Numerical Computing **2023**

Student: Hun Rim **Discussed with:** Georgy Batyrev

Solution for Project 3 **Due date:** Wednesday, 8 November 2023, 11:59 PM

Numerical Computing 2023 — 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, MATLAB). 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.

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

1. Run the Matlab function *datasets/getPoints()*. A graphical output of 6 different non-convex sets will be produced. Use this function to output the coordinate list of the different cloud-points and replace the variable *Pts dummy* with the set you wish to cluster. Consider the set named half-kernel: can you identify the two obvious clusters in this dataset? Describe them briefly and explain what difficulties a clustering algorithm could eventually encounter in a scenario of this kind.

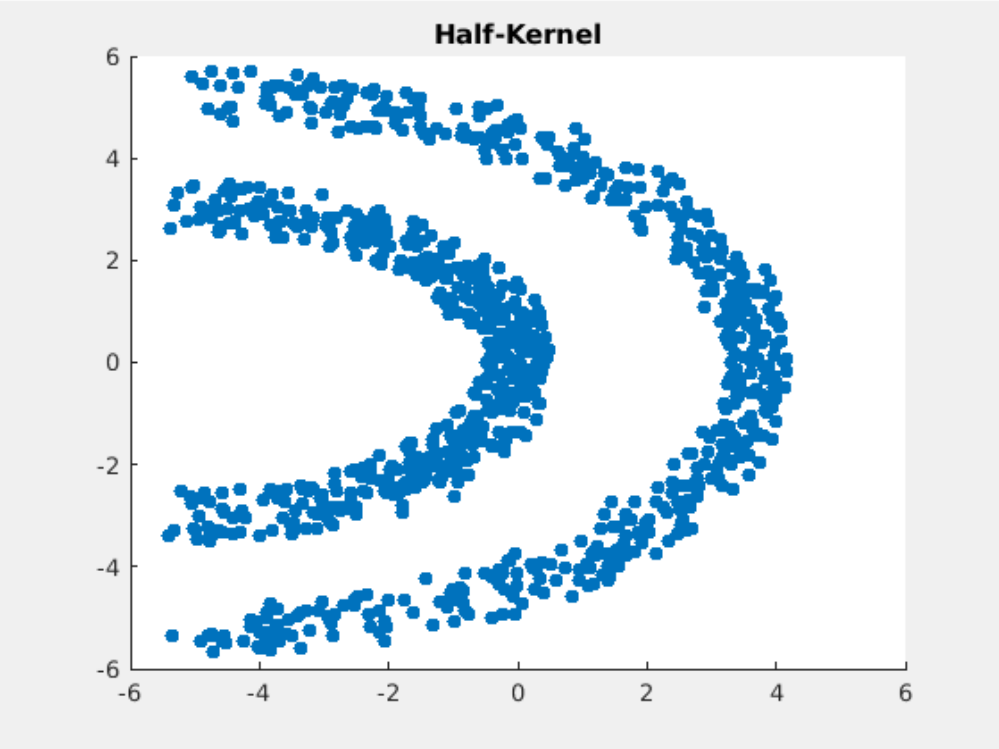

Figure 1: Graphical Output Of Half Kernel Dataset

As you can see from the graphical representation, the two clusters are very easy to spot. The first cluster is the smaller waxing crescent moon shaped collection of data points, and the second cluster is also waxing crescent moon shaped collection of data points but bigger than the first cluster and it surrounds the first cluster.

One example of the challenge this kind of clustering algorithm could face is incorrect partitioning of the datapoints into multiple clusters due to varying density if the algorithm relies on the density to partition. Even with the bare-eye it is clear that some parts of the graph is more densely populated than the others so it is a very possible issue.

Another problem can be when the cluster regions are close to each other. If the cluster regions are close, it would be very hard to differentiate separate clusters as there isn't any other factor to classify them.

Finally, non-convex clusters could be problematic for a basic clustering algorithm like **k-means**, as the distances between centroids are a important factor in calculating it but it is not so clear in non-convex sets. As a result, it might create an inccurate splits due to the center of the cluster which is very clustered.

2. Use the function *minSpanTree()* to find the minimal spanning tree of the full graph. Use this information to determine the ϵ factor for the ϵ-similarity graph.

   Minimum span tree(MST) is the minmum subset of the edges that can connect all the vertices in an undirected graph G with n vertices and m edges. Spectral clustering uses eigenvalues of the similarity matrix formed from the data points to reduce number of edges, and it is very useful when identifying clusters. Unlike k-means clustering, spectral clustering is very good with identifying clusters which are not linearly separable.

   We can define the MST and find the ϵ as following:

```
G = pdist2(Pts_halfk, Pts_halfk);
A = double(G < max(G(:)));
MST = minSpanTree(A);
eps = max(MST(MST > 0));
```

   The following is the plotted graph of MST.

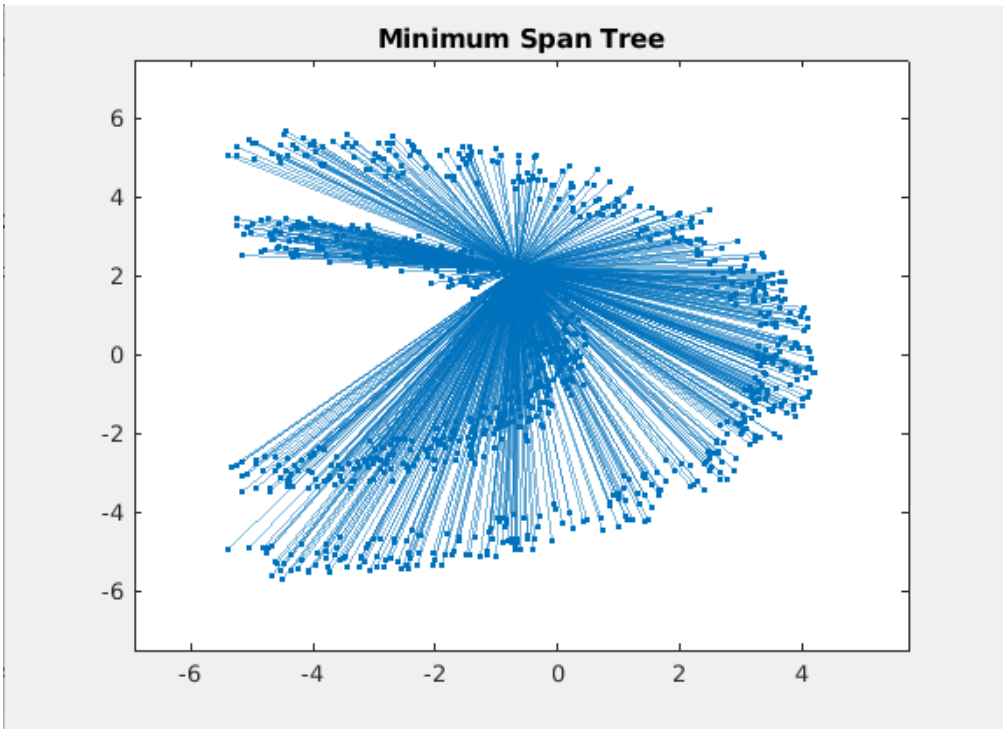

Figure 2: Plotted Minimum Span Tree of Half Kernel

3. Complete the Matlab function *epsilonSimGraph()*. It should generate the similarity matrix of the ϵ-similarity graph.

```
n = size(Pts, 1);
G = zeros(n, n);

for i = 1:n
    for j = 1:n
        if i == j
            continue;
        end
```

```
            distance = norm(Pts(i, :) - Pts(j, :));

            if distance < epsilon
                G(i, j) = 1;
            end
        end

    end
```

The *epsilonSimGraph()* can be completed like above to generate the similarity matrix of the ϵ-similarity graph.

The epsilon similarity graph is constructed by only connecting points which has distance less than epsilson units to emphasize the local neighborhood structure.

