Università della Svizzera italiana | Faculty of Informatics | Institute of Computing CI

Numerical Computing **2022**

Student: Albert Cerfeda

Solution for Project 4

Due date: Wednesday, 23 November 2022, 23:59 AM

Numerical Computing 2022 — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to iCorsi (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, Julia). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

Contents

1. Spectral clustering of non-convex sets [50 points]

1.1. Plotting non-convex sets

Let us plot the 'Two spirals' non-convex[1] graph:

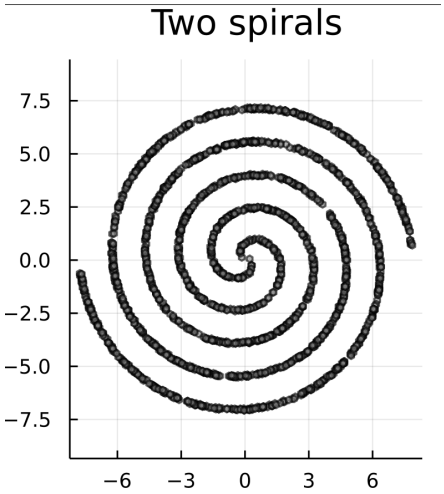

Figure 1: `pts_spiral` graph.

By briefly looking at the graph we can easily identify the two main clusters: the two interleaved spirals that meet in the center of the cartesian plane at coordinates $(0, 0)$.
For us it is intuitive to cluster points that have spacial continuity to each other like in this case, but each clustering algorithm might consider different factors for determining what makes two points 'similar' and therefore belonging to the same cluster.

The k-means algorithm on the eigenvectors of the graph's Laplacian matrix (*Spectral method*) makes the cluster centroids adjust based on the Eigenvector coordinates. As a result the Spectral method values the relative distance between the points, and the two resulting clusters are the ones we are able to identify intuitively. Calculating eigenvectors and eigenvalues of large graphs can get computationally expensive. Fortunately there are efficient methods for calculating the first K eigenvectors of sparse matrices.

On the other hand, running the standard k-means algorithm yields two clusters where we can roughly guess the position of each cluster's centroid is, as they adjust based on the "raw" input graph points. We notice how standard k-means method struggles correctly clustering non-convex[1] sets.

[1] Convex sets: In Euclidean space, an object is convex if, given any two points inside the object, the line between them is also within the object. [Source: stanford.edu]

1.2. Computing the ϵ factor for $K = 2$ and $K = 4$ clusters

As we have a set of points in space, we need to define a function to define and compute the similarity of any two points in the set, fundamental for our clustering algorithms.
We define the *Gaussian kernel similarity function* as follows: Opposed to the *Euclidean distance*

$$s\left(x_i, x_j\right) = e^{\frac{-\left\|x_i - x_j\right\|^2}{2\sigma^2}}$$

Figure 2: The *Gaussian Kernel* similarity function

function that measures the *distance* between two points, the Gaussian kernel similarity function ranges between 0 and 1 as it expresses absolute *similarity* and decreases with distance.

By using the *Gaussian Similarity function* we define the weighted adjacency matrix S that expresses the gaussian similarity between the points in our set.
The choice of the sigma σ parameter is important. It controls the size of the neighborhood therefore it allows us to have a more densely connected graph or the opposite. W chose it to be $\mathbf{2} * \log \mathbf{n}$.

Afterwards we compute the minimal spanning tree[2] of matrix S, useful to simplify the graph by removing redundant edges that could hinder our cluster computation.

We can finally choose our ϵ factor for the ϵ-similarity graph.
In order for the resulting graph to be safely connected we choose it to be the longest edge in the previously-computed Minimal Spanning Tree.

```
# ...
mintree = minspantree(S)
e = maximum(mintree)
# ...
```

Figure 3: Julia code for computing the Minimal Spanning Tree and the ϵ factor.

Choosing ϵ to be the maximum value in the minimum spanning tree of the Gaussian Similarity matrix yields the following results:
Note: there is a random component as the datasets are generated at runtime, so results may vary.

Mesh	ϵ factor
`pts_spiral`	0.8729153954604377
`pts_clusterin`	0.8834278441571511
`pts_corn`	0.8623495359026014
`pts_halfk`	0.8735819886907046
`pts_moon`	0.8637545218347059
`pts_outlier`	0.7459541372062404

Table 1: ϵ factors for the various graphs

[2]Minimal Spanning Tree (MST): A subgraph that includes every vertex of the graph with the minimum number of edges, the minimal amount of summed edge weights and that does not contain loops.

1.3. Generating the ϵ similarity graphs

We first need to compute the ϵ-neighborhood graph $G \in \mathbb{R}^{n \times n}$ defined as follows:

$$g_{ij} = \begin{cases} 1, & \text{if } \underbrace{\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}}_{\text{Euclidean Distance}} < \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

We can see how our ϵ factor is used for connecting only vertices whose distance is less than our ϵ factor.

```
function epsilongraph(epsilon, pts)
    n = size(pts, 1);
    G = zeros(n, n);
    for i = 1:n
        for j = 1:n
            if norm(pts[i,:]-pts[j,:]) < epsilon
                G[i,j] = 1
            end
        end
    end
    return sparse(G)
end
# ...
G_e = epsilongraph(e, pts)
```

Figure 4: Julia code defining the function for computing the ϵ-neighborhood graph

1.4. Computing and visualizing the adjacency matrix for the ϵ similarity graph

Once we multiply the resulting ϵ-neighborhood graph G with our Gaussian Similarity matrix S we obtain the weighted adjacency matrix for the ϵ-neighborhood graph. Like this we make sure to only include the edges that fall under the upperbound imposed by the threshold value ϵ.

```
# Create the adjacency matrix for the epsilon case
W_e = S .* G_e;
draw_graph(W_e, pts)
```

Let us plot the adjacency matrices for some ϵ-neighborhood graphs:

1.5. Implementing Spectral Method

The **graph Laplacian** is a symmetric positive semi-definite matrix that encodes various graph properties. It is computed through an adjacency matrix and it associated weight matrix. It can be easily computed using a provided function.

By running k-means on the eigenvectors corresponding to the kth smallest eigenvalues of the Laplacian matrix we are considering the graph through a new representation of its coordinates.

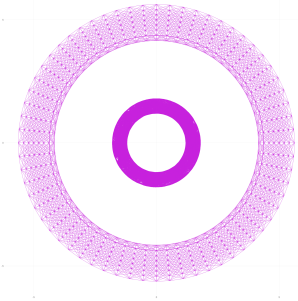

Figure 5: `pts_clusterin` ϵ adjacency graph.

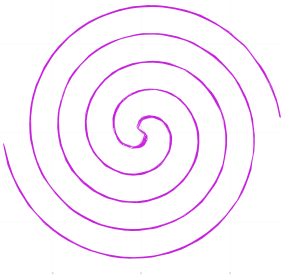

Figure 6: `pts_spiral` ϵ adjacency graph.

In this eigenvector representation the already existing similarity traits of points get accentuated, making the k-means' centroids find their ideal positions faster. We notice how the *Spectral method* is generally more effective in clustering non-convex sets.
An important observation is how sensitive spectral clustering is to the construction of the similarity graph and the choice of its parameters ϵ and σ.

```
# ...
K = 2
L, D = createlaplacian(W_e);

#   Spectral method
lambda = eigvals(L);
Y = eigvecs(L);
ind = sortperm(lambda);
Y = Y[:,ind[begin:K]]

#   Cluster rows of eigenvector matrix of L corresponding to K smallest eigenvalues.
R = kmeans(Y', K);
spec_assign = R.assignments;
# ...
```

Figure 7: Julia code implementing the Spectral method.

