Low rank approximation techniques for Graph Based Clustering

GENÍS FLORIACH PIGEM

Master degree in Telecommunications Engineering
Date: May 29, 2019
Supervisors: Daniel Palomar, Sandeep Kumar, Montse Nájar
Universitat Politècnica de Catalunya

Abstract

Clustering analysis is one of the main tools for exploratory data analysis, with applications from statistics, image processing, biology to social sciences. Generally, it is used as a process to find meaningful structure, explanatory underlying processes and generative features. Its goal is to group set of objects in such a way that objects in the same group (clusters) are similar to each other (in some sense) whilst objects from different clusters are dissimilar.

One way to perform clustering analysis is to to look at data as a graph, then clustering becomes a graph cutting problem. Within this set of techniques, *Spectral Clustering* stands for its simplicity and great performance. Recently, a new approach [12] radically different from spectral clustering has emerged and it consists on learning a graph that has the desired properties, namely, that it has the desired number of connected components or clusters.

In this thesis we have explored these techniques and we have proposed new ones with the goal of designing an efficient algorithm that can exploit the additional information in directed graphs, with respect to achieve good clustering performance. Experimental results on synthetic datasets exhibit the effectiveness of the proposed methods.

Contents

Chapter 1

Introduction

This thesis is focused on machine learning. More precisely, it tackles the problem of Clustering from a graph cutting point of view. In this section, an introductory approach to the problem is presented as well as the objective and contribution of this thesis.

1.1 Objectives and challenges

The aim of this master thesis is to implement an algorithm that can perform clustering by learning a disconnected graph. Based on the work of [12] who was the first to tackle the clustering problem as graph estimation problem, we expect to extend their work by generalizing their algorithm for non-symmetric matrices, with the hope of getting better results. Considering directed graph or non symmetric graph matrices possess additional problems as many of the nice properties of graph matrices gets lost.

1.2 Contribution

In this work we propose an approach for graph learning based clustering where the main novelty is the use of non-symmetric matrices that allows to better encode the local neighbors of eahc node.

We provide several algorithms that are simple as they only require linear algebra operators and efficient.

Then, we show that our algorithms improves the results on [12].

1.3 Overview

This is the organization of the thesis. Chapter 2 presents work related to ours and that in some cases will be used as base for our methods. In Chapter 3 we describe our method and its variants. Afterwards, the experimental setup, evaluation methods and results are described and discussed in Chapter 4. Finally, in Chapter 5 we summarize this work also providing some possible future work.

Chapter 2

Background

In this section we will formally define Clustering and the main clustering algorithms in the literature.

2.1 Clustering Problem

The clustering problem can be stated as follows: Given a set of data points, group them into clusters so that,

- Points within each cluster are similar to each other
- Points from differents clusters are dissimilar

Note however that the notion of cluster is not precisely defined, when we are clustering we are implicitly making some assumptions on the underlaying distribution of the data. That's why there are so many different clustering models:

- Connectivity models: for example, hierarchical clustering builds models based on distance connectivity.
- Centroid models: for example, the k-means algorithm represents each cluster by a single mean vector.
- Distribution models: clusters are modeled using statistical distributions, such as multivariate normal distributions used by the expectation-maximization algorithm.
- Density models: for example, DBSCAN and OPTICS defines clusters as connected dense regions in the data space.

2.1.1 K-means clustering

k-means is possibly the most well-known clustering algorithm. However, it was originally designed from the signal processing point of view as a method for vector quantization. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a representative of the cluster. This results in a partitioning of the data space into Voronoi cells.

More formally, given set of observations $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$, k-means aims to partition the n observation into $k < n$ sets $\mathbf{S} = \{S_1, \ldots, S_n\}$ so as to minimize the within-cluster sum of squares,

$$\underset{\mathbf{S}}{\operatorname{argmin}} \sum_{i}^{k} \sum_{x \in S_i} \|\mathbf{x} - \mu_i\|^2 \tag{2.1}$$

where μ_i is the mean of points in S_i.

The most common algorithm is Lloyd's algorithm. Given an initial set of k-means, the algorithm alternates between the two following steps:

- Assign each observation to the mean whose nearest (with Euclidean distance).
- Actualize the set of means by assigning the mean of the observations in each cluster.

Note that this problem is NP-hard and the solution will be dependent of the initial k-means

2.2 Graph-based clustering

This set of techniques cast the clustering problem into a graph-cutting problem. Given some notion of similarity $s_{ij} \geq 0$ between all pairs of points $\mathbf{x}_i$ and $\mathbf{x}_j$, we can represent the data in form of the *similarity graph* $G = (V, E)$. Each vertex v_i in this graph represents a data point $\mathbf{x}_i$. Two vertices are connected if the similarity s_{ij} between the corresponding data points $\mathbf{x}_i$ and $\mathbf{x}_j$ is positive or larger than a certain threshold, and the edge is weighted by s_{ij}. Then, the clustering problem is reformulated as: we want to find a partition of the graph such that the edges between different groups have very low weights (which means that points in different clusters are dissimilar from each other) and the edges within a group have high weights (which means that points within the same cluster are similar to each other) [19].

Similarity function

Before thinking how to construct a graph, we need to define a similarity function $s(\mathbf{x}_i, \mathbf{x}_j)$ between pairs of points $\mathbf{x}_i$ and $\mathbf{x}_j$ of our dataset. This similarity function induces the local neighborhoods that we will have later on when we build the graph. Then, we have to make sure we use a "meaningful" function for the application the data comes from. Instead of a similarity function, we can also define a distance function.

The most common case is when data points belong to the euclidean space $\mathbb{R}^d$, then a reasonable similarity function is the Gaussian function $s(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/(2\sigma^2))$. Where the parameter σ should be sensibly chosen.

2.2.1 Graph notation

In this section we will define all the mathematical objects used by graph based clustering techniques.

A graph G is defined by its vertices $V = \{v_i, \ldots, v_n\}$ and its connections or edges E, $G = (V, E)$. In our case, each vertex in the graph v_i represents a data point $\mathbf{x}_i$ and an edge between two vertices (v_i, v_j) represents the similarity or weight w_{ij} between its associated points $(\mathbf{x}_i, \mathbf{x}_j)$. The graph $G = (V, E)$ is undirected which means that $w_{ij} = w_{ji}$ and weighted as the edges are weighted by the similarity of the conneted vertices. The weighted *adjacency matrix* of the graph is the matrix $\mathbf{W} = (w_{ij})_{i,j=1,\ldots,n} \in \mathbb{R}^{n \times n}$.

Definitions

- The degree of a vertex $v_i \in V$ is defined as $d_i = \sum_{j=1}^{n} w_{ij}$.
- The degree matrix is defined as $\mathbf{D} = \text{diag}(d_1, \ldots, d_n)$
- Given a subset of vertices $A \subset V$, we denote its complement $V \setminus A$ by $\bar{A}$.
- The indicator vector $\mathbb{1}_A \in \mathbb{R}^n$ with its entries 0 if $v_i \notin A$ and 1 otherwise.
- A subset $A \subset V$of a graph is connected if any two vertices in A can be joined by a path such that all intermediate points also lie in A.
- A subset A is called a connected component if it is connected and if there are no connections between vertices in A and $\bar{A}$.

- For two sets $A, B \subset V$ we define $W(A, B) = \sum_{i \in A, j \in B} w_{ij}$. Where we introduce the notation $i \in A$ for the set of indices $\{i | v_i \in A\}$. This metric is just the sum of edges that connect two different sets, notice that $W(A, \bar{A}) = 0$.
- To measure the size of subset we can define two metrices,
  - $|A|$ = number of vertices in A.
  - vol$(A) = \sum_{i \in A} d_i$

Directed graph

We will also give a brief overview of directed graphs as they will come up later in the section 3. Directed graph or digraph is also defined by its vertices $V = \{v_i, \ldots, v_n\}$ and its connections or edges E, $G = (V, E)$, the difference is that know edges are directed, i.e the connection that edges indicate is not bidirectional. The first consequence is that the affinity matrix $\mathbf{W}$ is no longer symmetric, $w_{ij} \neq w_{ji}$. This kind of graphs appear a lot in networks such as social networks where each node represents a vertex and the edge may represent that person i follows person j ($w_{ji} = 1$), however, person j may not follow person i ($w_{ij} = 0$), or also the webpages of the internet where edges indicate if one webpage links to another. A very different kind of directed graph is the so called random walk graph, in this case the fact that $w_{ij} \neq w_{ji}$ is used to highlight the importance of node for the other given its neighbour. If node i is only connected to j the weight w_{ji} should be higher, however if node j is connected to N more points apart from i the weight w_{ij} should be lower.

Another important fact is that now we can define two degrees for a vertex v_i, the in-degree where we consider the weight of the edges going to i and out-degree where we consider the weight of the edges leaving i. Similarly there are three ways to define connectivity.

- Strong connectivity: A digraph is strongly connected if any ordered pair of distinct nodes can be joined by a strong path. Where a strong path is a path where we respect the directions of the edges.
- Weak connectivity: A digraph is weakly connected if any pair of distinct nodes can be joined by a weak path. Where a weak path is path where directions of the edges are not respected.
- Quasi strong connectivity: A digraph is quasi strongly connected if for every pair of nodes v_i and v_j there exist a node v_r that can reach both nodes following a strong path.

2.2.2 Different similarity graphs

There are several popular constructions to transform a given set $\mathbf{x}_1, \ldots, \mathbf{x}_n$ of data points with pairwise similarities s_{ij} or pairwise distances d_{ij} into a graph. When constructing similarity graphs the goal is to model the local neighborhood relationships between the data points.

