A MATLAB Spectral Clustering Implementation

Computational Linear Algebra for Large Scale Problems - Homework I

Giulio Zabotto, s279487@studenti.polito.it

February 2022

1 Graph theory

Proposition 1. *The normalized Laplacian matrix $L_{sym} \in \mathbb{R}^{n\times n}$ is positive semi-definite and have n non-negative real-valued eigenvalues $0 = \lambda_1 \leq \cdots \leq \lambda_n$.*

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, W)$ be a weighted undirected graph. Let the D be the diagonal degree matrix $D = Id$, where $d = W\mathbf{1}$. It is easier to prove the proposition above if we consider the case of a *simple* graph, i.e., an undirected graph with $w_{ij} \in W$ such that

$$w_{ij} = \begin{cases} 1 & \text{if } (i,j) \in \mathcal{E} \\ 0 & \text{otherwise} \end{cases}$$

The Laplacian is defined as

$$L = D - W.$$

Given a vector $x \in \mathbb{R}^n$, the quadratic form of the Laplacian is

$$\begin{aligned} x^T Lx &= x^T(D-W)x \\ &= x^T Dx - x^T Wx \\ &= \sum_{i=1}^{n} d_i x_i^2 - \sum_{(i,j)\in\mathcal{E}} 2x_i x_j \\ &= \sum_{i=1}^{n} \sum_{(i,j)\in\mathcal{E}} x_i^2 - \sum_{(i,j)\in\mathcal{E}} 2x_i x_j \\ &= \sum_{(i,j)\in\mathcal{E}} (x_i^2 + x_j^2 - 2x_i x_j) \\ &= \sum_{(i,j)\in\mathcal{E}} (x_i - x_j)^2 \end{aligned} \tag{1}$$

Now, if we remove the assumption of simple graph and we consider the normalized Laplacian, the only difference in (1) is

$$x^T L_{sym} x = \sum_{(i,j)\in\mathcal{E}} \frac{w_{ij}}{d_i}(x_i - x_j)^2$$

Proposition 2. *Let $\mathcal{G}$ be and undirected graph with non-negative weights. Then the multiplicity m of the eigenvalue 0 of $\mathbf{L}_{sym}$ equals the number of connected components $A_1, ..., A_m$ in the graph. The eigenspace of the eigenvalue 0 is spanned by the vectors $\mathbf{D}^{\frac{1}{2}}\mathbb{1}_{A_i}$, where $\mathbb{1}_{A_1}, \mathbb{1}_{A_2}, ..., \mathbb{1}_{A_m}$ are the indicator vectors. Prove the proposition above for the case $m = 1$.*

The proposition can be rephrased as proving that if and only if $\mathcal{G}$ is connected, then its second smallest eigenvalue $\lambda_2 > 0$. First, assume that $\mathcal{G}$ is disconnected, that it has at least two connected component A_1 and A_2. It is easy to figure that these connected components are smaller graphs on their own. Hence, we can arrange the indices of the vertices such that the laplacian of the whole graph is

$$L_{\mathcal{G}} = \begin{bmatrix} L_{A_1} & 0 \\ 0 & L_{A_2} \end{bmatrix}$$

Hence, $L_{\mathcal{G}}$ has at least two eigenvectors in the kernel:

$$\begin{bmatrix} \mathbf{0} \\ \mathbf{1} \end{bmatrix}, \quad \begin{bmatrix} \mathbf{1} \\ \mathbf{0} \end{bmatrix}.$$

On the other hand, if $\mathcal{G}$ is connected, and that x is an eigenvector of $L_{\mathcal{G}}$ of the kernel. Since $L_{\mathcal{G}}x = 0$, then

$$x^T L_{\mathcal{G}} x = \sum_{(i,j)\in\mathcal{E}} (x_i + x_j)^2 = 0$$

which means that for every connected edge (i, j), $x_i = x_j$. This applies for every pair of vertices connected by a path – and a path connecting each pair of vertices always exists, for the single connected component hypothesis. Therefore, x must be a constant vector, $x = c\mathbf{1}$. So, we can conclude that the kernel of a connected graph has unitary dimension.