1.6. Performing k-means clustering on the input points

Implementing the basic k-means algorithm on the input points is rather trivial as we use the provided `kmeans` function:

```
# ...
R = kmeans(pts', K);
data_assign = R.assignments;
```

Figure 8: Julia code for performing k-means clustering on the input points

1.7. Clustered datasets with k-means and spectral clustering in $K = 2$ and $K = 4$

We run the *Spectral* and k-means clustering algorithms for $K = 2$ and $K = 4$ clusters.
Let us plot the resulting clusters for $K = 2$ [Figure 15] and $K = 4$ [Figure 20] respectively.
One major weakness off the k-means algorithm that can be observed is how points that are unusually far away from the rest of the graph (i.e *outliers*) influence the centroid (and therefore cluster) positioning, resulting in badly-clustered points. This effect is particularly noticeable in Figure 18.
We also notice how the basic k-means method treats all the centroids clusters as of the same size meaning that it does not take in account for clusters of heterogeneous volume. This is particularly noticeable in graphs where there are clusters that are very different in volume.
All the generated plots are aggregated in the next page.

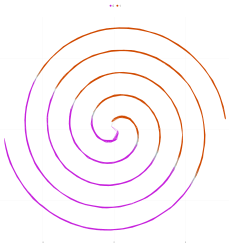

Figure 9: `pts_spiral` k-means clustering.

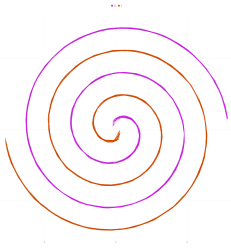

Figure 10: `pts_spiral` spectral clustering.

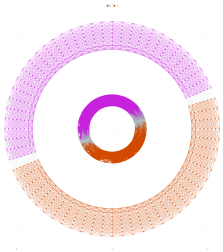

Figure 11: `pts_clusterin` k-means clustering.

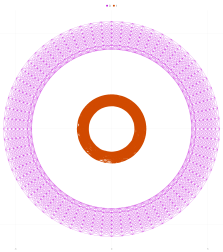

Figure 12: `pts_clusterin` spectral clustering.

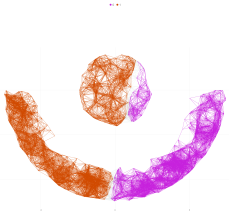

Figure 13: `pts_spiral` k-means clustering.

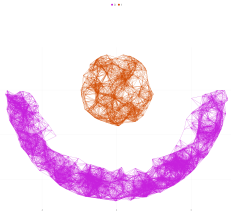

Figure 14: `pts_spiral` spectral clustering.

Figure 15: Graph clustering for $K = 2$

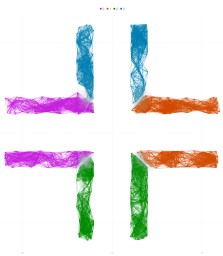

Figure 16: `pts_corn` k-means clustering.

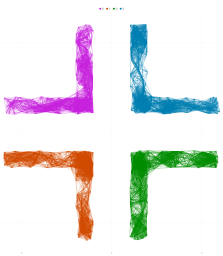

Figure 17: `pts_corn` spectral clustering.

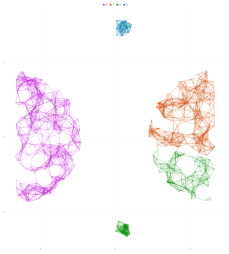

Figure 18: `pts_outlier` k-means clustering.

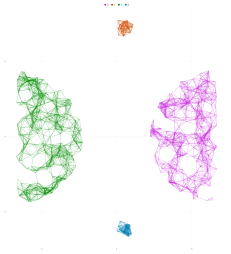

Figure 19: `pts_outlier` spectral clustering.

Figure 20: Graph clustering for $K = 4$

2. Spectral clustering of real-world graphs [35 points]

2.1. Computing the Laplacian Matrix and the eigenvector graph

We compute the graph Laplacian matrix L through the provided function and matrix Y consisting of the eigenvectors associated with the K smallest eigenvalues as usual.

