Ward, David
Matrikelnummer: 2005661
Studiengang: Wirtschaftsmathematik

04.07.2023

Prof. Dr. André Uschmajew
Mathematical Data Science
Institut für Mathematik an der Universität Augsburg
Sommersemester 2023

Bachelor Thesis

Spectral Clustering: Basic Concepts and Applications

by

David Ward

Contents

1. Introduction

Enormous amounts of data are collected and analysed every day with the majority of decisions made using these results. Ronald Coase remembers one of his most famous quotes as being "Torture the data enough, nature will always confess" in [3]. The economics noble prize winner seeks for answers in data and is backed up by the current expansion of data analytics. On a small scale our personal choices are decided by the data that our brains have collected throughout our lives. With technology evolving faster than politics can adapt, research and development will dictate the world of tomorrow posing both chances and challenges. Data will become one of the most important resources and hold a lot of power.

Therefore, one of the most in demand tools are cheap algorithms that give first impressions on massive data sets. One of these first impressions is the grouping or clustering of data. It divides the information into several clusters where all corresponding data is assumed to be relatively similar. The similarity can be defined in many different ways and is strongly dependent on the subject of the analysis.

Throughout this thesis we will focus on the spectral clustering algorithm. This approach translates data into a mathematical graph with a matrix representation. The idea of an ideal clustering can be defined as a calculation of the cost of separating the graph into the clusters. A small adaptation leads us to an eigenvalue problem that presents a relatively low-cost solution using the right methods. Generally this thesis poses an overview about spectral clustering based upon the tutorial [18]. Additionally, it presents a fully developed Python code and places a major focus on the algebraic connectivity of a graph.

We will start the thesis in section 2 recapitulating some notational standards and important results of linear algebra and graph theory. Some basic level understanding of mathematical graphs, matrices and eigenvalues is assumed and we will therefore not include foundational details. Afterwards we will define the general setting and our objective based on several examples demonstrated throughout our work. In section 4 we will begin by using the k-means algorithm and then illustrating its limitations and possible adaptations.

In section 5 we will introduce the most important tools of spectral clustering and mathematically justify the approach. After presenting the algorithm we will compare the obtained results to our earlier presented solution of k-means. Ultimately we will deliver additional cause on the functionality of spectral clustering by the means of algebraic connectivity and perturbation theory.

In section 6 we consider the improved normalized clustering approach and drag some connections to the unnormalized case. We will then once more clarify the underlying concepts and justify it by the means of a random walk on a graph. To complete our findings we refer to some of the connected fields of spectral clustering and present the finalised code that produced all mentioned results regarding our examples.

2. Basics

Spectral clustering is based mostly upon two fields of mathematics: graph theory and linear algebra. In the following we will summarize the most important concepts and notations regarding this thesis. For the majority of the concepts we will provide references for a more detailed analysis.

2.1. Linear Algebra

Firstly, we will introduce notational standards that are used throughout this thesis. For a matrix $A \in \mathbb{R}^{n\times n}$ the transposed matrix is denoted as A^T while the trace is referred to as $\mathrm{tr}(A)$. The standard basis $e_i, \quad i \in \{1, ..., n\}$ of the n-dimensional vector space consists of the vectors only having a 1 in the i-th coordinate. The constant vector $e \in \mathbb{R}^n$ represents $e = \begin{bmatrix}1 & 1 & ... & 1\end{bmatrix}^T$. During matrix computations we commonly encounter double sums that have the same maximum index. We are merging those into

$$\sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} = \sum_{i,j=1}^{n} a_{ij}.$$

The primary concept behind spectral clustering is a real-valued eigenvalue problem. Generally an eigenvector occurs if a vector x remains on its own span during a linear transformation. The eigenvector x is only squished or stretched by a factor λ. That scalar is the corresponding eigenvalue. If the linear transformation is represented as a matrix $A \in \mathbb{R}^{n\times n}$ we can write this concept with the eigenvector $x \in \mathbb{R}^n \setminus \{0\}$ and the eigenvalue $\lambda \in \mathbb{R}$ as

$$Ax = \lambda x \quad \Leftrightarrow \quad (A - \lambda I)x = 0. \tag{1}$$

For this to be true the determinant of $A - \lambda I$ has to be 0, which subsequently leads to the computational approach of solving the newly formed n-dimensional polynomial based on $\det(A - \lambda I) = 0$. The roots of the so called characteristic polynomial are the eigenvalues of our matrix A. Especially inside of an environment with large matrices this approach is highly inefficient and we will point out different methods in section 5.4.

Any pair of x and λ that satisfies equation (1) will be called an eigenpair (λ, x) of the matrix A. For the spectral clustering algorithm, we mostly work with symmetric, positive semi-definite matrices. This makes the search for eigenvalues easier because those matrices can only contain real-valued, non-negative eigenvalues. Therefore, we can order the eigenvalues

$$\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n$$

in ascending order. From now on we assume the eigenvalues ordered accordingly. For a thorough explanation of the mentioned concepts we refer to [8]. For a more visual understanding we highly recommend watching the video series "Essence of linear algebra" by 3Blue1Brown [1].

Many of the results in this thesis are directly linked to optimization problems which are solved by the eigenvectors of a matrix. Firstly, we require the Courant-Fischer theorem mentioned in [10][p.179].

Theorem 2.1 (Courant-Fischer)**.** *For a symmetric matrix $A \in \mathbb{R}^{n\times n}$ with eigenvalues $\lambda_1 \leq ... \leq \lambda_n$ and X_i being the set of all i-dimensional subspaces of $\mathbb{R}^n$ then*

$$\lambda_i = \max_{X\in X_{n-i+1}} \min_{x\in X, x\neq 0} \frac{x^T Ax}{x^T x}.$$

Using the characteristics of the well known Rayleigh quotient mentioned in [10][chapter 4.2] leads to this display for each eigenvalue of A:

$$\lambda_k = \min_{\substack{x\in\mathbb{R}^n \\ x\perp x_1,x_2,...,x_{k-1}}} \frac{x^T Ax}{x^T x} = \min_{\substack{x\in\mathbb{R}^n, x^T x=1 \\ x\perp x_1,x_2,...,x_{k-1}}} x^T Ax. \tag{2}$$

The second equality holding as the Rayleigh quotient is untouched by linear scaling of x. Therefore we can norm it without impacting the result.

The second required optimization problem is related to the trace of a matrix. For a detailed introduction and proof we refer to [9]. The standard trace minimization problem is given by

$$\min_{H\in\mathbb{R}^{n\times n}} \operatorname{tr}(H^T AH) \quad \text{s.t.} \quad H^T H = \mathbb{I}. \tag{3}$$

This problem is solved by the matrix H containing the eigenvectors of A in its columns ordered by their associated eigenvalue in ascending order.

2.2. Graph Theory

During the process of spectral clustering, we will construct a mathematical graph. Most basic concepts we are referring to can be found in [4]. A graph $G = (V, E)$ consists of a set of vertices V and a set of edges E. The vertices in V are typically numbered from 1 to $|V|$. For our cause we will use the weight matrix W or sometimes more specifically W_G for most of the time to represent the edges. For each two vertices $u, v \in V$ the corresponding weight matrix entry w_{uv} represents the weight of the edge between u and v. Generally, weights are defined for each pair of vertices while also being symmetric. For our cause the weights represent similarity and therefore we assume them to be between 0 and 1. We do not allow loops on a graph resulting in the weight between the same vertex $w_{vv} = 0$. This might seem unfit considering our idea of similarity, but it benefits us more than it does harm. To summarize, we are looking at an undirected, weighted and simple graph.

Throughout this thesis we will stumble upon unweighted graphs from time to time. They can be interpreted as a special case with the weight between vertices only having a value of 0 or 1. A graph G contains the edge $e_{uv} \in E$ if the corresponding weight matrix entry $w_{uv} > 0$.

The degree of a vertex $v \in V$ is defined as $d_v := \sum_{u\in V} w_{uv}$ or alternatively if the vertices v_i are numbered d_i which sums up the weight of all the adjacent edges.

When capturing the size of a graph G we can use the order $|V|$ which represents the total number of vertices. When mentioning a graph we always consider its order to be n if not mentioned otherwise. Alternatively $\operatorname{vol}(G)$ is the sum over all degrees for each vertex: $\operatorname{vol}(G) := \sum_{v\in V} d_v$.

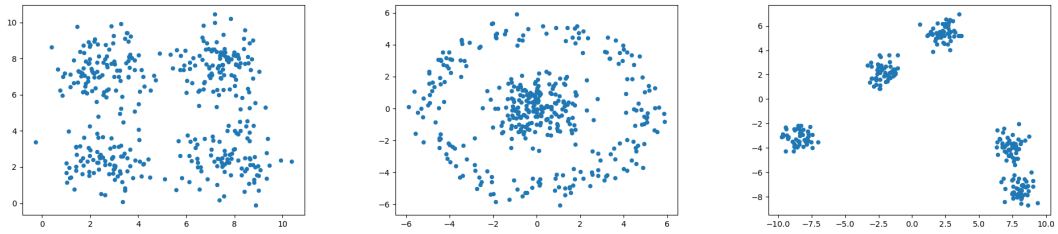
A subgraph G' is defined by a subset $V' \subset V$ carrying over all edges where both adjacent vertices are in V'. The complementary graph $\overline{G} = (V, \overline{E})$ is defined on the same vertex set V with $W_{\overline{G'}} = J - W_G - \mathbb{I}$, with J representing the matrix having a 1 in every cell. A partition of G divides the vertex set into subgraphs $A_i = (V_i, E_i)$ with the union of vertices $\bigcup V_i = V$.

G is connected if all vertices are reachable amongst each other. The connected components of G are a partition with each subgraph being the biggest possible connected subgraph of G containing its vertices. For the unweighted case the complete graph K_n contains all of the possible edges on n vertices.

3. Objective and Examples

To starting off let us set the general objective that we are trying to achieve and with that define the ideal solution. Our starting point is any type of dataset that we want to separate into clusters. More specifically we want data points that are reasonably similar in the same cluster and the ones that are dissimilar enough in different clusters. This is a rather abstract task, but we will develop methods to mathematically capture this optimization problem in section 5.2.

For a visualization of the methods and a better comparison of the clustering concepts we utilize three examples. To have it visually accessible, all of them will only include data points that are located in the $\mathbb{R}^2$ vector space. More abstractly this can be thought of as a data set where we only measure the similarity based upon two numerical attributes which are the x and y value of the said data point.



(a) (b) (c)

Figure 1: three examples to test our clustering methods.

- Example (1a) consists of 400 data points divided into four normal distributions. Both axes contain two distribution centres located at 5/2 and 15/2 with a standard deviation of 1. The chance of a data point appearing in either of the four clusters is equally distributed.
- Example (1b) consists of 400 data points distributed equally into either a normal distribution with a centre $(0, 0)$ or another normal distribution around a random point on the circle with the centre $(0, 0)$ and a radius of 5. Both have a standard deviation of 1.

- Example (1c) consists of five dense blobs created by the makeblobs function of scikit-learns[2].

With the human eye and our knowledge about the origin of the distribution we can easily decide on each of these data the ideal clustering that we are seeking. As for Example (1a) we want to have a cluster for each of the normal distributions. For Example (1b) we want two clusters, one containing all data of the outer ring while the other cluster refers to the centre. Lastly, for Example (1c) we want one clusters for each of the data blobs.

4. K-means

Before starting our spectral clustering approach another clustering algorithm will be implemented. K-means is much stricter in its input but considering our two dimensional space it suffices. The Euclidean distance between points represents its idea of similarity. Therefore, we are trying to cluster data points on a metric space with the objective that the distance of points inside of the same clusters is low while for data points of different clusters the distance is rather high. We do not need to normalize the distance for the k-means approach. Later when working with spectral clustering we will make use of k-means once again.

4.1. General Construction

The general structure of k-means can be formulated in five steps.

1. Beforehand supply k-means with the number of clusters it is supposed to create. This is our k represented in the name. This already expects us to have a basic understanding of the desired result and the underlying data which we discuss in section 4.2.

2. For each of the k clusters assign a random starting point in our data space. The starting point is called the cluster centre and each cluster is defined by its centre.

3. For each data point compute the distance to every cluster centre and assign it to the cluster that has the lowest distance. By the end of this step all points belong to one of the clusters.

