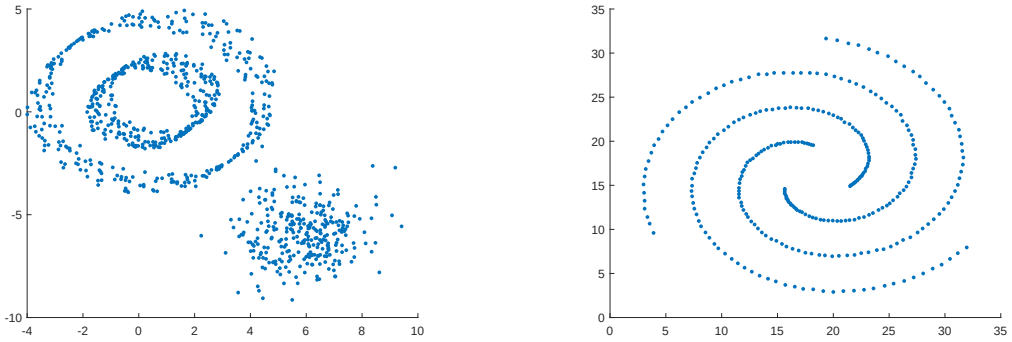
Computational Linear Algegbra For Large Scale Problems

Nenna Giulio, Ornella Elena Grassi

Spectral Clustering Homework

The aim of this homework is to implement and apply **Spectral Clustering** to two different sets of datapoints in $\mathbb{R}^2$. The two sets are shown in Figure 1



(a) Scatterplot of the data stored in `Circle.mat`

(b) Scatterplot of the data stored in `Spiral.mat`

Figure 1: Scatterplot of the two Datasets

As it is clearly visible through visual inspection, both datasets contain 3 different shapes that can be classified as different clusters. In the `Circle` dataset there are two concentrical circles and a cloud of points in the bottom right while in the `Spiral` dataset there are 3 spirals. Traditional clustering algorithms, that mainly rely on euclidean distance, may fail in recognizing the presence of shapes in our data hence our need to rely on a different technique called **Spectral clustering**.

1 K-Nearest Neighborhood Graph

First, we need to define a similarity function that measures "how much our points are similar to each other". Let X_i and X_j be two points in our data, then we will use a similarity measure defined as:

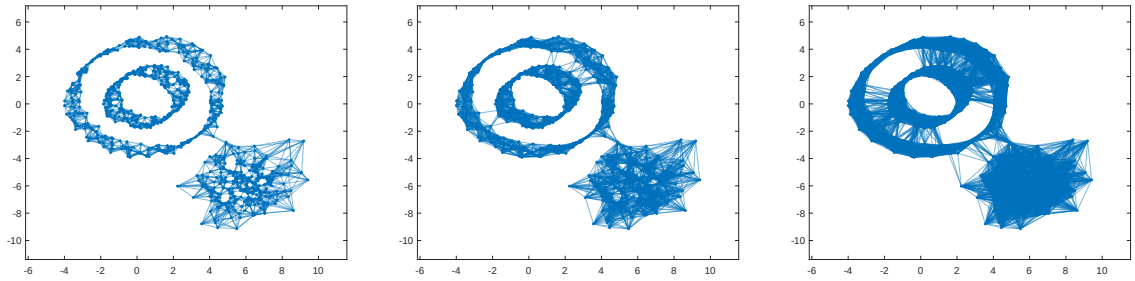
$$s_{i,j} = \exp\left(-\frac{\|X_i - X_j\|^2}{2\sigma^2}\right) \tag{1.1}$$

Then, a *K-Nearest Neighborhood* similarity graph is a Graph $G = (V, E)$ where each vertex $v_1, \dots v_n$ represents a point and two vertices v_i and v_j are connected by an undirected edge $e_{i,j}$ if the similarity between v_i and v_j is among the K-th highest similarities between v_i and other vertices in V. For such graph we can define the relative adjacency matrix as $W_{i,j} = s_{i,j}$ where each entry $W_{i,j}$ is nonzero only if there exists an edge between v_i and v_j. W has zero-values on the diagonal by definition.