We plot the graph with coordinates supplied by the 2nd and 3rd smallest eigenvalues, by definition:

$(x_i, y_i) = (v_2(i), v_3(i))$ where v_j is the eigenvector associated with the jth smallest eigenvalue

```
L, D = createlaplacian(W)

lambda = eigvals(L);
Y = eigvecs(L);
ind = sortperm(lambda);
Y = Y[:,ind[begin:K]]

vertices = Y[:,2:3]
draw_graph(W, vertices)
```

Figure 21: Julia code for computing the Laplacian matrix and plotting the eigenvector graph

Below are some obtained plots:

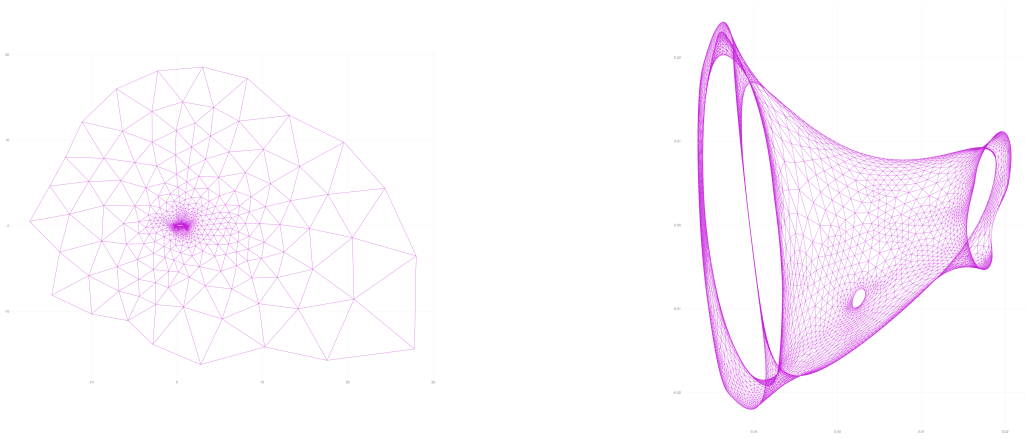

Figure 22: Visualization for `barth4` graph, on the left the Laplacian matrix plot, on the right its eigenvector plot

2.2. Clustering each graph in $K = 4$ clusters with the spectral and k-means method

When plotting the provided graphs some clear differences between the two algorithms arise:
We know that basic k-means does not have the concept of a cluster density, and it really shows as the cartesian plane is divided into "areas" all of which have heterogeneous densities of points, leading to clusters that vary greatly in size.
For the *Spectral method*, choosing a lower K number of clusters may lead to clearly separate clusters to be clustered together and choosing a higher K may lead to unnecessary fragmented clusters instead.

Figure 23: `barth4` graph clustering. On the left with k-means clustering and on the right with spectral clustering, both for $K = 4$.

Figure 24: `3elt` graph clustering. On the left with k-means clustering and on the right with spectral clustering, both for $K = 4$.

Figure 25: `airfoil1` graph clustering. On the left with k-means clustering and on the right with spectral clustering, both for $K = 4$.

2.3. Cluster size comparison

As stated before, the obvious difference between the two algorithms is that the basic k-means algorithm lacks knowledge of cluster density, partitioning the points based solely on their spacial distance. This results in clusters that are extremely heterogeneous in size.

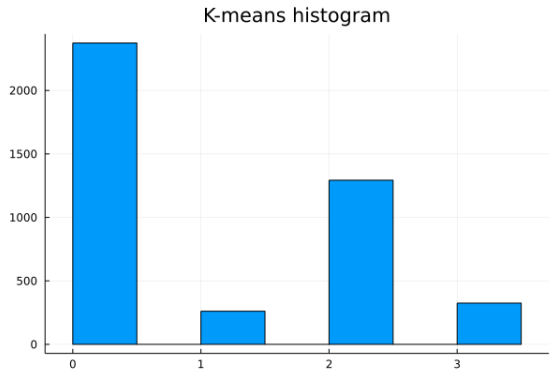

Figure 26: `airfoil1` k-means clustering.

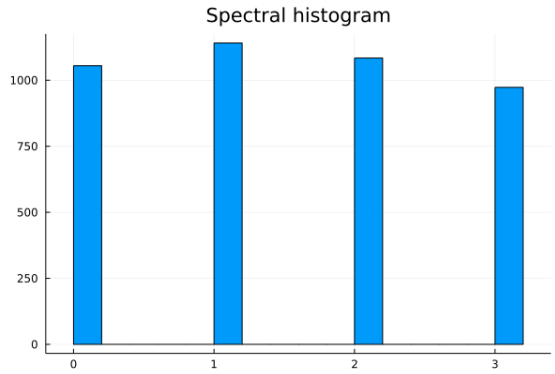

Figure 27: `airfoil1` spectral clustering .

