Approximation of eigenvalues and eigenvectors of a Laplacian matrix for spectral clustering with Kmeans*

*Course: Computational linear algebra for large scale problems

Michele Veronesi
Polytechinc University of Turin
Turin, Italy
s296599@studenti.polito.it

Leonardo Tredese
Polytechinc University of Turin
Turin, Italy
s302294@studenti.polito.it

***Abstract*—In this report we present the work done to approximate eigenvalues and eigenvectors of a symmetric Laplacian matrix, in order to perform spectral clustering on a given dataset. In particular we focused on the power method with Wielandt deflation technique, and implemented it with a recursive program. The correctness of such code is provided among the matlab files as a comment, which is located at the end of the document in the appendix C.**

1. Introduction

Our objective is to perform clustering on a given dataset. In this case, such set is composed of 900 points in $\mathbb{R}^2$, and is represented as a matrix $X \in \mathbb{R}^{900\times 2}$. In order to apply spectral clustering, we need to compute the eigenvalues of smallest modulus and the associated eigenvectors of the normalized symmetric Laplacian matrix (which will be introduced later on). To do so, we implemented our function for the power method and Wielandt deflation. From tests we have seen that it works pretty well with a problem of this dimension, anyway it can still be improved in order to get better performances and scale properly to larger tasks.

The outcomes of clustering are reported in appendix B. Moreover, results of eigenvalues computation with different techniques can be found in appendix A. Both will be later described in this report. For reproducibility purposes, the code used during our experiment is attached to this document in appendix C, and it can be downloaded from [1].

2. Construction of the adjacency matrix

We compute each element of the full adjacency matrix $S \in \mathbb{R}^{n\times n}$ (where n is the number of samples) using the similarity function reported in equation 1. When setting the tolerance to 10^{-8} and $\sigma = 1$, S is a symmetric dense matrix, as shown in figure 1, .

$$s_{ij} = \exp(-\frac{||x_i - x_j||^2}{\sigma^2}) \tag{1}$$

From S we extract the *k-nearest neighbourhood similarity graph* as a matrix W. You can see the sparsity pattern of W computed using $k = 13$ in figure 2 and $k = 40$ in figure 3.

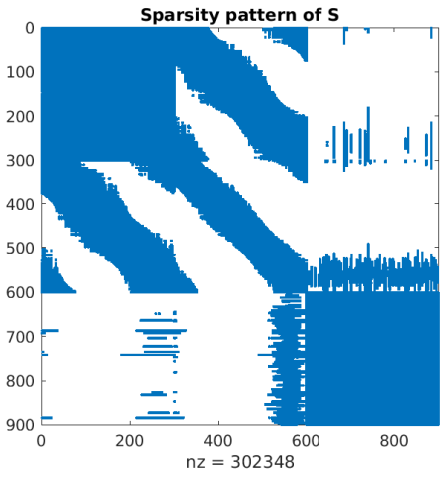


Figure 1: Sparsity pattern of S

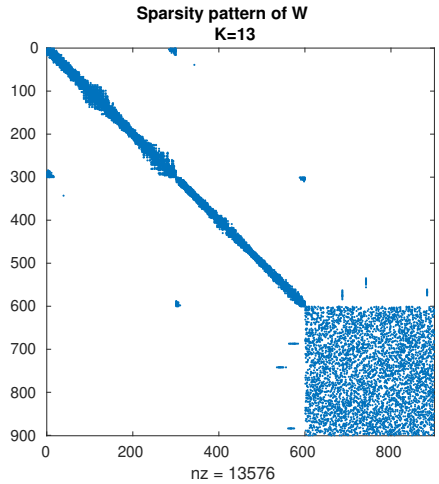


Figure 2: Sparsity pattern of W using 13-nearest neighbourhood similarity graph

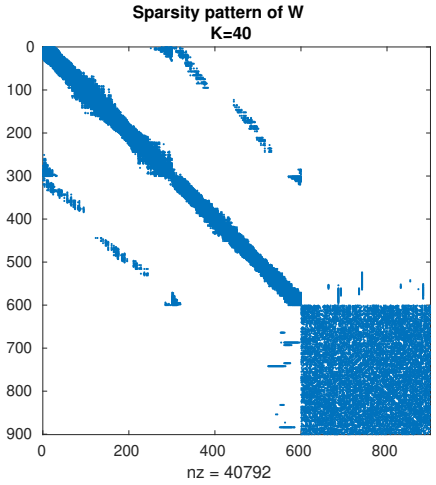


Figure 3: Sparsity pattern of W using 40-nearest neighbourhood similarity graph

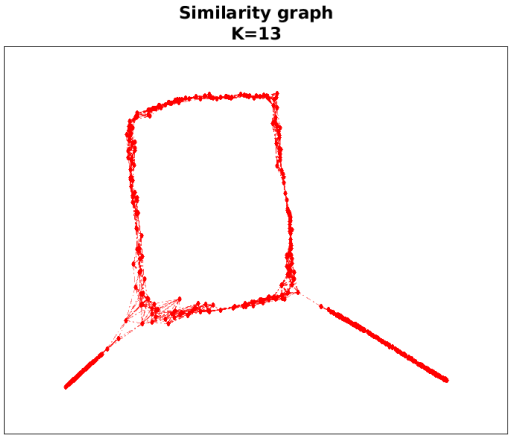


Figure 4: Similarity graph obtained using 13-nearest neighbourhood

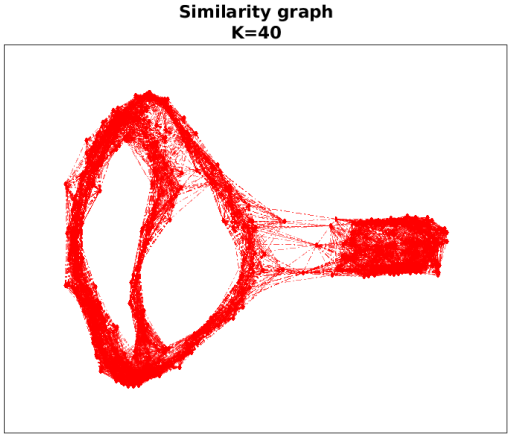


Figure 5: Similarity graph obtained using 40-nearest neighbourhood

Furthermore, the number of neighbours represented in W could be slightly grater than k. This happens when a node is in the k neighbours of another, but not the opposite. We include this connection as well, which can result in rows and columns with more than k non zero elements. Finally, we impose that W has only zeros on the main diagonal.

