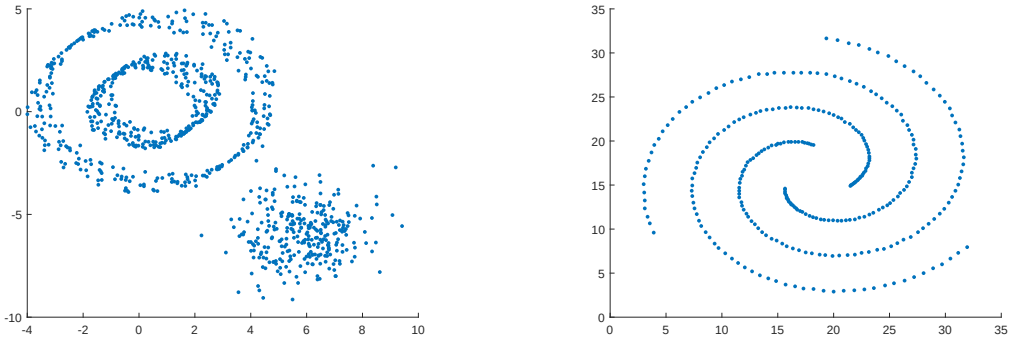
Computational Linear Algegbra For Large Scale Problems

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:

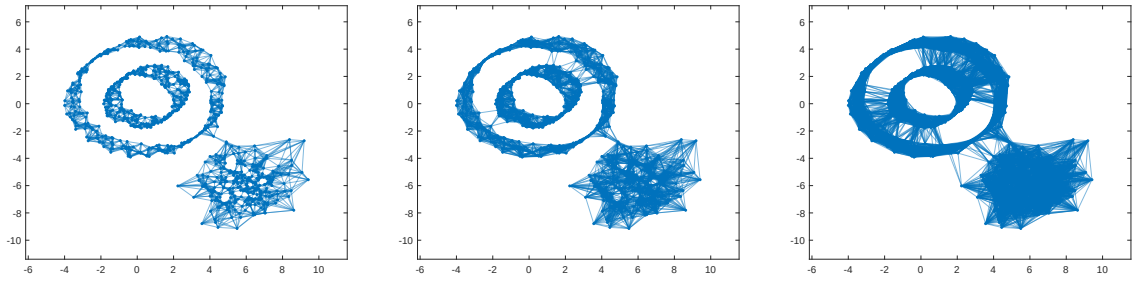
$$s_{i,j} = \exp\left(-\frac{\|X_i - X_j\|^2}{2\sigma^2}\right) \tag{1.1}$$

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:

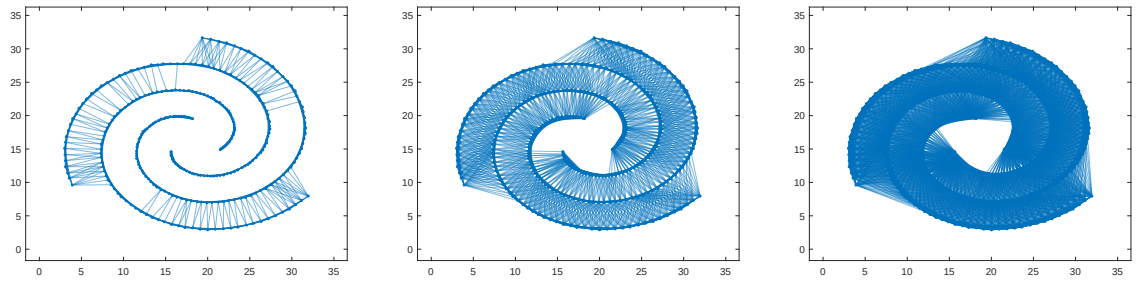
```
function [G, W] = knn_graph(X, K, sigma)
%An adjacency matrix of a K-nn graph has, for each column and row, only K
%nonzero elements that correspond to the nearest neighbors of each point.

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

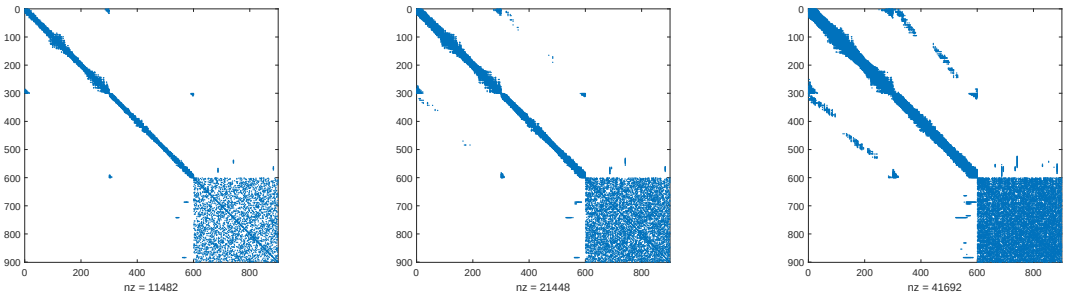
The *Degree matrix* D is a diagonal matrix that has the degree of each node on the diagonal where the degree of a node v_i is the sum of the weights of all the edges that are connected to such node. Since in our case edge weights correspond to similarities, the degree of each node is the sum of the similarities 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
```

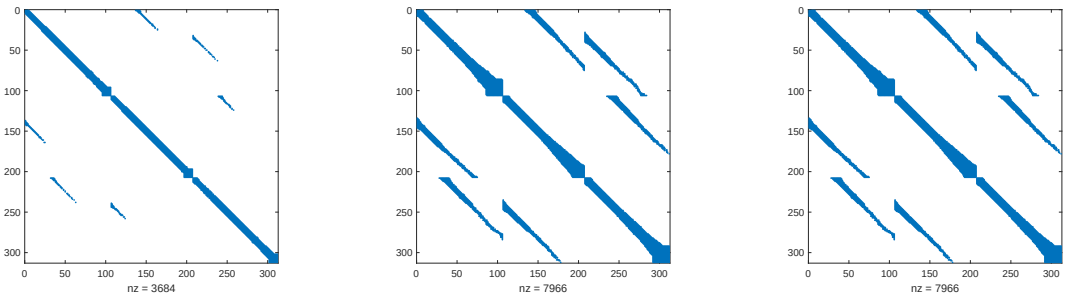
After computing both matrices for each dataset and for each value of K we are testing for the K-NN graph we can inspect each Laplacian matrix using the `spy` command in MATLAB:



(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 $C = 3$ eigenvalues of both graphs obtaining the following results: