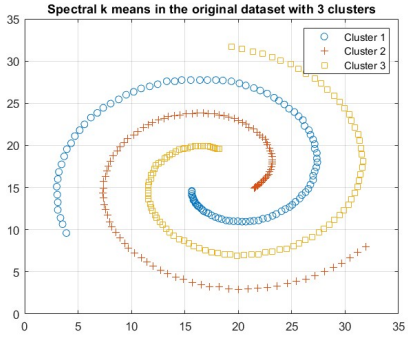


(b) Choice of M, k = 20

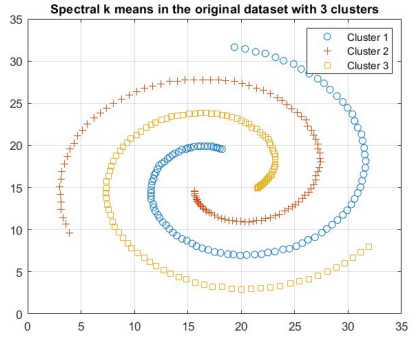


(c) Choice of M, k = 40

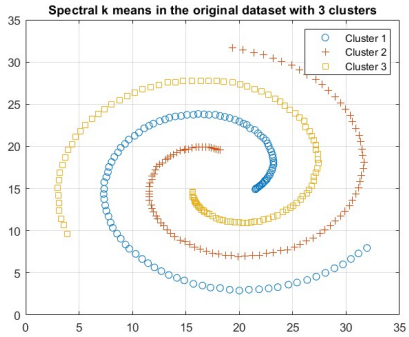
4.2.5 Exercise 6, 7, 8



(a) Spectral k-means, k = 10 neighbors



(b) Spectral k-means, k = 20 neighbors



(c) Spectral k-means, k = 40 neighbors

It is observed that the method correctly separates the three spirals into distinct clusters for all values of k considered, demonstrating the robustness of spectral clustering in this case.

As expected, the division is the same in all cases (the numbers assigned to the clusters are arbitrary) and is done correctly, as we can see by comparing the results with the correct clusters.

```
%% Difference with original clusters
% Plot the original clusters
figure;
markers = ['o', '+', 's'];
gscatter(X_tot(:,1), X_tot(:,2), X_tot(:, 3), [], markers, [], 5);
title('Correct clusters');
legend('Cluster 1', 'Cluster 2', 'Cluster 3');
grid on;

% Shift the numbers in X_spect_clust(:,3) to have the same names as in
% X_tot

first_value = X_spect_clust(1, 3);
second_value = X_spect_clust(160, 3);
third_value = X_spect_clust(310, 3);

for i = 1:length(X_spect_clust(:, 3))
    if X_spect_clust(i, 3) == first_value
        X_spect_clust(i, 3) = 3;
    elseif X_spect_clust(i, 3) == second_value
        X_spect_clust(i, 3) = 1;
    elseif X_spect_clust(i, 3) == third_value
        X_spect_clust(i, 3) = 2;
    end
end

% Compute the difference
diff_clust = sum((X_tot(:,3) - X_spect_clust(:,3)));
%for M = 3, it is 0 as expected from the plots
```

```
diff_clust = 0
```

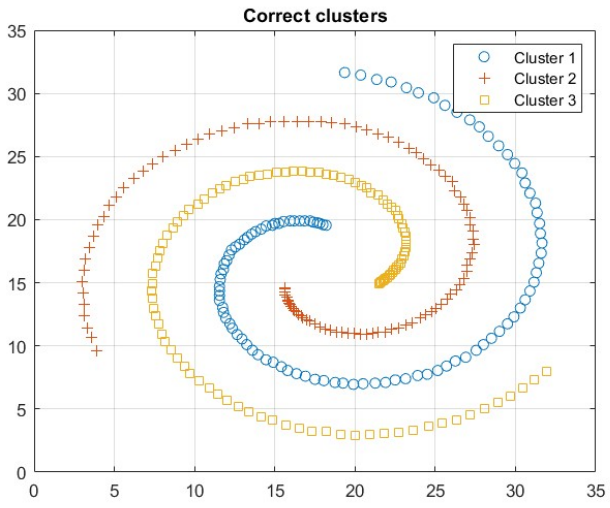
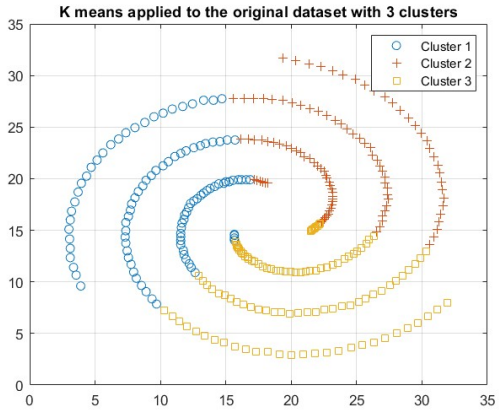


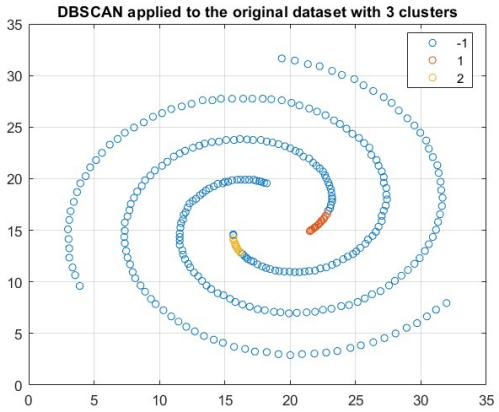
Figure 20: Correct clusters

4.2.6 Exercise 9

The figures below show the results obtained by applying k-means and DBSCAN to the data set of the three spirals in order to identify three clusters. Both methods show limitations in correctly separating the spirals. K-means tends to divide the clusters according to their geometric compactness, rather than following the spiral structure. DBSCAN identifies part of the spirals but fails to separate them completely, showing difficulty in dealing with the density variation along the spirals. These results highlight the difficulties of these algorithms in capturing non-linear and density-varying data structures. Therefore, k-means spectral clustering is again confirmed as the most effective method for this particular data set, capable of accurately representing its intrinsic structure.