Next, considering the normalized Laplacian, we show that $D^{\frac{1}{2}}\mathbf{1} \in ker(L_{sym})$.

$$\begin{aligned} L_{sym}D^{\frac{1}{2}}\mathbf{1} &= D^{-\frac{1}{2}}LD^{-\frac{1}{2}}D^{\frac{1}{2}}\mathbf{1} \\ &= D^{-\frac{1}{2}}(D - W)\mathbf{1} \\ &= D^{\frac{1}{2}}\mathbf{1} - D^{\frac{1}{2}}W\mathbf{1} \\ &= \sum_i^n \sqrt{d_i} - \sum_i^n \frac{d_i}{\sqrt{d_i}} = 0 \end{aligned}$$

2 Build adjacency matrix W

We are given a set of observation $X = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$, where $n = 900$. The want to cluster these data implementing the spectral clustering method. First we need to compute the Laplacian yielded by the graph built onto this set of data. The graph is defined by its adjacency matrix W, hence, we need to compute it. We

Table 1: Top-5 largest eigenvalues computed with MATLAB *eigs* function

	λ_1	λ_2	λ_3	λ_4	λ_5
$\mathbf{W}_{13}$	-1.1810e-18	4.3077e-4	0.0014	0.0043	0.0060
$\mathbf{W}_{40}$	-1.5669e-17	0.0055	0.0364	0.0523	0.0803

arbitrary impose that the weight of each edge is given by the *k-nearest neighbors*, algorithm, $k \in \{13, 40\}$, and the following Gaussian kernel as similarity function

$$s_{ij} = exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||}{\sigma^2}\right),$$

where $\sigma = 1$.

Figures 1a and 1c shows the sparsity patterns for the respective adjacency matrix $\mathbf{W}_{13}$ and $\mathbf{W}_{40}$, while figures 1b and 1d shows their graph plot.

From the images we can guess the number of cluster one may expect. In figure 1b we can clearly distinguish three clusters. Figure 1d is less obvious: the number of clusters ranges from two to four if we either consider the larger cluster on the right as a single one or three separate ones.

3 Assemble the symmetric Laplacian

As we already assembled the weight matrix $\mathbf{W}$, we only need to compute the diagonal degree matrix $D^{\frac{1}{2}}$. We directly compute the diagonal entries as

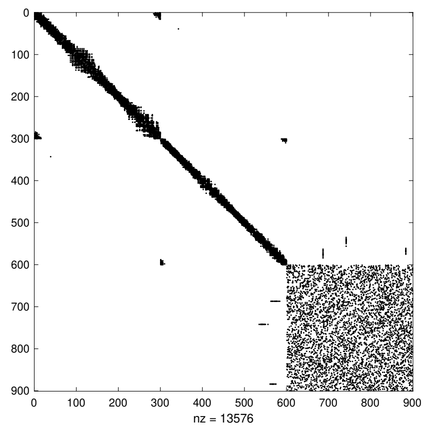
$$d_{ii} = \left(\sum_{j=1}^{n} w_{ij}\right)^{-\frac{1}{2}} \qquad i = 1, 2, ..., n.$$

The next task requires us to compute the five smallest eigenvalues of the computed Laplacian. As we had proven in the **proposition 2**, the expected multiplicity of the null eigenvalue equals the number of connected components of the graphs; hence, as the figures 1b and 1d show, the multiplicities should be three for $k = 13$ and ranging from two to four for $k = 40$. The matlab-native implementation for the computation of the smallest eigenvalues yields the eigevalues in 1 in order of ascending absolute magnitude.

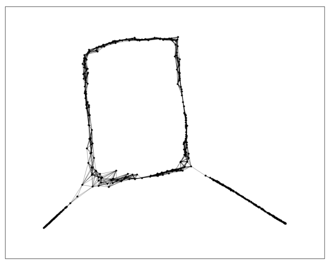
The calculation are in IEEE double precision, which means that the machine epsilon $\epsilon_{machine}$ is approximately 1.1×10^{-16}. For the 13-NN case, the first eigenvalue is within the machine precision, hence it is a zero. All the remaining are not zero.