ϵ-neighborhood

All vertices whose pairwise distance d_{ij} is smaller than ϵ get connected. Then, remaining connections could be weighted by s_{ij}, however, this step is normally skipped as weights are roughly of the same scale (at most ϵ). Hence, the ϵ-neighborhood graph is usually considered unweighted.

k-nearest neighbor

We connect vertex v_i with vertex v_j if v_j is among the k-nearest neighbour of v_i, where the k-nearest neighborhood of v_i is the set of k points closer to it. However, this approach may lead to a directed graph as the relationship is not symmetric. In order to make it undirected there are two options, one is to ignore the direction of the edges, then two pairs of points get connected if at least one is in the k-nearest neighbour of the other, the resulting graph is called k-nearest neighbour graph. The second option is to connect only when both points v_i and v_j are in the k-nearest neighbour of each other, the resulting graph is called mutual k-nearest graph. Then, in both cases, the resulting edges should be weighted according to the weights w_{ij}.

Fully connected graph

In that case, we connect all points with positive weight w_{ij} with each other.

2.2.3 Parameters of the similarity graph

In general, when choosing the connectivity parameters ϵ, k, or σ we would like our similarity graph to be connected or "almost" connected and with no isolated vertices. Because otherwise we are already clustering in the similarity graph construction process and we are not using the graph at all to aid good clustering performance.There are some results on the connectivity of random graphs but those are only useful in the limit $n \to \infty$. For example, if n data points are drawn i.i.d. from a density with a connected support, the k -nearest neighbor graph will be connected if k is on the order of $\log(n)$. While being

of theoretical interest, all those results do not really help us for choosing k on a finite sample. However, we will try to give some rules of thumb when choosing this parameters

ϵ-neighborhood

This is very simple, we compute the *minimum spanning tree* of the fully connected graph and choose ϵ as the maximum weight in the tree. The *minimum spanning tree* is defined as the subset of edges that connect all the vertices with the minimum possible total edge weight. This ensures the connectivity of the graph. However, it is important to observe that this may give problems if there are outliers or the data contains tight clusters very far apart from each other.

k-nearest neighbor

For not very large datasets, by just trying with some values we can get this. If the graph is very large, a first approximation could be to choose k in the order of $\log(n)$.

Fully connected graph

Since σ measures the "size" of the neighborhood we can choose σ such that the resulting graph behaves similarly to a k-nearest neighbor or a ϵ-neighborhood. Thus we may choose $\sigma = \epsilon$ or σ equal to the mean distance of a point to its k-th nearest neighbor.

2.2.4 Graph Laplacian

The main tool for all graph based clustering technique are the graph Laplacians. There is a whole field dedicated to the study of those matrices, called spectral graph theory [6]. In this section, we will introduce the properties of this matrices that make it so useful for clustering, mainly, the fact that its eigenvalues and rank are tightly connected to the connectivity of the graph (algebraic connectivity).

In the following, we will assume that G is an undirected weighted graph with weighted adjacency matrix $\mathbf{W} \in \mathbb{R}^{n \times n}$, where $w_{ij} > 0$. When using eigenvalues we will always consider them ordered increasingly. By "the first k eigenvectors" we refer to the eigenvectors corresponding to the k smallest eigenvalues.

There exist many different graph Laplacians, in this section we will define the most common ones,

The Unnormalized Graph Laplacian

The Unnormalized Graph Laplacian matrix is defined as,

$$\mathbf{L} = \mathbf{D} - \mathbf{W} \tag{2.2}$$

An extensive analysis of this matrix can be found here [10]. Next we summarize the most important properties of $\mathbf{L}$ for graph based clustering,

Proposition 1.

1. *For every vector* $\mathbf{f} \in \mathbb{R}^n$

$$\mathbf{f}^T\mathbf{L}\mathbf{f} = \frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}(f_i - f_j)^2 \tag{2.3}$$

2. $\mathbf{L}$ *is symmetric and positive semidefinite*

3. *The smallest eigenvalue of* $\mathbf{L}$ *is* 0*, its corresponding eigenvector is the constant one vector* $\mathbf{1}$.

4. $\mathbf{L}$ *has* n *non-negative, real-valued eigenvalues* $0 = \lambda_1 \leq \lambda_2 \leq \ldots \lambda_n$.

5. *The multiplicity* k *of the eigenvalue* 0 *of* $\mathbf{L}$ *equals the number of connected components* $A_1, \ldots, A_k$ *of the graph.*

Proof.

1. By the definition of $\mathbf{D}$,

$$\begin{aligned}
\mathbf{f}^T\mathbf{L}\mathbf{f} = \mathbf{f}^T\mathbf{D}\mathbf{f} - \mathbf{f}^T\mathbf{W}\mathbf{f} &= \sum_{i=1}^{n} d_i f_i^2 - \sum_{i,j}^{n} f_i f_j w_{ij} = \\
&= \frac{1}{2}(\sum_{i=1}^{n} d_i f_i^2 - 2\sum_{i,j=1}^{n} f_i f_j w_{ij} + \sum_{j=1}^{n} d_j f_j^2) \\
&= \frac{1}{2}(\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} f_i^2 - 2\sum_{i,j=1}^{n} f_i f_j w_{ij} + \sum_{j=1}^{n}\sum_{i=1}^{n} w_{ji} f_j^2) \\
&= \frac{1}{2}\sum_{i,j=1}^{n} w_{i,j}(f_i - f_j)^2
\end{aligned} \tag{2.4}$$

2. Symmetry follows from the symmetry of $\mathbf{D}$ and $\mathbf{W}$, and the positive semidefinite property follows from the property 1 as well as $w_{ij} \geq 0$.

3. It follows from the construction of $\mathbf{L}$, since $\mathbf{D} = \text{diag}(\sum_j w_{ij}) = \text{diag}(\mathbf{W1})$, then the row sum is 0 for all the rows.

4. Non-negativity follows from the property 1, and real-valued eigenvalues from the symmetry of $\mathbf{L}$.

□

For its importance to the graph based clustering technique, we consider this last property separately,

Proposition 2. *The multiplicity k of the eigenvalue 0 of $\mathbf{L}$ equals the number of connected components $A_1, \ldots, A_k$ of the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors $\mathbb{1}_{A_1}, \ldots, \mathbb{1}_{A_k}$ of those components.*

Proof. We start with the case $k = 1$, that is the graph is connected. Assume that $\mathbf{f}$ is an eigenvector with eigenvalue 0. Then by property 1,

$$0 = \mathbf{f}^T \mathbf{L} \mathbf{f} = \sum_{i,j=1}^{n} w_{ij}(f_i - f_j)^2. \tag{2.5}$$

As $w_{ij} > 0$, this sum only vanishes to 0 if all the terms vanish. Therefore, if two vertices v_i and v_j are connected (i.e, $w_{ij} \geq 0$), then f_i has to be equal to f_j. Following this argument, we can see that $\mathbf{f}$ needs to be constant for all vertices which can be connected by a path in the graph. As all vertices of a connected component in an undirecred graph can be connected by a path, $\mathbf{f}$ needs to be constant.Then, in a graph with only one connected component we only have the eigenvector $\mathbf{1}$ as eigenvector with eigenvalue 0, which obviusly, is the indicator vector of the connected component.

The extension to k connected components is straighforward. Without loss of generality we assume that the vertices are ordered according to the component they belong to. In this case, the adjacency matrix $\mathbf{W}$ has a block diagonal form, and the same is true for the matrix $\mathbf{L}$:

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_1 & & \\ & \ddots & \\ & & \mathbf{L}_k \end{bmatrix} \tag{2.6}$$

Each block $\mathbf{L}_i$ is a graph laplacian itself. Then, as it is the case for all block diagonal matrices, the spectrum of $\mathbf{L}$ is given by the union of the spectra of $\mathbf{L}_i$,

and the corresponding eigenvectors of $\mathbf{L}$ are the eigenvectors $\mathbf{L}_i$ padded with 0 at the positions of the other blocks. As each $\mathbf{L}_i$ is a graph Laplacian of a connected graph, we know that every $\mathbf{L}_i$ has eigenvalue 0 with multiplicity 1, and the corresponding eigenvector is the constant one vector on the i-th connected component. Thus, the matrix $\mathbf{L}$ has as many eigenvalues 0 as there are connected components, and the corresponding eigenvectors are the indicator vectors of the connected components. □

The Normalized Graph Laplacians

There exist two normalizations for the graph laplacian matrix,

$$\mathbf{L}_{sym} = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$$
$$\mathbf{L}_{rw} = \mathbf{D}^{-1}\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1}\mathbf{W}$$

These two matrices have similar properties to the ones that the unnormalized laplacian have. The first matrix is denoted $\mathbf{L}_{sym}$ as it is a symmetric matrix, the second one is denoted $\mathbf{L}_{rw}$ as it is closely related to a random walk.

2.2.5 Spectral Clustering

Once we have defined the Laplacian matrix we can explain the spectral clustering technique. Before, explaining how and why it works, let's first define the algorithm. We assume that our data consists of n data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ which can be arbitrary objects. We measure their pairwise similarities $s_{ij} = s(\mathbf{x}_i, \mathbf{x}_j)$ by some similarity function which is symmetric and non-negative, and we denote the corresponding similarity matrix by $\mathbf{S} = (s_{ij})_{i,j=1\ldots n}$.

Algorithm 1 Unnormalized spectral clustering

Input: set of points $\mathbf{x}_1, \ldots \mathbf{x}_n$, Similary matrix $\mathbf{S}$, number of clusters k
Output: Clusters $A_1, \ldots, A_k$

1: Construct a similarity graph by any method and obtain its weighted adjacency matrix $\mathbf{W}$.
2: Compute the unnormalized laplacian $\mathbf{L}$
3: Compute the first k eigenvectors $\mathbf{u}_1, \ldots \mathbf{u}_k$ of the laplacian matrix.
4: Let $\mathbf{U} \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$ as columns.
5: Consider as new set of points the rows of $\mathbf{U}$, $\mathbf{y}_i \in \mathbb{R}^k \;\; i = 1, \ldots, n$
6: Run k-means to cluster the points $\mathbf{y}_i$ into the clusters $A_1, \ldots A_k$.