(a) k-means, k = 3 clusters



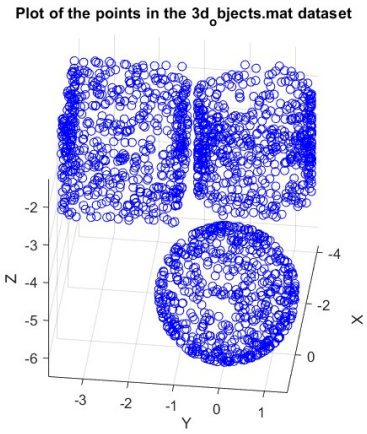
(b) DBSCAN

5 First optional dataset: 3d_objects.mat

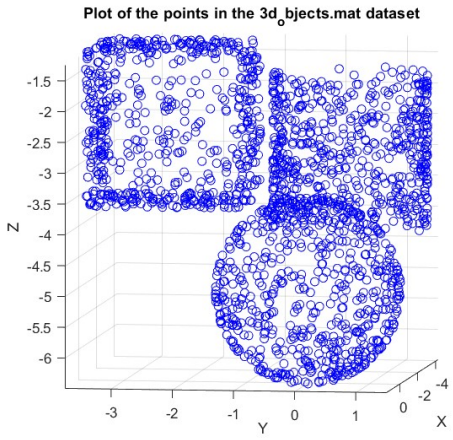
This dataset consists of 1800 rows (points) in 3 dimensions (3 columns). We created the dataset using the file `datasets_generation.m`. Three objects were created: a sphere, a cube and a cylinder. The objects were created by placing the points on the surfaces of the objects ("randomly", with a fixed random seed); the cylinder has no points on the top and bottom surfaces.

5.1 Upload the dataset and plot the points

From the first graph (Figure 22a), it can clearly be seen that the three objects (cylinder, cube and sphere) are distinct and spatially separated from each other, with no obvious overlaps or intersections.



(a) Plot of the points in 3d._objects.mat from above

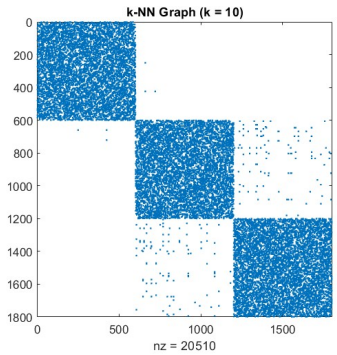


(b) Plot of the points in 3d._objects.mat from the front

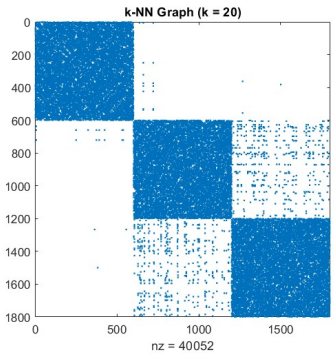
5.2 Results for k = 10, 20, 40

Similarly to what has been done previously, given the consistency of the results obtained when the parameters considered vary, a joint discussion will be held analysing the characteristics common to all scenarios.

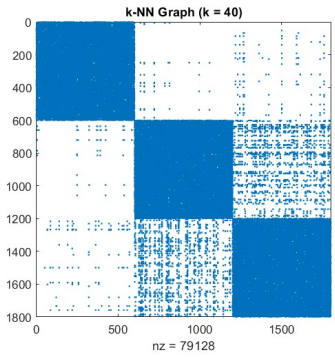
5.2.1 Exercise 1



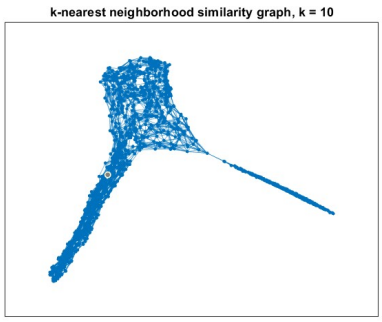
(a) Similarity matrix, k = 10



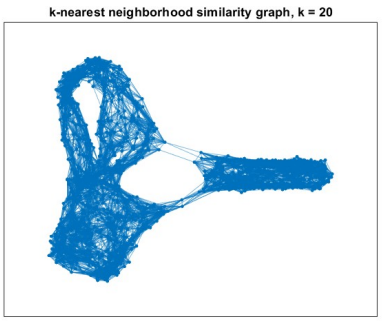
(b) Similarity matrix, k = 20



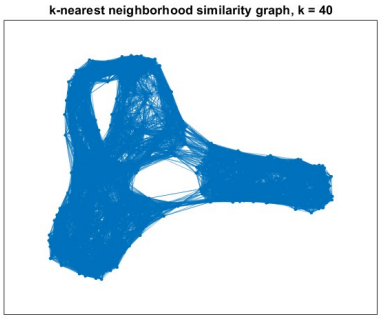
(c) Similarity matrix, k = 40



(a) Similarity graph, k = 10



(b) Similarity graph, k = 20



(c) Similarity graph, k = 40

As we can see, there are 3 components in the dataset (the three figures) with strong connections of each point with the closest ones belonging to the same object and also some connections with the points of the other objects, especially between the cylinder and the cube, since they are the closer ones.

5.2.2 Exercise 2

We did the computation also using the normalized symmetric Laplacian matrix, as the second optional point states: apply the same process to the normalized symmetric Laplacian matrix $L_{\text{sym}} \in \mathbb{R}^{N \times N}$ that is defined as

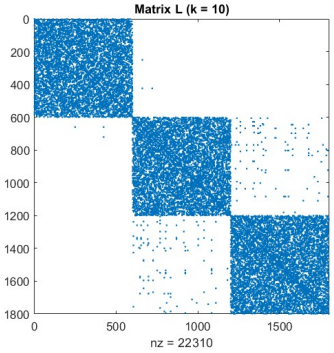
$$L_{\text{sym}} := D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}.$$

Solution:

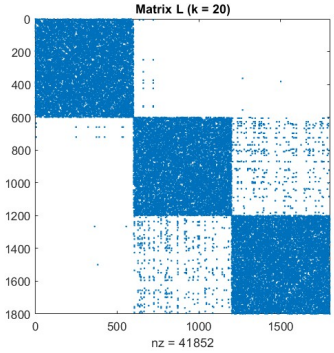
```
%% 2.
% Number of points
N = size(W, 1);

% Inizialize the degree matrix D
D = zeros(N, N);

% The degree of each point is given by the sum of the elements of each
    row in W
D = diag(sum(W, 2));

% Save D in a sparse format
D = sparse(D);
```

```
% Compute the Laplacian matrix L
L = D - W;
L = sparse(L);

if issparse(L) % issparse(L)==1 means that L is stored in a sparse
    format
     disp("The matrix L is stored in a sparse format")
end

% Compute the normalized Laplacian matrix L(sym)
D_inv_sqrt = diag(1 ./ sqrt(diag(D)));
L = D_inv_sqrt * L * D_inv_sqrt;

% Plot the Laplacian matrix L
figure;
spy(L);
title(['Matrix L (k = ', num2str(k), ')']);
```

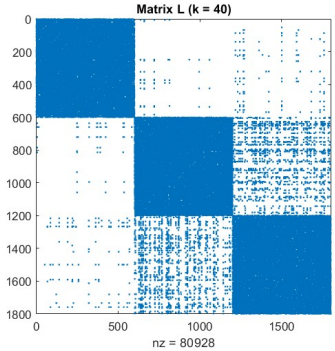
Obviously, the L-matrix is the same in both cases in the following illustration.



(a) Laplacian matrix, k = 10



(b) Laplacian matrix, k = 20



(c) Laplacian matrix, k = 40

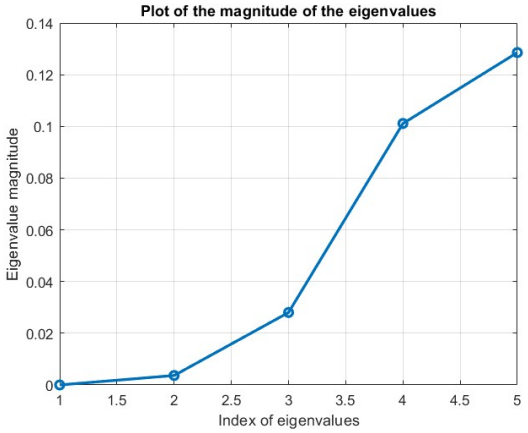
5.2.3 Exercise 3

The result is: "`Number of connected components: 1`". This means that the connections between the objects are sufficiently strong to group the points into one connected component.

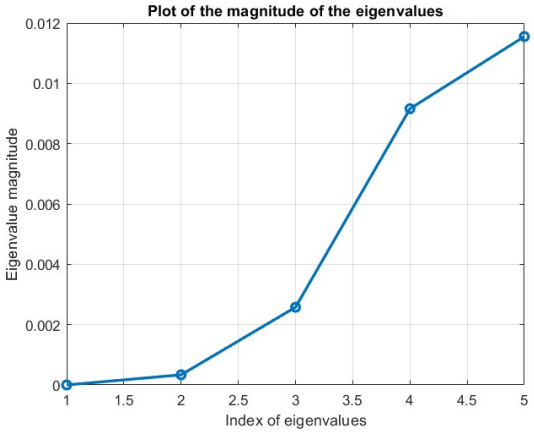
5.2.4 Exercise 4, 5

We now compare the results obtained with **L** and **L_sym**. When computing the eigenvalues of **L_sym**, we apply the inverse power method from the beginning since from theory

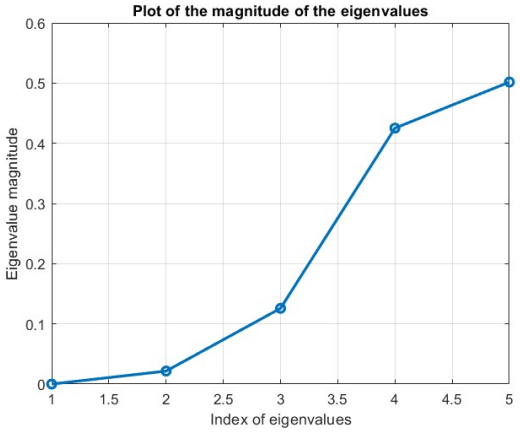
we don't know the value of the first eigenvector.