Yet, we expected to find one more non-zero result does not satisfy our expectation, as the second and third eigenvalues are not zero. Still we have to take into consideration that our laplacian matrix may be ill-conditioned. The computation of condition number yields

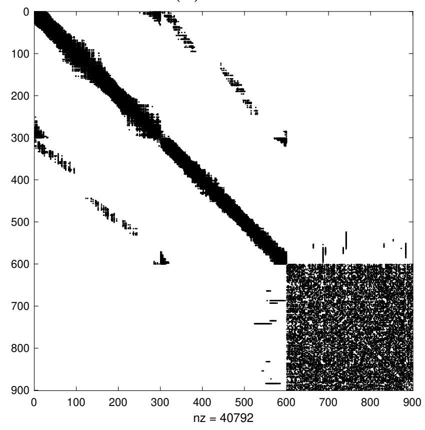
$$\kappa(L_{sym}) = \|L_{sym}\|_2 \left\|L_{sym}^{-1}\right\|_2 = 1.6176e^{18}$$



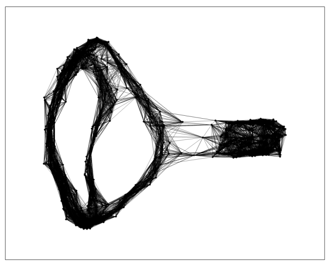
(a) $\mathbf{W}_{13}$



(b) $\mathbf{G}_{13}$



(c) $\mathbf{W}_{40}$



(d) $\mathbf{G}_{40}$

Figure 1: Left-side: sparse matrices inspection graphs; a black dot is a non-zero entry. Right-side: graph plot yielded by the respective weight matrices on the left

L_{sym} is ill-conditioned. This might explain why the eigenvalues are so inaccurate. Generally, eigenvalue computation of symmetrical matrices is a well-conditioned problem, yet, the algorithm may not preserve the conditioning.

We overcome the *impasse* employing the *eigengap* heuristic: we set the number of clusters to the m-th eigenvalue that maximize the gap between successive eigenvalues in ascending order.

$$m = \max\{\lambda_{i+1} - \lambda_i\}_{i=1}^{n}$$

With this reasoning, the $m_{13} = 3$, and $m_{40} = 2$.

Now that we know what to expect, we implement our own algorithm to compute the top-5 smallest eigenvalues.

4 Power iteration with Wielandt deflation

The *power iteration* method is an iterative method to compute the largest eigenvalue of a given matrix. Yet, we need to compute the smallest, not the largest eigenvalues, but we exploit the following relations.

$$L_{sym} = I - B$$

where $B = D^{-1/2}WD^{-1/2}$. Then, by the definition of eigenvalue and eigenvector,

$$\begin{aligned} L_{sym}v &= \lambda v \\ (I - B)v &= \lambda v. \end{aligned}$$

For better conditioning, we want B to be diagonally dominant, so we add constant $\mu = \max_i \sum_j b_{ij}$ to B. Therefore,

$$\begin{aligned} (I - B)v + \mu v &= \lambda v + \mu v \\ (I - B + \mu I)v &= (\lambda + \mu)v \\ (B + \mu I)v &= (1 - \lambda + \mu)v \end{aligned}$$

Let us define $B_{mod} \coloneqq B + \mu I$; its eigenvalue is $\beta = 1 - \lambda + \mu$. Hence, we can compute the largest eigevalues of B_{mod} and transform them into those of L_{sym} according to

$$\lambda = 1 - \beta + \mu. \tag{2}$$

The power iteration method computes only the largest eigenvalue, but we the next four as well. We can still make use of it with the integration of a deflation technique. The deflation consists of removing an eigenvector from the eigenspace of the input matrix A, thus obtaining a *deflated* matrix B.

Suppose that the eigenvalues of a matrix A are $\lambda_1, \lambda_2, ...\lambda_n$, and the the largest eigenvalue λ_1 has multiplicity 1; their associated eigenvectors are $v_1, v_2, ..., v_n$;

suppose that x is a vector such that $x^* v_1 = 1$. Then, it can be shown that the matrix

$$B = A - \lambda_1 v_1 x^*$$

has the same eigenvalues of A but $\lambda_1 = 0$ and its eigenvectors are $v_1, w_2, ..., w_n$, with

$$w_i = (\lambda_i - \lambda_1) w_i + \lambda_1 (x^* w_i) v_i \quad i = 2, 3, ..., n$$

We implemented the *Wielandt* deflation, which impose

$$x = \frac{1}{\lambda_1 v_1(1)} A(i, :)$$

where $v_1(1)$ is first component of v_1 and $A(i, :)$ is the i-th row of A.