4. Create the adjacency matrix for the ϵ similarity graph and visualize the resulting graph using the function *gplotg()*.

   Like the name states, adjacency matrix denotes the neighbor relation between the nodes and states which vertices are next(adjacent) to other vertices. Because in an unweighted graph, the presence of adjacency is represented with binary, we can do the following to get an adjacency graph:

   ```
   W = double(D < eps);
   ```

   According to the implementation, the adjacency graph looks like the following.

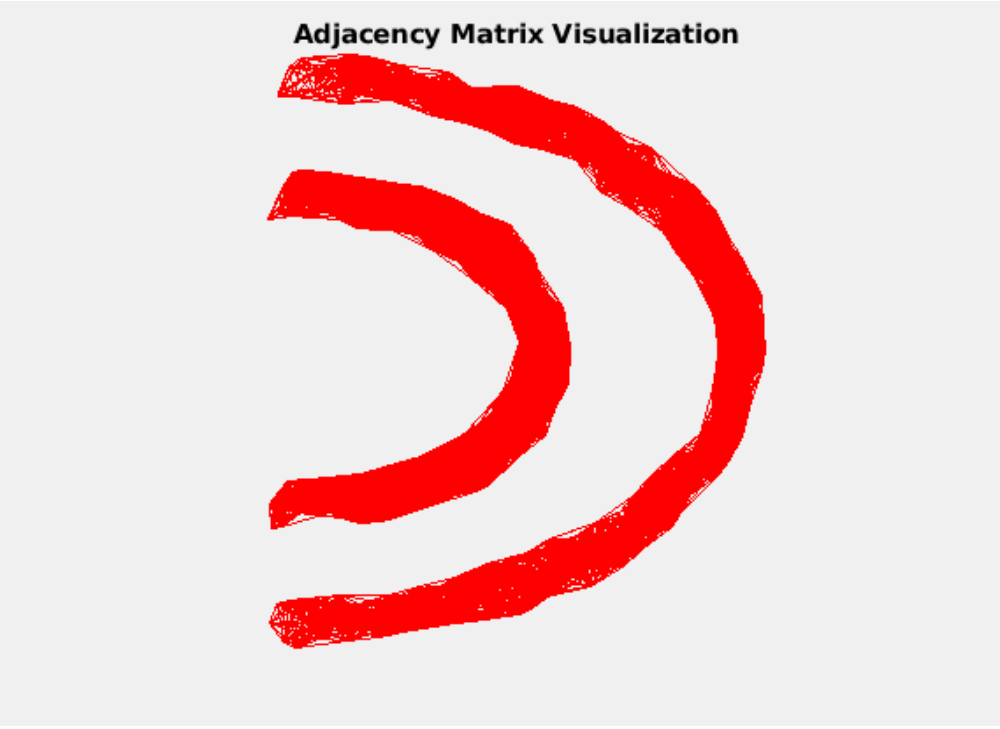

Figure 3: Adjacency Matrix Visualization

5. Create the Laplacian matrix and implement spectral clustering. Your goal is to find the eigenvectors of the Laplacian corresponding to the $K = 2$ smallest eigenvalues. Afterwards, use the function *kmeans_mod*() to cluster the rows of these eigenvectors.

   The implementation is as following:

```
[L, Diag] = CreateLapl(W);
[eigVec, eigVal] = eig(L);
[sortedEigVal, ind] = sort(diag(eigVal));
H = eigVec(:, ind(1:K));
H_normalized = diag(1./sqrt(sum(H.^2, 2))) * H;
```

   Then we just have to replace dummy value with *H_normalized* we obtained.

6. Use the *kmeans_mod*() function to perform k-means clustering on the input points. Visualize the two clustering results using the function *gplotmap()*. You can debug your implementation by using as a reference the results reported in Figure 1, which are relative to the half-kernel dataset.