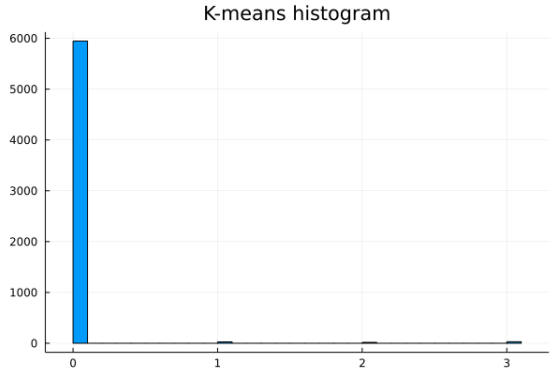

Figure 28: `barth4` k-means clustering.

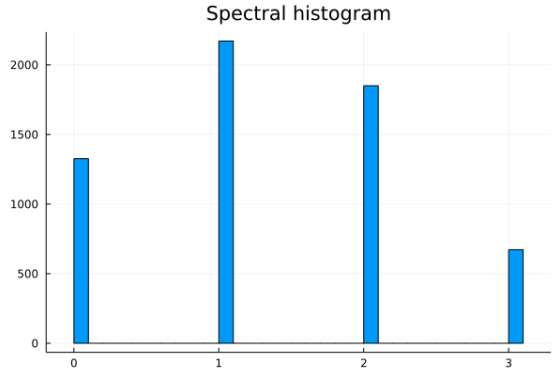

Figure 29: `barth4` spectral clustering.

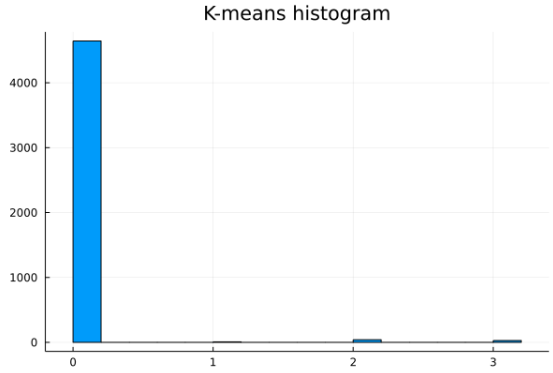

Figure 30: `3elt` k-means clustering.

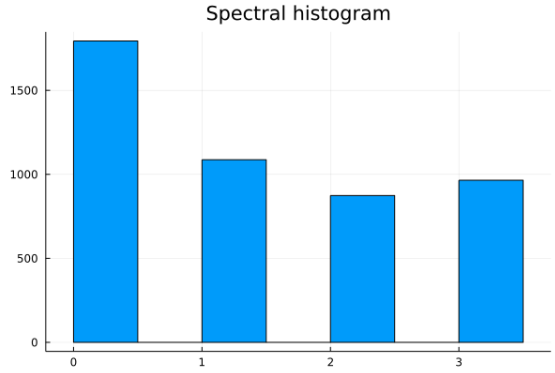

Figure 31: `3elt` spectral clustering.

Figure 32: Cluster size comparison with histograms

Case	Spectral	K-Means
airfoil1	1055,1141,1084,973	2374,261,1293,325
barth4	1326,2171,1850,672	5945,27,17,30
3elt	1794,1087,874,965	4644,8,40,28

Table 2: Cluster size comparison for $K = 4$

3. Reproducing the obtained results

In the `src/` folder inside the submission archive you can find a `Makefile`.

Run command `make` while having the current working directory set as the `src/` folder to plot and store all the results used for this report. All the plots and tables are saved inside the `src/out` folder. Make sure to uncomment the `GLMakie.save(...)` statements for the plots you want to generate.

A known bug on my `arm64` machine is the Julia REPL crashing unexpectedly for a segmentation error. Copying the code and pasting it in the Julia REPL running inside a standalone terminal prevents it from crashing for me.