Applying this method recursively for five times we obtain the top-5 largest eigenvalues. The pseudo-code for this algorithm is the following.

```
lambda, v = powerIteration(A, tol, nmax, x)
for k = 2 to 5
    A = deflate(A, lambda, v)
    lambda, v = powerIteration(A, tol, nmax, x)
    store lambda
```

As the code shows, the power iteration method needs an input vector x, which is randomly generated and two parameters as stopping condition. The power iteration stops if either the difference $\lambda_t - \lambda_{t-1} < tol$ or the number of iteration exceeds *nmax*. The values this parameters are in table 2.

Table 2: Power iteration with Wielandt deflation parameters. The *tol* is the error tolerance, while *nmax* is the maximum number of iteration.

parameter	value
tol	10^{-8}
nmax	5000

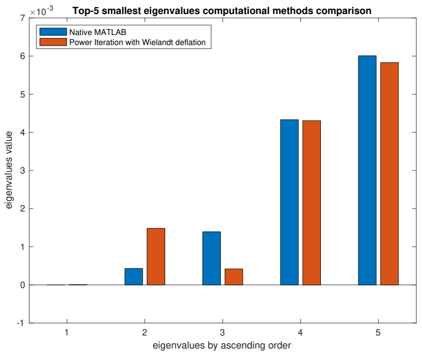
The maximum number of iteration is high because the power iteration method accuracy scales linearly (quadratically for the eigenvalue) according to the relations [TB97]

$$\left\| \tilde{x}^{(k)} - v_1 \right\| = O\left(\left| \frac{\lambda_2}{\lambda_1} \right|^k \right)$$

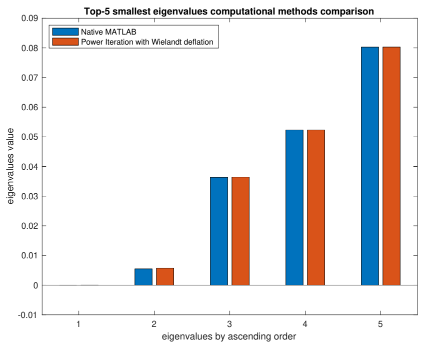
$$|\tilde{\lambda}^{(k)|-\lambda_1} = O\left(\left| \frac{\lambda_2}{\lambda_1} \right|^{2k} \right)$$

where k is the iteration index, $\tilde{x}^k$ is the computed eigenvector, $\tilde{\lambda}$ is the computed eigenvalue at iteration k. On the other hand, the tolerance is higher than expected — table 1 shows that the precision necessary to distinguish the eigenvalues is 10^{-4} at most, while are are using the square of that precision. Indeed, when using $tol = 10^{-4}$ the algorithm is unstable. The deflated matrices are not ill-conditioned, hence, the algorithm must be unstable.

Figures 2a and 2b show that the results are comparable with those obtained by the MATLAB native implementation.



(a) L_{sym}^{13} eigenvalues



(b) L_{sym}^{40} eigenvalues

5 The eigenvectors of L_{sym}

The previous algorithm yielded us the largest 5 eigenvalues of B_{mod}. To get those of L_{sym}, we simply plug them in equation (2). Next, we need the eigenvectors of L_{sym}. We only those related to the smallest m eigenvalues. To compute them we employ the *Rayleigh quotient iteration* method, alias the inverse power method with shift starting from an approximation of an eigenvector. It can be shown that the algorithm convergence is cubic, indeed, if q and λ an eigenvector and an eigenvalue of A and the starting eigenvector $v^{(0)}$ is close to the eigenvector q, then

$$\left\| v^{(k+1)} - q \right\| = O(\left\| v^{(k)} - q \right\|^3)$$
$$|\lambda^{(k+1)} - \lambda| = O(|\lambda^{(k)} - \lambda|^3).$$

Thus, in a few iterations we should get an accuracy within the tolerance $10e-4$. Indeed, in the experiments, the number of iterations required to satisfy such a tolerance threshold ranged from two to six. The parameters used for this task are showed in table 3.

Since B_{mod} and L_{sym} share the same eigenspace, we decide to apply the algorithm on B_{mod}, as the their eigenvalues are non-zero and the condition number of B_{mod} lower by is orders of magnitude.

6 Comparison of algorithms

The eigenvectors computed with the Rayleigh quotient iteration methods end up in the matrix U; its column vectors are then normalized.