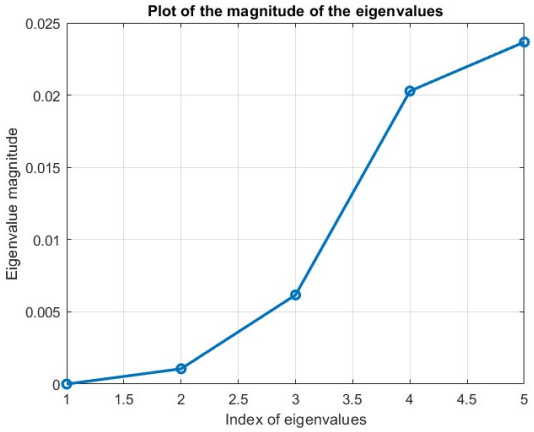
(a) Choice of M with L, k = 10



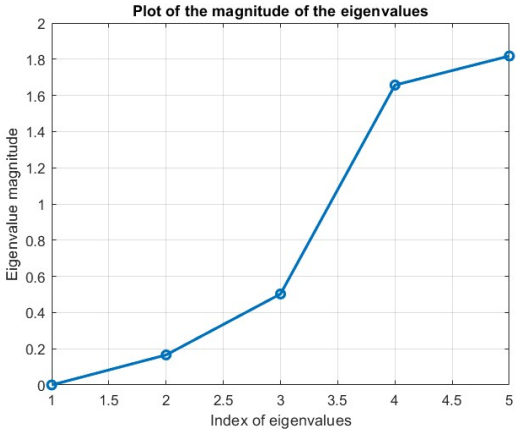
(b) Choice of M with L_sym, k = 10



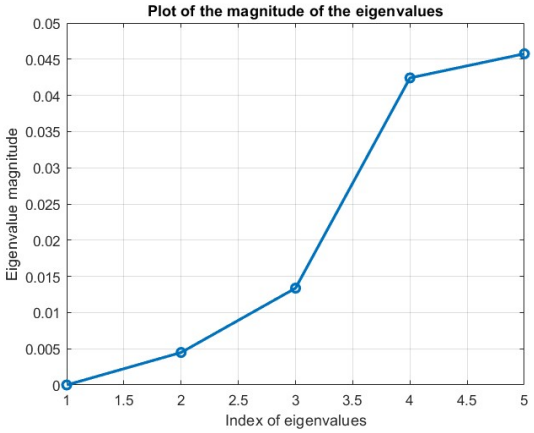
(a) Choice of M with L, k = 20



(b) Choice of M with L_sym, k = 20



(a) Choice of M with L, k = 40

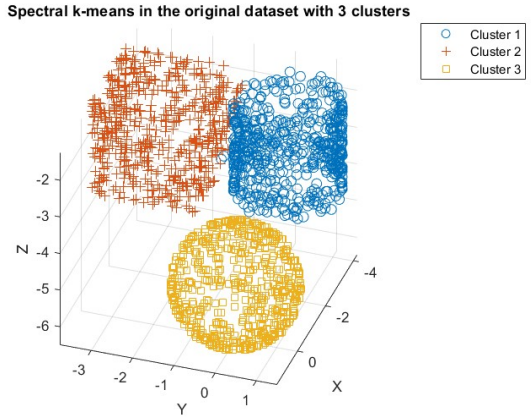


(b) Choice of M with L_sym, k = 40

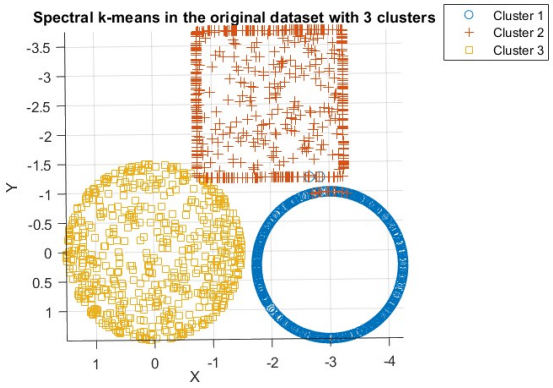
The normalised symmetric Laplacian matrix normalises the entries of L over the "degrees" of the vertices, thus balancing the contributions of higher and lower degree vertices. This can be seen in particular in the difference between the second and third

smallest eigenvalue, which becomes much less significant in the normalised case, making it easier to choose $M = 3$. If we were to choose by only looking at L, we would probably choose $M = 2$. For this reason we have decided to use the normalised matrix.

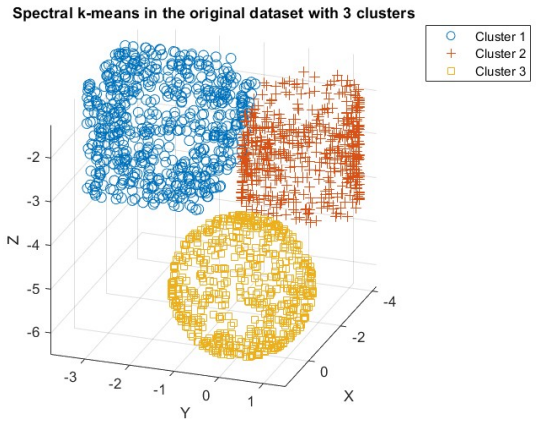
5.2.5 Exercise 6, 7, 8



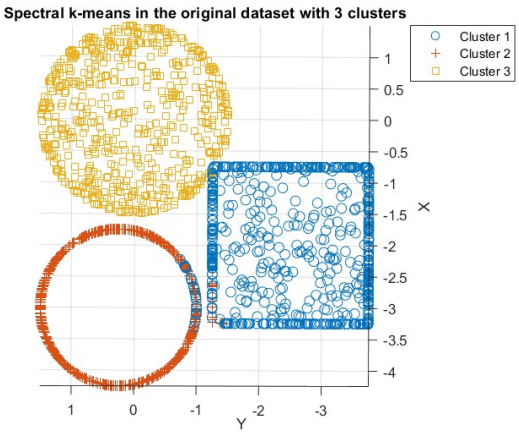
(a) Spectral k-means, k = 10 neighbors from the front



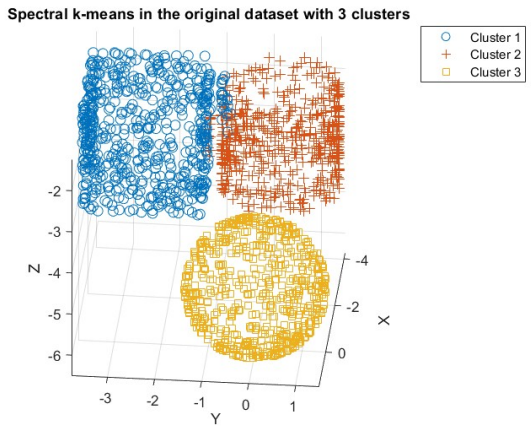
(b) Spectral k-means, k = 10 neighbors from above



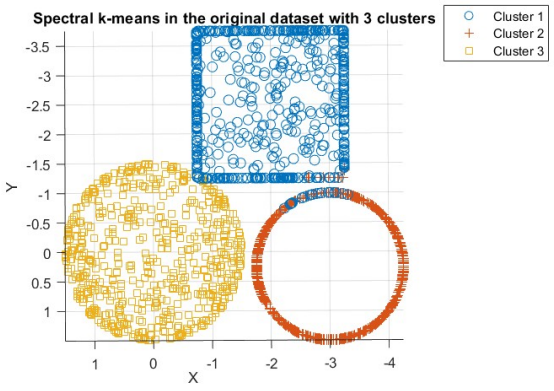
(a) Spectral k-means, k = 20 neighbors from the front



(b) Spectral k-means, k = 20 neighbors from above



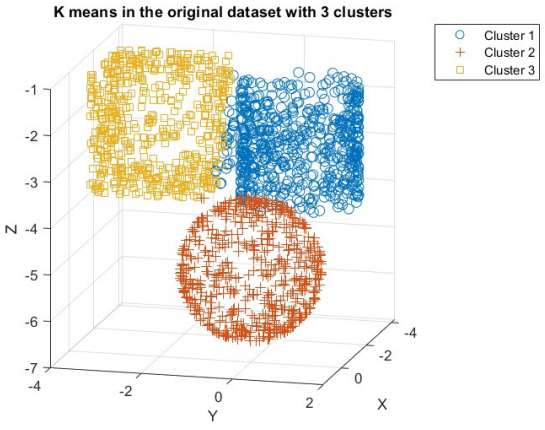
(a) Spectral k-means, k = 20 neighbors from the front



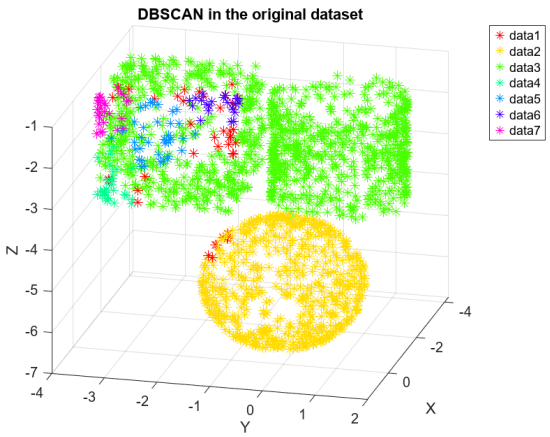
(b) Spectral k-means, k = 20 neighbors from above

From the pictures we can see that spectral k-means works quite well, but has some difficulties in dividing the points in the border between the cylinder and the cube, especially as k increases.

5.2.6 Exercise 9



(a) k - means



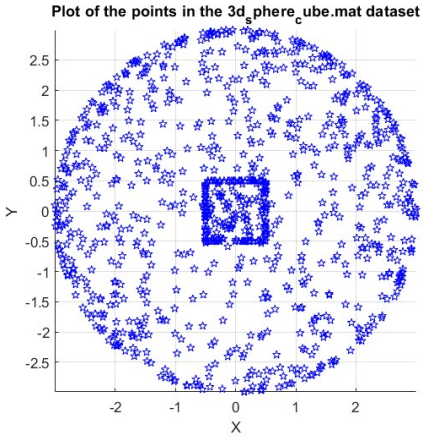
(b) DBSCAN

In this case, there is not much difference between k-means and spectral k-means, although the former has more difficulty in clustering the points on the edge of the cylinder close to that of the cube. The DBSCAN algorithm is prone to identifying an excess of clusters, thereby failing to differentiate between objects.

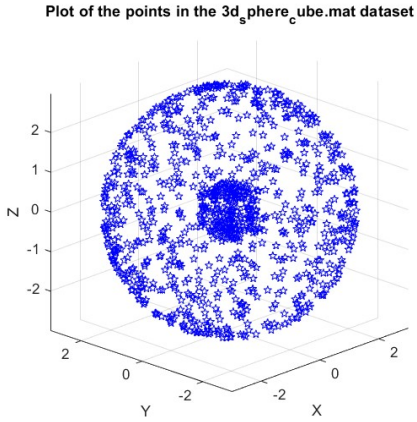
6 Second optional dataset: 3d_sphere_cube.mat

This dataset consists of 1200 rows (points) in 3 dimensions (3 columns). We have also created this dataset using the `datasets_generation.m` file. It consists of 2 objects: a small cube inside a sphere. The points are placed only on the surfaces of the objects.

6.1 Upload the dataset and plot the points



(a) Plot of the points in 3d_sphere_cube.mat from above



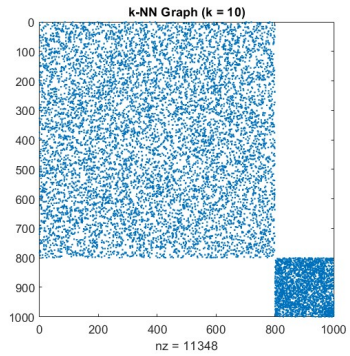
(b) Plot of the points in 3d_sphere_cube.mat from the front

6.2 Results for k = 10, 20, 40

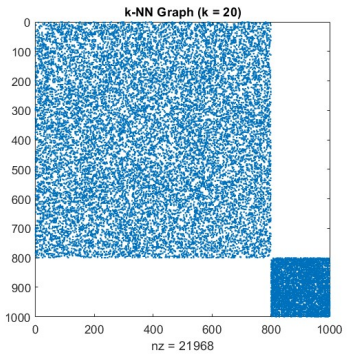
As in previous cases, the following analysis will jointly discuss the results obtained for the different configurations, given their substantial similarity.

6.2.1 Exercise 1

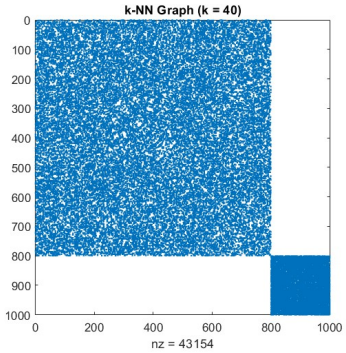
The visualizations consistently reveal two distinct connected components within the graph, corresponding to the two spatially separated objects present in the dataset. This separation is evident across all values of k, indicating that the k-NN graph effectively captures the underlying structure of the data, maintaining the separation between the objects despite variations in the neighborhood size. The consistent presence of two components across different k values suggests a robust separation between the two clusters.