4. Determine new cluster centres based upon the mean of all data points now associated to that cluster.

5. Return to the third step until a certain criterion is met. Mostly this discontinuation happens if the cluster centres are stagnant, however, a threshold is also a possibility if there are massive amounts of data in play.

```
def k_means(X, k):
    N,n = np.shape(X)
    centers = rngCenters(X, n, k)

    for i in range(maxSteps):
        old_centers = centers
        labels = pairwise_distances_argmin(X, centers)
        centers = newCenters(X, k, labels, centers, n, N)
        if np.all(old_centers == centers):
            break
    return (labels, centers)

def newCenters(X, k, labels, centers, n, N):
    new_centers = np.zeros((n,k))
    cnt = np.zeros(k)
    for i in range(N):
        new_centers[:,labels[i]] += X[i]
        cnt[labels[i]] += 1
    for j in range(k):
        if cnt[j] == 0:
            new_centers[:,j] = centers[j,:].T
            cnt[j] = 1
    new_centers = np.divide(new_centers, cnt)
    return new_centers.T

def rngCenters(X, n, k):
    uborder = np.max(X,axis=0)
    lborder = np.min(X,axis=0)
    centers = np.random.random((k,n))
    centers = np.multiply(centers, uborder-lborder)
    centers = np.add(centers, lborder)
    return centers
```

Figure 2: Python Code for the k-means algorithm

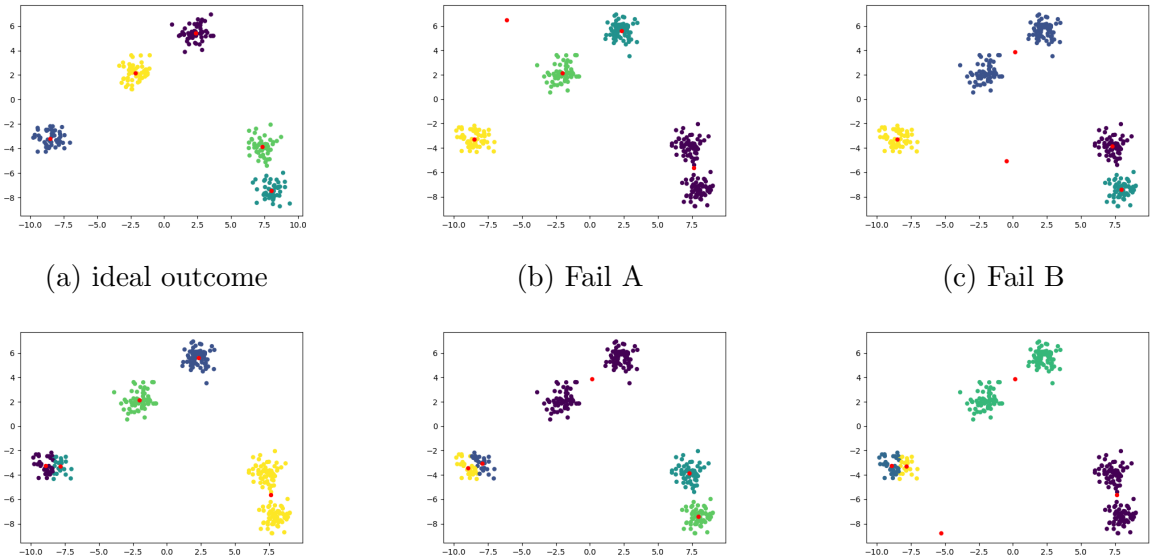
The k-means implementation makes use of the pairwise_distance_argmin function mentioned in [2] to immediately return the label for the cluster centre closest to each data point. The initialization of the cluster centres is totally random only restricted by the extremal values on each dimensional axis for all data points.

4.2. Limitations and Enhancements

After generally being familiar with the k-means algorithm we should point out a couple of issues emerging and propose possible improvements.

The first issue is the non-deterministic clustering result. Running k-means a dozen of times on the same example can lead to a variety of different results as presented in Figure

3.



(a) ideal outcome (b) Fail A (c) Fail B

(d) Fail C (e) Fail D (f) Fail E

Figure 3: different results of k-means on Example (1c
).

The source of this problem is the random assignment of the cluster centres. Specifically in this case, this mostly occurs due to the high distances between the blobs in comparison to their size. This results in many cluster centres randomly being placed into the same space between clusters. Subsequently, this can cause a cluster to have no data points which are closest to their cluster centre. Ultimately, this forces a search for less clusters than desired.

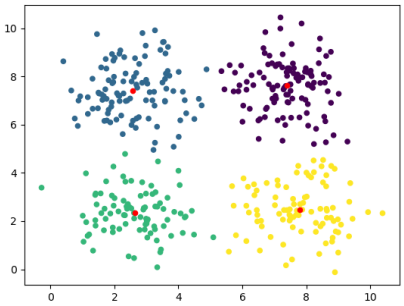
Another issue resulting due to similar reasons is when two cluster centres are close to the same data blob. If all other blobs are relatively far away, this results in them splitting the blob as they are their only connected data points. This once again leads to a lack of one cluster for the rest of the graph. All in all, this causes significant problems in trusting a singular result of k-means.

The simplest solution to this issue is running k-means multiple times and numerically value each cluster distribution. By comparing the resulting values of how well the clustering was evaluated we can choose the highest valued partition. This evaluation is done mostly by calculating the sum of the distances between each points and their respective cluster centre.

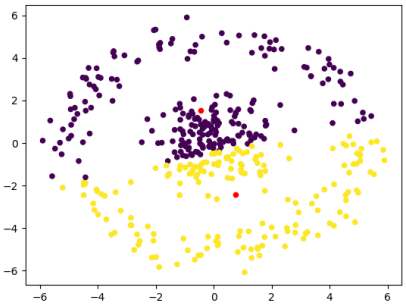
Another solution is a more intelligent approach of "randomly" assigning cluster centres. There are a handful of options to create more favourable starting positions including placing the centres on the same position as a random data points, relocating cluster centres which no data point connections or even more complicated ideas based upon the individual data set.

In contrast to that for our other two examples k-means gives us relatively constant

results irrespective of the random cluster centres distribution. These results are displayed in figure 4 and point out the key issue regarding k-means.



(a) k-means works brilliantly on this example.



(b) k-means does not yield the desired result.

Figure 4: results of k-means on Example (1a) and Example (1b).

Due to the convexity of distances the possible geometry of cluster k-means can distinguish into is restricted. A prominent case for this issue is seen in Example (1b). K-means has no ability to find the ideal partition disregarding the random starting cluster centres. Even allowing ourselves to place the cluster centres at any location, quickly yields in the realization that there are no cluster centres to make k-means work on this example.

Finishing up on a positive note, Example (1a) is easily solvable for k-means. With the presented adaptations k-means shows consistent correct results for Example (1c). Therefore with some additional numerical cost we can increase our chances of getting the desired result massively and make k-means a powerful tool.

5. Unnormalized Spectral Clustering

Following the mentioned unavoidable issue of k-means we want to turn our attention to another algorithm. The spectral clustering approach provides better results while also being able to commence any given data. This among others assures its functionality on our previously used examples in Figure 1. For now our starting input for the algorithm changes to a graph or a representational object that uniquely defines a graph (e.g. weight matrix). Our idea of similarity which was previously given by the Euclidean distance is more customizable and represented by the edge weights of our graph.

The idea of an ideal partition changes slightly in the setting of a graph. We are trying to find a partition of the graph such that the sum of weights in between each cluster, which are the edges cut in the process of partitioning is minimized while also having a high sum of edge weights inside each subgraph.

5.1. The Graph Laplacian

To effectively work on the relatively complex structure of a graph we already introduced the weight matrix. Building upon this idea we can construct the most important piece of information for our clustering algorithm, the graph Laplacian matrix.

Definition 5.1. *For a graph G the degree matrix $D \in \mathbb{R}^{n\times n}$ is defined as*

$$d_{ij} := \begin{cases} d_i, & \text{if } i = j \\ 0, & \text{if } i \neq j \end{cases}.$$

The degree of each vertex is displayed on the representative diagonal element of D. All other entries of the matrix are 0. With additional help of the weight matrix W we can define the graph Laplacian matrix as

$$L := D - W.$$

Note that the entries of both matrices are not interfering with each other. The graph Laplacian combines the information about negative edge weights while also displaying the degrees of all vertices in one matrix.

Taking a close look at $L \in \mathbb{R}^{n\times n}$ we summarise the following characteristics:

Lemma 5.2 (Properties of the graph Laplacian L)**.**

1. *L is symmetrical.*
2. *For any given $x \in \mathbb{R}^n$ the quadratic form of L is*

$$x^T Lx = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2$$

and with that L is positive semi-definite.

3. *L has only real eigenvalues which are non-negative.*

4. *$(0, e)$ is always an eigenpair of L.*

Proof.

1. The matrices W and D do not interfere with each other. As D is only responsible for the diagonal entries it suffices to show that W is symmetrical. Knowing that we are only considering undirected graphs the weight function is symmetrical.

2. A straightforward computation using the binomial theorem yields

$$\begin{aligned} x^T Lx = x^T(D-A)x = x^T Dx - x^T Ax = \sum_{i=1}^{n} d_i x_i^2 - \sum_{i,j=1}^{n} x_i x_j w_{ij} \\ = \frac{1}{2}(\sum_{i=1}^{n} d_i x_i^2 - 2(\sum_{i,j=1}^{n} x_i x_j w_{ij}) + \sum_{j=1}^{n} d_j x_j^2) \\ = \frac{1}{2}(\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} x_i^2 - 2(\sum_{i,j=1}^{n} x_i x_j w_{ij}) + \sum_{j=1}^{n}\sum_{i=1}^{n} w_{ij} x_j^2) \\ = \frac{1}{2}\sum_{i,j=1}^{n}(w_{ij}x_i^2 - 2w_{ij}x_i x_j + w_{ij}x_j^2) = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(x_i^2 - 2x_i x_j + x_j^2) \\ = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2. \end{aligned}$$

3. Follows directly from (1.) and (2.).

4. A simple investigation of each row of L concludes that the sum is always 0. The diagonal value is the degree of the connected vertex while all other entries in that row are the negative weights off adjacent edges. If we multiply L with the vector e the result is the vector containing the sum of each row. Therefore $(0, e)$ is an eigenpair of L.

□

On a quick side note, we could allow loops in our graph and with that the rather intuitive similarity of 1 between the same vertex. The resulting graph Laplacian would not change. If vertex v would gain an additional loop with the weight w_{vv} then d_v would be raised equally leaving the affected entry of the graph Laplacian l_{vv} unchanged as it consists of $l_{vv} = d_v - w_{vv}$.

Throughout the following sections we will work massively with the eigenvalues of L. Mostly they are referred to as the Laplacian eigenvalues and we use λ_i or $\lambda_i(G)$ if clarification on the origin of the eigenvalue is needed. Additionally we assume

$$0 = \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n.$$

The two most important eigenvalues are the non-trivial extremal eigenvalues λ_2 and λ_n. Both of them are discussed in detail in section 5.7 but for now only one correlation between a graphs connectivity and its second smallest Laplacian eigenvalue λ_2 is needed.

Lemma 5.3. *Considering a graph G and its graph Laplacian L. The second smallest eigenvalue λ_2 is positive if and only if G is connected.*

Proof. Assuming G is a connected graph and x is a corresponding eigenvector to Laplacian eigenvalue $\lambda_2 = 0$ that is linearly independent from e. Multiplying the resulting eigenvalue equation from the left with x^T and with the help of Lemma 5.2(2) we conclude

$$Lx = 0 \quad \Leftrightarrow \quad x^T L x = \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2 = 0.$$

For this to hold all summands have to be 0. That means that for any adjacent vertices v_i and v_j, meaning $w_{ij} > 0$ the corresponding entries in x have to be identical ($x_i = x_j$). With the assumption that G is connected all vertices are connected amongst each other and therefore all entries of x have to be identical. This results in the only possible eigenvector being a scaled variant of e. Similarly, if G is not connected only vertices in the same connected component must have identical values and therefore a linearly independent vector corresponding to eigenvalue 0 exists. □

Building upon the last idea of the proof we can squeeze additional information from the multiplicity of eigenvalue 0.