The following MATLAB code was use to generate the K-NN similarity graph of our data:

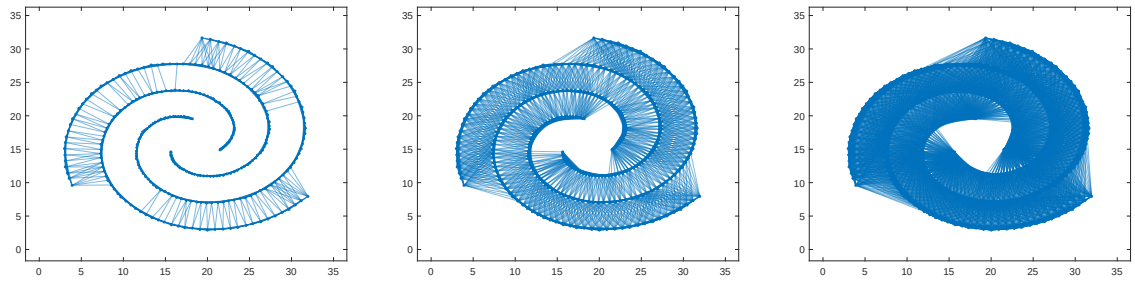
```
function [G, W] = knn_graph(X, K, sigma)
%An adjacency matrix of a K-nn graph has, for each column and row, only K
%nonzero elements that correspond to the nearest neighbors of each point.

N = length(X);
X = X(:,1:2);

W = sparse(N, N);
for i = 1:N %for each point
    %Compute the similarity for every other point
    sim = exp(-((X(i, 1)- X(:, 1)).^2 + (X(i, 2)- X(:, 2)).^2)/(2*sigma^2));
    sim(i) = 0; %set similarity with itself to 0
    [sim, idx] = maxk(sim, K); %Compute the K most similar points and its indices
    W(i, idx) = sim; %set values of the adjacency matrix
    W(idx, i) = sim;
end
G = graph(W);
```

Running `knn_graph` on both dataset using $\sigma = 1$ and testing with $K = 10, 20, 40$ gave the following results:



(a) $K = 10$ (b) $K = 20$ (c) $K = 40$

Figure 2: K-NN graphs plots for `Circle` data with different values of K



(a) $K = 10$ (b) $K = 20$ (c) $K = 40$

Figure 3: K-NN graphs plots for `Spiral` data with different values of K

Our objective is to find a K-NN graph that "separates well" the shapes, meaning that in the best case each shape is a connected component of the graph. As we can see, for both `Circle` and `Spiral` data, the value $K = 10$ seems to separate the shapes well. We will quantify the concept of *good cluster separation* in the next section.

2 Degree and Laplacian Matrices

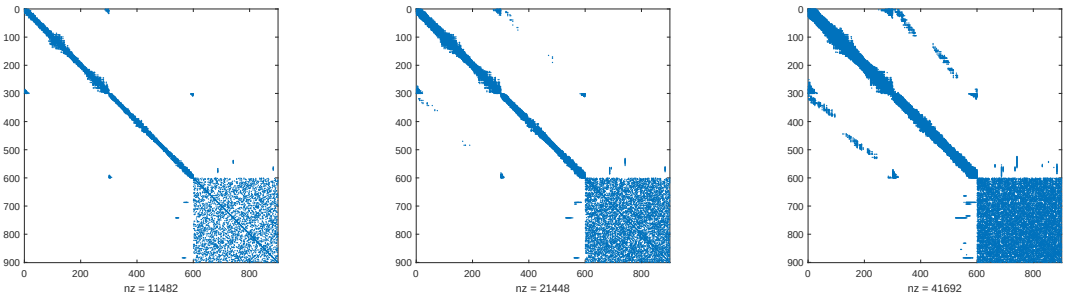
The *Degree matrix* D is a diagonal matrix that has the degree of each node on the diagonal where the degree of a node v_i is the sum of the weights of all the edges that are connected to such node. Since in our case edge weights correspond to similarities, the degree of each node is the sum of the similarities with all of its neighbors. This can be simply achieved with the following:

$$D = \text{diag}(W\mathbb{1}) \tag{2.1}$$

Since the Laplacian matrix is defined as $L = D - W$, we use the following code to compute both D and L:

```
function [L, D] = graph_laplacian(W)
    D = sparse(diag(sum(W)));
    L = D-W;
end
```

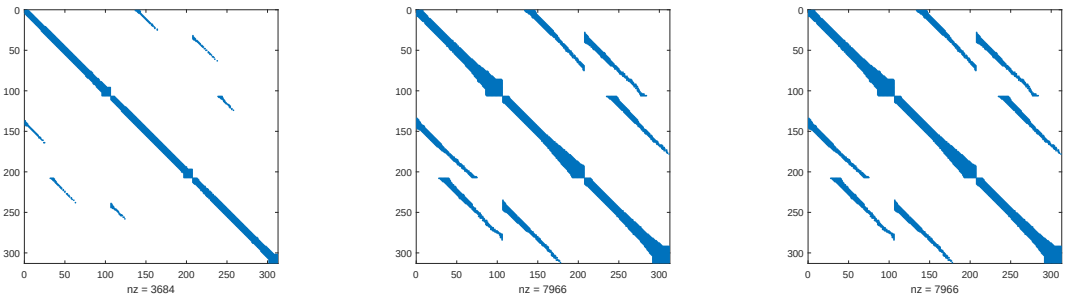
After computing both matrices for each dataset and for each value of K we are testing for the K-NN graph we can inspect each Laplacian matrix using the `spy` command in MATLAB:



(a) $K = 10$ (b) $K = 20$ (c) $K = 40$

Figure 4: `spy` plots of the Laplacian matrix for `Circle` data with different values of K

In Figure 4 we can see that, for each Laplacian matrix inspected, there are 3 sections. Each section defines a "*shape cluster*" in the `Circle` dataset: the first one is clearly visible in the bottom right corner of each matrix and it is relative to the point cloud. The other two sections are along the diagonal in the top left of the matrix, they are somewhat visible for $K = 10$ and $K = 20$ and they are relative to the two "circles" visible in the scatterplot in Figure 1. The Laplacian matrix inspected for $K = 40$ clearly shows a sort of **data pollution** from one shape cluster to another, this might suggest that the value $K = 40$ may be too high for deploying the clustering.



(a) $K = 10$ (b) $K = 20$ (c) $K = 40$

Figure 5: `spy` plots of the Laplacian matrix for `Spiral` data with different values of K

The same concept can be applied for Figure 5. Along the diagonal, three sections are visible and they represent the three different spirals seen in Figure 1. As for `Circle` data, the KNN graph

computed on the `Spiral` data with $K = 40$ shows pollution between each spiral and this is visibile since there are a lot of non-zero elements away from the diagonal.

The recurrent theme for both dataset is that the more K is higher, the more each shape will "pollute" another. This means that the corresponding Laplacian matrix will show a **lot of non-zero elements away from the diagonal** and this is a phenomenon that can (and will) impact computational costs.

3 Connected components computation

In order to compute the number of connected components in each graph we use the following result:

Theorem 3.1. *Let $G = (\mathcal{V}, W)$ be a finite graph and let L be its laplacian matrix. Then L has $\lambda = 0$ as an eigenvalue and its algebraic multiplicity correspond to the number of connected components in G.*

Using the Matlab function `eigs` we can compute the smallest $C = 6$ eigenvalues of both graphs laplacian matrices L and for each value of K obtaining the following results:

$K = 10$	$K = 20$	$K = 40$
8.5482e-17	2.4854e-16	8.1617e-16
3.7050e-16	0.0111	0.0482
0.0048	0.0652	0.7028
0.0286	0.1620	0.7797
0.0425	0.1724	0.9065
0.0429	0.3220	1.376

Table 1: Smallest 6 eigenvalues of the Laplacian matrix for the `circle` dataset

$K = 10$	$K = 20$	$K = 40$
1.0496e-16	1.7853e-16	4.0554e-16
1.9667e-04	0.0018	0.0023
2.7219e-04	0.0020	0.0025
0.0041	0.0048	0.0049
0.0044	0.0054	0.0062
0.0046	0.0056	0.0067

Table 2: Smallest 6 eigenvalues of the Laplacian matrix for the `spiral` dataset

From a visual inspection of both dataset we would expect to find 3 connected components and consequently an eigenvalue $\lambda = 0$ of both laplacian matrices with algebraic multiplicity $M = 3$.

Considering the case $K = 10$ for both datasets we can see that the first 3 eigenvalues are "almost" equal to 0, and we can see a jump in order of magnitude from the fourth eigenvalue. On the other hand the cases $K = 20$ and $K = 40$ show that only the first eigenvalue could be considered equal to 0 (which is always true) but the second and third eigenvalues are somewhat smaller than the others.
This behaviour is perfectly in line with the "data pollution" concept seen graphically in 2. We can in fact interpret an "almost null" eigenvalue as an eigenvalue corresponding to an "almost connected component". The less the component is "isolated", the greater its eigenvalue will be. In the case $K = 10$ we clearly have 3 connected components that are isolated fairly well, hence the value of the first 3 eigenvalues are almost 0 for both graphs. In the cases $K = 20$ and $K = 40$, the 3 connected

components that we expected are not well isolated and present a lot of edges from one to another, hence the first 3 eigenvalues are not small enough to be considered null.