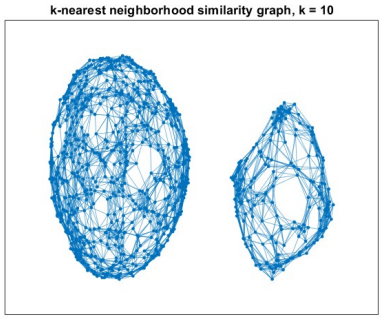
(a) Similarity matrix, k = 10



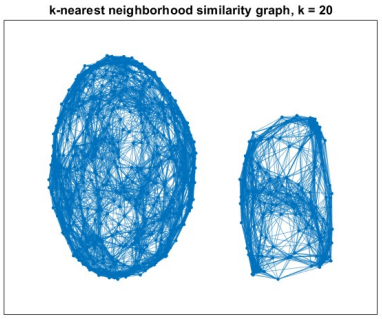
(b) Similarity matrix, k = 20



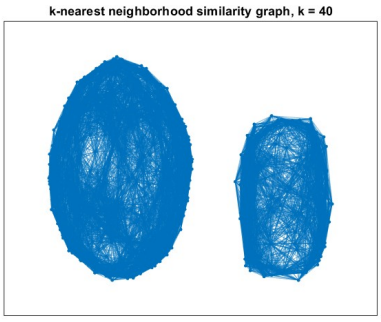
(c) Similarity matrix, k = 40



(a) Similarity graph, k = 10

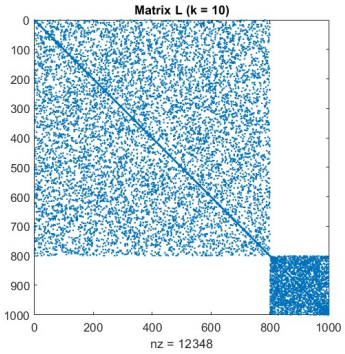


(b) Similarity graph, k = 20

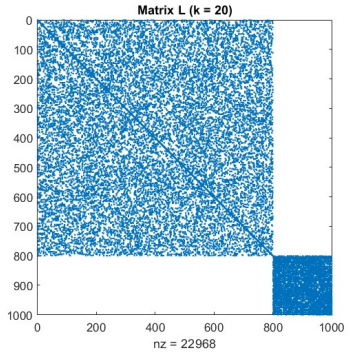


(c) Similarity graph, k = 40

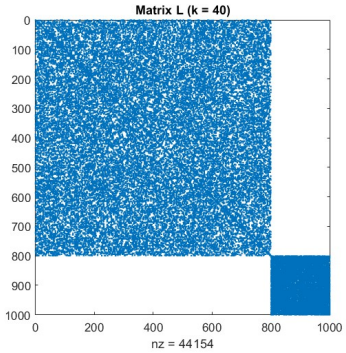
6.2.2 Exercise 2



(a) Laplacian matrix, k = 10



(b) Laplacian matrix, k = 20



(c) Laplacian matrix, k = 40

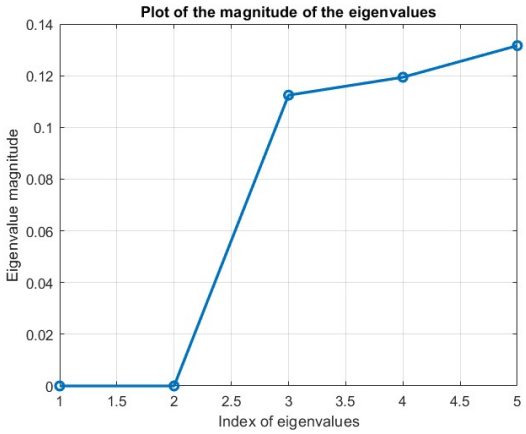
We tried computing both $\mathbf{L}$ and the normalized $\mathbf{L}_{\text{sym}}$, ultimately deciding to keep the latter.

6.2.3 Exercise 3

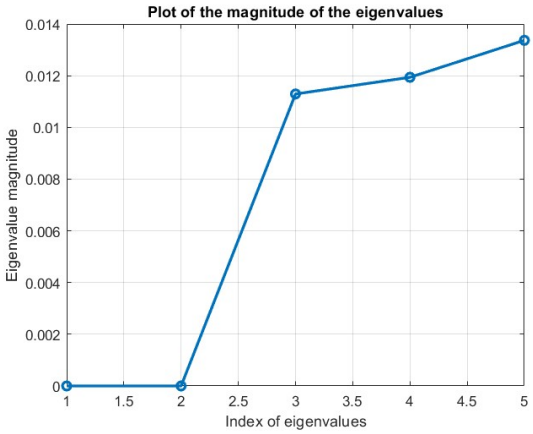
The result is: "`Number of connected components:  2`".

6.2.4 Exercise 4, 5

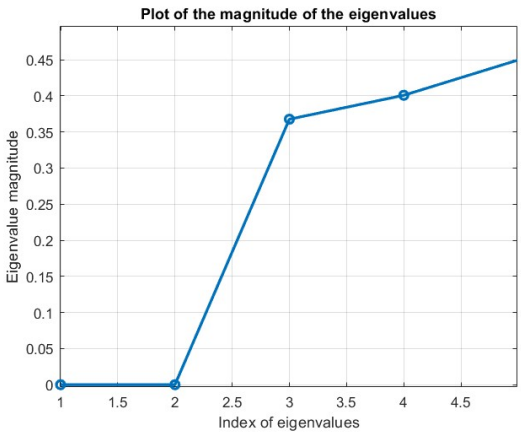
We now compare the results obtained with **L** and **L_sym**.