Theorem 5.4. *Let G be a graph. The multiplicity of the Laplacian eigenvalue 0 equals the number of connected components of G. The associated eigenvector for the j-th component C_j is given by*

$$(e_{C_j})_i := \begin{cases} 1, & \text{if } v_i \in C_j \\ 0, & \text{if } v_i \in \overline{C_j} \end{cases}.$$

Proof. Let G be a union of k connected components. We can renumber the vertices in a way that the component are side by side in the graph Laplacian. This results in L being a block matrix with each block representing the vertices of one component.

$$L = \begin{bmatrix} L_1 & 0 & & & \\ 0 & L_2 & 0 & & \\ & 0 & L_3 & 0 & \\ & & 0 & \ddots & 0 \\ & & & 0 & L_k \end{bmatrix}$$

Each of the component blocks L_j is a graph Laplacian itself representing component C_j. Proposed by Lemma 5.2(4) each blocks has the eigenvalue 0 with corresponding eigenvector e_{C_j}. Clearly all those eigenvectors are linearly independent as each vertex can only be part of one connected component. Hence we got at least multiplicity k of

eigenvalue 0. With Lemma 5.3 we know that each connected component and with that each block has exactly multiplicity one of eigenvalue 0. This gives us a total multiplicity of k for eigenvalue 0 as the eigenvalues of a block matrix consist of the union of eigenvalues from each individual block. □

5.2. Cuts and Partitions

Now that we have defined the graph Laplacian we can start describing our clustering problem mathematically. As a by-product of our partitioning process, we always cut a number of edges if two adjacent vertices end up in different subgraphs. To make further definitions easier we introduce the sum of weights between two subsets $A, B \subset V$ as

$$W(A,B) = \sum_{u \in A, v \in B} w_{uv} = \sum_{u \in B, v \in A} w_{uv} = W(B,A).$$

One of our main goals will always be to reduce the weight of edges in between clusters. To give this concept a numerical value we introduce the Cut.

Definition 5.5 (Cut of a partition)**.** *For a partition $A = \{A_1, A_2, ..., A_k\}$ of a graph G the Cut of A is defined as*

$$\mathrm{Cut}(A) := \frac{1}{2} \sum_{i=1}^{k} W(A_i, \overline{A_i}).$$

The factor $1/2$ is introduced as all edges are counted twice during this process. From an optimization standpoint this does not yield any changes regarding the optimal Cut.

All of our attempts to define the ideal partition are based upon the minimization of the Cut. But merely focussing our attention on the Cut function leads to quite undesirable results. It often favours partitions where a singular vertex is separated from the rest of the graph. This results from the fact that we are ignoring the objective to achieve relatively dense clusters. The minimization of the Cut function returns clusters that have the lowest sum of edges between them. This issue can be spotted in Figure 5.

To comply with our concept of an ideal partition we introduce adapted versions of the Cut. The main idea is to introduce a quotient of the size of each partition into our optimization. With that we are rating each of the partitions based on the quotient between the Cut and the size of each cluster. Therefore, we prioritise more balanced partitions. We explicitly used the term size here as we use both earlier introduced concepts of graph size. The order and the volume creates two concepts to rate a partition.

Definition 5.6. *For a partition $A = \{A_1, A_2, ..., A_k\}$ of a graph G we define the following cost measures:*

$$\mathrm{RatioCut}(A) = \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \overline{A_i})}{|V_i|} = \sum_{i=1}^{k} \frac{\mathrm{Cut}(A_i, \overline{A_i})}{|V_i|},$$

$$\mathrm{NCut}(A) = \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \overline{A_i})}{\mathrm{vol}(A_i)} = \sum_{i=1}^{k} \frac{\mathrm{Cut}(A_i, \overline{A_i})}{\mathrm{vol}(A_i)}.$$

The NCut was mostly popularised by [16] and evaluates the costs of cutting edges based upon the percentage the intersecting edges take upon the total number of edges in that partition. We will work with NCut in section 5 as it leads to the normalized spectral clustering approach.

The other valuation of RatioCut puts the cost of an edge in perspective to the number of vertices that are in that partition. This is a simpler approach that leads to unnormalized spectral clustering.

Considering Figure 5 we deepen our understanding on how these concepts alter our ideal partition while pointing out the issues with the Cut function.

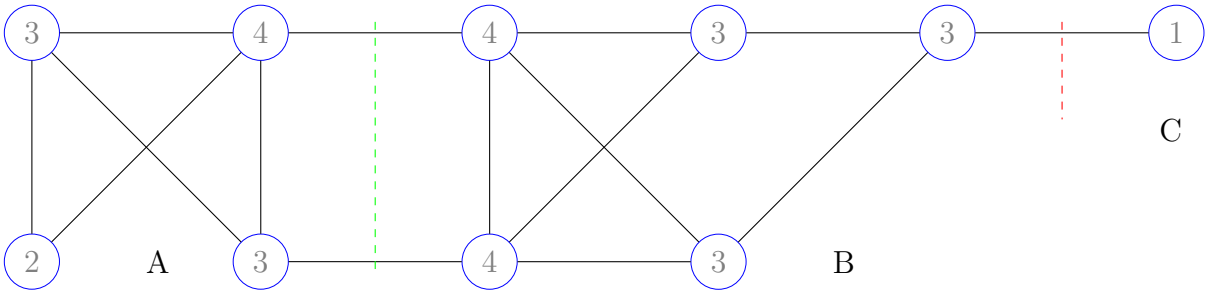


Figure 5: example of Cut, NCut and RatioCut.

In this setting our goal is to find an ideal bipartition based upon the rating of the previously defined cut functions. In dealing with an unweighted graph we assume all weights of existing edges to be 1. The numbers inside each vertex represents its degree. For the desired partition we would ideally separate along the green line but as already mentioned, the Cut favours the partition along the red line.

Partition	$(A, B \cup C)$	$(A \cup B, C)$
Cut	2	1
NCut	$\frac{2}{12} + \frac{2}{18} = \frac{5}{18}$	$\frac{1}{29} + 1 = \frac{30}{29}$
RatioCut	$\frac{2}{4} + \frac{2}{6} = \frac{5}{6}$	$\frac{1}{9} + 1 = \frac{10}{9}$

Table 1: results of the different cut functions on figure 5

Cutting the lowest amount of edges to create two connected components is represented by the red line and backed up by the Cut function. Solving the clustering problem, we additionally pay attention to higher densities inside of each cluster. The NCut and RatioCut both prefer a more balanced partition even if the splitting itself is more expensive. From this point on we trust the latter two concepts to find the ideal partition.

5.3. Normalisation of RatioCut

Simply trying to optimize the RatioCut is extremely challenging which leaves us searching for an alternative solution. This leads to an equivalent optimization using the graph Laplacian and a vector $x \in \mathbb{R}^n$.

For now, we stick to the case of two clusters as it is much easier to understand and follow the ideas. The vector x should be thought of as a label that indicates which cluster each vertex belongs to. Each coordinate $i \in \{1, ..., n\}$ represents the correspondent vertex v_i. For the two dimensional case it is enough to distinguish the coordinates of x, and thereby the vertices, by the sign. With that idea in the back of our minds, we begin investigating the connection to the RatioCut.

Theorem 5.7. *Given a graph G of order n that we want to cluster into the partition $(A, \overline{A})$. Then*

$$\min_{A \subset V} \text{RatioCut}(A, \overline{A}) \quad \Leftrightarrow \quad \min_{A \subset V} x^T L x \quad s.t. \quad x \perp e, \ ||x|| = \sqrt{n}$$

where x is defined as

$$x_i := \begin{cases} \sqrt{|\overline{A}|/|A|}, & \text{if } v_i \in A \\ -\sqrt{|A|/|\overline{A}|}, & \text{if } v_i \in \overline{A} \end{cases}.$$

Proof. Inserting our chosen x into the term obtained by Lemma 5.2(2) we compute

$$\begin{aligned} x^T L x &\overset{5.2}{=} \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2 \\ &= \frac{1}{2} \left(\sum_{i \in A, j \in \overline{A}} w_{ij} \left(\sqrt{\frac{|\overline{A}|}{|A|}} + \sqrt{\frac{|A|}{|\overline{A}|}} \right)^2 + \sum_{i \in \overline{A}, j \in A} w_{ij} \left(-\sqrt{\frac{|A|}{|\overline{A}|}} - \sqrt{\frac{|\overline{A}|}{|A|}} \right)^2 \right). \end{aligned}$$

This results from equal coordinates of x if the corresponding vertices are located in the same cluster. Realising that both quadratic terms are identical we aggregate

$$\begin{aligned} x^T L x &= \frac{1}{2} \left(\sum_{i \in A, j \in \overline{A}} w_{ij} + \sum_{i \in \overline{A}, j \in A} w_{ij} \right) \left(\frac{|\overline{A}|}{|A|} + \frac{|A|}{|\overline{A}|} + 2\sqrt{\frac{|\overline{A}|}{|A|}\frac{|A|}{|\overline{A}|}} \right) \\ &= \frac{1}{2}(W(A, \overline{A}) + W(\overline{A}, A)) \left(\frac{|\overline{A}|}{|A|} + \frac{|A|}{|\overline{A}|} + 2 \right). \end{aligned}$$

With the first part being our Cut function and some relocating of the terms we ultimately conclude

$$\begin{aligned} x^T Lx &= \mathrm{Cut}(A,\overline{A})\left(\frac{|\overline{A}|}{|A|}+\frac{|A|}{|\overline{A}|}+2\right) = \mathrm{Cut}(A,\overline{A})\left(\frac{|\overline{A}|}{|A|}+\frac{|A|}{|\overline{A}|}+\frac{|A|}{|A|}+\frac{|\overline{A}|}{|\overline{A}|}\right) \\ &= \mathrm{Cut}(A,\overline{A})\left(\frac{|V|}{|A|}+\frac{|V|}{|\overline{A}|}\right) = |V|\left(\frac{\mathrm{Cut}(A,\overline{A})}{|A|}+\frac{\mathrm{Cut}(A,\overline{A})}{|\overline{A}|}\right) \\ &= |V|\cdot \mathrm{RatioCut}(A,\overline{A}). \end{aligned}$$

With the order of G being a constant, the minimization of RatioCut can be achieved by minimizing $x^T Lx$ given this specific definition of x. The other constraints follow naturally as

$$\begin{aligned} \langle e, x\rangle &= \sum_{i=1}^{n} x_i = \sum_{i\in A}\sqrt{\frac{|\overline{A}|}{|A|}} - \sum_{i\in\overline{A}}\sqrt{\frac{|A|}{|\overline{A}|}} = |A|\sqrt{\frac{|\overline{A}|}{|A|}} - |\overline{A}|\sqrt{\frac{|A|}{|\overline{A}|}} \\ &= \sqrt{|A|^2\frac{|\overline{A}|}{|A|}} - \sqrt{|\overline{A}|^2\frac{|A|}{|\overline{A}|}} = \sqrt{|V|} - \sqrt{|V|} = 0. \end{aligned}$$

Directly resulting in x being orthogonal to e. Lastly, we check on the squared length of x

$$||x||^2 = \sum_{i=1}^{n} x_i^2 = |A|\frac{|\overline{A}|}{|A|} + |\overline{A}|\frac{|A|}{|\overline{A}|} = |\overline{A}| + |A| = |V| = n.$$

□

Even now that each coordinate of x only has two possible discrete values, with large graph size 2^n possible solutions emerge. Therefore, we want to relax the problem to make use of the earlier mentioned standard optimization problems. This is achieved by allowing arbitrary values for x. The problem transforms into

$$\min_{x\in\mathbb{R}^n} x^T Lx \quad \text{s.t.} \quad x \perp e, \quad ||x|| = \sqrt{n}.$$

With the knowledge that e is the eigenvector corresponding to the smallest eigenvalue of L the optimal solution is given as he second smallest Laplacian eigenvector λ_2 by equation (2).

Let us take a quick moment to rewind what we have just done. We are searching for a bipartition of a graph with the goal of minimizing the RatioCut function. If we define a vector $x \in \mathbb{R}^n$ following the blueprint of Theorem 5.7, representing a label for each vertex. Then minimizing $x^T Lx$ yields the same result as minimizing the RatioCut. Relaxing the resulting problem by allowing arbitrary values for x we create a slightly tempered problem that is solved by the second smallest eigenvalue of L.

Assuming arbitrary values for x, we lost the clear indication which cluster each vertex belongs to. Finalizing the result, we need to split the labels into the two clusters. In the two-dimensional case a simple separation based upon the sign of each coordinate will suffice. As soon as we want to create more clusters, this rule is no longer applicable. Therefore, we will use our k-means algorithm to divide, the embedded vertex labels, into the two clusters.

Example 5.8. *Let us consider a quite obvious example of a bipartition on the graph in Figure 6 which consists of two connected components.*

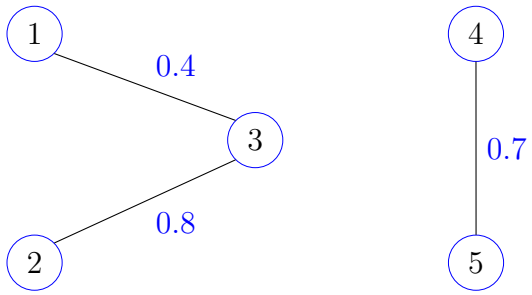


Figure 6: A graph G with the obvious bipartition into the two connected components.