In any case we consider $M = 3$ the number of connected components and construct the matrix $U \in \mathbb{R}^{N\times 3}$ using the $M = 3$ eigenvectors u_1, u_2, u_3 corresponding to the first 3 eigenvalues as columns.

The following Matlab code was used for the computation in this section:

```
%% --- TASK 3-4-5 ---
n_eigs=6;
U_circle={};
U_spiral={};
eigs_circle={};
eigs_spiral={};

for i=1:length(K) %for each value of K tested
    [U_circle{i}, eigs_circle{i}] = eigs(L_circle{i}, n_eigs, 'smallestabs'); %compute the n_eigs
     smallest eigenvalues for circle
    [U_spiral{i}, eigs_spiral{i}] = eigs(L_spiral{i}, n_eigs, 'smallestabs'); %same for spiral
end

for i = 1:length(K)
    %simple code to generate a matrix where in each column the eigenvalues
    %are listed for each value of K tested
    eigs_circle_tot(:,i) = diag(eigs_circle{i});
    eigs_spiral_tot(:,i) = diag(eigs_spiral{i});
end
M=3; %using the first 3 eigenvectors
for i = 1:length(K)
    %trim the matrix U to the first M columns
    U_circle{i} = U_circle{i}(:, 1:M);
    U_spiral{i} = U_spiral{i}(:, 1:M);
end
```

4 Performing Spectral Clustering

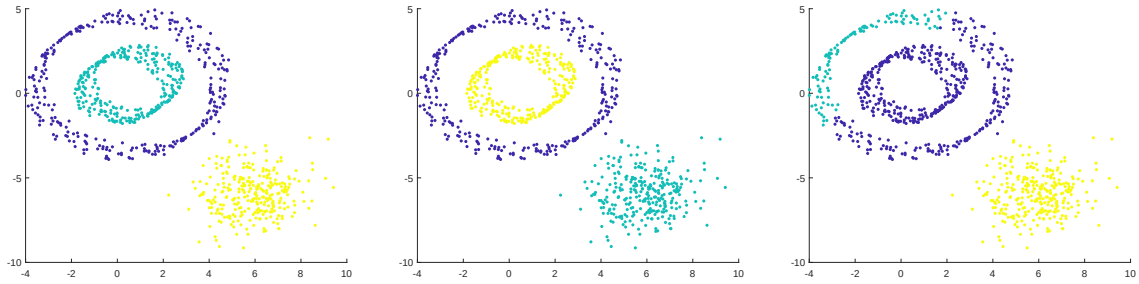
After computing the matrix $U \in \mathbb{R}^{N\times 3}$ for both the `circle` and the `spiral` datasets and for each $K \in \{10, 20, 40\}$ we can now finally deploy a clustering algorithm.

Let $y_i \in \mathbb{R}^M$ be the vector corresponding to the i-th row of the matrix U. Then we can use the *K-Means* algorithm to cluster the points y_i, $i \in \{1, \dots, N\}$ into M clusters and assign the original datapoints to the same cluster as their corresponding rows in U.

The following code was used in order to perform clustering:

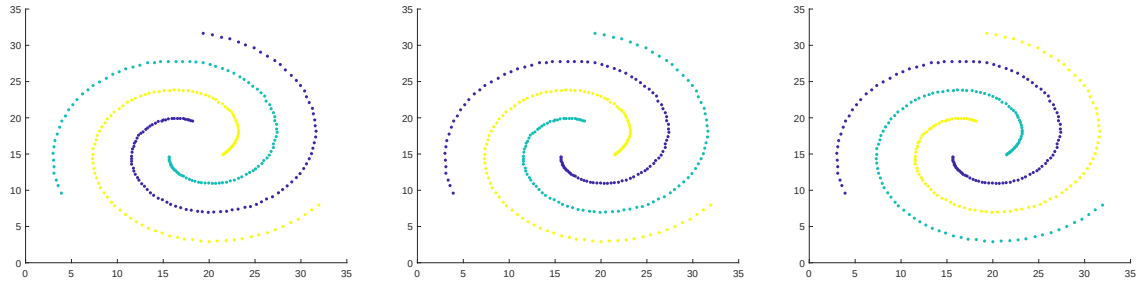
```
%% --- TASK 6-7-8---
M=3;
idx_circle={}; %cluster labels for circle dataset
idx_spiral={}; %same for spiral


for i = 1:length(K)
    idx_circle{i} = kmeans(U_circle{i}, M);
    idx_spiral{i} = kmeans(U_spiral{i}, M);
end
```

Hence for both dataset we obtain 3 different clusterings, one for each value of K tested. Plotting the results we obtain:



(a) $K = 10$ (b) $K = 20$ (c) $K = 40$

Figure 6: Spectral clustering for `circle` data with different values of K



(a) $K = 10$ (b) $K = 20$ (c) $K = 40$

Figure 7: Spectral clustering for `spiral` data with different values of K

As it is clearly visible in Figure 6, performing spectral clustering on the `circle` data yields good results both in the case with $K = 10$ and $K = 20$. The same couldn't be said about the case $K = 40$ where the effect of pollution from one shape to another leads to poor results in terms of the proficiency of the algorithm to distinguish shapes.