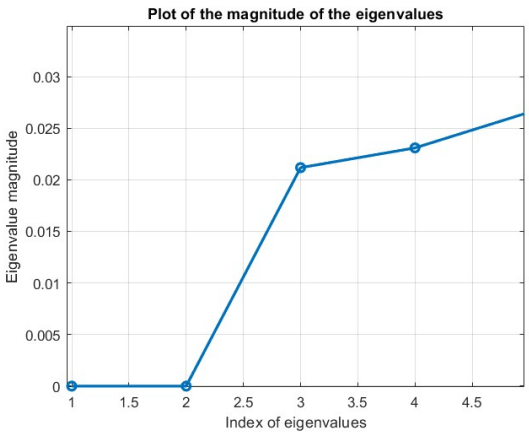
(a) Choice of M with L, k = 10



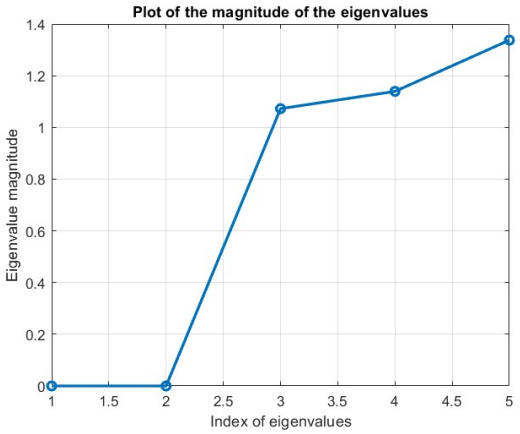
(b) Choice of M with L_sym, k = 10



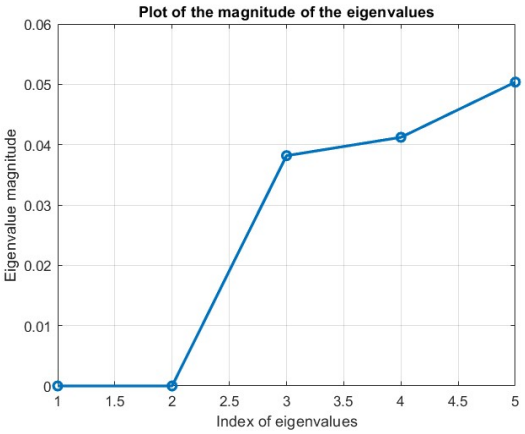
(a) Choice of M with L, k = 20



(b) Choice of M with L_sym, k = 20



(a) Choice of M with L, k = 40

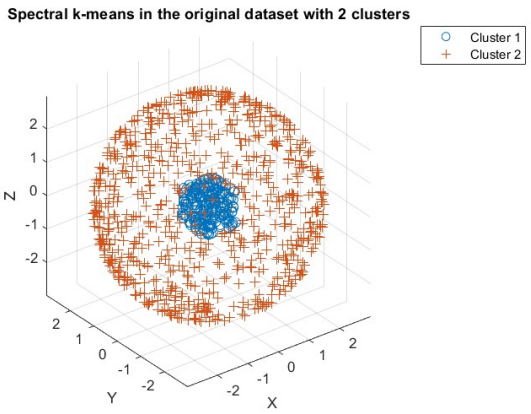


(b) Choice of M with L_sym, k = 40

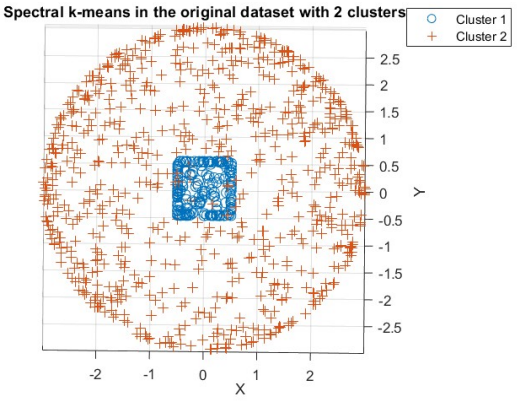
Similar considerations can be made as before, but here the difference between the second and third smallest eigenvalues is not negligible, so we will keep $M = 2$.

6.2.5 Exercise 6, 7, 8

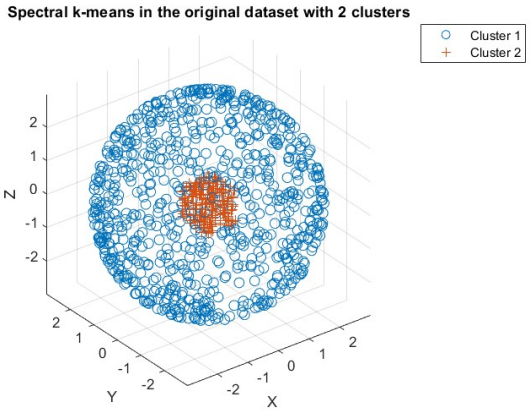
The results obtained with k-means spectral clustering demonstrate a clear separation of the cube and sphere into two distinct clusters, in line with their actual spatial separation in the dataset. This correct partitioning confirms the effectiveness of the method in capturing the intrinsic structure of the data, identifying the two objects as separate, homogeneous groups.



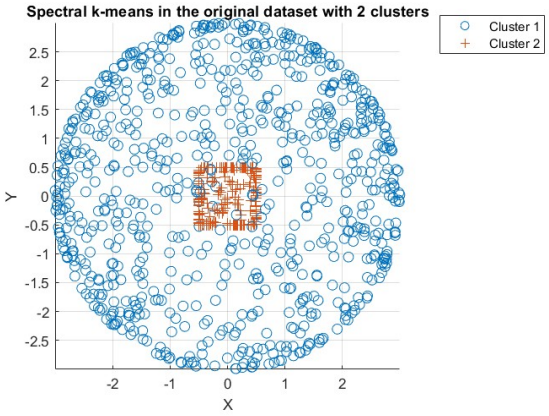
(a) Spectral k-means, k = 10 neighbors from the front



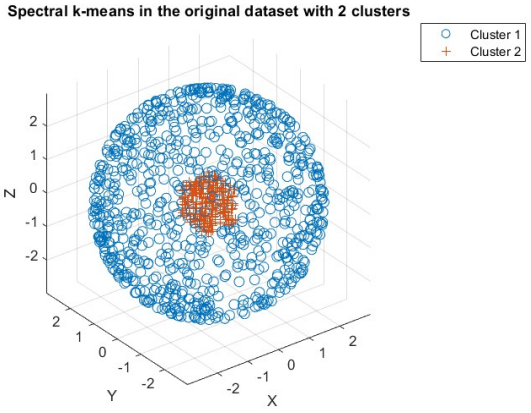
(b) Spectral k-means, k = 10 neighbors from above



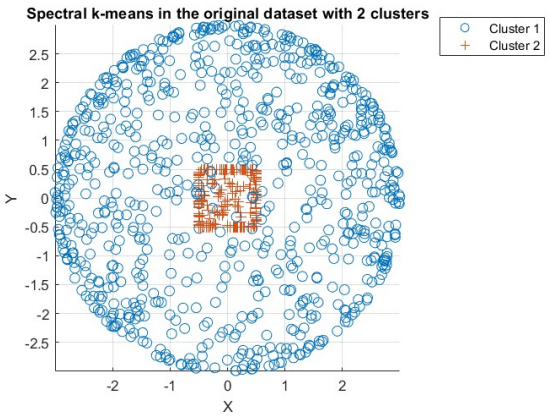
(a) Spectral k-means, k = 20 neighbors from the front



(b) Spectral k-means, k = 20 neighbors from above



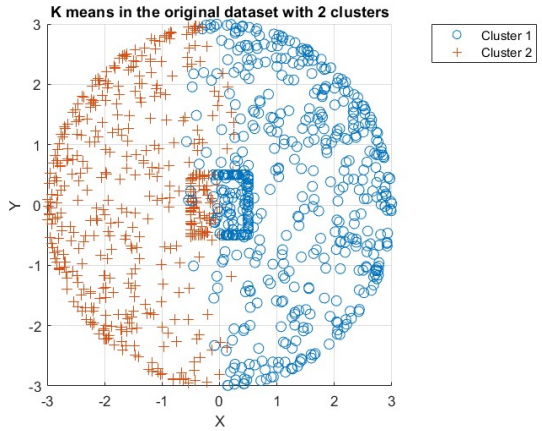
(a) Spectral k-means, k = 40 neighbors from the front



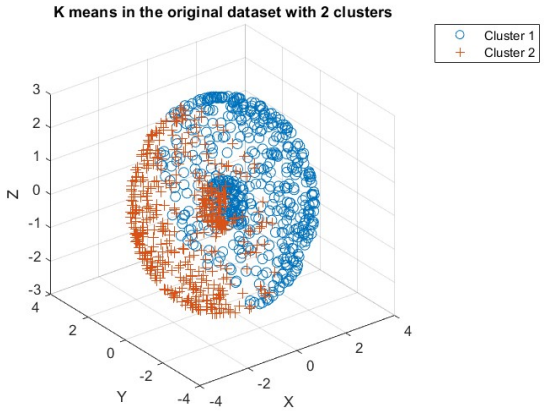
(b) Spectral k-means, k = 40 neighbors from above

6.2.6 Exercise 9

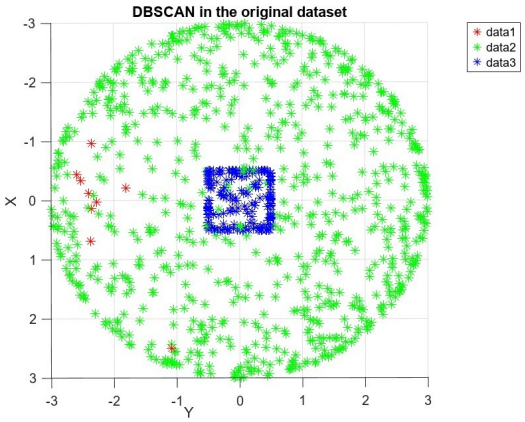
As can be seen from the figures, the k-means algorithm again manifests its limitations in distinguishing between the cube and the sphere, failing to separate the two structures correctly. The DBSCAN algorithm, on the contrary, provides more satisfactory results, largely identifying the two objects as distinct clusters, with the exception of a limited number of points classified as outliers. This performance of DBSCAN, although better than that of k-means, still shows a certain sensitivity to the distribution of the data.



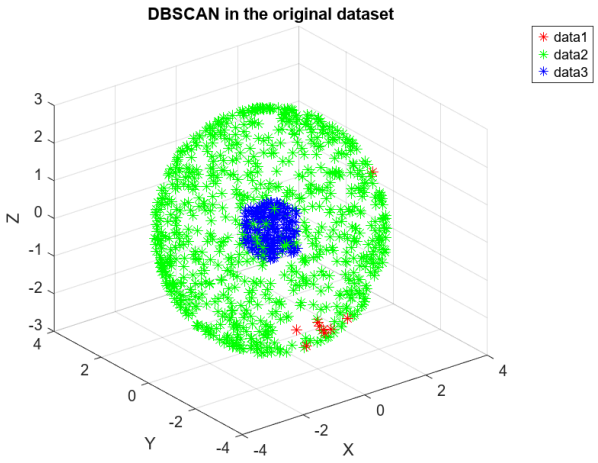
(a) k - means from above



(b) k - means from the front



(a) DBSCAN from above



(b) DBSCAN from the front

7 Final Considerations and Conclusions

This work has demonstrated the effectiveness of spectral clustering as a superior alternative to traditional clustering algorithms, particularly in scenarios characterised by non-convex clusters, poorly separated in the original space and with interconnections between points that do not emerge from a simple Euclidean distance analysis. The application of the k-means algorithm in the reduced dimensionality space defined by the M-selected eigenvectors has consistently produced more accurate cluster separation than the direct application of k-means on the original data matrix. This underscores the efficacy of spectral transformation in facilitating the identification of complex patterns by projecting data

into a space where similarity relationships are more accurately represented. Concurrently, the analysis revealed the inherent limitations of the k-means algorithm, particularly its challenges in handling clusters with non-globular shapes. Additionally, the performance of DBSCAN was found to be adversely affected by the presence of clusters with varying densities and by increasing the dimensionality of the data. Specifically, k-means is prone to dividing the space into convex regions, resulting in the failure to capture the complex geometries present in datasets such as spirals. Conversely, DBSCAN, while effective in identifying clusters of arbitrary shape, encounters challenges in distinguishing clusters with significantly different densities or in high-dimensional spaces, where neighbourhood definition based on Euclidean distance may become less informative. Spectral clustering, by contrast, has been shown to be a valid solution for complex clustering problems by analysing the structure of the similarity graph.

References

[1] Berrone, S. *Spectral Clustering*. Notes from *Computational Linear Algebra for Large Scale Problems*, Politecnico di Torino.

[2] Della Santa, F. *Principal Component Analysis*. Presentation slides for *Computational Linear Algebra for Large Scale Problems*, Politecnico di Torino.

[3] Baralis, E., & Cerquitelli, T. (2023). *Clustering fundamentals*. Presentation slides for Data Science Lab: process and methods, Politecnico di Torino.