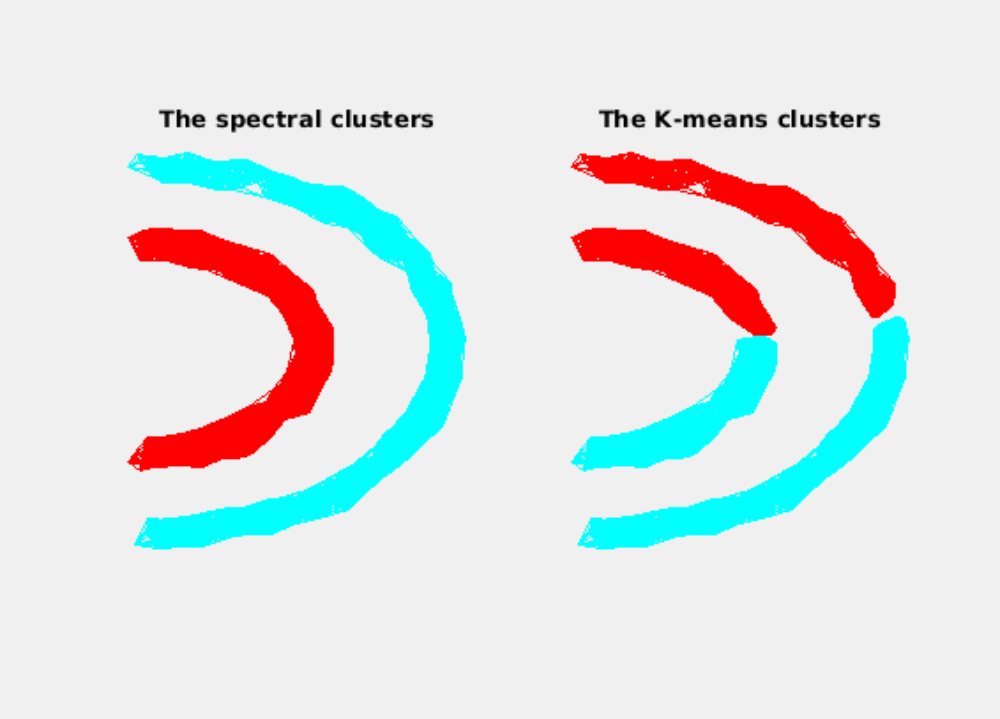

Figure 4: Half-kernel dataset in spectral clusters(left) and K-means-clusters(right), $K = 2$

7. Cluster the datasets *i*) Two spirals, *ii*) Cluster in cluster, and *iii*) Crescent & full moon in $K = 2$ clusters, and visualize the results obtained with spectral clustering and k-means directly on the input data. Do the same for *i*) Corners, and *ii*) Outlier for $K = 4$) clusters.

   To get the results, I am manually manipulating the values of the script.

1) $K = 2$

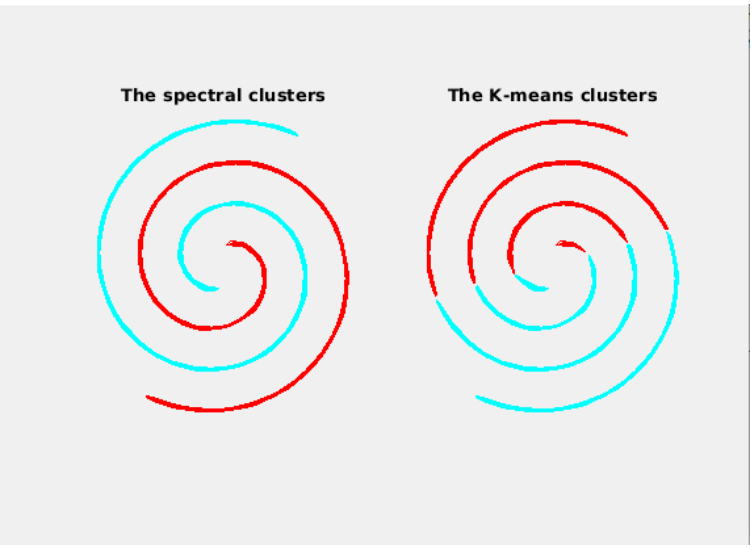

Figure 5: Two Spirals dataset in spectral clusters(left) and K-means-clusters(right), $K = 2$