With Lemma 5.3 and Theorem 5.4 we are aware that $\lambda_1, \lambda_2 = 0$ *and suitable eigenvectors are* $e_{C_1} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \end{bmatrix}^T$ *and* $e_{C_2} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \end{bmatrix}^T$*. These would certainly indicate the ideal clusters but we want to use the more general approach of Theorem 5.7 which always assumes the first eigenvector* e*. This will result in an alternative way of spanning the same eigenspace. We start with the graph Laplacian*

$$L = \begin{bmatrix} 0.4 & 0 & -0.4 & 0 & 0 \\ 0 & 0.8 & -0.8 & 0 & 0 \\ -0.4 & -0.8 & 1.2 & 0 & 0 \\ 0 & 0 & 0 & 0.7 & -0.7 \\ 0 & 0 & 0 & -0.7 & 0.7 \end{bmatrix}.$$

The earlier mentioned candidates e_{C_1} *and* e_{C_2} *are both violating the constraint of being orthogonal to* e*. Still with those two in mind we can easily find another eigenvector* x^2 *corresponding to eigenvalue 0 while also orthogonal to* e*.*

$$x^2 = \begin{bmatrix} a & a & a & b & b \end{bmatrix}^T$$

Clearly with $a \neq b$ *the two eigenvectors* x^2 *and* e *will span the same eigenspace as* e_{C_1} *and* e_{C_2}*. We just have to make sure that* x^2 *is orthogonal to* e *and its length is* $\sqrt{n}$*. Therefore,*

$$\begin{aligned} \langle x^2, e \rangle &= 3a + 2b = 0 \quad \textit{and} \\ ||x^2|| &= \sqrt{3a^2 + 2b^2} = \sqrt{n} \Leftrightarrow 3a^2 + 2b^2 = n. \end{aligned}$$

Solving this linear system results in $a = -\frac{\sqrt{6}}{3}$ *and* $b = \frac{\sqrt{6}}{2}$*. Together with our first eigenvector this constructs the eigenvector matrix*

$$U = \begin{bmatrix} 1 & -\frac{\sqrt{6}}{2} \\ 1 & -\frac{\sqrt{6}}{2} \\ 1 & -\frac{\sqrt{6}}{2} \\ 1 & \frac{\sqrt{6}}{2} \\ 1 & \frac{\sqrt{6}}{2} \end{bmatrix}.$$

Even if this notation does not seem very useful at this point it is the key to understand the separation of the graph into more than two clusters. The general idea is that the first k eigenvectors all give their indication on which vertex belongs to which cluster. The combined information for each vertex is held in the corresponding row of U. These rows hold the embedded coordinates for each vertex which k-means will use to cluster the graph.

Plotting the rows of U results in all vertices of the same connected component sitting on identical locations and making it trivial for k-means to detect the two clusters. Alternatively, the simple separation after the sign of the components in x^2 would have yielded the same result.

Now we focus our attention to the case of more than two clusters. The basic proceeding is similar but certain definitions must be adapted to yield similar results.

Theorem 5.9. *For any given $k > 2$, let G be a graph that we want to partition into $A = (A_1, A_2, ..., A_k)$. Then*

$$\min_A \text{RatioCut}(A) \Leftrightarrow \min_A \text{tr}(H^T L H) \quad s.t. \quad H^T H = \mathbb{I}.$$

where $H \in \mathbb{R}^{n \times k}$ is defined as:

$$h_{ij} := \begin{cases} 1/\sqrt{|A_j|}, & \text{if } v_i \in A_j \\ 0, & \text{if } v_i \notin A_j \end{cases}.$$

Proof. Firstly, we want to confirm that the columns of H create an orthonormal basis. Each vertex can only be part of one cluster which means that each row of H only contains one cell that is not zero. Therefore, all columns are orthogonal. Furthermore, each column is normalized due to the division by the square root of the number of vertices in that partition. Altogether, we conclude that $H^T H = \mathbb{I}$ as all cells of $H^T H$ represent the inner product of the corresponding vectors.

The trace of $H^T L H$ consists of the diagonal cells

$$(H^T L H)_{jj} = h_j^T L h_j,$$

with h_j being the j-th column of H. This term is mentioned in Lemma 5.2(2) and we can utilise similar techniques as in Theorem 5.7 to show the correspondence of each trace component with the Cut function:

$$\begin{aligned} h_j^T L h_j &\overset{5.2}{=} \frac{1}{2} \sum_{s,t=1}^{n} w_{st}(h_{sj} - h_{tj})^2 \\ &= \frac{1}{2} \left(\sum_{s \in A_j, t \in \overline{A_j}} w_{st} \left(\sqrt{\frac{1}{|V_j|}} \right)^2 + \sum_{s \in \overline{A_j}, t \in A_j} w_{st} \left(-\sqrt{\frac{1}{|V_j|}} \right)^2 \right) \\ &= \frac{1}{|V_j|} \cdot \frac{1}{2} \left(\sum_{s \in A_j, t \in \overline{A_j}} w_{st} + \sum_{s \in \overline{A_j}, t \in A_j} w_{st} \right) = \frac{\text{Cut}(A_j, \overline{A_j})}{|V_j|}. \end{aligned}$$

Putting everything together leads to

$$\text{RatioCut}(A) = \sum_{j=1}^{k} \frac{\text{Cut}(A_j, \overline{A_j})}{|V_j|} = \sum_{j=1}^{k} h_j^T L h_j = \sum_{j=1}^{k} (H^T L H)_{jj} = \text{tr}(H^T L H).$$

□

Repeating our earlier relaxation approach we allow H to take arbitrary values and with that the problem becomes

$$\min_{H \in \mathbb{R}^{n \times k}} \text{tr}(H^T L H) \text{ s.t. } H^T H = I.$$

This is the standard trace minimization problem from equation (3) and it is solved by the matrix H containing the eigenvectors of L in ascending order of their corresponding eigenvalue as columns. As we are only clustering into k clusters our matrix has dimension $n \times k$ with the best approximation given by the matrix containing only the first k eigenvectors. This is a direct result from the singular value decomposition mentioned in [17][p.77-80]. Note that in this case the singular values are just the eigenvalues as L is symmetric and positive semi-definite.

5.4. Presenting the Algorithm

Being familiar with the basic ideas behind spectral clustering we can design a blueprint for our algorithm and inspect the implemented code. Starting off with the basic idea of the implementation for spectral clustering. For now, we assume the number of clusters to be known. For a discussion to figure out the ideal number of clusters based on the graph we refer to section 5.7.2.

Unnormalized Spectral Clustering
Input: G - graph, k - desired number of clusters
Output: $C_1, ..., C_k$ clusters of G

---

1. Compute the graph Laplacian of the graph G.
2. Solve for its k smallest Laplacian eigenvalues and put the corresponding eigenvectors into the columns of a matrix $U \in \mathbb{R}^{n \times k}$. Each row of U represents the spectrally embedded coordinates of a vertex in G.
3. On the newly embedded coordinates let k-means distinguish into k clusters and return the results.

These few steps are quickly implemented and result in the main part of the spectral clustering algorithm displayed in figure 7.

Firstly, we transform the graph Laplacian into a sparse type matrix and change its cell type into a float. This allows us to utilize the SciPy eigsh function which computes

```
def spectralClustering(W, k):
    L = sparse.csr_matrix(getGraphLaplacian(W))
    L = L.asfptype()
    lamb, eigv = sparse.linalg.eigsh(L, k = k, which='SM')
    kmeans = KMeans(n_clusters=k, random_state=2).fit(eigv)
    return kmeans.labels_
```

Figure 7: the main part of the spectral clustering code.

the k smallest eigenvalues with their corresponding eigenvectors given the used input parameters. The eigenvectors are collected in the matrix eigv and are divided into clusters by k-means.

The computationally most expensive step of the algorithm is the solving of the eigenvalue problem. The eigsh method does not take all favourable characteristics of the graph Laplacian into consideration. It is surely more effective than the classical solution of using the characteristic polynomial but massive research on eigenvalues produced dozens of highly effective approaches to compute the eigenpairs for specific types of matrices.

The graph Laplacian presents all kinds of opportunity to effectively search for the eigenvalues. Irrespectively of the base graph the graph Laplacian is always symmetrical, positive semi-definite and in the case of an unweighted graph it is relatively sparse. Furthermore, all diagonal cells are positive while all other are non-positive and we are already familiar with the first eigenpair. We could continue on, but it should be obvious that we work with an unique class of matrices. Therefore, methods of finding the eigenvalues can be improved massively if only dealing with graph Laplacians matrices.

We will not cover any effective way of computation in this thesis, but we still want to give a starting point to any interested reader. During many years of research in the field of spectral clustering, one particular computational method for eigenvalues has been popularized. This is the Lanczos method which can be further investigated in [14, 13]. For a general overview and discussion on different methods to solve eigenvalue problems we refer to [17].

5.5. Similarity Measures

With the main part of our algorithm implemented there is one more thing we need to address. Our clustering is supposed to be capable of commencing any kind of data. In contrast to that, our current input is the weight matrix of a graph. To ensure that any data set can be processed, we need to translate it into a graph.

The first step is always the assumption that each of our data points represents a vertex of the graph. The more noteworthy challenge is to figure out how to decide if there is an edge between two data points and what the corresponding weight would be if we intent to create a weighted graph. The clusters ought to include vertices that are similar. Therefore, the edges will represent this idea of similarity in regards to the specific data set.

5.5.1. Similarity Functions

To put the word similarity into a numeric value, we utilize a similarity function. It takes an input of two vertices and outputs a numeric representation regarding how similar those two vertices are. The specific implementation largely depends on the data set and the aspects of interest in the analysis. Nonetheless, there are a few properties that all similarity functions should have in common.

Definition 5.10. *For a given data set D we may call the function $s : D \times D \to \mathbb{R}$ similarity function if it satisfies the following criteria. For all $u, v \in D$:*

- $s(u, v)$ *is defined.* *(universally defined)*
- $s(u, v) \in [0, 1]$. *(positivity and normalization)*
- $s(u, v) = s(v, u)$. *(symmetry)*

This leaves many possibilities, and rarely gives a direct answer on how to exactly define similarity on a data set. Nonetheless, most similarity functions should yield similar values as there exists a general understanding on what similarity means. On the other hand, if we are only interested in a small portion of the attributes that a data set offers, we may achieve vastly differing results. Positively speaking, this provides us with a broad spectrum of possibilities to decide which aspect of the data we wish to focus on. For example, a dataset containing basic information on humans could be compared purely by focusing on foot and hand sizes, while another research might only compare hair and eye colour. The similarity function would only be influenced by the aspects that we choose. A relatively intuitive way of thinking about a similarity function is given by comparing each attribute itself with a similarity value. Those single attribute comparisons are put into a convex combination with each attribute weighted according to its importance regarding our idea of similarity.

At this point, reconsider the earlier used examples in Figure 1. The most natural starting point in regards to similarity is the already used Euclidean distance. It is the only immediate notion of similarity that already satisfies most requirements of a similarity function. Still the normalization remains unsatisfied while the function also has a reverse idea of similarity. Realising that high distances should yield low similarity values results in reversing the projection onto the interval between 0 and 1. Luckily, there is a quite convenient way to complete this through the use of the exponential function.

Definition 5.11. *For two data points $x, y \in \mathbb{R}^n$ and $\sigma > 0$ the Gaussian similarity function is defined as:*

$$s_G(x, y) = \exp(-||x - y||^2/(2\sigma^2))$$

It is easily provable that this satisfies a similarity function in the sense of Definition 5.10. Considering two data points that are dissimilar, which is depicted in a high distance between them. Therefore, the negative value inside of the exponential function of the Gaussian similarity function is larger, ultimately yielding a similarity value closer to 0.

For close-by points exactly the opposite development is observed which results in similarity values closer to 1.

Additionally, this term might remind us of the formula for the standard normal distribution. This is exactly how we can think about the Gaussian similarity. It represents a normal distribution centred around each data point with similarities rising in immediate proximity of each data point. Furthermore, this gives a nice interpretation for the used σ which functions exactly as the standard deviation. It spreads the distribution and therefore creates higher similarities further out.

5.5.2. Similarity Graphs

After having defined a similarity function there are a multitude of options to translate the dataset into an actual graph. We will look at some of the most common ones, all of which are based upon a general similarity function.

Definition 5.12 (Common Similarity Graphs)**.**

- *The fully connected graph - This is the most obvious idea to create a graph. We simply connect each pair of vertices with the given value of the similarity function. Clearly this results in a weighted graph which does not loose any information provided by the similarity function. The drawback is the higher computational cost compared to sparse weight matrices.*

- *The ϵ-neighbourhood graph - Here we choose an $\epsilon \in [0,1]$ which decides the minimum required similarity to connects two vertices. For any two vertices $u, v \in V$:*

$$uv \in E \quad \Leftrightarrow \quad s(u,v) \geq \epsilon.$$

  This loses quite a bit of information, as any similarity value is interpreted as either being similar or being non-similar. This is an easy rule to apply that results in an unweighted similarity graph.