Table 3: Rayleigh quotient iteration method parameters. The *tol* is the error tolerance, while *nmax* is the maximum number of iteration.

parameter	value
tol	10^{-4}
nmax	10

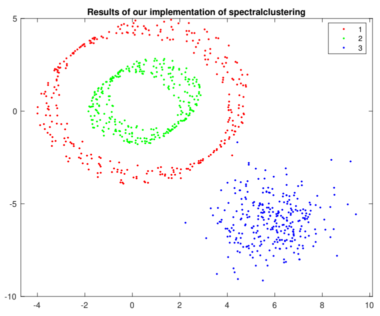
Finally, we cluster points of the orginal dataset trasformed in the space of U with the *kmeans* algorithm as implemented by the homonymous MATLAB function. Figures 3a and 3d shows the results yielded by our algorithm, while figure 3b and 3d those obtained with MATLAB function *spectralcluster*.

We can clearly see from the figures our results are the same as those yielded by the MATLAB version.

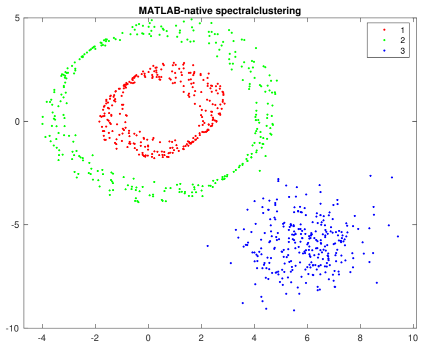
We end this analysis comparing the spectral clustering with the k-means applied to the original, non-transformed, dataset X. The limitation of this technique are showed in figures 4a and 4b. When the cluster are non globular, the k-means fails. On the other hand, the spectral clustering shines. This method is both synthetic, as it requires a number of features at most equal to the number of clusters, and powerful in discriminating, as the eigenvectors of a symmetric matrix are always orthogonal.

References

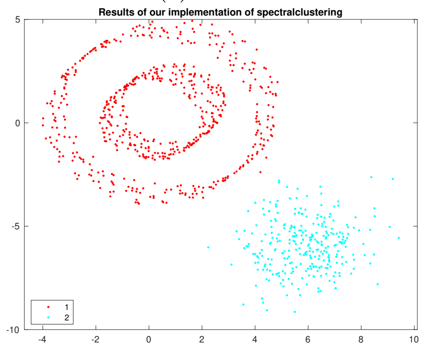
[TB97] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra.* SIAM, 1997.



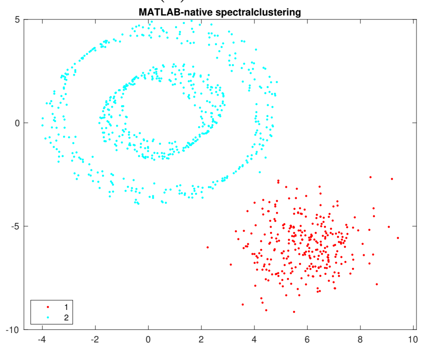
(a) $k = 13$



(b) $k = 13$

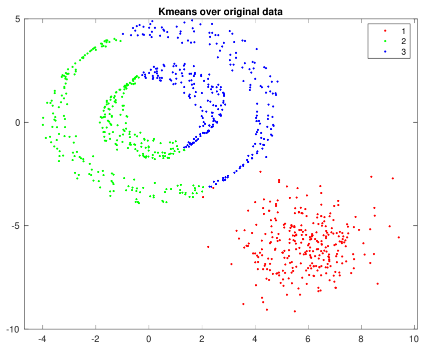


(c) $k = 40$

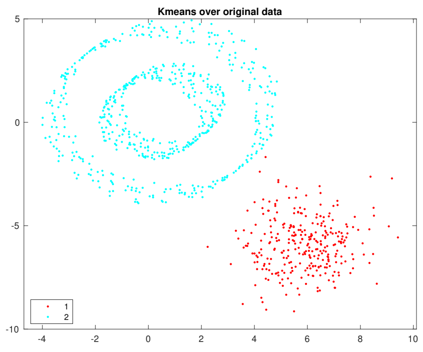


(d)

Figure 3: Left-side: scatter plot of our implementation of spectral clustering; Right-side: scatter plot of the MATLAB-native spectral clustering



(a) $k = 13$, $m = 3$



(b) $k = 40, m = 2$

Figure 4: Scatter plot of the original data labeled according to the k-means algorithm