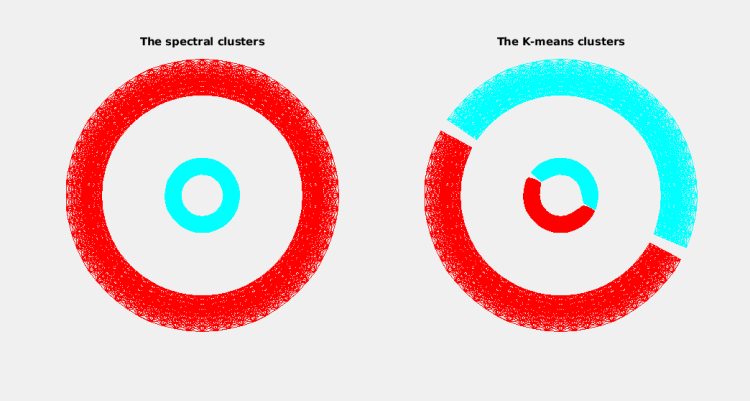

Figure 6: Cluster in cluster dataset in spectral clusters(left) and K-means-clusters(right), $K = 2$

Figure 7: Crescent full moon dataset in spectral clusters(left) and K-means-clusters(right), $K = 2$

2) $K = 4$

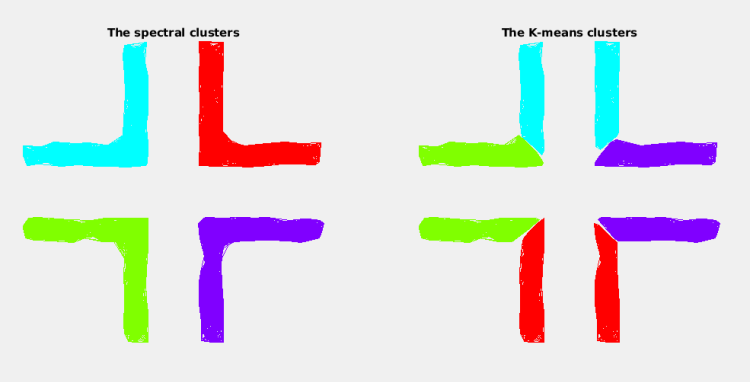

Figure 8: Cluster in cluster dataset in spectral clusters(left) and K-means-clusters(right), $K = 2$

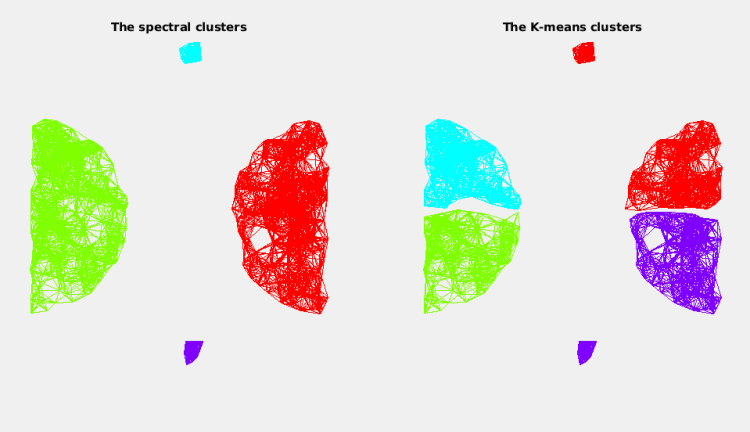

Figure 9: Cluster in cluster dataset in spectral clusters(left) and K-means-clusters(right), $K = 2$

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

1. Construct the Laplacian matrix, and draw the graphs using the eigenvectors to supply coordinates. Locate vertex i at position:

$$(x_i, y_i) = (v_2(i), v_3(i))$$

where $v2, v3$ are the eigenvectors associated with the 2nd and 3rd smallest eigenvalues. Figure 2 illustrates these 2 ways of visualizing the *airfoil1* graph. Plot your results for all the three additionally supplied meshes, grid2, barth, 3elt.