- *The k-nearest neighbour graph - As the name suggest we define the set of the k-nearest neighbours of $v \in V$ as*

$$\mathrm{knN}_k(v) := \{u \in D | s(u,v) \leq a, \quad a := \mathrm{argmax}_\epsilon |\{u' \in V | s(u',v) \geq \epsilon\}| = k+1\}.$$

  The idea is to include an edge from each vertex to its k nearest neighbours. The problem occurring is that this results in a directed graph. To solve this issue, the most common approach is to ignore the direction of the edges. Equivalently, we might say for $u, v \in V$ is holds that

$$uv \in E \quad \Leftrightarrow \quad v \in \mathrm{knN}_k(u) \quad or \quad u \in \mathrm{knN}_k(v).$$

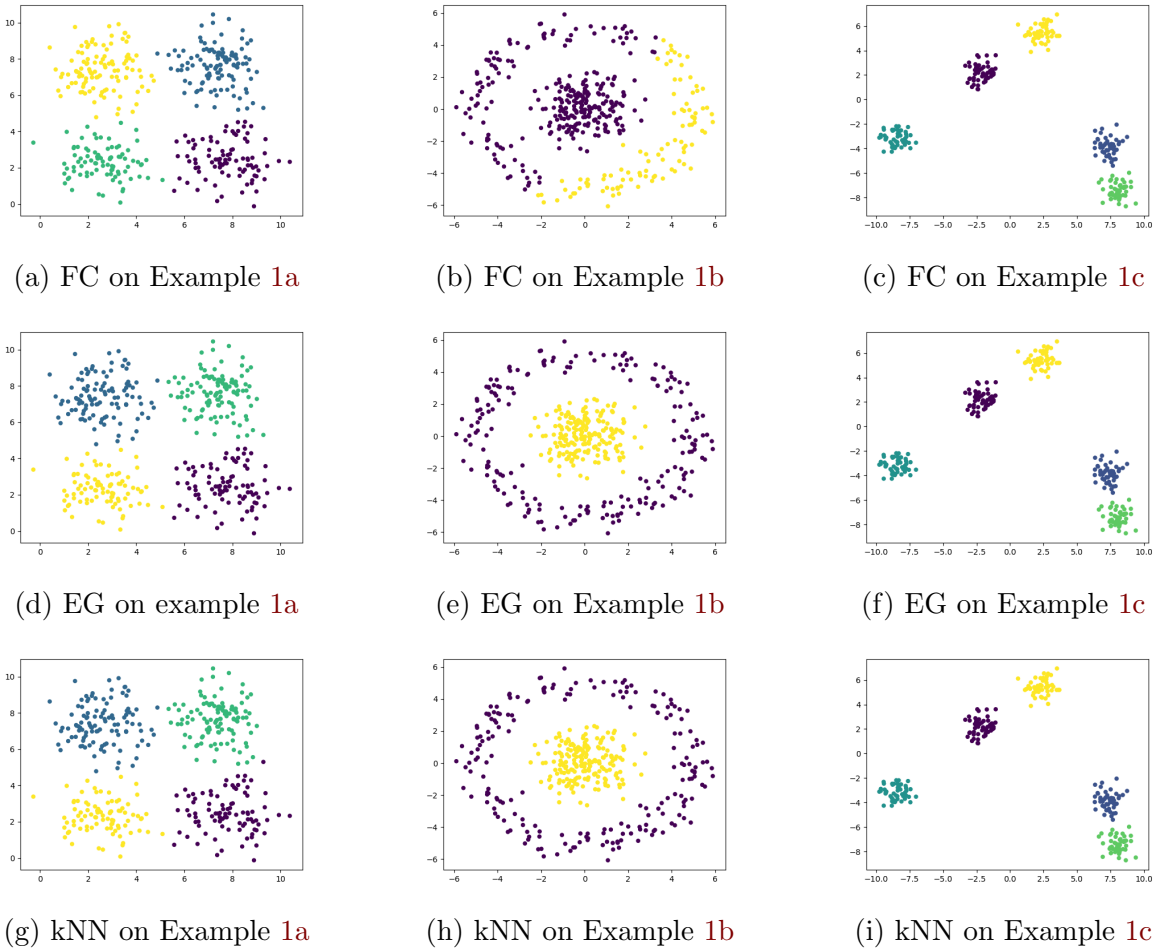
  Alternatively we could add the edge (uv) if u is a k-nearest neighbour of v and v is a k-nearest neighbour of u. For $u, v \in D$ it follows that

$$uv \in E \quad \Leftrightarrow \quad v \in \mathrm{knN}_k(u) \quad and \quad u \in \mathrm{knN}_k(v).$$

  The latter is less common and is referred to as the mutual k-nearest neighbour graph. Both approaches result in an unweighted graph.

5.6. Display of Results

Having met every requirement to build our own algorithm we are ready to test our examples in Figure 1 with the spectral clustering algorithm. The results are displayed in Figure 8 with the full code displayed in section A.



(a) FC on Example 1a (b) FC on Example 1b (c) FC on Example 1c

(d) EG on example 1a (e) EG on Example 1b (f) EG on Example 1c

(g) kNN on Example 1a (h) kNN on Example 1b (i) kNN on Example 1c

Figure 8: Different results for spectral clustering using the fully connected graph (FC) based upon the Gaussian Similarity with $\sigma = 3$, the ϵ-neighbourhood graph (EG) with $\epsilon = 0.6$ and the k-nearest neighbour graph (kNN) with 20 neighbours.

The most interesting results appear in the middle column on the example k-means could not handle. Firstly, we can positively note that spectral clustering is able to distinguish into the desired clusters. Moreover, using the fully connected graph did not achieve the desired results given the used parameters. This is mostly due to the small differences between inner and outer circle that are not as clearly separated. The fully connected graph keeps all similarities, therefore no large differences of distance occur between the two desired clusters. Especially, when considering a point on the outer ring, the similarity towards most points in the inner circle will be higher than most other points on the ring.

Additionally, given that $\sigma = 0.5$ for the Gaussian similarity yields the desired result for

the fully connected graph. The updated σ reduces the similarities significantly between all data points. However, this interferes with other results making it the reason that it is not displayed. A similar line of argument also explains why the other two graphs work particularly well for this example. Both cut off similarity under a certain threshold, which with the right choice of parameters results in the majority of connections only existing inside each of the desired clusters.

Altogether, an understanding of each parameter as well as knowledge of the data mixed with some trial and error is needed to effectively work with spectral clustering.

5.7. Additional Concepts

After concluding with the algorithm, we will now focus on two connected fields of spectral clustering. These give us an even deeper understanding of the way spectral clustering operates, makes us more comfortable with many of the newly introduced variables and generally showcases how powerful of a tool the graph Laplacian is. Mostly we want to focus on the second smallest Laplacian eigenvalue λ_2 and later on, take a short detour into perturbation theory. The following section is largely inspired by the results in [7, 12].

5.7.1. Algebraic Connectivity

Fiedler named the second smallest Laplacian eigenvalue the algebraic connectivity of a graph. Lemma 5.3 already provides a very important connection to the connectivity of a graph. In the following pages we investigate different values of the algebraic connectivity, by proposing bounds depending on the degree of vertices and missing edges on a graph. Additionally, we will list the algebraic connectivity for a number of specific graphs.

Starting off, there is a close connection between the Laplacian eigenvalues and the degree of all vertices in G.

Lemma 5.13. *For a graph $G = (V, E)$ and its graph Laplacian L*

$$\sum_{i=1}^{n} \lambda_i = 2|E| = \sum_{v \in V} \mathrm{d}(v).$$

Proof. As the trace of a matrix is also defined by the sum of the eigenvalues it simply follows that

$$\sum_{i=1}^{n} \lambda_i = \mathrm{tr}(L) = \sum_{v \in V} \mathrm{d}_v = 2|E|.$$

□

For our next result we require additional assistance from Weyl's inequality [10][p.181]. It bounds the maximum change of the eigenvalues if a matrix is perturbed which we will discuss in section 5.7.2.

Theorem 5.14 (Weyl's Inequality)**.** *For two matrices $L, P \in \mathbb{R}^{n\times n}$ and their sum $\widetilde{L} = L + P$ with respective eigenvalues $\lambda_i, \widetilde{\lambda}_i$ for $L, \widetilde{L}$ and ρ_i for P all in ascending order, it holds that*

$$\lambda_i + \rho_1 \leq \widetilde{\lambda}_i \leq \lambda_i + \rho_n.$$

Bringing Lemma 5.13 and Theorem 5.14 together we conclude how eigenvalues change if we add an extra edge to a graph.

Corollary 5.15. *Let G be a graph and $G' = G + e$ the same graph with an extra edge e. Then with Theorem 5.14 if we consider P to be the matrix associated with the additional edge we immediately conclude*

$$0 = \lambda_1(G) = \lambda_1(G') \leq \lambda_2(G) \leq \lambda_2(G') \leq \lambda_3(G) \leq ... \leq \lambda_n(G').$$

Together with Lemma 5.13 more precisely we follow

$$\sum_{i=1}^{n} (\lambda_i(G') - \lambda_i(G)) = 2.$$

This means that on the one hand, at least one inequality must be strict and from a wider perspective, it clearly shows that more edges in a graph result in larger Laplacian eigenvalues, hence the name algebraic connectivity.

Gathering some of the basic properties of the algebraic connectivity we continue with the following lemma.

Lemma 5.16. *Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be two graphs on an identical vertex set with $W_1, W_2 \in \mathbb{R}^{n\times n}$ their respective weight matrix.*

1. *If G_1 is included in G_2 implying that $W_1 \leq W_2$ coordinate wise. Then $\lambda_2(G_1) \leq \lambda_2(G_2)$.*
2. *If for all $v_i, v_j \in V$ $(W_1)_{ij} = 0 \quad or \quad (W_2)_{ij} = 0$ meaning the edges of the two graphs are not intersecting. Then the algebraic connectivity of $G = (V, W_1 + W_2)$ satisfies*

$$\lambda_2(G_1) + \lambda_2(G_2) \leq \lambda_2(G).$$

Proof.

1. Follows right away from Corollary 5.15
2. We can easily verify that $L_G = L_{G_1} + L_{G_2}$. Therefore with usage of equation (2)

$$\begin{aligned}\lambda_2(G) = \lambda_2(G_1 + G_2) &= \min_{x\in\mathbb{R}^n, x\perp e} x^T L_{G_1} x + x^T L_{G_2} x \\ &\geq \min_{x\in\mathbb{R}^n, x\perp e} x^T L_{G_1} x + \min_{x\in\mathbb{R}^n, x\perp e} x^T L_{G_2} x = \lambda_2(G_1) + \lambda_2(G_2).\end{aligned}$$

□

Let us return to the graph Laplacian matrix. To get an upper bound for the algebraic connectivity, we want to inspect the densest graph possible.

Lemma 5.17. *In the unweighted case the graph Laplacian of the complete graph K_n is*

$$L(K_n) = nI - J, \tag{4}$$

with J being the matrix consisting of a 1 in each cell. Considering $L(K_n)$ we compute that

$$\lambda_2(K_n) = n.$$

Proof. For the complete graph all vertices are connected amongst each other. Therefore, each vertex has degree $n-1$. With each possible edge existing every off-diagonal cell equals -1.

We will demonstrate that the eigenvalue n has the geometrical multiplicity of $n-1$. Therefore, only our known first eigenvalue 0 and n can exist. Define the following $n-1$ vectors

$$x^i = \begin{bmatrix} -1 & 0 & ... & 0 \end{bmatrix}^T + e_i, \quad i \in \{2, ..., n\}.$$

Multiplying our graph Laplacian L with x^i results in

$$L(K_n)x^i = \begin{cases} -n, & \text{for } x^i_1 \\ n, & \text{for } x^i_i \\ 0, & \text{for } x^i_{\{2,...,n\}\setminus\{i\}} \end{cases}.$$

This clearly states that all x^i are eigenvectors associated to the eigenvalue n. In addition to that, each of them are the only vector to have a non-zero coordinate at the i-th position making all x^i linearly independent. This results in the corresponding eigenspace having at minimum the dimension of $n-1$. Quickly confirming that e is another linearly independent vector, we found all eigenvalues and therefore, especially results in $\lambda_2 = n$. □

In conclusion, excluding double edges or edges with a higher weight than one, the largest possible algebraic connectivity on a graph is its order.