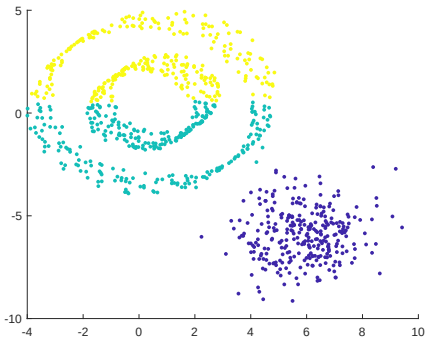
Although not every value of K yields good results for the `circle` data, the same isn't true for the `Spiral` data, where the presence of edges between spirals in the adjacency graph seems to have no effects on the fitness of the clustering, even with $K = 40$.

5 Comparison with other clustering algorithms

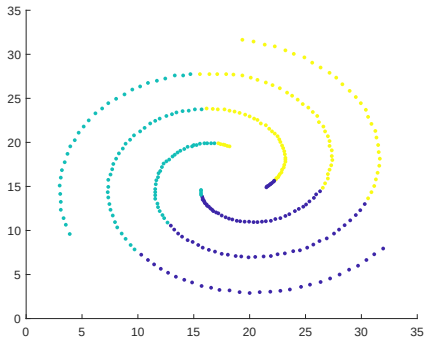
In this section we will test other clustering algorithm on the data, visualizing how they perform in comparison with the spectral clustering performed above.

5.1 Plain *Kmeans* Algorithm

Performing the plain *K-means* algorithm on our data yields poor results since the algorithm is based on euclidean distance. This is the reason it fails to recognize shapes other than balls.



(a) *K-means* algorithm performed on `circle`



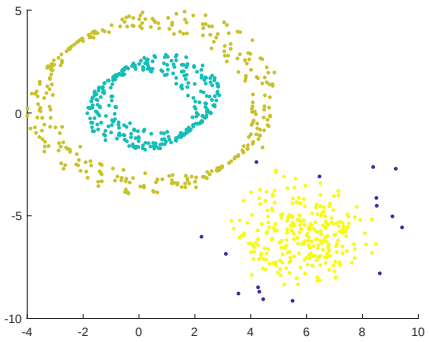
(b) *K-means* algorithm performed on `spiral`

Figure 8: *K-means* algorithm performed onto the datasets

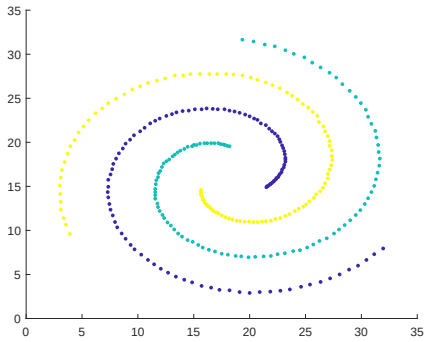
As it is clearly visible in figure 8, the *K-means* algorithm performed on spatial data is only able to recognize the cloud of points in the `circle` data since is euclidean-ball shaped. It clearly fails to recognize any other shape and this results in a very poor clustering with regard to shaped data.

5.2 DBSCAN algorithm

The DBSCAN algorithm is an "explorative" approach to clustering. Without going into details, at its core the DBSCAN algorithm populate each cluster by searching the neighborhood of each point in the cluster hence performs much better at recognizing shapes than the plain *K-means* algorithm.



(a) DBSCAN algorithm performed on `circle`



(b) DBSCAN algorithm performed on `spiral`

Figure 9: DBSCAN algorithm performed onto the datasets

The results of the clustering are shown in Figure 9. Those are obtained using the `dbscan` function in MATLAB performing some minor tuning of the parameters. In particular the clustering for the `circle` data was obtained using the default metric of the algorithm which is the euclidean distance while for the `spiral` data the Mahalanobis distance was used.

As it is visible in Figure 9 the DBSCAN algorithm is perfectly capable of recognizing shapes and correctly classifies all points of the two datasets with the only exception of some points in the `circle` dataset.