1) *grid2*

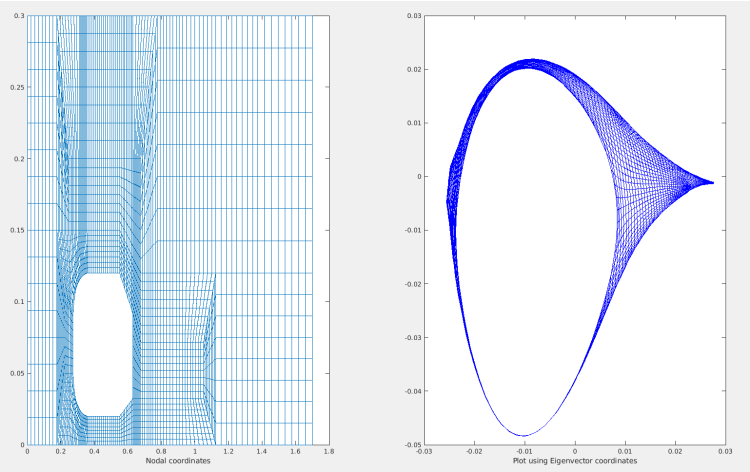

Figure 10: Nodal coordinates(left), Eigenvectors(right)

2) *barth*

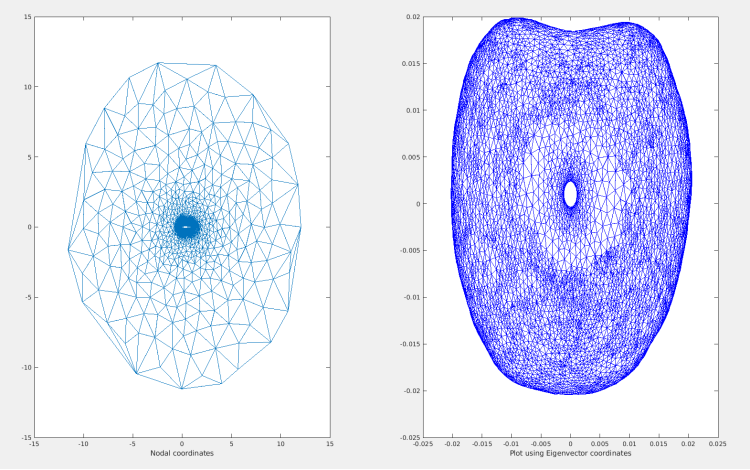

Figure 11: Nodal coordinates(left), Eigenvectors(right)

3) *3elt*

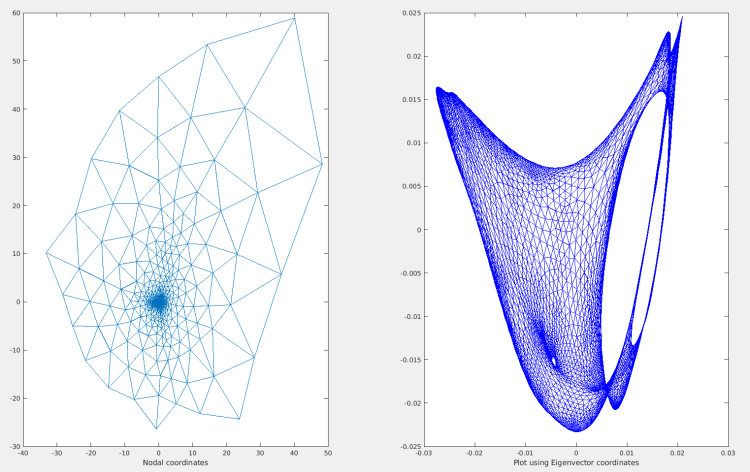

Figure 12: Nodal coordinates(left), Eigenvectors(right)