Returning to the graph Laplacian of the complete graph, we can connect equation (4) to the complementary graph. The union of a graph with its complement always results in the complete graph. Therefore, with G having order n we expand to

$$L(G) + L(\overline{G}) = L(K_n) = nI - J. \tag{5}$$

Now we can determine the eigenvalues of the complementary graph $\overline{G}$, based upon those of the original graph G.

Lemma 5.18. *Let G be a graph of order n and $\lambda_i, \quad i \in \{1, ..., n\}$ the corresponding eigenvalues. The Laplacian eigenvalues of the complementary graph $\overline{G}$ are given by $\overline{\lambda}_1 = 0$ and*

$$\overline{\lambda}_i = n - \lambda_{n-i+2} \quad \text{for } i \in \{2, ..., n\}.$$

Additionally the connected eigenvalues are associated with the same eigenspace.

Proof. Let x^i for $i \in \{2, ..., n\}$ be the associated eigenvectors of $L(G) \in \mathbb{R}^{n \times n}$ excluding e. By multiplying equation (5) with eigenvector x^i we conclude that

$$L(G) + L(\overline{G}) = nI - J \Leftrightarrow L(G)x^i + L(\overline{G})x^i = \lambda_i x^i + L(\overline{G})x^i = (nI - J)x^i.$$

Trying to seek the Laplacian eigenvalues $\overline{\lambda}_i$ of the complementary graph and by using the knowledge that x^i is orthogonal to e as it is not the first eigenvalue of $L(\overline{G})$, we continue

$$L(\overline{G})x^i = (nI - J - \lambda_i I)x^i = ((n - \lambda_i)I)x^i - Jx^i = (n - \lambda_i)x^i.$$

Wanting to have them in the same ascending order we simply re-index according to magnitude

$$\overline{\lambda_i} = n - \lambda_{n+2-i}$$

□

Using our newly gained information, we can compute the eigenvalues of the graph K_{n-p}. It can be thought of as a complete graph K_n but with a subset $P \subset V$ with $p = |P|$ where each edge in between vertices of P is removed. The complementary graph $\overline{K}_{n-p}$ consists of the complete graph K_p and $n - p$ isolated vertices. Therefore, the graph Laplacian of $\overline{K}_{n-p}$ is a block matrix containing the eigenvalue 0 in multiplicity $n - p + 1$ for each isolated vertex. Additionally, the complete graph contributes eigenvalue p in multiplicity $p - 1$ according to Lemma 5.17.

Altogether, with Lemma 5.18 the eigenvalues of K_{n-p} consist of 0, $(n-p)$ in multiplicity $p - 1$ and n in multiplicity $n - p$. Together with Lemma 5.16(1) we have proven the following theorem.

Theorem 5.19. *Let G be a graph with order n that contains an unconnected set of p vertices $P \subset V$. Then*

$$\lambda_2(G) \leq n - p.$$

An immediate consequence is that if we have a graph G of order n that is not the complete graph, then we can immediately name an upper bound for λ_2. With G containing at least two not directly connected vertices, Theorem 5.19 states

$$\lambda_2(G) \leq n - 2.$$

For further algebraic connectivities of well known graphs and deeper analysis we refer to [5, 7].

To mention another set of bounds we need an alternative definition for the algebraic connectivity which can be verified using the earlier mentioned results.

Lemma 5.20. *For a graph G of order n, the algebraic connectivity*

$$\lambda_2 = n \min_{x \in \mathbb{R}^n, x \perp e} \frac{\sum_{i,j=1}^n w_{ij}(x_i - x_j)^2}{\sum_{i,j=1}^n (x_i - x_j)^2}.$$

Proof. Using equation (2) we know that

$$\lambda_2 = \min_{x \in \mathbb{R}^n, x \perp e} \frac{x^T L x}{x^T x}.$$

Additionally, Lemma 5.2(2) directly results in the numerator of one half of the desired term hence it suffices to show that the denominator $x^T x = \frac{1}{2n} \sum_{i,j=1}^n (x_i - x_j)^2$. Equivalently we compute

$$\begin{aligned} \sum_{i,j=1}^n (x_i - x_j)^2 &= \sum_{i=1}^n x_i^2 - 2 \sum_{i,j=1}^n x_i x_j + \sum_{j=1}^n x_j^2 \\ &= n x^T x - 2 x^T e + n x^T x = 2n x^T x. \end{aligned}$$

□

Using the same techniques we show that

$$\lambda_n = n \cdot \max_{x \in \mathbb{R}^n, x \perp e} \frac{\sum_{i,j=1}^n w_{ij}(x_i - x_j)^2}{\sum_{i,j=1}^n (x_i - x_j)^2}.$$

Examining the quotient in Lemma 5.20 we notice that the addition of a by $c \in \mathbb{R}$ scaled e yields the same result. Fiedler suggested in [6] that it suffices that x is not a scaled variant of e itself instead of being orthogonal to e. Therefore,

$$\lambda_2 = n \min_{x \in \mathbb{R}^n, x \neq ce} \frac{\sum_{i,j=1}^n w_{ij}(x_i - x_j)^2}{\sum_{i,j=1}^n (x_i - x_j)^2}.$$

The same result holds for the largest eigenvalue and allows us to find bounds for both extremal eigenvalues.

Proposition 5.21. *Let G be a graph with its minimum degree $\delta(G)$ and maximum degree $\Delta(G)$ and $c \in \mathbb{R}$. Then the extremal Laplacian eigenvalues are bounded by*

$$\lambda_2(G) \leq \frac{n}{n-1}\delta(G) \leq \frac{n}{n-1}\Delta(G) \leq \lambda_n(G) \leq 2\Delta(G).$$

Proof. Choosing any vertex $v_i \in V$ and taking its corresponding normal basis vector e_i we can utilize the results from Lemma 5.20 to estimate our extremal Laplacian eigenvalues. Firstly

$$\lambda_2 = n \min_{x \in \mathbb{R}^n, x \neq ce} \frac{\sum_{i,j=1}^n w_{ij}(x_i - x_j)^2}{\sum_{i,j=1}^n (x_i - x_j)^2} \overset{x=e_i}{\leq} n \frac{2d_i}{2(n-1)} = \frac{n}{n-1} d_i.$$

Secondly for λ_n using the same e_i but this time resulting in a smaller term as a result of the maximum:

$$\lambda_n \geq n\frac{2d_i}{2(n-1)} = \frac{n}{n-1}d_i$$

The last inequality in the proposition follows from the Perron-Frobenius theorem and is proven in [12]. □

5.7.2. Perturbation Theory

Another approach to justify spectral clustering arises from the field of perturbation theory. In particular, matrix perturbation theory tries to measure or bound the changes in eigenvalues or eigenvectors of a slightly altered matrix. This is usually done by a perturbation matrix H which creates the tampered matrix $\widetilde{A} = A + H$. There are several discovered upper boundaries between the eigenpairs of $\widetilde{A}$ and A with one earlier mentioned example being Theorem 5.14. Generally speaking, the consent is that a minimal perturbation of a matrix can only change the eigenvalues to a certain extent.

Theorem 5.4 states that the graph Laplacian of k connected components has eigenvalue 0 in multiplicity k. If we consider a graph where only a few edges with reasonably small weights exist between the clusters, the first k eigenvalues should remain considerably close to our ideal case of 0. Similarly, the embedded coordinates belonging to the same component were found on the same spot. The same argument would suggest that the labels still have reasonably small distance between each other which leads to k-means being able to identify the clusters. To visualize this idea, we alter the ideal case of k connected components to being only almost ideal.

Example 5.22. *Reconsidering Example 6 with the little tweak of adding an additional edge with weight w between vertex 3 and 4. This results in the graph displayed in Figure 9.*

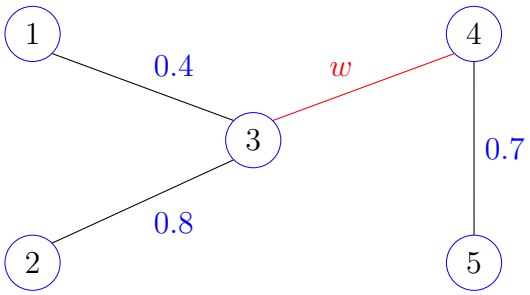


Figure 9: The additional red edge connects the previously two connected components.

Computing the eigenvalue λ_2 and corresponding eigenvector x^2 for different weights of the red edge is displayed in Table 2.

For all three values of w k-means would still detect the same two clusters.

w	0.1	0.3	0.5
λ_2	0.08	0.2	0.28
x^2	$\begin{bmatrix} -0.7 \\ -0.63 \\ -0.57 \\ 0.89 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -0.79 \\ -0.53 \\ -0.4 \\ 0.72 \\ 1 \end{bmatrix}$	$\begin{bmatrix} -0.9 \\ -0.43 \\ -0.28 \\ 0.6 \\ 1 \end{bmatrix}$

Table 2: The second smallest eigenvalue λ_2 and its corresponding eigenvector x^2 subject to the weight of the additional edge in Figure 9.

Directly linked to these results, arises the idea of figuring out the ideal number of clusters to distinguish into. Up until this point we always provided the desired number of clusters as an input to the algorithm. For a starting point consider the graph consisting of n separate vertices. The graph Laplacian has eigenvalue 0 in multiplicity n. Connecting any two vertices results in $\lambda_n = 2$ according to Lemma 5.17. All other eigenvalues are unchanged due to Corollary 5.15 and the ideal number of clusters is the multiplicity of eigenvalue 0. We generally want to start from the smallest eigenvalue and search through the ordered eigenvalues until we find an unusually large gap. This approach is backed up further by the ideal case of k connected components resulting in eigenvalue 0 in multiplicity k.

Regarding all three cases of Table 2 the third smallest eigenvalue λ_3 always hovers around $1/2$. For the smaller weights this seems like a rather large step from our second smallest eigenvalue but looking at $w = 0.5$ we could argue that the increase from $\lambda_2 = 0.28$ to $\lambda_3 = 0.5$ is not as big of a gap. Therefore, we should include a third cluster. Considering our graph, this seems reasonable as the resulting three clusters $(\{1\}, \{2, 3\}, \{4, 5\})$ are all strongly connected, and we have an at most equal distribution of vertices. The big downsides of this approach are that we possibly need to compute unnecessary eigenvectors which are only used to find the reasonable large gap. Furthermore, the precise formulation of a generally large gap seems rather idealistic and might not always yield the desired result, if even any. Nevertheless, this might be a good start point for trying to remove the precondition of knowing the number of clusters, and therefore result in needing less information on the data.

6. Normalized Spectral Clustering

Over the years the spectral clustering approach has been optimized and use of the graph Laplacian as we introduced it has become outdated. A more advanced approach uses a different variant of the graph Laplacian L_{rw} introduced in [16]. Different literature might call the variants of the graph Laplacians slightly different names, however, we stick to the notations in [18].

6.1. The normalized Graph Laplacian

We will only scratch the surface of the theory in regards to the normalized graph Laplacian and refer to the earlier mentioned [16] for a thorough analysis. The biggest issue regarding the normalized Laplacian is the loss of symmetry which the theory so far is built upon. For this reason we introduce another graph Laplacian L_{sym} which to think of as a symmetric representative of the L_{rw}. The main objective of L_{sym} is to simplify the following proofs. For the proof without the symmetric normalized graph Laplacian we again refer to [16].

Definition 6.1. *For a graph G with weight matrix W and degree matrix D the normalized graph Laplacians are*

$$L_{rw} := D^{-1}L = \mathbb{I} - D^{-1}W$$
$$L_{sym} := D^{-\frac{1}{2}}LD^{-\frac{1}{2}} = \mathbb{I} - D^{-\frac{1}{2}}WD^{-\frac{1}{2}}.$$

To use the symmetric variant to our advantage we need to clarify some connections between the two matrices.