3. Normalized and unormalized Laplacian matrices

We define the unormalized Laplacian matrix $L \in \mathbb{R}^{n\times n}$ as $L = D - W$, where D is a diagonal matrix with each vertices degree (as defined in equation 2) on the main diagonal.

$$d_{ii} = \sum_{j=1}^{n} w_{ij} \tag{2}$$

We then define $L_{sym} \in \mathbb{R}^{n\times n}$ the normalized symmetric Laplacian matrix as described in equation 3.

$$L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - B \tag{3}$$

$$B = D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \tag{4}$$

Proposition 1. *$L_{sym} \in \mathbb{R}^{n\times n}$ is positive semi-definite and it has n non-negative real-valued eigenvalues $0 = \lambda_1 \leq \ldots \leq \lambda_n$*

3.1. Proof of proposition 1

Now we provide a proof for proposition 1. For every non-vanishing vector $v \in \mathbb{R}^n$ and $\mathcal{V} = D^{-\frac{1}{2}} v \in \mathbb{R}^n$ the equation 5 holds.

$$v^T L_{sym} v = \frac{1}{2} \sum_{i,j=1}^{n} (\mathcal{V}_i - \mathcal{V}_j)^2 \geq 0 \tag{5}$$

Here the proof.

$$
\begin{aligned}
&v^T L_{sym} v = \\
&= v^T D^{-\frac{1}{2}} L D^{-\frac{1}{2}} v = \mathcal{V}^T L \mathcal{V} \\
&= \mathcal{V}^T D \mathcal{V} - \mathcal{V}^T W \mathcal{V} \\
&= \sum_{i=1}^{n} d_i \mathcal{V}_i^2 - \sum_{i,j=1}^{n} \mathcal{V}_i \mathcal{V}_j w_{ij} \\
&= \frac{1}{2}(\sum_{i=1}^{n} d_i \mathcal{V}_i^2 - 2 \sum_{i,j=1}^{n} \mathcal{V}_i \mathcal{V}_j w_{ij} + \sum_{j=1}^{n} d_j \mathcal{V}_j^2) \\
&= \frac{1}{2}[\sum_{i=1}^{n} (\mathcal{V}_i^2 \sum_{j=1}^{n} w_{ij}) + \sum_{j=1}^{n} (\mathcal{V}_j^2 \sum_{i=1}^{n} w_{ij}) + \\
&- 2 \sum_{i,j=1}^{n} (w_{ij} \mathcal{V}_i \mathcal{V}_j)] \\
&= \frac{1}{2}(\sum_{i,j=1}^{n} w_{ij} \mathcal{V}_i^2 + \sum_{i,j=1}^{n} w_{ij} \mathcal{V}_j^2 - 2 \sum_{i,j=1}^{n} (w_{ij} \mathcal{V}_i \mathcal{V}_j) \\
&= \frac{1}{2} \sum_{i,j=1}^{n} (w_{ij} \mathcal{V}_i^2 + w_{ij} \mathcal{V}_j^2 - 2 w_{ij} \mathcal{V}_i \mathcal{V}_j) \\
&= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} \cdot (\mathcal{V}_i^2 + \mathcal{V}_j^2 - 2 \mathcal{V}_i \mathcal{V}_j) \\
&= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij} (\mathcal{V}_i - \mathcal{V}_j)^2 \geq 0 \quad \square
\end{aligned}
\tag{6}
$$

The final equalities of the proof, are true thanks to the definition in equation 2. By acknowledging that $w_{ij} \geq 0 \; \forall i, j$ and proof of equation 5, we can state that L_{sym} is semi positive definite. $\square$

Then we proceed in showing that L_{sym} has at least one zero as eigenvalue. Given the definition of the adjacency matrix W and the diagonal degrees matrix D, equation 7 holds.

$$
\begin{aligned}
W e_n &= \begin{bmatrix} d_1 \\ \vdots \\ d_n \end{bmatrix} \text{ with } e_n = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^n \\
L e_n &= (D - W) e_n = D e_n - W e_n = \\
&= \begin{bmatrix} d_1 \\ \vdots \\ d_n \end{bmatrix} - \begin{bmatrix} d_1 \\ \vdots \\ d_n \end{bmatrix} = 0 e_n
\end{aligned}
\tag{7}
$$

$$
\begin{aligned}
L_{sym} e_{sym} &= D^{-\frac{1}{2}} L D^{-\frac{1}{2}} D^{\frac{1}{2}} e_n = \\
&= D^{-\frac{1}{2}} L e_n = 0 e_{sym}
\end{aligned}
\tag{8}
$$

Let us now define a vector $e_{sym} = D^{\frac{1}{2}} e_n$. Since the degree of each node is greater than zero we can state that $e_{sym} \neq 0 \in \mathbb{R}^n$. Combining the previous sentence with equation 8, we can state that e_{sym} is an eigenvector of L_{sym} for its eigenvalue 0, therefore proving proposition 1.

3.2. Eigenvalues and eigenvectors of the normalized symmetric Laplacian matrix

To approximate the minimum eigenvalues in modulus and associated eigenvectors of L_{sym} we use the power method with deflation applied to B_{mod} defined in equation 9, where B is the matrix defined in equation 3 and μ is the largest row sum of B. In particular, we seek for its highest eigenvalues in modulus.

$$
B_{mod} = B + \mu I \tag{9}
$$

Since B_{mod} is a Hermitian diagonally dominant matrix we can state that its eigenvalues are non-negative. Given an eigenvalue of B_{mod}, let us say λ_B, we can get the corresponding eigenvalue λ_L of L_{sym} with equation 10. With this relation, given the highest in modulus eigenvalues of B_{mod}, we obtain the minimum in modulus eigenvalues of L_{sym}.

$$
\lambda_L = 1 + \mu - \lambda_B \tag{10}
$$

Since B_{mod} is defined as a shift of L_{sym}, we can state that an eigenvector of L_{sym} is linearly dependent on the corresponding eigenvector of B_{mod}. Therefore we can use eigenvectors of B_{mod} by normalizing them in the clustering phase, since they have the same direction (but in the opposite way). To obtain the exact direction we should multiply eigenvectors by -1, but this operation is not necessary for our purpose.

You can read the code of the functions for eigenvalues and eigenvectors approximation in appendices C.1 and C.2. In particular we defined it with a recursive approach. You can find a proof of correctness of the power deflation function among the code reported in appendix C.1. The function requires two mandatory parameters:

- A: a square matrix in $\mathbb{R}^{n \times n}$.
- M: a scalar strictly greater than zero, indicating the number of eigenvalues and eigenvectors to be found.

The function returns three values:

- λ: a vector in $\mathbb{R}^M$ containing the M highest eigenvalues of A in modulus.
- V: a matrix in $\mathbb{R}^{n \times M}$ with the M associated eigenvectors as columns.
- iter: a vector in $\mathbb{R}^M$ indicating the number of iterations needed by the power method to compute each eigenvalue and the relative eigenvector.

The iterations performed for each eigenvalue and for $k = 13, 40$ are reported in figures 6 and 7 respectively. Meanwhile table 1 reports the error commited in computing eigenvalues with respect to those found by *eigs* function [2] and the overall time needed to compute them.

TABLE 1: Eigenvalues and eigenvectors computation analysis

K	Eigs time (sec)	Power method time (sec)	Error
13	0.02	52.26	$4.132 \cdot 10^{-8}$
40	0.04	9.19	$5.126 \cdot 10^{-8}$

When we apply the metho the tolerance is set to 10^{-16} and the maximum number of iterations is set to 10^5. With these parameters the tolerance is reached for every eigenvalue without a premature stop. If we compute the error between eigenvalues approximated by our function and those of the *eigs* [2] function, we have a mean error in the order of 10^{-8}: thus we have a pretty good results.

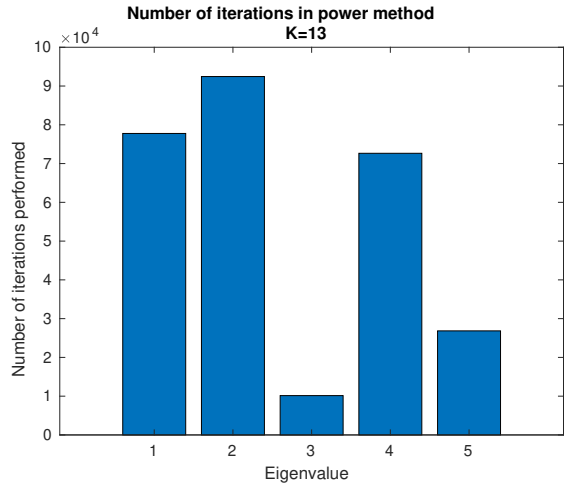


Figure 6: Number of iterations for computing eigenvalues with k=13

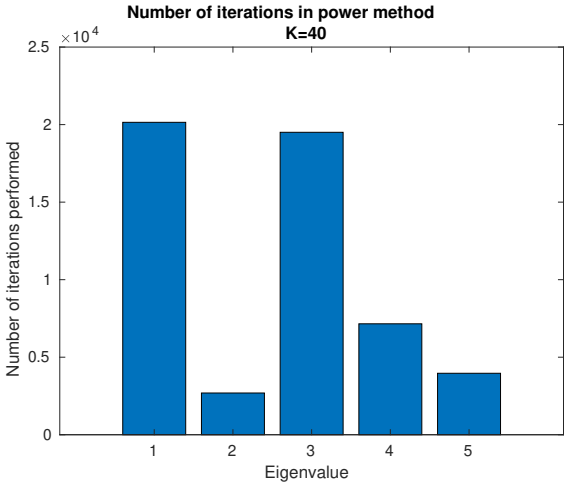


Figure 7: Number of iterations for computing eigenvalues with k=40

3.3. Eigengap heuristic

In order to find the eigengap, in appendix A we confront eigenvalues computed using different methods:

- our function for power method with Wielandt deflation
- the *eigs* function [2]
- the *spectralcluster* function [3]

You can see them in figures 8 and 9. The idea behind eigengap heuristic, is to choose m (the number of eigenvectors used to represent samples in the spectral clustering technique) such that eigenvalues $\lambda_1, \dots, \lambda_m$ are small, but $\lambda_{m+1} > 0$ is relatively large (so there is a gap between λ_m and λ_{m+1}.

In the case of $k = 13$ we take the first 3 eigenvectors since the gap appears after the third eigenvalue. In the case of $k = 40$, instead, we take only the first two eigenvectors. This is related to proposition 2.

Proposition 2. *Let $\mathcal{G}$ be an undirected graph with non-negative weigths. Then the multiplicity m of the eigenvalue 0 of L_{sym} equals the number of connected components $A_1, \dots, A_m$ in the graph. The eigenspace of eigenvalue 0 is spanned by the vectors $D^{\frac{1}{2}}\mathbb{1}_{A_i}$, where $\mathbb{1}_{A_1}, \dots, \mathbb{1}_{A_m}$ are the indicator vectors.*

3.4. Proof of proposition 2

Now we are going to prove proposition 2 for $m = 1$ (i.e. a single connected component). Consider again equations 5 and 6, assuming that v is an eigenvector of L_{sym} associated with eigenvalue zero and $\mathcal{V} = D^{-\frac{1}{2}}v$, then equation 11 holds.

$$v^T L_{sym} v = \sum_{i,j=1}^{n} w_{ij}(\mathcal{V}_i - \mathcal{V}_j)^2 = 0 \tag{11}$$

Given the vertices of the graph $p_1, \dots p_n$, we assume that they are ordered in a way such that there is a connection between p_i and $p_{i+1} \forall i \in \{1, \dots, n-1\}$. We also have that the matrix W (representing arches) is composed by non-negative entries $w_{i,j} \geq 0 \ \forall i, j \in \{1, \dots, n\} \subset \mathbb{N}$. Thus, the sum in equation 11 can only vanish if all terms vanish. If two verteces p_1 and p_2 are connected ($w_{1,2} = w_{2,1} > 0$) then $\mathcal{V}_1$ must be equal to $\mathcal{V}_2$. Now let us assume that if two vertices p_k and p_{k+1} are connected, then $w_{k,k+1} = w_{k+1,k} > 0$ and $\mathcal{V}_k = \mathcal{V}_{k+1}$. Consider p_{k+2}, which is directly connected to p_{k+1} because of our ordering criteria of vertices: we have that $w_{k+1,k+2} = w_{k+2,k+1} > 0$, therefore $\mathcal{V}_k = \mathcal{V}_{k+1} = \mathcal{V}_{k+2}$. By induction, $\mathcal{V}$ must be constant for all vertices connected by a path in the graph (i.e. all vertices because $m = 1$). □