We also introduce the algorithms using the Normalized graph laplacians,

Algorithm 2 Normalized spectral clustering [15]

Input: set of points $\mathbf{x}_1, \dots \mathbf{x}_n$, Similary matrix $\mathbf{S}$, number of clusters k
Output: Clusters $A_1, \dots, A_k$

1: Construct a similarity graph by any method and obtain its weighted adjacency matrix $\mathbf{W}$.
2: Compute the unnormalized laplacian $\mathbf{L}$
3: Compute the first k generalized eigenvectors $\mathbf{u}_1, \dots \mathbf{u}_k$ of the generalized eigenproblem $\mathbf{Lu} = \lambda \mathbf{Du}$.
4: Let $\mathbf{U} \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $\mathbf{u}_1, \dots, \mathbf{u}_k$ as columns.
5: Consider as new set of points the rows of $\mathbf{U}$, $\mathbf{y}_i \in \mathbb{R}^k \quad i = 1, \dots, n$
6: Run k-means to cluster the points $\mathbf{y}_i$ into the clusters $A_1, \dots A_k$.

This algorithm is very similar to the one presented before, with the exception that we are considering the generalized eigenvectors of $(\mathbf{L}, \mathbf{D})$ which are in fact the eigenvectors of $\mathbf{L}_{rw}$. However, solving the generalized eigenvector problem of $(\mathbf{L}, \mathbf{D})$ is preferred over the eigenvectors of $\mathbf{L}_{rw}$ because $\mathbf{L}_{rw}$ is non-symmetric, unlike $\mathbf{L}$ and $\mathbf{D}$.

There are more algorithms that uses the normalized laplacian $\mathbf{L}_{sym}$, however we will leave it as a reference for the reader [11]. The idea is the same in all the methods, that is, to change the representation of the points $\mathbf{x}_i \in \mathbb{R}^d$ to points $\mathbf{y}_i \in \mathbb{R}^k$ in such a way that it is easier for k-means to identify clusters. More formally, we are projecting the data points onto a subspace, so that similar points are close by (in the euclidean space $\mathbf{R}_k$), and dissimilar points are far apart.

There are three different ways to formally explain why spectral theory works, here we will focus on two of them, graph cutting and perturbation theory point of view.

Graph cut point of view

If we recall, clustering aims to separate points in different groups according to their similarities. If data is given in a graph form, we can restate the problem as follows: we want a partion of the graph such that edges between different groups have low weight and edges within groups have high weight.

Given a similarity graph with adjacency matrix $\mathbf{W}$, the simplest and most direct way to construct a partition of the graph is to solve the *mincut* problem. For a given number k of clusters, the *mincut* problem consists in choosing a

partition $A_1, \ldots, A_k$ which minimizes,

$$\text{cut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{i}^{k} W(A_i, \bar{A}_i) \tag{2.7}$$

The *mincut* problem can be solved efficiently [REFERENCE], however, it often leads to poor solutions, as it tends separate individual vertices from the rest of the graph. To overcome this problem, we can explicitly enforce large sets $A_1, \ldots, A_k$. To do so, we can normalize the cost of each cut by the size of the resulting set. As said before, there are two ways to measure the size of a set, $|A|$ or vol(A), this two measures lead to the RatioCut and Ncut respectively [11] [15],

$$\text{RatioCut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{i}^{k} \frac{W(A_i, \bar{A}_i)}{|A_i|} \tag{2.8}$$

$$\text{Ncut}(A_1, \ldots, A_k) = \frac{1}{2} \sum_{i}^{k} \frac{W(A_i, \bar{A}_i)}{\text{vol}(A_i)} \tag{2.9}$$

This formulation apart from enforcing big sets A_i, it also enforces the size of the sets to be similar as the minimum of the function $\sum_i^k (1/|A_i|)$ is achieved if all $|A_i|$ coincide. Unfortunately, introducing this normalization turns the *mincut* problem into an NP-hard problem. Spectral clustering is a way to relax these problems, relaxing RatioCut leads to the Unnormalized spectral clustering algorithm and relaxing Ncut leads to the normalized spectral clustering.

Approximating RatioCut

In order to relax the RatioCut problem we first have to transform into a more convenient format.

Given a partition V into k sets $A_1, \ldots, A_k$ we define k indicator vectors $\mathbf{h}_j \in \mathbb{R}^n$ $j = 1, \ldots, k$ by,

$$h_{i,j} = \begin{cases} 1/\sqrt{|A_j|} & \text{if} \quad \text{v}_i \in A_j \\ 0 & \text{otherwise} \end{cases} \qquad \text{(i=1, \ldots, n; j=1,\ldots,k)} \tag{2.10}$$

We define $\mathbf{H} = [\mathbf{h}_1, \ldots, \mathbf{h}_k] \in \mathbb{R}^{n \times k}$. Note that the columns of $\mathbf{H}$ are

orthogonal to each other, that is $\mathbf{H}^T\mathbf{H} = \mathbf{I}$. Now, we can see that,

$$\begin{aligned}
\mathbf{h}_j^T\mathbf{L}\mathbf{h}_j &= \frac{1}{2}\sum_{z,t=1}^{n} w_{zt}(h_{jz} - h_{jt}) \\
&= \frac{1}{2}\sum_{z\in A_j, t\in \bar{A}_j}^{n} w_{zt}(\frac{1}{\sqrt{|A_j|}})^2 + \frac{1}{2}\sum_{z\in \bar{A}_j, t\in A_j}^{n} w_{zt}(-\frac{1}{\sqrt{|A_j|}})^2 \\
&= \frac{1}{|A_j|}(\frac{1}{2}\sum_{z\in A_j, t\in \bar{A}_j}^{n} w_{zt} + \frac{1}{2}\sum_{z\in \bar{A}_j, t\in A_j}^{n} w_{zt}) \\
&= \frac{\text{cut}(A_j, \bar{A}_j)}{|A_j|}
\end{aligned} \tag{2.11}$$

Moreover,

$$\mathbf{h}_j^T\mathbf{L}\mathbf{h}_j = (\mathbf{H}^T\mathbf{L}\mathbf{H})_{jj} \tag{2.12}$$

Combaining two previous results,

$$\text{RatioCut}(A_1, \dots A_k) = \sum_{j=1}^{k}\mathbf{h}_j^T\mathbf{L}\mathbf{h}_j = \sum_{j=1}^{k}(\mathbf{H}^T\mathbf{L}\mathbf{H})_{jj} = \text{Tr}\,(\mathbf{H}^T\mathbf{L}\mathbf{H}) \tag{2.13}$$

Finally, we can rewrite the RatioCut problem as,

$$\begin{aligned}
\underset{A_1,\dots,A_k}{\text{minimize}} \quad & \text{Tr}\,(\mathbf{H}^T\mathbf{L}\mathbf{H}) \\
\text{subject to} \quad & \mathbf{H}^T\mathbf{H} = \mathbf{I} \\
& \mathbf{H} \text{ as defined in Eq. (2.10)}
\end{aligned} \tag{2.14}$$

Then, to relax it we just dropped the constraint on the entries of $\mathbf{H}$ and we let $\mathbf{H} \in \mathbb{R}^{n\times k}$,

$$\begin{aligned}
\underset{\mathbf{H}}{\text{minimize}} \quad & \text{Tr}\,(\mathbf{H}^T\mathbf{L}\mathbf{H}) \\
\text{subject to} \quad & \mathbf{H}^T\mathbf{H} = \mathbf{I}
\end{aligned} \tag{2.15}$$

This is the trace minimization problem and the solution is given by the first k eigenvectors of the matrix $\mathbf{L}$. We can see that the matrix $\mathbf{H}$ coincides with the matrix U used in the unnormalized spectral clustering algorithm. Once we have solved the relaxed problem we need to obtain a feasible solution to the original problem, i.e a discrete partition $A_1, \dots, A_k$. That's why we use k-means on the rows of .

For the Ncut algorithm we can do a very similar development that we will skip for brevity.

Perturbation theory point of view

The justification is the followign, if we suppose that the graph is already partitioned and has k connected components we would have that the indicator vectors would span the eigenspace of eigenvalue 0, then $\mathbf{U} = [\mathbb{1}_{A_1}, \mathbb{1}_{A_2}, \ldots, \mathbb{1}_{A_k}] \in \mathbb{R}^{n \times k}$ and we would only have k distinct points $\mathbf{y}_i \in \mathbb{R}^k$ and they would have the form $(0, \ldots, 0, 1, 0, \ldots, 0)^T$ where the position of the 1 indicates the connected components this point belongs to. And all points belonging to the same cluster will coincide. Then, k-means can trivially find the solution by placing a center point of each of the points $(0, \ldots, 0, 1, 0, \ldots, 0)^T$. Now, suppose that the graph does not have exactly k connected components and that there are small edges between points from different clusters, then we can consider the resulting laplacian $\mathbf{L}$ a perturbation of the ideal one. Then, perturbation theory tells us that the eigenvectors of the perturbed laplacian will be very close to the indicator vectors. The resulting points will no longer coincide with $(0, \ldots, 0, 1, 0, \ldots, 0)^T$, but they will be close up to some small term. Therefore, if perturbations are not too large, then k-means algorithm will recover the solution.

We will leave the formal analysis of the perturbation theory approach for the reader [17] [11].

2.2.6 Graph learning Clustering

In this section we introduce the work from [12] which is the starting point for this thesis.