Lemma 6.2. *The normalized graph Laplacians L_{rw} and L_{sym} satisfy the following:*

1. *For any given $x \in \mathbb{R}^n$ the quadratic form of L_{sym} is*

$$x^T L_{sym} x = \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}\left(\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}}\right)^2.$$

2. *(λ, x) is an eigenpair of L_{rw} if and only if $(\lambda, D^{\frac{1}{2}}x)$ is an eigenpair of L_{sym}.*
3. *L_{rw} and L_{sym} both have n non-negative, real-valued eigenvalues.*
4. *The smallest eigenvalue of L_{rw} is 0 with corresponding eigenvector e. Furthermore for both normalized graph Laplacians, the multiplicity of the eigenvalue 0 indicates the number of connected components in G.*

We will not include the full proof for all of the mentioned statements as they are proven similarly to the unnormalized case of Lemma 5.2. The only thing that we will cover is the connection of the eigenvalues between the two matrices. Starting from the eigenvalue equation (1) of L_{rw} we conclude

$$L_{rw}x = \lambda x \Leftrightarrow D^{-1}Lx = \lambda x \Leftrightarrow D^{\frac{1}{2}}D^{-1}Lx = \lambda D^{\frac{1}{2}}x.$$

To include L_{sym} in the equation we insert the identity matrix in form of $\mathbb{I} = D^{-\frac{1}{2}}D^{\frac{1}{2}}$, resulting in

$$\Leftrightarrow D^{-\frac{1}{2}}LD^{-\frac{1}{2}}D^{\frac{1}{2}}x = L_{sym}D^{\frac{1}{2}}x = \lambda D^{\frac{1}{2}}x.$$

6.2. Normalisation of NCut

With almost identical techniques as used regarding RatioCut we can rewrite the problem of minimizing NCut using another discretely defined vector x. We once again begin with the case of a bipartition.

Theorem 6.3. *Given a graph G of order n with vertices v_i, $i \in \{1, ..., n\}$ that we want to cluster into the partition $(A, \overline{A})$:*

$$\min_{A \subset V} \mathrm{Ncut}(A, \overline{A}) \quad \Leftrightarrow \quad \min_{A \subset V} x^T L x \quad s.t. \quad Dx \perp e, \quad x^T D x = \mathrm{vol}(V)$$

where x is defined as:

$$x_i := \begin{cases} \sqrt{\mathrm{vol}(\overline{A})/\mathrm{vol}(A)}, & \text{if } v_i \in A \\ -\sqrt{\mathrm{vol}(A)/\mathrm{vol}(\overline{A})}, & \text{if } v_i \in \overline{A} \end{cases}$$

Proof. Using similar techniques as in Theorem 5.7 we showcase that

$$\begin{aligned} x^T L x &\overset{5.2}{=} \frac{1}{2}\sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2 \\ &= \frac{1}{2}\left(\sum_{i\in A, j\in\overline{A}} w_{ij}\left(\sqrt{\frac{\mathrm{vol}(\overline{A})}{\mathrm{vol}(A)}} + \sqrt{\frac{\mathrm{vol}(A)}{\mathrm{vol}(\overline{A})}}\right)^2 + \sum_{i\in\overline{A}, j\in A} w_{ij}\left(-\sqrt{\frac{\mathrm{vol}(A)}{\mathrm{vol}(\overline{A})}} - \sqrt{\frac{\mathrm{vol}(\overline{A})}{\mathrm{vol}(A)}}\right)^2\right) \\ &= \frac{1}{2}\left(\sum_{i\in A, j\in\overline{A}} w_{ij} + \sum_{i\in\overline{A}, j\in A} w_{ij}\right)\left(\frac{\mathrm{vol}(\overline{A})}{\mathrm{vol}(A)} + \frac{\mathrm{vol}(A)}{\mathrm{vol}(\overline{A})} + 2\right) \\ &= \mathrm{Cut}(A, \overline{A})\left(\frac{\mathrm{vol}(V)}{\mathrm{vol}(A)} + \frac{\mathrm{vol}(V)}{\mathrm{vol}(\overline{A})}\right) = \mathrm{vol}(V) \cdot \mathrm{NCut}(A, \overline{A}). \end{aligned}$$

Along with our given x we compute

$$(Dx)^T e = \sum_{i\in A} d_i\sqrt{\frac{\mathrm{vol}(\overline{A})}{\mathrm{vol}(A)}} - \sum_{i\in\overline{A}} d_i\sqrt{\frac{\mathrm{vol}(A)}{\mathrm{vol}(\overline{A})}} = \sqrt{\mathrm{vol}(A)\mathrm{vol}(\overline{A})} - \sqrt{\mathrm{vol}(\overline{A})\mathrm{vol}(A)} = 0.$$

For the last constraint we conclude

$$\begin{aligned} x^T D x &= \sum_{i=1}^{n} d_i x_i^2 = \frac{\mathrm{vol}(\overline{A})}{\mathrm{vol}(A)}\sum_{i\in A} d_i + \frac{\mathrm{vol}(A)}{\mathrm{vol}(\overline{A})}\sum_{i\in\overline{A}} d_i \\ &= \mathrm{vol}(A)\frac{\mathrm{vol}(\overline{A})}{\mathrm{vol}(A)} + \mathrm{vol}(\overline{A})\frac{\mathrm{vol}(A)}{\mathrm{vol}(\overline{A})} = \mathrm{vol}(\overline{A}) + \mathrm{vol}(A) = \mathrm{vol}(V). \end{aligned}$$

□

Transforming this optimization problem into the known form of equation (2) we remove the constraint of the discrete values for x. Now our L_{sym} comes into play as by substitution $g := D^{\frac{1}{2}}x$ our term becomes

$$\min_{g\in\mathbb{R}^n} g^T L_{sym} g \quad \text{s.t.} \quad g \perp D^{\frac{1}{2}}e, \quad x^T D^{\frac{1}{2}} D^{\frac{1}{2}} x = ||g|| = \text{vol}(V).$$

With $D^{\frac{1}{2}}e$ being the first eigenvector of L_{sym} according to Lemma 6.2(2,4) this is the optimization problem of equation (2) that is solved by the second smallest eigenvector g^2 of L_{sym}. In retrospect our pre-substituted problem

$$\min_{x\in\mathbb{R}^n} x^T L x \quad \text{s.t.} \quad Dx \perp e, \quad x^T D x = \text{vol}(V)$$

is therefore optimized by x^2 in $g^2 = D^{\frac{1}{2}}x^2$ which is the second smallest eigenvector of L_{rw} according to Lemma 6.2(2).

Continuing with the case of more than two desired clusters we obtain similar results.

Theorem 6.4. *For any given $k > 2$, let G be a graph of order n that we want to partition into $A = (A_1, A_2, ..., A_k)$. Then if D is the graphs degree matrix*

$$\min_A \text{NCut}(A) \Leftrightarrow \min_A \text{tr}(H^T L H) \quad \text{s.t.} \quad H^T D H = \mathbb{I}$$

where $H \in \mathbb{R}^{n\times k}$ is defined as

$$h_{ij} := \begin{cases} 1/\sqrt{\text{vol}(A_j)}, & \text{if } v_i \in A_j \\ 0, & \text{if } v_i \in \overline{A_j} \end{cases}.$$

Proof. Firstly, we can rewrite each diagonal element $h_j L h_j$ using the Cut definition as

$$\begin{aligned} h_j^T L h_j &\overset{5.2}{=} \frac{1}{2}\sum_{s,t=1}^{n} w_{st}(h_{sj} - h_{tj})^2 \\ &= \frac{1}{2}\left(\sum_{s\in A_j, t\in\overline{A_j}} w_{st}\left(\frac{1}{\text{vol}(A_j)}\right) + \sum_{s\in\overline{A_j}, t\in A_j} w_{st}\left(\frac{1}{\text{vol}(A_j)}\right)\right) \\ &= \frac{1}{2}\left(\sum_{s\in A_j, t\in\overline{A_j}} w_{st} + \sum_{s\in\overline{A_j}, t\in A_j} w_{st}\right)\frac{1}{\text{vol}(A_j)} = \frac{\text{Cut}(A_j, \overline{A_j})}{\text{vol}(A_j)}. \end{aligned}$$

Putting everything together we achieve

$$\text{tr}(H^T L H) = \sum_{j=1}^{k}(H^T L H)_{jj} = \sum_{j=1}^{k} h_j^T L h_j = \sum_{j=1}^{k}\frac{\text{Cut}(A_j, \overline{A_j})}{\text{vol}(A_j)} = \text{NCut}(A).$$

Verifying our only constraint we compute

$$h_j^T D h_j = \sum_{v_i \in A_j} (h_j)_i^2 d_i = \sum_{v_i \in A_j} \frac{d_i}{\mathrm{vol}(A_j)} = \frac{1}{\mathrm{vol}(A_j)} \sum_{v_i \in A_j} d_i = 1,$$

while each off diagonal cell of $H^T DH$ is 0 as each vertex can only belong to one cluster and therefore one of each coordinates during multiplication is always 0. Additionally, D only functions as a scaling factor. □

By relaxing and substituting $R = D^{\frac{1}{2}} H$ we get the following problem:

$$\min_{T \in \mathbb{R}^{n \times k}} \mathrm{tr}(R^T L_{sym} R) \quad \text{s.t.} \quad R^T D^{-\frac{1}{2}} D D^{-\frac{1}{2}} R = R^T R = \mathbb{I}.$$

which is solved by identical argumentation as in Theorem 5.9 by the matrix containing the k eigenvectors corresponding to the smallest eigenvalues of L_{sym}. The pre-substituted problem of

$$\min_{H \in \mathbb{R}^{n \times k}} \mathrm{tr}(H^T L H) \quad \text{s.t.} \quad H^T D H = \mathbb{I}$$

is solved, using identical argumentation as in Theorem 6.3, by the k eigenvectors corresponding to the smallest k eigenvalues of L_{rw}.

6.3. Random Walk Theory

The name L_{rw} arises from the close connection to the statistical field of random walks. The idea of minimizing the NCut can be equally interpreted as minimizing the chance of transitioning to another cluster while randomly walking on the edges of a graph. We want to mathematically grasp the random walk and display the parallelism to the NCut function and with that justify the spectral clustering approach using L_{rw}.

The concept of a random walk is a sequence of single edge walks from vertex to vertex on a given graph. While located on vertex v_i the chance of walking to another vertex v_j is given by the probability

$$p_{ij} = \frac{w_{ij}}{d_i}.$$

For a whole graph with order n the transition matrix $T \in \mathbb{R}^{n \times n}$ displays all possibilities for an edge walk and is given by

$$T(G) = D^{-1} W.$$

Immediately the connection between T and L_{rw} strikes as $L_{rw} = \mathbb{I} - T$. The probabilities of transitioning are not symmetrical with different degrees and therefore neither T nor L_{rw} are necessarily symmetrical.

Considering a connected, non-bipartite graph where a random walk is taking place. If at any given moment, we freeze the walk in its current position, the possibility for the current location of the random walk can be described by the unique stationary distribution. The chance of the walk currently being located in the vertex v_i is given by

$$\pi_i = \frac{d_i}{\mathrm{vol}(V)}.$$

Lemma 6.5. *Let G be a connected and non-bipartite graph. Given a random walk $(X_t)_{t\in\mathbb{R}}$ that started in the previously defined unique stationary distribution π. For $A, \overline{A} \subset V$ the probability of the random walk, if currently located on any vertex in A, to transition subset to $\overline{A}$ in a single edge walk is denoted by $P(\overline{A}|A) := P(X_1 \in \overline{A}|X_0 \in A)$. The NCut function is given by*

$$\mathrm{NCut}(A, \overline{A}) = \frac{1}{2}(P(\overline{A}|A) + P(A|\overline{A})).$$

Proof. The only statistic background needed is the definition of a conditional probability. The probability that event A occurs when event B is already fulfilled is given by the quotient of the probability of both events occurring and the probability of B:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

Firstly, analysing the numerator with the joint possibility as

$$\begin{aligned} P(X_1 \in \overline{A} \cap X_0 \in A) &= \sum_{v_i \in A, v_j \in \overline{A}} P(X_0 = v_i, X_1 = v_j) = \sum_{i \in A, j \in \overline{A}} \pi_i p_{ij} \\ &= \sum_{i \in A, j \in \overline{A}} \frac{d_i}{\mathrm{vol}(V)} \frac{w_{ij}}{d_i} = \frac{1}{\mathrm{vol}(V)} \sum_{i \in A, j \in \overline{A}} w_{ij}. \end{aligned}$$

Now we can compute the sum of the conditional probability of transitioning between subsets A and $\overline{A}$ by

$$\begin{aligned} P(\overline{A}|A) + P(A|\overline{A}) &= \frac{P(X_1 \in \overline{A} \cap X_0 \in A)}{P(X_0 \in A)} + \frac{P(X_1 \in A \cap X_0 \in \overline{A})}{P(X_0 \in \overline{A})} \\ &= \frac{1}{\mathrm{vol}(V)} \left(\sum_{i \in A, j \in \overline{A}} w_{ij} \left(\frac{\mathrm{vol}(A)}{\mathrm{vol}(V)} \right)^{-1} + \sum_{i \in \overline{A}, j \in A} w_{ij} \left(\frac{\mathrm{vol}(\overline{A})}{\mathrm{vol}(V)} \right)^{-1} \right) \\ &= \frac{\sum_{i \in A, j \in \overline{A}} w_{ij}}{\mathrm{vol}(A)} + \frac{\sum_{i \in \overline{A}, j \in A} w_{ij}}{\mathrm{vol}(\overline{A})} = 2 \cdot \mathrm{NCut}(A, \overline{A}). \end{aligned}$$

□

In Conclusion, minimizing NCut lowers the chance of leaving our cluster by randomly walking towards an adjacent vertex based on the similarity. Alternatively, this means to minimize the chance of finding connections of similarity between data that is located in different clusters.