As a corollary of the proof of proposition 1, we must conclude that this eigenvector is always equal to the constant one vector $\mathbb{1}$. Then, as we defined $\mathcal{V}$, we can state that $v = D^{\frac{1}{2}}\mathcal{V} = D^{\frac{1}{2}}\mathbb{1}$ is the eigenvector associated with eigenvalue 0. □

Now we show that the algebraic multiplicity of eigenvalue 0 is equal to the number of connected components (for our specific case one connected component). Suppose that the graph is connected but the algebraic multiplicity of eigenvalue zero is greater than 1. Then there exists at least two linearly independent eigenvectors v_1, v_2 with zero as associated eigenvalue. Then $v_1^T L_{sym} v_1 = v_2^T L_{sym} v_2 = 0$. But, from equation 11, we have that

$$\sum_{i,j=1}^{n} w_{ij}(\mathcal{V}_{i1} - \mathcal{V}_{j1})^2 = \sum_{i,j=1}^{n} w_{ij}(\mathcal{V}_{i2} - \mathcal{V}_{j2})^2 = 0$$

and we can clearly see that $\mathcal{V}_1$ and $\mathcal{V}_2$ are a scalar multiplier of each other because they are both constant, therefore they are linearly dependent, thus the algebraic multiplicity of eigenvalue zero must be equal to 1. □

4. Clustering results

Finally, we discuss the results obtained performing spectral clustering on eigenvectors computed with the power method. Furthermore, we compare them to the results obtained with the *spectralcluster* function and the K-means algorithm applied directly to the original dataset. All the figures representing clustering results are located in appendix B. In general, performances are better when we have an eigengap equal to the number of clusters.

This fact follows directly from proposition 2, since the multiplicity of the eigenvalue zero corresponds to the number of connected components (i.e. the number of clusters). Therefore, if we want to properly use spectral clustering we need to find the right way to construct the similarity graph (e.g. the right value for K if we use K-nearest-neighbours or the right parameter σ in the similarity function).

We can clearly see the advantage of performing spectral clustering with respect to applying K-means naively to original points when we want to obtain three clusters. K-means is capable of detecting globular clusters, making it impossible to distinguish the two concentric round clusters in a proper way when applying it to the original dataset.

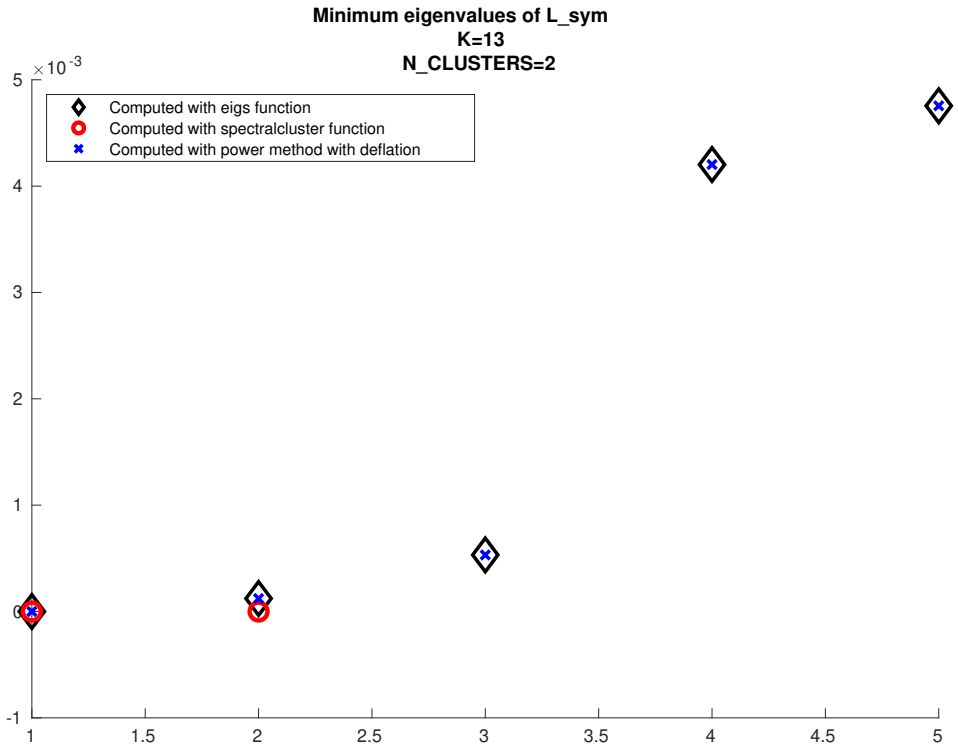
References

[1] L. Tredese and M. Veronesi. (2022, 2) Code repository. [Online]. Available: https://github.com/mveronesi/SpectralClustering/releases/tag/final

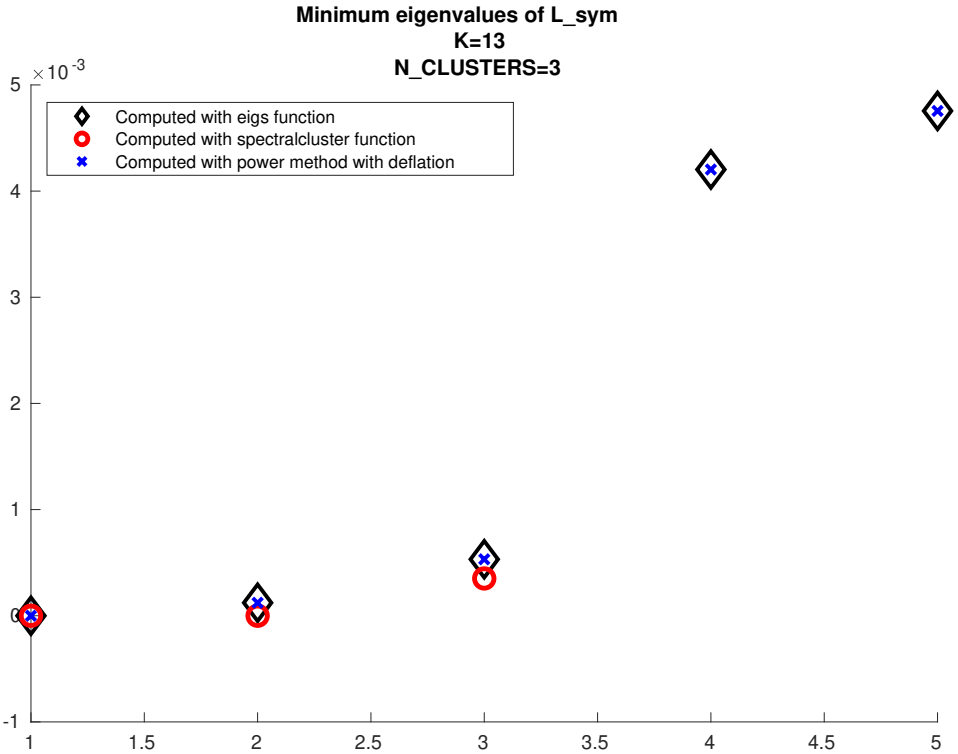
[2] Matlab eigs function. [Online]. Available: https://www.mathworks.com/help/matlab/ref/eigs.html

[3] Matlab spectral clustering function. [Online]. Available: https://www.mathworks.com/help/stats/spectralcluster.html

Appendix A.
Eigenvalues comparison

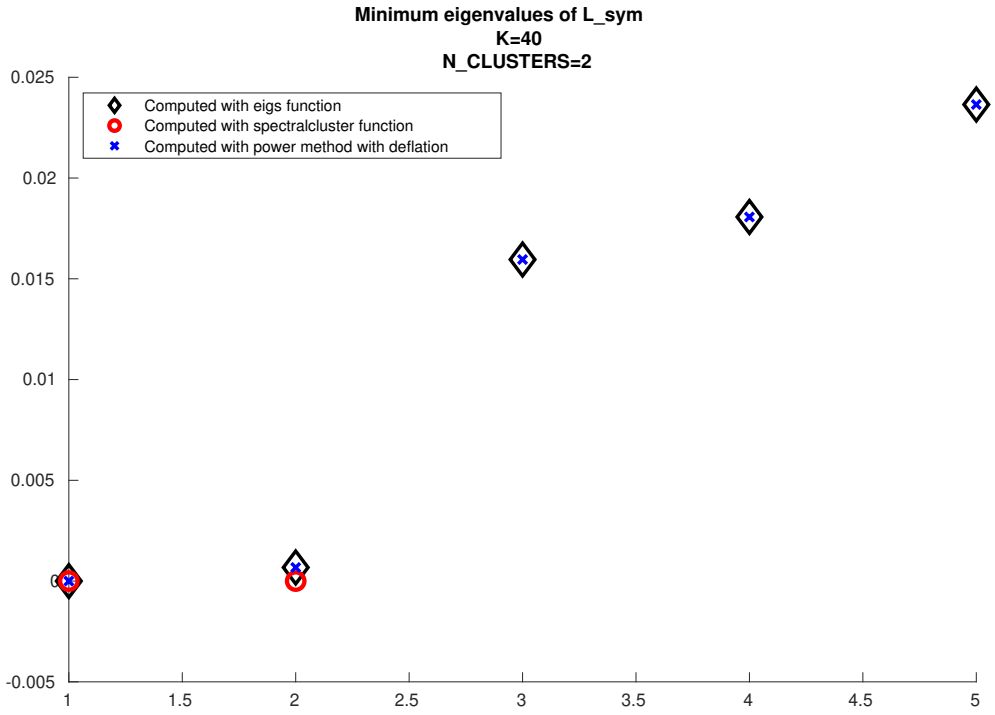


(a) 2 clusters

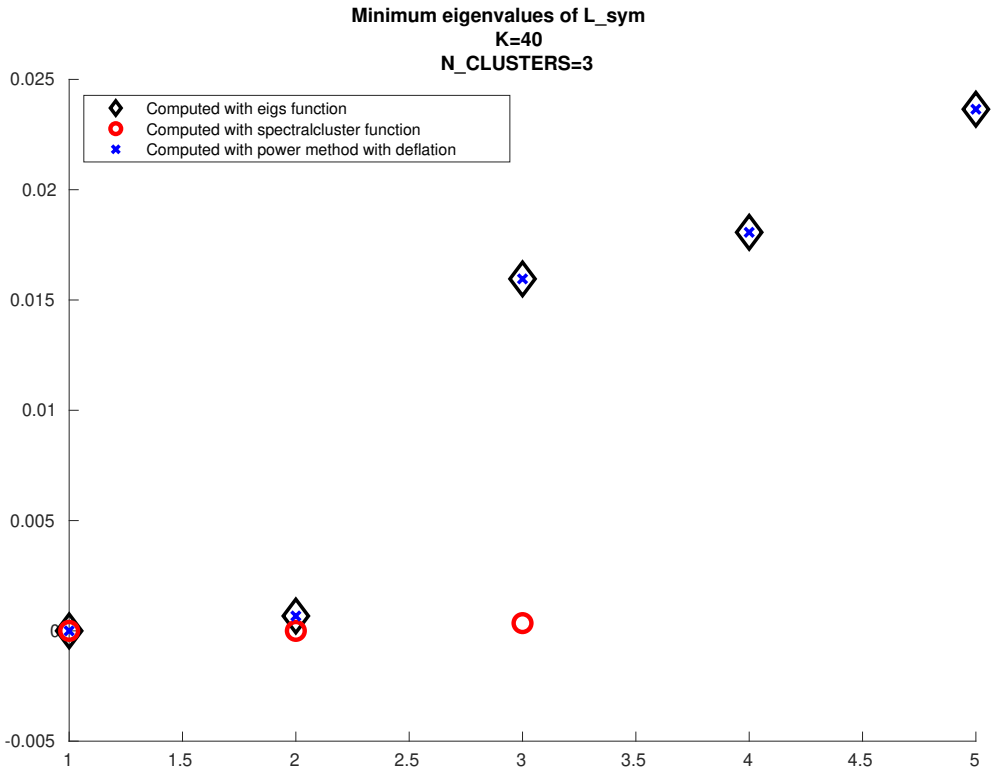


(b) 3 clusters

Figure 8: Eigenvalues comparison with K=13



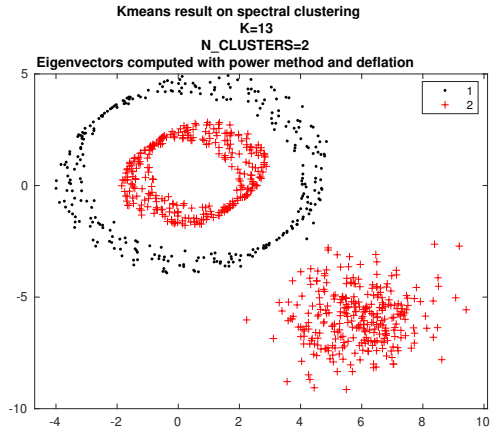
(a) 2 clusters



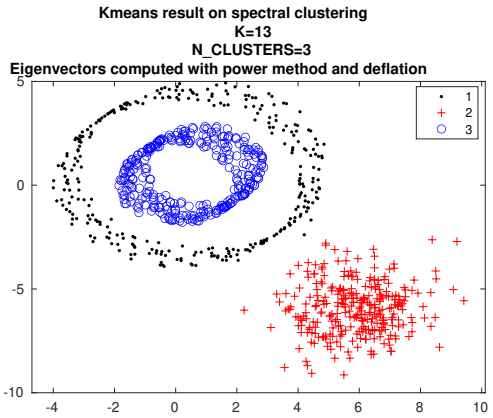
(b) 3 clusters

Figure 9: Eigenvalues comparison with K=40

Appendix B.
Clustering results figures

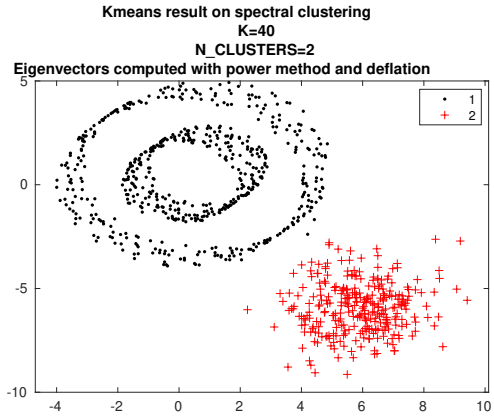


(a) 2 clusters

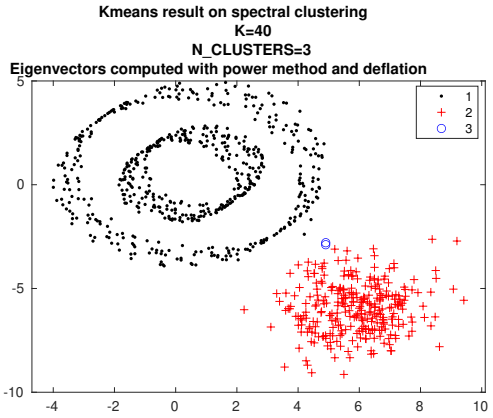


(b) 3 clusters

Figure 10: Spectral clustering with k=13, eigengap=3

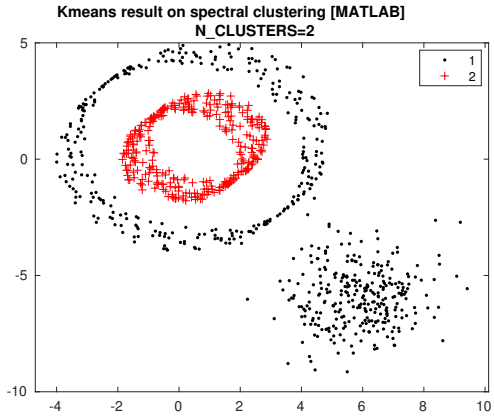


(a) 2 clusters

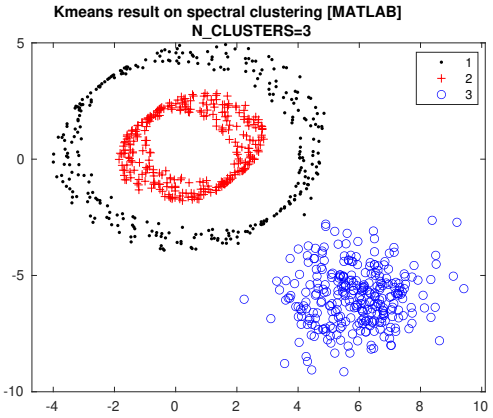


(b) 3 clusters

Figure 11: Spectral clustering with k=40, eigengap=2

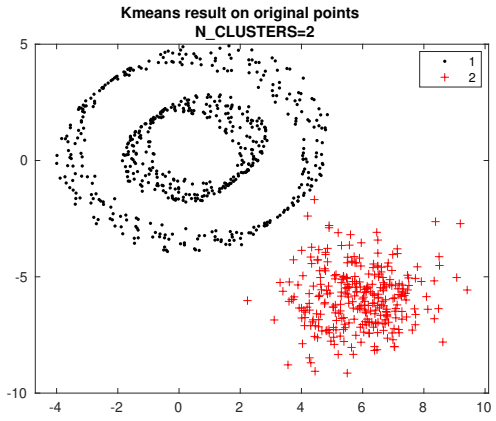


(a) 2 clusters

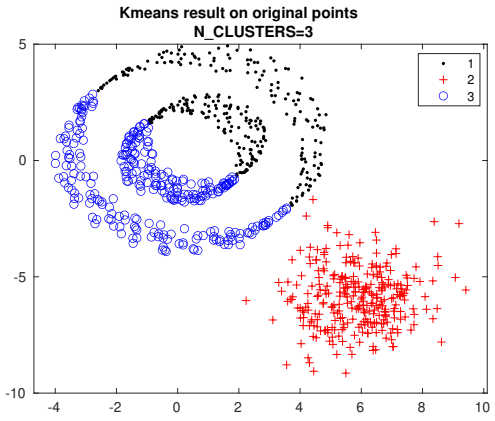


(b) 3 clusters

Figure 12: Spectral clustering with MATLAB function



(a) 2 clusters



(b) 3 clusters

Figure 13: K-means on original dataset

Appendix C. MATLAB code

C.1. Power method with deflation

```
function [lambda, v, iter] = power_deflation_rec(A, M, tol, nmax, x0)
% PRE-conditions:
% - A is a square matrix of size n-by-n
% - M > 0 is the number of largest eigenvalues of A that we want to find
% - tol, nmax, x0 are optional parameters used to apply the iterative
% procedure power_method to find the largest eigenvalue of A and the
% associated eigenvector. Furthermore the size of x0 is n-by-1, a column
% unitary vector
%
% POST-conditions:
% - lambda is a vector M-by-1 with the largest in modulus M eigenvalues of A
% - v is a matrix n-by-M where the columns are the associated eigenvectors
%
% Among the code we will place comments to demonstrate the correctness of
% the function with respect to the PRE and POST conditions stated above.

% Initialization
if M < 1
    error('M must be ≥ 1');
end
[n, m] = size(A);
if n ≠ m
    error('Matrix A is not square');
end
if nargin == 2
    tol = 1e-16;
    nmax = 1e5;
    x0 = ones(n, 1);
end
x0 = x0 / norm(x0);
% The PRE-condition of the function power_method is respected.
[lambda_1, v_1, iter_1] = power_method(A, tol, nmax, x0);
% Therefore the POST-condition hold since we assume the correctness of the
% function:
% - lambda_1 is the eigenvector with highest modulus of A
% - v_1 is the associated eigenvector
if M == 1
    % BASE CASE (M=1)
    % lambda_1 is the highest eigenvalue of A and v_1 is n-by-1, the
    % associated eigenvector. This condition match exactly the
    % POST-condition of power_deflation_rec, so no more operations are
    % needed.
    lambda = lambda_1;
    v = v_1;
    iter = iter_1;
```

```
else
    % INDUCTIVE CASE (M > 1)
    % now we deflate the matrix A obtaining B of size (n-1)-by(n-1), then
    % we find the other M-1 eigenvalues and eigenvectors of B using
    % inductive properties of this recursive function
    i = find(abs(v_1) == max(abs(v_1)), 1);
    if(i ≠ 1)
        B(1:i-1, 1:i-1) = A(1:i-1,1:i-1)-v_1(1:i-1)/v_1(i)*A(i,1:i-1);
        if(i ≠ n)
            B(i:n-1,1:i-1) = A(i+1:n,1:i-1)-v_1(i+1:n)/v_1(i)*A(i,1:i-1);
            B(1:i-1,i:n-1) = A(1:i-1,i+1:n)-v_1(1:i-1)/v_1(i)*A(i,i+1:n);
        end
    end
    if(i ≠ n)
        B(i:n-1,i:n-1) = A(i+1:n,i+1:n)-v_1(i+1:n)/v_1(i)*A(i,i+1:n);
    end
    % The recursive PRE-Condition hold:
    % - B is a square matrix (n-1)-by-(n-1)
    % - M-1>0 (M>1) is the number of eigenvalues of B to find
    [lambda_next, v_next, iter_next] = power_deflation_rec(B, M-1);
    % Therefore, by inductive assumption, the recursive POST-condition hold:
    % - lambda_next is a vector (M-1)-by-1 with the M-1 highest in modulus
    % eigenvalues of B
    % - v_next is a matrix (n-1)-by-(M-1) whose columns are the associated
    % eigenvectors
    %
    % now we need to transform the eigenvector returned by the previous
    % call in order to satisfy the general POST-condition of this function,
    % then by induction the correctness is fulfilled.
    %
    % because of the deflation properties we do not need to apply
    % any operations to eigenvalues, we just respect the size of the vector
    % lambda imposed by the POST-condition
    lambda = [lambda_1; lambda_next];
    iter = [iter_1; iter_next];
    v = zeros(n, M);
    v(:, 1) = v_1;
    % now we transform eigenvectors returned by the recursive call to
    % dimensionality n, while we satisy the constraints on the matrix v
    % stated in the POST-condition
    for j = 1:(M-1) % scanning columns of v_next
        w_prime = v_next(:, j);
        w = zeros(n, 1);
        if(i ≠ 1)
            w(1:i-1) = w_prime(1:i-1);
        end
        w(i) = 0;
        if(i ≠ n)
            w(i+1:n) = w_prime(i:n-1);
        end
        v(:, j+1) = w;
    end
end
end
```

C.2. Power method

```
function [lambda, x, iter] = power_method(A, tol, nmax, x0)
% PRE-conditions:
% - A is a square matrix of size n-by-n
% - tol, nmax, x0 are parameters used to apply the power method.
% Furthermore x0 has size n-by-1 and it is a unitary vector.
%
% POST-conditions:
% - lambda is the eigenvalues with the highest modulus of A
% - x is the associated eigenvector
% - iter is the number of iteration needed to reach the tolerance
%
% We assume the correctness of this function and do NOT demonstrate it
% with respect to the PRE and POST conditions stated above.

    pro = A*x0;
    lambda = x0'*pro ;
    err = tol*abs(lambda)+1;
```

```
    iter = 0;
    while (err > tol*abs ( lambda )) && (iter ≤ nmax)
        x = pro;
        x = x/norm(x);
        pro = A*x;
        lambdanew = x'*pro ;
        err = abs(lambdanew-lambda);
        lambda = lambdanew ;
        iter = iter + 1;
    end
end
```

C.3. Main script

```
% Initialization
clear; close all; clc;
format long;
load('X.mat');
sigma = 1; tol = 1e-8;
similarity = @(x, y) exp(-norm(x-y, 2).^2 ./ sigma.^2);
% parameters for clustering results plots
symbols = ['.', '+', 'o'];
colors = ['k', 'r', 'b'];

for K = [13, 40]
    disp(['############ K = ', num2str(K), ' ##############']);
    % Generating similarity matrix between points
    [n, ¬] = size(X);
    S1 = zeros(n, n);
    for i=1:(n-1)
        p1 = X(i, :);
        for j=(i+1):n
            p2 = X(j, :);
            s = similarity(p1, p2);
            if s > tol
                S1(i,j) = s;
                S1(j,i) = s;
            end
        end
    end
    figure();
    spy(S1);
    title('Sparsity pattern of S');
    S2 = S1;
    % Generating similarity graph with k-nearest neighbourhood graph
    for i = 1:n
        [¬, kmax_pos] = maxk(S1(i, :), K);
        S1(i, setdiff(1:n, kmax_pos)) = 0;
    end
    for j = 1:n
        [¬, kmax_pos] = maxk(S2(:, j), K);
        S2(setdiff(1:n, kmax_pos), j) = 0;
    end
    max_nz = nnz(S1) + nnz(S2);
    column_indices = zeros(max_nz, 1);
    row_indices = zeros(max_nz, 1);
    values = zeros(max_nz, 1);
    nz = 0;
    for i=1:n
        for j=1:n
            if S1(i,j) > tol && S2(i,j) ≤ tol
                nz = nz + 1;
                column_indices(nz) = j;
                row_indices(nz) = i;
                values(nz) = S1(i,j);
            elseif S1(i,j) ≤ tol && S2(i,j) > tol
                nz = nz + 1;
                column_indices(nz) = j;
                row_indices(nz) = i;
                values(nz) = S2(i,j);
            elseif S1(i,j) > tol && S2(i,j) > tol
                if S1(i,j) - S2(i,j) > tol % should be equals
                    warning("Something went wrong...");
                end
```

```
                nz = nz + 1;
                column_indices(nz) = j;
                row_indices(nz) = i;
                values(nz) = S1(i,j);
            end
        end
    end
    column_indices = column_indices(1:nz);
    row_indices = row_indices(1:nz);
    values = values(1:nz);
    W = sparse(row_indices, column_indices, values, n, n);

    figure();
    spy(W);
    title({'Sparsity pattern of W', ['K=', num2str(K)]});

    figure();
    plot(graph(W), '-.dr', 'NodeLabel', {});
    title({'Similarity graph', ['K=', num2str(K)]});

    % Computing the normalized symmetric Laplacian
    degrees = full(sum(W, 2));
    D = spdiags(degrees, 0, n, n);

    M = 5;

    D_tmp = sqrt(inv(D));
    B = D_tmp * W * D_tmp;
    L_sym = speye(n) - B;

    % Computing eigenvalues and eigenvectors with eigs function
    tic;
    [U_orig, lambda_orig] = eigs(L_sym, M, 'smallestabs');
    elapsed = toc;
    fprintf('Elapsed time with eigs: %.2f sec\n', elapsed)
    lambda_orig = diag(lambda_orig);

    % Computing eigenvalues and eigenvectors with power method and deflation
    mu = max(full(sum(B, 2)));
    B_mod = B + mu .* speye(n);
    tic;
    [lambda, U, iter] = power_deflation_rec(B_mod, M);
    elapsed = toc;
    fprintf('Elapsed time with power_deflation_rec: %.2f sec\n', elapsed)
    lambda = 1-lambda+mu;
    rel_err_eigs = norm(lambda - lambda_orig, 2);
    fprintf('Error in the computation of eigenvalues: %e\n', rel_err_eigs);

    figure();
    bar(1:M, iter);
    title({'Number of iterations in power method', ['K=', num2str(K)]});
    xlabel('Eigenvalue');
    ylabel('Number of iterations performed');

    switch K
        case 13
            EIGENGAP = 3;
        case 40
            EIGENGAP = 2;
        otherwise
            warning('Eigengap not computed for this value of K, setting to 3');
            EIGENGAP = 3;
    end

    U = U(:, 1:EIGENGAP);

    % normalizing eigenvectors
    for j=1:EIGENGAP
        U(:,j) = U(:,j) ./ norm(U(:,j));
    end

    % Computating matrix U containing first m eigenvectors
    % and normalizing its rows
    for i=1:n
        U(i,:) = U(i,:) ./ norm(U(i,:));
    end

    % Applying K-means for spectral clustering
```

```
    for N_CLUSTERS = [2, 3]

        rng(42); % setting random state for kmeans reproducibility
        idx = kmeans(U, N_CLUSTERS);
        figure();
        gscatter(X(:,1), X(:,2), idx, colors, symbols);
        title({'Kmeans result on spectral clustering', ...
            ['K=', num2str(K)], ['N\_CLUSTERS=', num2str(N_CLUSTERS)], ...
            'Eigenvectors computed with power method and deflation'});

        % Using spectral cluster of MATLAB.
        rng(42); % setting random state for kmeans reproducibility
        [idx, V, D] = spectralcluster(X, N_CLUSTERS);
        figure();
        gscatter(X(:,1), X(:,2), idx, colors, symbols);
        title({'Kmeans result on spectral clustering [MATLAB]', ...
            ['N\_CLUSTERS=', num2str(N_CLUSTERS)]});

        % Applying K-means on original points
        rng(42); % setting random state for kmeans reproducibility
        idx = kmeans(X, N_CLUSTERS);
        figure();
        gscatter(X(:,1), X(:,2), idx, colors, symbols);
        title({'Kmeans result on original points', ...
            ['N\_CLUSTERS=', num2str(N_CLUSTERS)]});

        % comparing eigenvalues computed by the function
        % spectralcluster to those of our function
        figure();
        scatter(1:M, lambda_orig, 200, 'd', 'MarkerEdgeColor', 'black', ...
            'LineWidth', 2);
        hold on;
        scatter(1:length(D), D, 100, 'o', 'MarkerEdgeColor', 'red', ...
            'LineWidth', 2.5);
        scatter(1:M, lambda, 50, 'x', 'MarkerEdgeColor', 'blue', ...
            'LineWidth', 2);
        hold off;
        title({'Minimum eigenvalues of L\_sym', [' K=', num2str(K)], ...
            [' N\_CLUSTERS=', num2str(N_CLUSTERS)]});
        legend('Computed with eigs function', ...
            'Computed with spectralcluster function', ...
            'Computed with power method with deflation', ...
             'Location', 'northwest');
    end
end
```