The main concern with the spectral clustering technique is that it is a two-steps approach where first a graph is formed from the data and then various optimization procedures are invoked on this fixed input graph. A disadvantage of this approach is that the final clustering structures are not represented explicitly in the data graph, i.e we use k-means to post-process the results to get the clustering indicators. Then, the motivation is that a strategy in which the optimization phase is allowed to change the data graph could have advantages with respect to the two-steps approach.

Problem Formulation

Having that in mind, the authors aim to learn a graph $\mathbf{S}$ based on a given data graph $\mathbf{A}$ such that the new data graph is more suitable for clustering. Recalling the property 2 of graph laplacians we have,

Proposition. *The multiplicity k of the eigenvalue 0 of the graph laplacian matrix $\mathbf{L}$ is equal to the number of connected components in the graph.*

Hence, if we manage to obtain graph $\mathbf{S}$ such that $\text{rank}(\mathbf{L_S}) = n - k$, we already have partioned the data points into k clusters, without the need of performing k-means or other discretization techniques.

Motivated by this property, the authors propose, given an initial weighted affinity matrix $\mathbf{A}$ learn an additional weighted affinity matrix $\mathbf{S}$, not necessarily symmetric, close to $\mathbf{A}$ under some metric and such that its corresponding laplacian matrix $\mathbf{L_S}$ has rank $n - k$. Important to note that the authors on this paper do not define the weighted affinity matrices $\mathbf{A}$ and $\mathbf{S}$ as symmetric matrices, hence its associated graphs are not undirected, and its laplacians are not properly defined. However, the authors in [12] define the laplacian as the symmetrized version of the weighted affinity matrix,

$$\mathbf{L_S} = \mathbf{D_S} - (\mathbf{S}^T + \mathbf{S})/2$$

where $(\mathbf{D_S})_{ii} = \sum_j (s_{ij} + s_{ji})/2$. And now, the laplacian is symmetric and fullfills all the properties stated before. Under this constraint the learned $\mathbf{S}$ with a proper permutation is a block diagonal matrix and thus we can directly partition the data points into k clusters based on $\mathbf{S}$. Besides that, the authors propose to add the additional constraint on the rows of $\mathbf{S}$,

$$\sum_j^n s_{ij} = 1 \quad j = 1, \ldots, n \tag{2.16}$$

This constraint ensures that there is no row of $\mathbf{S}$ with all zero elements (we are considering graphs with non-negative weights), which would mean that there is an isolated data point.

Finally, we can properly formulate the problem to be solved,

$$\begin{aligned} \underset{\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{S} - \mathbf{A}\|_F^2 \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \\ & \text{rank}(\mathbf{L_S}) = n - k. \end{aligned} \tag{2.17}$$

This problem is not convex, hence it cannot be solved efficiently at a first glance. It its not true that all non-convex problems cannot be solved efficiently but for this case and most of the cases it is so.

It is not convex due to the constraint rank$(\mathbf{L_S}) = n - k$ which is highly non-convex, to see that we just need to apply the definition of a convex set, i.e, a set is convex if the convex combination of any two elements of the set falls within the set, in this case the set is the matrices $\mathbf{R}^{n\times n}$ of rank $= n - k$. A quick example that does not meet this property is the average of two diagonal matrices with zeros in the diagonals in different places. In this case for sure, the averaged matrix has rank higher than the original matrices.

Optimization Algorithm

To tackle this problem the authors propose the following approach, let $\lambda_i(\mathbf{L_S})$ be the i-th smallest eigenvalue of $\mathbf{L_S}$, recall from the properties of the graph laplacian that $\lambda_i \geq 0 \quad \forall i$. Then, the problem (2.17) is equivalent to the following problem for a large enough value of ρ,

$$\begin{aligned} \underset{\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{S} - \mathbf{A}\|_F^2 + 2\rho \sum_{i=1}^{k} \lambda_i(\mathbf{L}_s) \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \end{aligned} \tag{2.18}$$

When ρ is large enough, the optimal solution $\mathbf{S}$ will make the second term $\sum_{i=1}^{k} \lambda_i(\mathbf{L}_s)$ equal to 0 and thus the constraint rank$(\mathbf{L_S}) = n - k$ in the problem (3.1) will be satisfied.

The trick to solve (2.18) is invoking the Ky Fan's Theorem,

$$\sum_{i=1}^{k} \lambda_i(\mathbf{L}_s) = \min_{\mathbf{F}\in\mathbb{R}^{n\times k}, \mathbf{F}^T\mathbf{F}=I} \operatorname{Tr}(\mathbf{F}^T\mathbf{L}_S\mathbf{F})$$

Then, the problem is further equivalent to,

$$\begin{aligned} \underset{\mathbf{S},\mathbf{F}}{\text{minimize}} \quad & \|\mathbf{S} - \mathbf{A}\|_F^2 + 2\rho \operatorname{Tr}(\mathbf{F}^T\mathbf{L}_s\mathbf{F}) \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \\ & \mathbf{F}^T\mathbf{F} = \mathbf{I} \\ & \mathbf{F} \in \mathbb{R}^{n\times k} \end{aligned} \tag{2.19}$$

Comparing with the original problem (2.17), the problem (2.19) looks more easy to solve. However, it is still non-convex as the constraint $\mathbf{F}^T\mathbf{F} = \mathbf{I}$ is

not convex. Nonethelesss, when we fix any of the two variables the resulting problems are solvable.

When $\mathbf{S}$ is fixed, the problem (2.19) becomes,

$$\begin{aligned} \underset{\mathbf{F}}{\text{minimize}} \quad & \operatorname{Tr}(\mathbf{F}^T \mathbf{L}_{\mathbf{S}} \mathbf{F}) \\ \text{subject to} \quad & \mathbf{F}^T \mathbf{F} = \mathbf{I} \\ & \mathbf{F} \in \mathbb{R}^{n \times k} \end{aligned} \tag{2.20}$$

Now, this problem is also not convex as we still have the constraint $\mathbf{F}^T\mathbf{F} = \mathbf{I}$. However, this one is solvable and the solution is given by the k eigenvectors of $\mathbf{L}_{\mathbf{S}}$ associated to the k smallest eigenvalues.

When $\mathbf{F}$ is fixed we have,

$$\begin{aligned} \underset{\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{S} - \mathbf{A}\|_F^2 + 2\rho \operatorname{Tr}(\mathbf{F}^T \mathbf{L}_s \mathbf{F}) \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \end{aligned} \tag{2.21}$$

Problem (2.21) can be solved for each row independently, for the term $\|\mathbf{S} - \mathbf{A}\|_F^2$ it is obvious that it can be split for each row, however, for the term $\operatorname{Tr}(\mathbf{F}^T \mathbf{L}_s \mathbf{F})$ it is more involved,

$$\operatorname{Tr}(\mathbf{F}^T \mathbf{L}_S \mathbf{F}) = \operatorname{Tr}(\mathbf{L}_S \mathbf{F}\mathbf{F}^T) \tag{2.22}$$

$\mathbf{F}\mathbf{F}^T$ is a matrix of the form,

$$\mathbf{F}\mathbf{F}^T = \begin{pmatrix} \|\mathbf{f}_1\|_2^2 & \mathbf{f}_1^T\mathbf{f}_2 & \dots & \mathbf{f}_1^T\mathbf{f}_n \\ \mathbf{f}_2^T\mathbf{f}_1 & \|\mathbf{f}_2\|_2^2 & \dots & \mathbf{f}_2^T\mathbf{f}_n \\ \vdots & \dots & \|\mathbf{f}_i\|_2^2 & \dots \\ \mathbf{f}_n^T\mathbf{f}_1 & \mathbf{f}_n^T\mathbf{f}_2 & \dots & \|\mathbf{f}_n\|_2^2 \end{pmatrix} \tag{2.23}$$

where $\mathbf{f}_i \in \mathbb{R}^k$ is the i-th row of the matrix $\mathbf{F}$ but in column form.

Then the first term in the diagonal of $2\mathbf{L}_{\mathbf{S}}\mathbf{F}\mathbf{F}^T$ is given by,

$$\sum_{j=1}^{n}(s_{1j}+s_{j1})\|\mathbf{f}_1\|_2^2 - \sum_{j=2}^{n}(s_{1j}+s_{j1})\mathbf{f}_1^T\mathbf{f}_j = \sum_{j=1}^{n}(s_{1j}+s_{j1})(\|\mathbf{f}_1\|_2^2 - \mathbf{f}_1^T\mathbf{f}_j) \tag{2.24}$$

where the last step follows from $s_{ii} = 0$. Finally the trace is given by,

$$\begin{aligned} 2\text{Tr}(\mathbf{L}_s\mathbf{F}\mathbf{F}^T) &= \sum_{i=1}^{n}\sum_{j=1}^{n}(s_{ij}+s_{ji})(\|\mathbf{f}_i\|_2^2 - \mathbf{f}_i^T\mathbf{f}_j) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} s_{ij}\|\mathbf{f}_i\|_2^2 - s_{ij}\mathbf{f}_i^T\mathbf{f}_j + s_{ji}\|\mathbf{f}_i\|_2^2 - s_{ji}\mathbf{f}_i^T\mathbf{f}_j \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} s_{ij}\|\mathbf{f}_i\|_2^2 - s_{ij}\mathbf{f}_i^T\mathbf{f}_j + s_{ij}\|\mathbf{f}_j\|_2^2 - s_{ij}\mathbf{f}_j^T\mathbf{f}_i \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n}\|\mathbf{f}_i - \mathbf{f}_j\|_2^2 s_{ij} \end{aligned} \tag{2.25}$$

Finally, the problem that we have to solve for each row of $\mathbf{S}$, $\mathbf{s}_i$, $\;i = 1, \dots n$, is the following,

$$\begin{aligned} \underset{\mathbf{s}_i}{\text{minimize}} \quad & \sum_{j=1}(s_{ij} - a_{ij})^2 + \rho\sum_j \|\mathbf{f}_i - \mathbf{f}_j\|_2^2 s_{ij} \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \end{aligned} \tag{2.26}$$

It is important to note that the second term is bounded by 0 and it follows from the positive semidefinitess of the symmetrized laplacian $\mathbf{L_S}$.