With that, the minimization of NCut is also statistically justified to solve the ideal clustering problem. The topic of random walks is thoroughly explained in [11] with concepts of Markov chains and the commute distance introduced.

7. Conclusion

Spectral clustering has been researched and improved since the seventies and has now been developed into a massive scientific field. Established connections expand into many different linked areas. Some of these include statistics [11], image segmentation [16] and many others that we did not mention in this thesis. There are endless publications on the topic of spectral clustering that showcase its usage for a variety of causes.

We mostly worked with the unnormalized graph Laplacian to construct an algorithm that showed improvements to a clustering carried out by k-means. This is mostly due to it not relying on the convexity of metric distances. Furthermore, the spectral clustering approach is not a highly complex process and only relies on basic linear algebra. We mentioned several justifications on the spectral clustering approach further cementing its usefulness. The graph Laplacian includes all the information a graph has to offer and we mostly utilized its eigenvalues to harvest different information regarding the graph. Lastly, we discussed possible enhancements that can be achieved with automatic detection of the number of clusters and the normalized graph Laplacians. For a wider comparison including more examples and different clustering algorithm we refer to [15].

Altogether, spectral clustering appears highly promising for its purposes. This is predominantly due to its ability to break down the issue of finding the ideal clusters on a data set to an extremely well known and optimized eigenvalue problem. It also benefits from its minimal requirements in regards to the input data, making it versatilely applicable, with the ability to alter results based on personal preference.

A. Complete Algorithms

For individual testing, possible improvements and the general comprehension we want to list the complete code used for figure 3, 4 and 8.

A.1. Full Code - Spectral Clustering

Starting off we present the implementation for the spectral clustering results in figure 8. We make use of the packages NumPy, SciPy, Matplotlib and the k-means implementation of [2].

```
import matplotlib.pyplot as plt
import numpy as np
from scipy import sparse
from sklearn.cluster import KMeans

#parameters for graphs, data sets and clusters
np.random.seed(763236924)
k2Circ = 2
k4ND = 4
k5blob = 5
vertices2Circ = 400
vertices4ND = 400
vertices5blob = 250
radius = 5
sigma = 3
epsilon = 0.6
neighbours = 20

#the Gaussian similarity function
def similarity(x, y):
    return np.exp(- (np.linalg.norm(x-y))**2/(2*sigma))

#create the fully connected graph by filling in the similarities.
def makeFCGraph(X):
    nodes = np.shape(X)[0]
    A = np.zeros((nodes,nodes))
    for i in range(nodes):
        for j in range(nodes):
            A[i,j] = similarity(np.array(X[i,:].T),
                                np.array(X[j,:].T))
    A -= np.identity(nodes)
    return A

#create epsilon-neighbour graph by making each entry of the FC
#graph 1 if the similarity is higher than epsilon.
def makeENGraph(X):
    A = makeFCGraph(X)
```

```
    A = np.where(A >= epsilon, 1, 0)
    return A

#create k-nearest neighbour graph by filling both corresponding
#entries for the nearest k(neighbours) with a 1.
def makekNNGraph(X, neighbours):
    nodes = np.shape(X)[0]
    A = makeFCGraph(X)
    B = np.zeros((nodes,nodes))
    for i in range(nodes):
        index = np.argpartition(A[:,i], nodes-neighbours)
        for j in index[nodes - neighbours:]:
            B[i,j] = 1
            B[j,i] = 1
    return B

#compute the graph Laplacian based upon the weight matrix
def getGraphLaplacian(A):
    nodes = np.shape(A)[0]
    A *= -1
    for i in range(nodes):
        A[i,i] = -np.sum(A[:,i])
    return A

#example of one circle surrounded by a circular track. The
#radius decides the distance to the central line of the other track.
def twocircPoints(nodes):
    X = np.zeros((nodes,2))
    for i in range(nodes):
        a = np.random.random()
        b = np.random.random()
        if a < 0.5:
            X[i][0] = np.random.normal(0, 1)
            X[i][1] = np.random.normal(0, 1)
        else:
            X[i][0] = np.random.normal(radius*np.sin(2*np.pi*b), 0.5)
            X[i][1] = np.random.normal(radius*np.cos(2*np.pi*b), 0.5)
    return X

#example of the four normal distributions created by two centres of
#distribution on two axis.
def dnDPoints(nodes):
    X = np.zeros((nodes,2))
    for i in range(nodes):
        a = np.random.random()
        b = np.random.random()
        if a < 0.5:
```

```
            X[i][0] = np.random.normal(7.5, 1)
        else:
            X[i][0] = np.random.normal(2.5, 1)
        if b < 0.5:
            X[i][1] = np.random.normal(7.5, 1)
        else:
            X[i][1] = np.random.normal(2.5, 1)
    return X

def spectralClustering(A, k):
    L = sparse.csr_matrix(getGraphLaplacian(A))
    L = L.asfptype()
    lamb, eigv = sparse.linalg.eigsh(L, k = k, which='SM')
    kmeans = KMeans(n_clusters=k, random_state=2).fit(eigv)
    return kmeans.labels_

#example of the 5 blobs
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=vertices5blob, centers=5,
                       cluster_std=0.60, random_state=42352)

#for each example with each graph compute the labels given by spectral
#clustering and plot them in different colours.
A = makeFCGraph(X)
plt.figure(1)
plt.scatter(X[:,0], X[:,1], s=20, c = spectralClustering(A,k5blob))

A = makeENGraph(X)
plt.figure(2)
plt.scatter(X[:,0], X[:,1], s=20, c = spectralClustering(A,k5blob))

A = makekNNGraph(X, neighbours)
plt.figure(3)
plt.scatter(X[:,0], X[:,1], s=20, c = spectralClustering(A,k5blob))

Y = twocircPoints(vertices2Circ)

A = makeFCGraph(Y)
plt.figure(4)
plt.scatter(Y[:,0], Y[:,1], s=20, c = spectralClustering(A,k2Circ))

A = makeENGraph(Y)
plt.figure(5)
plt.scatter(Y[:,0], Y[:,1], s=20, c = spectralClustering(A,k2Circ))

A = makekNNGraph(Y, neighbours)
plt.figure(6)
```

```
plt.scatter(Y[:,0], Y[:,1], s=20, c = spectralClustering(A,k2Circ))

Z = dnDPoints(vertices4ND)

A = makeFCGraph(Z)
plt.figure(7)
plt.scatter(Z[:,0], Z[:,1], s=20, c = spectralClustering(A,k4ND))

A = makeENGraph(Z)
plt.figure(8)
plt.scatter(Z[:,0], Z[:,1], s=20, c = spectralClustering(A,k4ND))

A = makekNNGraph(Z, neighbours)
plt.figure(9)
plt.scatter(Z[:,0], Z[:,1], s=20, c = spectralClustering(A,k4ND))
```

A.2. Full Code - K-means

For the examples regarding k-means we used the packages NumPy, Matplotlib and the pairwise_distances_argmin method of [2].

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import pairwise_distances_argmin

#parameters for graphs, data sets and clusters
np.random.seed(763236924)
maxSteps = 30
radius = 5
verticesblobs = 250
vertices2Circ = 400
vertices4ND = 400

#k-means body
def k_means(X, k):
    N,n = np.shape(X)
    centers = rngCenters(X, n, k)

    for i in range(maxSteps):
        old_centers = centers
        labels = pairwise_distances_argmin(X, centers)
        centers = newCenters(X, k, labels, centers, n, N)
        if np.all(old_centers == centers):
            break
    return (labels, centers)

#assignment of new cluster centres as the mean of all vertices
```

```
#belonging to that cluster.
def newCenters(X, k, labels, centers, n, N):
    new_centers = np.zeros((n,k))
    cnt = np.zeros(k)
    for i in range(N):
        new_centers[:,labels[i]] += X[i]
        cnt[labels[i]] += 1
    for j in range(k):
        if cnt[j] == 0:
            new_centers[:,j] = centers[j,:].T
            cnt[j] = 1
    new_centers = np.divide(new_centers, cnt)
    return new_centers.T

#first initialization of random cluster centres based upon
#the extremal values for all vertices.
def rngCenters(X, n, k):
    uborder = np.max(X, axis=0)
    lborder = np.min(X, axis=0)
    centers = np.random.random((k,n))
    centers = np.multiply(centers, uborder-lborder)
    centers = np.add(centers, lborder)
    return centers

#example of one circle surrounded by a circular track. The
#radius decides the distance to the central line of the other track.
def twocircPoints(N):
    X = np.zeros((N,2))
    for i in range(N):
        a = np.random.random()
        b = np.random.random()
        if a < 0.5:
            X[i][0] = np.random.normal(0, 1)
            X[i][1] = np.random.normal(0, 1)
        else:
            X[i][0] = np.random.normal(radius*np.sin(2*np.pi*b), 0.5)
            X[i][1] = np.random.normal(radius*np.cos(2*np.pi*b), 0.5)
    return X

#example of the four normal distributions created by two centres of
#distribution on two axis.
def dnDPoints(N):
    X = np.zeros((N,2))
    for i in range(N):
        a = np.random.random()
        b = np.random.random()
        if a < 0.5:
```

```
            X[i][0] = np.random.normal(7.5, 1)
        else:
            X[i][0] = np.random.normal(2.5, 1)
        if b < 0.5:
            X[i][1] = np.random.normal(7.5, 1)
        else:
            X[i][1] = np.random.normal(2.5, 1)
    return X

#example of the 5 blobs
from sklearn.datasets import make_blobs
X, y_true = make_blobs(n_samples=verticesblobs, centers=5,
                       cluster_std=0.60, random_state=42352)

#for each example k-means gives us the cluster labels and their
#cluster centre which we plot in different colours
labels, centers = k_means(X, 5)
plt.figure(1)
plt.scatter(X[:,0], X[:,1], s=20, c = labels)
plt.scatter(centers[:, 0], centers[:, 1], c = 'red',
            s=20)

Y = twocircPoints(vertices2Circ)
labels, centers = k_means(Y, 2)
plt.figure(2)
plt.scatter(Y[:,0], Y[:,1], s=20, c = labels)
plt.scatter(centers[:, 0], centers[:, 1], c = 'red',
            s=20)

Z = dnDPoints(vertices4ND)
labels, centers = k_means(Z, 4)
plt.figure(3)
plt.scatter(Z[:,0], Z[:,1], s=20, c = labels)
plt.scatter(centers[:, 0], centers[:, 1], c = 'red',
            s=20)
```

References

[1] 3Blue1Brown. Essence of linear algebra. https://www.youtube.com/watch?v=fNk_zzaMoSs&list=PLZHQObOWTQDPD3MizzM2xVFitgF8hE_ab&ab_channel=3Blue1Brown.

[2] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[3] R. H. Coase. How should economists choose. *Ideas, Their Origins and Their Consequences: Lectures to Commemorate the Life and Work of G. Warren Nutter*, 63, 1988.

[4] M. Dür. Nichtlineare und kombinatorische Optimierung, 2021, unpublished.

[5] M. Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

[6] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Mathematical Journal*, 25(4):619–633, 1975.

[7] M. Fiedler. Laplacian of graphs and algebraic connectivity. *Banach Center Publications*, 1(25):57–70, 1989.

[8] U. Frauenfelder. Lineare Algebra 2, 2021, unpublished.

[9] B. Ghojogh, F. Karray, and M. Crowley. Eigenvalue and generalized eigenvalue problems: Tutorial. *arXiv preprint arXiv:1903.11240*, 2019.

[10] R. A. Horn and C. R. Johnson. *Matrix analysis*. Cambridge university press, 2012.

[11] L. Lovász. Random walks on graphs. *Combinatorics, Paul erdos is eighty*, 2(1-46):4, 1993.

[12] B. Mohar. *Some applications of Laplace eigenvalues of graphs*. Springer, 1997.

[13] C. C. Paige. *The computation of eigenvalues and eigenvectors of very large sparse matrices*. PhD thesis, University of London, 1971.

[14] C. C. Paige. Computational variants of the Lanczos method for the eigenproblem. *IMA Journal of Applied Mathematics*, 10(3):373–381, 1972.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000.

[17] T. Stykel. Einführung in die Numerik, 2022, unpublished.

[18] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17:395–416, 2007.