The objective form in (2.26) can be further compacted, by denoting, $v_{ij} = \|\mathbf{f}_i - \mathbf{f}_j\|_2^2$ and denoting $\mathbf{v}_i$ as a vector with the j-th element equal to v_j, similarly, $\mathbf{s}_i$ and $\mathbf{a}_i$ denote the i-th row of $\mathbf{S}$ and $\mathbf{A}$ respectively,

$$\begin{aligned} \underset{\mathbf{s}_i}{\text{minimize}} \quad & \|\mathbf{s}_i - (\mathbf{a}_i - \frac{\rho}{2})\mathbf{v}_i\|_2^2 \\ \text{subject to} \quad & \mathbf{s}_i^T\mathbf{1} = 1. \\ & \mathbf{s}_i \geq 0. \end{aligned} \tag{2.27}$$

This is a convex problem, as the objective function is convex (quadratic) and the constraints as well (in this case they are linear), more precisely this problem falls in the Quadratic Programming (QP) problems, then we could use any QP solver [16]. However, this problem in particular follow into a narrower set of problems which can be efficiently solved through the lagrange duality and

the dual problem, the solution is given by the water filling solution algorithm widely used in communications.

The water filling algorithm goes as follow, we first construct the lagrangian of the problem 2.27 (the $\frac{1}{2}$ factor is to ease the calculations),

$$\mathcal{L}(\mathbf{s}_i, \eta, \boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{s}_i - (\mathbf{a}_i - \frac{\lambda}{2}\mathbf{v}_i)\|_2^2 - \eta(\mathbf{s}_i^T\mathbf{1} - 1) - \boldsymbol{\beta_i}^T\mathbf{s}_i \tag{2.28}$$

Taking the gradient with respect to $\mathbf{s}_i$ and setting it to 0 yields,

$$\nabla_{\mathbf{s}_i}\mathcal{L} = \mathbf{s}_i - (\mathbf{a}_i - \frac{\lambda}{2}\mathbf{v}_i) - \eta\mathbf{1} - \boldsymbol{\beta_i} = 0 \tag{2.29}$$

$$\mathbf{s}_i^* = (\mathbf{a}_i - \frac{\lambda}{2}\mathbf{v}_i) + \eta\mathbf{1} + \boldsymbol{\beta_i} \tag{2.30}$$

Then for j-th element of $\mathbf{s}_i^*$, we have

$$s_{ij}^* = (a_{ij} - \frac{\lambda}{2}v_{ij}) + \eta + \beta_{ij} \tag{2.31}$$

Now, by the complementary slackness condition, $s_{ij}\beta_{ij}$ = 0, and the fact that $\beta_{ij} \geq 0$ we have,

$$s_{ij}^*\beta_{ij} = 0 \implies \begin{cases} \beta_{ij} & = 0 \implies s_{ij}^* = (a_{ij} - \frac{\lambda}{2}v_{ij}) + \eta \geq 0 \\ s_{ij}^* & = 0 \implies \beta_{ij} = -(a_{ij} - \frac{\lambda}{2}v_{ij}) - \eta \geq 0 \end{cases} \tag{2.32}$$

Then, we can rewrite (2.31) as,

$$s_{ij}^* = (a_{ij} - \frac{\lambda}{2}v_{ij} + \eta)_+ \tag{2.33}$$

Now, we have to find η such that,

$$\sum_{j=1}^{n} s_{ij}^* = \sum_{j=1}^{n}(a_{ij} - \frac{\lambda}{2}v_{ij} + \eta)_+ = 1 \tag{2.34}$$

Equation (2.34) is piecewise linear function of one variable (η) which can be solved easily by bisections method.

Once we know how to solve each subproblem (2.20), and (2.21), a sensible approach would be to solve this two subproblems iteratively updating the variables $\mathbf{S}$ and $\mathbf{F}$ at each step. Note that this procedure has no guarantees to yield the same solution as the original problem whatsoever, even more, it has no guarantees to converge to a valid solution. However, we will see that in general it yields good solutions that meet the required constraints.

Finally we can formalize the constrained laplacian rank algorithm as follows,

Algorithm 3 The constrained graph Laplacian Algorithm

Input: $\mathbf{A} \in \mathbb{R}^{n \times n}$, number of clusters k, a large enough ρ
Output: $\mathbf{S} \in \mathbb{R}^{n \times n}$ with exactly k connected components
1: Initialize $\mathbf{F} \in \mathbb{R}^{n \times k}$ by the k smallest eigenvectors of $\mathbf{L_A}$
2: **while** not converge do **do**
3: Update $\mathbf{S}$ by solving problem (2.21)
4: Update $\mathbf{F}$ by solving problem (2.20)
5: **end while**

Where convergence is achieved when the k smallest eigenvalues of $\mathbf{S}$ are below a certain value ϵ. Once, we have the graph with exactly k connected components obtaining the clusters is trivial, using any graph search strategy such as *bread-first search* or *depth-first search* we can retrieve the components / clusters in linear time.

Initial Graph A

The algorithm introduced above start from an initial graph affinity matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ that should encode the local neighborhoods relationship in our data points, i.e it should be a similarity matrix. We have already presented 3 different similarity matrix (ϵ-neighborhood, k-nearest neighborhood and fully connected) and we could use them as initial graph affinity matrix $\mathbf{A}$. However, the matrix $\mathbf{S}$ that we aim to learn has the restriction of row sum equal to 1, then, it would be desirable that our initial graph also fulfills this requirement. That's why, the authors devise an algorithm to learn an initial affinity matrix.

Given the data points $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, we would like the affinity values of $\mathbf{A}$ such that smaller distance $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2$ between data points $\mathbf{x}_i$ and $\mathbf{x}_j$ corresponds to a large affinity value a_{ij}. Having that in mind, as well as, the row sum constraint, the authors propose to solve the following problem for each of the rows of $\mathbf{A}$,

$$\begin{aligned} \underset{\mathbf{a}_i}{\text{minimize}} \quad & \sum_{j=1}^{n} \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 a_{ij} + \gamma \sum_{j=1}^{n} a_{ij}^2 \\ \text{subject to} \quad & \mathbf{a}_i^T \mathbf{1} = 1. \\ & \mathbf{a}_i \geq 0. \end{aligned} \tag{2.35}$$

Note, that we are solving for each row independently, hence the result is not going to be symmetric. More precisely, the affinity value a_{ij} between points $\mathbf{x}_i$ and $\mathbf{x}_j$ is going to be dependent on the neighborhood of $\mathbf{x}_i$, that means, a_{ij} is going to measure how important is $\mathbf{x}_j$ for $\mathbf{x}_i$ given that $\mathbf{x}_i$ has a certain neighbour, then as the neighbor of $\mathbf{x}_j$ has not to be the same of $\mathbf{x}_i$, in general, $a_{ij} \neq a_{ji}$. Therefore, the matrix $\mathbf{A}$ will represent a directed graph, however, we prefer to work with undirected graphs, that's why when defining the laplacian matrix we symmetrize the affinity matrix by the mean $((a_{ij}+a_{ji})/2)$. This non-symmetry is a direct consequence of the constraint $\mathbf{a}_i^T\mathbf{1} = 1$, and is expected to encode better the local neighborhoods relationship in our data points.

The term $\sum_{j=1}^{n} a_{ij}^2$ is a regularization term. Moreover, we prefer a sparse affinity matrix $\mathbf{A}$ for efficiency and higher performance, that's why we fix $L0$-norm (number of non-zero components) of the vector to a tuning parameter m, $\|\mathbf{a}_i\|_0 = m$, this parameter, controls the number of neighbours of each vertex and therefore directly controls the connectivity of the graph.

Chapter 3

Method

The main contributions in this thesis builds upon the constrained graph laplacian rank algorithm [12] introduced in the previous chapter. Analyzing their approach, the following concerns arises:

- How much information are we losing in the symmetrization step of the graph laplacian matrix $\mathbf{L}$? Can we devise an algorithm that manages to get a solution and does not require this symmetrization?
- The row sum equal to 1 constraint seems rather arbritary. Can we improve it?

During this thesis we focused on the first problem and we proposed several methods, very similar in nature to the one of [12], but that does not require symmetrization.

3.1 Motivation

Recalling the constrained graph laplacian rank algorithm we were aiming to solve the following problem,

$$\begin{aligned} \underset{\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{S}-\mathbf{A}\|_F^2 \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \\ & \text{rank}(\mathbf{L_S}) = n-k. \end{aligned} \tag{3.1}$$

where $\mathbf{A}$ is the initial affinity matrix, $\mathbf{S}$ is the output affinity matrix to be optimized and $\mathbf{L_S}$ is the unnormalized and symmetrized laplacian of $\mathbf{S}$ defined as,

$$\mathbf{L_S} = \mathbf{D} - \frac{\mathbf{S}^T + \mathbf{S}}{2} \tag{3.2}$$

where $\mathbf{D}$ is a diagonal matrix with its entries $d_i = \sum_j \frac{s_{ij}+s_{ji}}{2}$.

Now, the reason for choosing this averaged symmetrization of $\mathbf{L_S}$ over the unsymmetrized one,

$$\begin{aligned} \hat{\mathbf{L}}_\mathbf{S} &= \hat{\mathbf{D}} - \mathbf{S} \\ (\hat{\mathbf{D}})_{ii} &= \sum_j^n s_{ij} \end{aligned} \tag{3.3}$$

are two fold,

- There is a deep theoretical analysis only on the rank of laplacian matrices of undirected graphs, that is, symmetric laplacian matrices. More precisely, the proposition (2) presented before, which relates the rank or multiplicity of the 0 eigenvalue of laplacian matrix to the number of connected components of its associated graph, is only true for undirected graphs.

  If $\mathbf{L_S}$ is not symmetric that statement is not exactly true anymore. However, it is still true that its rank is related to the connectivity of its associated directed graph. For the moment, we will not dive into the mathematical analysis of such relations and we will leave that for a future chapter. It will suffice to say that the rank is related to some sort of connectivity of the directed graph. However, we will leave the references for the reader [1][3].

- The other point is that if $\mathbf{L_S}$ is symmetric, its eigenvalues are real and its eigenvectors orthogonal to each other, plus we also showed that it is positive semidefinite, hence, eigenvalues are also positive. Unfortunately, if $\mathbf{L_S}$ is not symmetric its eigenvalues are complex and its eigenvectors are not orthogonal to each other. Then, the analysis that we did on the previous chapter that allowed as to transform the problem (3.1) into the

problem,

$$\begin{aligned} \underset{\mathbf{S},\mathbf{F}}{\text{minimize}} \quad & \|\mathbf{S}-\mathbf{A}\|_F^2 + 2\rho\,\mathrm{Tr}(\mathbf{F}^T\mathbf{L}_s\mathbf{F}) \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \\ & \mathbf{F}^T\mathbf{F} = \mathbf{I} \\ & \mathbf{F} \in \mathbb{R}^{n\times k} \end{aligned} \tag{3.4}$$

is not valid anymore.

If the symmetric laplacian matrix has so many good properties, one may ask why should we bother to consider the non-symmetric case. A non-symmetric affinity matrix, i.e a directed graph, allows us to embed more information about the local neighborhoods relationships of the data. By using two edges or affinities (a_{ij} and a_{ji}) for each pair of vertices v_i and v_j, we can encode relative importance of one with respect to the other given its neighborhood. More precisely, it allows to represent the following case very conveniently, if v_i is only connected to v_j but v_j is not only connected to v_i but also a set of other vertices, the node v_j should be more important for v_i than v_i is to v_j, hence $a_{ij} > a_{ji}$, where a_{ij} means the effect of vertex j to vertex i. Even more, the non-symmetric affinity matrix $\mathbf{A}$ resembles a lot to the random walk matrix and we have seen it being used successfully in the spectral clustering approach (Ncut [15]), yielding better performance than the unnormalized laplacian (symmetric). Finally, it is to expect that this additional information should lead to better clustering performance.

Having justified the desire to use the non symmetric laplacian, we could try to solve the problem (3.1) for the non-symmetric laplacian $\hat{L}_{\mathbf{S}}$ following similar arguments.

$$\begin{aligned} \underset{\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{S}-\mathbf{A}\|_F^2 \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \\ & \text{rank}(\hat{\mathbf{L}}_{\mathbf{S}}) = n-k. \end{aligned} \tag{3.5}$$

Although the non symmetric laplacian matrix $\hat{\mathbf{L}}$ has no real eigenvalues, it has a similar property to the positive semidefinitess property, which is that real part of all the eigenvalues is non negative. It follows from the fact that the

laplacian matrix (both the symmetric and the non symmetric) are $\mathbf{M}$ matrices [13]. Even more, the number of eigenvalues of real part 0 of the laplacian matrix is tightly related to its connectivity [3]. Then, we could try to devise an algorithm similar to the one in [12] but considering the real part of the k-smallest eigenvalues,

$$\begin{aligned} \underset{\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{S}-\mathbf{A}\|_F^2 + 2\rho \sum_{i=1}^{k} \text{Re}(\lambda_i(\hat{\mathbf{L}}_{\mathbf{S}})) \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \end{aligned} \tag{3.6}$$

However, working with the real part of the complex eigenvalues is not convenient and the analysis gets too involved.

Another trivial extension that one may think of is considering singular values σ_i and the SVD decomposition instead of the eigenvalues and eigenvectors, remember that the rank equals the number of 0 singular values. Hence, similarly to [12] in (2.19), we could consider the following problem,

$$\begin{aligned} \underset{\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{S}-\mathbf{A}\|_F^2 + 2\rho \sum_{i=1}^{k} \sigma_i(\hat{\mathbf{L}}_s) \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \end{aligned} \tag{3.7}$$

And now one, might be tempted to apply a theorem similar to the Ky Fan's theorem and say,

$$\sum_{i=1}^{k} \sigma_i(\hat{\mathbf{L}}_s) = \min_{\mathbf{U}\in\mathbb{R}^{n\times k}, \mathbf{V}\in\mathbb{R}^{n\times k} \mathbf{U}^T\mathbf{U}=I, \mathbf{V}^T\mathbf{V}=I} \text{Tr}(\mathbf{U}^T\hat{\mathbf{L}}_S\mathbf{V})$$

However, that is not true, consider that case that $\mathbf{U}$ and $\mathbf{V}$ are the left and right singular vectors of $\hat{\mathbf{L}}_{\mathbf{S}}$, then $\mathbf{U}^T\mathbf{U} = I, \mathbf{V}^T\mathbf{V} = I$, it is the case that,

$$\sum_{i=1}^{k} \sigma_i(\hat{\mathbf{L}}_s) = \text{Tr}(\mathbf{U}^T\hat{\mathbf{L}}_S\mathbf{V})$$

However, this $\mathbf{U}$ and $\mathbf{V}$ are not the minimizers of the problem,

$$\min_{\mathbf{U}\in\mathbb{R}^{n\times k}, \mathbf{V}\in\mathbb{R}^{n\times k} \mathbf{U}^T\mathbf{U}=\mathbf{I}, \mathbf{V}^T\mathbf{V}=\mathbf{I}} \text{Tr}(\mathbf{U}^T\hat{\mathbf{L}}_S\mathbf{V})$$

Because if we take $\hat{\mathbf{U}} = -\mathbf{U}$, we still have $\hat{\mathbf{U}}^T\hat{\mathbf{U}} = \mathbf{I}$, however,

$$\mathrm{Tr}(\hat{\mathbf{U}}^T\hat{\mathbf{L}}_S\mathbf{V}) = -\sum_{i=1}^{k}\sigma_i(\hat{\mathbf{L}}_\mathbf{S})$$

In this section we have motivated the desire to use the non-symmetric laplacian matrix and have shown that the arguments used by the authors in [12] do not easily extend for the non-symmetric case. In the following sections we give solutions to this problem, by representing the rank of the laplacian matrix by means of a low rank factorization and by means of its singular values σ_i in a slightly different way.

3.2 Proposed methods

In this section we present all the different proposed methods to solve the problem,

$$\begin{aligned} \underset{\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{S}-\mathbf{A}\|_F^2 \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \\ & \text{rank}(\hat{\mathbf{L}}_\mathbf{S}) = n-k. \end{aligned} \tag{3.8}$$

where $\mathbf{S}$ and $\mathbf{A}$ are weighted affinity matrix of a directed graph, then they are not symmetric, $\hat{\mathbf{L}}_\mathbf{S} = \hat{\mathbf{D}} - \mathbf{S}$ with $(\hat{\mathbf{D}})_{ii} = \sum_j^n s_{ij}$, n is the number of vertices in our graph and k the desired number of connected components.

3.2.1 Matrix Factorization: QP Problems

The idea behind this method is based on the following property,

Proposition 3. *Given a matrix $\mathbf{X} \in \mathbb{R}^{n\times n}$ its rank$(\mathbf{X}) \leq r$ if and only if $\mathbf{X}$ can be factorized as $\mathbf{X} = \mathbf{FG}$ where $\mathbf{F} \in \mathbb{R}^{n\times r}$ and $\mathbf{G} \in \mathbb{R}^{r\times n}$.*

Then, instead of imposing the rank constraint on $\hat{\mathbf{L}}_\mathbf{S}$ we can impose that

$\hat{\mathbf{L}}_\mathbf{S} = \mathbf{FG}$,

$$\begin{aligned} \underset{\mathbf{S},\mathbf{F},\mathbf{G}}{\text{minimize}} \quad & \|\mathbf{S} - \mathbf{A}\|_F^2 \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \\ & \hat{\mathbf{L}}_\mathbf{S} = \mathbf{FG} \end{aligned} \tag{3.9}$$

This problem is equally hard as the rank constrained one, because the constraint $\hat{\mathbf{L}}_\mathbf{S} = \mathbf{FG}$ is not-convex in $(\hat{\mathbf{L}}_\mathbf{S}, \mathbf{F}, \mathbf{G})$. However, if we fix either $\mathbf{F}$, or $\mathbf{G}$ the problem is convex in $(\hat{\mathbf{L}}_\mathbf{S}, \mathbf{G})$ or $(\hat{\mathbf{L}}_\mathbf{S}, \mathbf{F})$ respectively. Then, a posible heuristic for (3.9) is solving these two subproblems iteratively,

$$\begin{aligned} \underset{\mathbf{S},\mathbf{F}}{\text{minimize}} \quad & \|\mathbf{S} - \mathbf{A}\|_F^2 \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \\ & \hat{\mathbf{L}}_\mathbf{S} = \mathbf{FG} \end{aligned} \qquad \begin{aligned} \underset{\mathbf{S},\mathbf{G}}{\text{minimize}} \quad & \|\mathbf{S} - \mathbf{A}\|_F^2 \\ \text{subject to} \quad & \sum_j s_{ij} = 1. \\ & s_{ij} \geq 0. \\ & \hat{\mathbf{L}}_\mathbf{S} = \mathbf{FG} \end{aligned} \tag{3.10}$$

However, this two subproblems may be unfeasible, we may not be able to fulfill the affinity matrix constraint at the same time as the equality constraint on $\hat{\mathbf{L}}_\mathbf{S}$. That's why we relax it a bit more by moving the constraint $\hat{\mathbf{L}}_\mathbf{S} = \mathbf{FG}$ to the objective,

$$\underset{\mathbf{S}>0,\ \mathbf{S1}=\mathbf{1},\ \mathbf{G}\in\mathbb{R}^{r\times n}}{\text{minimize}} \|\mathbf{S} - \mathbf{A}\|_F^2 + \lambda\|\hat{\mathbf{L}}_\mathbf{S} - \mathbf{FG}\|_F^2 \tag{3.11}$$

$$\underset{\mathbf{S}>0,\ \mathbf{S1}=\mathbf{1},\ \mathbf{F}\in\mathbb{R}^{n\times r}}{\text{minimize}} \|\mathbf{S} - \mathbf{A}\|_F^2 + \lambda\|\hat{\mathbf{L}}_\mathbf{S} - \mathbf{FG}\|_F^2 \tag{3.12}$$

where the parameter λ controls how "Low rank" we want the solution. Also, note that if the row sum constraint $\sum_j s_{ij} = 1$ has to be satisfied, the laplacian matrix $\hat{\mathbf{L}}_\mathbf{S}$ can be written as follows,

$$\hat{\mathbf{L}}_\mathbf{S} = \mathbf{I} - \mathbf{S} \tag{3.13}$$

Then we can rewrite it as,

$$\underset{\mathbf{S}>0,\ \mathbf{S1}=\mathbf{1},\ \mathbf{G}\in\mathbb{R}^{r\times n}}{\text{minimize}} \|\mathbf{S} - \mathbf{A}\|_F^2 + \lambda\|\mathbf{S} - \mathbf{I} + \mathbf{FG}\|_F^2 \tag{3.14}$$

$$\underset{\mathbf{S}>0,\ \mathbf{S1}=\mathbf{1},\ \mathbf{F}\in\mathbb{R}^{n\times r}}{\text{minimize}} \|\mathbf{S}-\mathbf{A}\|_F^2 + \lambda\|\mathbf{S}-\mathbf{I}+\mathbf{F}\mathbf{G}\|_F^2 \tag{3.15}$$

Having all that in mind, the final algorithm is the following,

Algorithm 4 Factorized graph Laplacian Algorithm

Input: $\mathbf{A}\in\mathbb{R}^{n\times n}$, number of clusters k, a large enough λ, stopping criteria ϵ
Output: $\mathbf{S}\in\mathbb{R}^{n\times n}$ with $\sigma_k(\hat{\mathbf{L}}_\mathbf{S})\leq\epsilon$

1: Define $r=n-k$
2: Compute the SVD decomposition of $\mathbf{A}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$
3: Take $\mathbf{F}^{(0)}=\mathbf{U}_{k+1:n}\mathbf{\Sigma}_{k+1:n}^{1/2}$
4: **while** $\sigma_k(\hat{\mathbf{L}}_\mathbf{S})>\epsilon$ **do**
5: $\quad(\tilde{\mathbf{S}}^{(t)},\mathbf{G}^{(t)})=\underset{\mathbf{S}>0,\ \mathbf{S1}=\mathbf{1},\ \mathbf{G}\in\mathbb{R}^{r\times n}}{\text{argmin}} \|\mathbf{S}-\mathbf{A}\|_F^2+\lambda\|\mathbf{S}-\mathbf{I}+\mathbf{F}^{(t-1)}\mathbf{G}\|_F^2$
6: $\quad(\mathbf{S}^{(t)},\mathbf{F}^{(t)})=\underset{\mathbf{S}>0,\ \mathbf{S1}=\mathbf{1},\ \mathbf{F}\in\mathbb{R}^{n\times r}}{\text{argmin}} \|\mathbf{S}-\mathbf{A}\|_F^2+\lambda\|\mathbf{S}-\mathbf{I}+\mathbf{F}\mathbf{G}^{(t)}\|_F^2$
7: **end while**

where the notation $\mathbf{U}_{k+1:n}$, indicates the last $n-k$ columns of $\mathbf{U}$ and the same for $\mathbf{\Sigma}_{k+1:n}^{1/2}$. Remember that we are considering that singular values are ordered incresingly.

Problems (3.14) and (3.15) fall in the Quadratic Programming problems given that the restrictions are linear and the objective quadratic. To solve it, we could just resort to a modeling software for convex optimization CVX [8]. In this software we can define the optimization problem in the same way that we define it here in (3.14, 3.15) and the software will transform it into a canonical form so that it can be given to a solver [14][18] [5]. However, the solver chosen may be for a more generic problem. That's why, if we know that the problem falls in the Quadratic Programming problems, the most convenient is to choose a solver specfific for QPs. However, transform a problem into the canonical forms is not trivial.

Canonical QP

The canonical form of a QP problem is the following,

$$\begin{aligned}\underset{\mathbf{x}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x}+\mathbf{c}^T\mathbf{x}\\ \text{subject to} \quad & \mathbf{G}\mathbf{x}\succeq\mathbf{h}.\\ & \mathbf{D}\mathbf{x}=\mathbf{b}.\end{aligned} \tag{3.16}$$

where the matrix $\mathbf{Q}$ has to be positive semidefinite otherwise the objective function will not be convex.

To write the first optimization (3.14) problem as a generic QP we define as a decision variable,

$$\mathbf{X} = \begin{bmatrix} \mathbf{S} \\ \mathbf{G} \end{bmatrix} \in \mathbb{R}^{(n+r)\times(n+r)}$$

For convenience we also define,

$$\tilde{\mathbf{F}} = [\mathbf{I}_n \;\; \mathbf{F}] \in \mathbb{R}^{n\times(n+r)}$$

Now, the term $\|\mathbf{S} - \mathbf{I} + \mathbf{FG}\|_F^2$ can be rewritten as,

$$\|\mathbf{S} - \mathbf{I} + \mathbf{FG}\|_F^2 = \|\tilde{\mathbf{F}}\mathbf{X} - \mathbf{I}\|_F^2 \tag{3.17}$$

We can decompose the Frobenious norm as follow,

$$\|\tilde{\mathbf{F}}\mathbf{X} - \mathbf{I}\|_F^2 = \operatorname{Tr}((\tilde{\mathbf{F}}\mathbf{X} - \mathbf{I})^T(\tilde{\mathbf{F}}\mathbf{X} - \mathbf{I})) = \operatorname{Tr}(\mathbf{X}^T\tilde{\mathbf{F}}^T\tilde{\mathbf{F}}\mathbf{X}) - 2\operatorname{Tr}(\tilde{\mathbf{F}}\mathbf{x})$$

Finally, we can decompose the Trace using the kronecker product $\otimes$ and the $\operatorname{vec}(\mathbf{X})$ notation which means stacking the columns of $\mathbf{X}$,

$$\begin{aligned} &\operatorname{Tr}(\mathbf{X}^T\tilde{\mathbf{F}}^T\tilde{\mathbf{F}}\mathbf{X}) - 2\operatorname{Tr}(\tilde{\mathbf{F}}\mathbf{x}) = \\ &= \operatorname{vec}(\mathbf{X})^T(\mathbf{I}_n \otimes \tilde{\mathbf{F}}^T\mathbf{F})\operatorname{vec}(\mathbf{X}) - 2\operatorname{vec}(\tilde{\mathbf{F}}^T)^T\operatorname{vec}(\mathbf{X}) \end{aligned} \tag{3.18}$$

We can make a similar development for the term $\|\mathbf{S} - \mathbf{A}\|_F^2$ but we will skip it for brevety.

For the end result we just need to define some additional matrices,

$$\mathbf{\Phi} = \begin{bmatrix} (1+\lambda)\mathbf{I}_n & \lambda\mathbf{F} \\ \lambda\mathbf{F}^T & \lambda\mathbf{F}^T\mathbf{F} \end{bmatrix} \in \mathbb{R}^{(n+r)\times(n+r)}$$

$$\mathbf{R} = \begin{bmatrix} \mathbf{A}^T + \lambda\mathbf{I}_N & \lambda\mathbf{F} \end{bmatrix} \in \mathbb{R}^{n\times(n+r)}$$

Then the objective function can be written as,

$$\operatorname{vec}(\mathbf{X})^T\left[\mathbf{I}_n \otimes \mathbf{\Phi}\right]\operatorname{vec}(\mathbf{X}) - 2\operatorname{vec}(\mathbf{R}^T)^T\operatorname{vec}(X)$$

Hence,

$$\mathbf{Q} = \mathbf{I}_n \otimes \mathbf{\Phi} \in \mathbb{R}^{(n^2+nr)\times(n^2+nr)}$$

$$\mathbf{c} = -2\operatorname{vec}(\mathbf{R}^T) \in \mathbb{R}^{1\times(n^2+nr)}$$

For the constraints we define,

$$\mathbf{K} = \begin{bmatrix} \mathbf{I}_n & 0_{n\times r} \end{bmatrix} \in \mathbb{R}^{n\times(n+r)}$$

$$\mathbf{D} = \begin{bmatrix} \mathbf{1}_n^T \otimes \mathbf{K} \end{bmatrix} \in \mathbb{R}^{n\times(n^2+nr)}$$

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_n \otimes \mathbf{K} \end{bmatrix} \in \mathbb{R}^{n^2\times(n^2+nr)}$$

$$\mathbf{b} = \mathbf{1}_n \in \mathbb{R}^{n\times 1}$$

$$\mathbf{h} = \mathbf{0}_{n^2} \in \mathbb{R}^{n^2\times 1}$$

The reason for showing this analysis is two fold, first of all, given a convex problem it is not trivial how to turn it into the canonical form, and even more when we are optimizing with respect to matrices, as things get big and messy. The second reason is to show the huge size of the matrices involved, for exemple $\mathbf{Q} \in \mathbb{R}^{(n^2+nr)\times(n^2+nr)}$, and how sparse they are. Then, it is desirable to use a QP solver that allows the use of sparse matrices, luckily we found one in the literature [16].

3.2.2 Matrix Factorization: Block Coordinate descent

We have just presented and heuristic method to obtain a low rank non symmetric laplacian matrix based on the matrix factorization property, and this algorithm requires solving two QP iteratively. In a desire for achieving a faster algorithm with cheaper updates we devise another approach. In this case we are considering $\mathbf{S}, \mathbf{F}, \mathbf{G}$ as three separate agents and we iteratively update them in parallel. In every iteration, we perform a two-steps procedure. First, we determine the descent direction at $[\mathbf{S}^{(k)}, \mathbf{F}^{(k)}, \mathbf{G}^{(k)}]$ by minimizing with respect to each variable independetly from the others. Then, we obtain the solutions $[\hat{\mathbf{S}}^{(k)}, \hat{\mathbf{F}}^{(k)}, \hat{\mathbf{G}}^{(k)}]$. The descent direction is then given by, $[\hat{\mathbf{S}}^{(k)} - \mathbf{S}^{(k)}, \hat{\mathbf{F}}^{(k)} - \mathbf{F}^{(k)}, \hat{\mathbf{G}}^{(k)} - \mathbf{G}^{(k)}]$. In the second step, we obtain the variables for the next iteration $[\mathbf{S}^{(k+1)}, \mathbf{F}^{(k+1)}, \mathbf{G}^{(k+1)}]$ by descending through the previously computed direction with a proper step size.

$$\begin{aligned} [\mathbf{S}^{(k+1)}, \mathbf{F}^{(k+1)}, \mathbf{G}^{(k+1)}] = \\ = [\mathbf{S}^{(k)} + \alpha^{(k)}(\hat{\mathbf{S}}^{(k)} - \mathbf{S}^{(k)}), \\ \mathbf{F}^{(k)} + \alpha^{(k)}(\hat{\mathbf{F}}^{(k)} - \mathbf{F}^{(k)}), \\ \mathbf{G}^{(k)} + \alpha^{(k)}(\hat{\mathbf{G}}^{(k)} - \mathbf{G}^{(k)})] \end{aligned} \tag{3.19}$$

Descent Direction Computation

- If $\mathbf{F}$ and $\mathbf{G}$ are fixed at $\mathbf{F}^{(k)}$ and $\mathbf{G}^{(k)}$ respectively, the optimization problem is,

$$\begin{aligned} \underset{\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{S}-\mathbf{A}\|_F^2 + \lambda\|\mathbf{S}-\mathbf{I}+\mathbf{F}^{(k)}\mathbf{G}^{(k)}\|_F^2 \\ \text{subject to} \quad & \mathbf{S}\mathbf{1}=\mathbf{1}, \\ & s_{ij} \geq 0. \end{aligned}$$

We can separate the problem for each row,

$$\begin{aligned} \underset{\mathbf{s}_i}{\text{minimize}} \quad & \|\mathbf{s}_i-\mathbf{a}_i\|_2^2 + \lambda\|\mathbf{s}_i-\mathbf{k}_i\|_2^2 \\ \text{subject to} \quad & \mathbf{s}_i\mathbf{1}=1, \\ & s_{ij} \geq 0. \end{aligned}$$

where $\mathbf{K}=\mathbf{I}-\mathbf{F}^{(k)}\mathbf{G}^{(k)}$.

To ensure that this method work, the objective function has to be strongly convex. In this case the hessian of the objective function is given by,

$$\mathbf{H}=\mathbf{I}+\lambda\mathbf{I} \succ 0$$

which, given that $\lambda \geq 0$, is strictly positive and hence strongly convex.

This problem is very similar to the one solved in (2.27) which can be solved by the water filling algorithm.

$$\mathcal{L}(\mathbf{s}_i,\eta,\beta)=(1+\lambda)\|\mathbf{s}_i\|_2^2-2(\mathbf{k}_i+\lambda\mathbf{a}_i)\mathbf{s}_i-\eta(\mathbf{s}_i^T\mathbf{1}-\mathbf{1})-\beta^T\mathbf{s}.i$$

Taking the derivative,

$$\nabla_{\mathbf{s}_i}\mathcal{L}(\mathbf{s}_i,\eta,\beta)=2(1+\lambda)\mathbf{s}_i-2(\mathbf{k}_i+\lambda\mathbf{a}_i)-\eta\mathbf{1}-\beta=0$$

By the complementary slackness ($s_{ij}\beta_j=0$), we have that,

$$\hat{\mathbf{s}}_i=\Big(\frac{2(\mathbf{k}_i+\lambda\mathbf{a}_i)+\eta\mathbf{1}}{2(1+\lambda)}\Big)_+$$

where η is given by the following piecewise linear function,

$$\sum_{j=1}^{N} s_{ij}=\sum_{j=1}^{N}(2(k_{ij}+\lambda a_{ij})+\eta)_+=2(1+\lambda)$$

- If $\mathbf{S}$ and $\mathbf{G}$ are fixed at $\mathbf{S}^{(k)}$ and $\mathbf{G}^{(k)}$ respectively, the optimization problem is,

$$\underset{\mathbf{F}}{\text{minimize}} \quad \|\mathbf{S}^{(k)} - \mathbf{I} + \mathbf{F}\mathbf{G}^{(k)}\|_F^2$$

  In this case the objective function is not necessarily strongly convex as the hessian is given by $\mathbf{H} = \mathbf{G}^{(k)}\mathbf{G}^{(k)T} \succeq 0$ (it may have a 0 eigenvalue). That's why we add the smoothing term $\tau\|\mathbf{F} - \mathbf{F}^{(k)}\|_F^2$.

  If we define $\mathbf{B} = \mathbf{I} - \mathbf{S}^{(k)}$, we have,

$$\underset{\mathbf{F}}{\text{minimize}} \quad \|\mathbf{F}\mathbf{G}^{(k)} - \mathbf{B}\|_F^2 + \tau\|\mathbf{F} - \mathbf{F}^{(k)}\|_F^2$$

  It also admits a closed form solution given by,

$$\hat{\mathbf{F}} = (\mathbf{B}\mathbf{G}^{(k)^T} + \tau\mathbf{F}^{(k)})(\mathbf{G}^{(k)}\mathbf{G}^{(k)T} + \tau\mathbf{I})^{-1}$$

- If $\mathbf{S}$ and $\mathbf{F}$ are fixed at $\mathbf{S}^{(k)}$ and $\mathbf{F}^{(k)}$ respectively, the optimization problem is,

$$\underset{\mathbf{G}}{\text{minimize}} \quad \|\mathbf{S}^{(k)} - \mathbf{I} + \mathbf{F}^{(k)}\mathbf{G}\|_F^2$$

  Again, the objective function is not necessarily strongly convex as the hessian is given by $\mathbf{H} = \mathbf{F}^{(k)T}\mathbf{F}^{(k)} \succeq 0$. That's why we add the smoothing term $\tau\|\mathbf{G} - \mathbf{G}^{(k)}\|_F^2$.

  If we define $\mathbf{B} = \mathbf{I} - \mathbf{S}^{(k)}$, we have,

$$\underset{\mathbf{S}}{\text{minimize}} \quad \|\mathbf{F}^{(k)}\mathbf{G} - \mathbf{B}\|_F^2 + \tau\|\mathbf{G} - \mathbf{G}^{(k)}\|_F^2$$

  It also admits a closed form solution given by,

$$\hat{\mathbf{G}} = (\mathbf{F}^{(k)^T}\mathbf{F}^{(k)} + \tau\mathbf{I})^{-1}(\mathbf{F}^{(k)T}\mathbf{B} + \tau\mathbf{G}^{(k)})$$

Step size computation

To ensure convergence, the step size $^{(k)}$ have to meet the following requirements,

$$\begin{gathered}\tau > 0 \\ \alpha^{(k)} \in (0, 1] \\ \sum_k \alpha^{(k)} = +\infty \\ \sum_k \alpha^{(k)2} < +\infty\end{gathered}$$

We have considered two policies for $\alpha^{(k)}$,

$$\alpha^{(k)} = \frac{1}{1+\epsilon k}$$

and,

$$\alpha^{(k+1)} = \alpha^{(k)}(1 - \epsilon\alpha^{(k)})$$

However, in terms of speed convergence there is no significant differences between these two policies.

Backtracking
Instead of fixing the α we can also try to find the most convenient for that particular step.

The algorithm is the following,

- Set $\alpha^{(k)}$
- while

$$J(\mathbf{S}^{(k+1)}, \mathbf{F}^{(k+1)}, \mathbf{G}^{(k+1)}) - J(\mathbf{S}^{(k)}, \mathbf{F}^{(k)}, \mathbf{G}^{(k)}) > -\tau\alpha^{(k)} \| \left[\hat{\mathbf{S}}^{(k)}, \hat{\mathbf{F}}^{(k)}, \hat{\mathbf{G}}^{(k)}\right] - \left[\mathbf{S}^{(k)}, \mathbf{F}^{(k)}, \mathbf{G}^{(k)}\right] \|$$

- $\alpha^{(k)} = \alpha^{(k)}\rho$

where $\rho \in (0, 1)$ is the shrinkage parameter and $\tau > 0$ is the descent parameter.

Exact line search
Another option is to do a grid search over the step size domain and get the step size that leads to lower objective value.

Apart from considering the parallel update, we can update the three variable cyclically, actually, there are many different ways to update them [20]. However, we will only focus on this two,

Algorithm 5 Parallel Block Coordinate Descent

Input: $\mathbf{A} \in \mathbb{R}^{n \times n}$, number of clusters k, a large enough λ, stopping criteria ϵ
Output: $\mathbf{S} \in \mathbb{R}^{n \times n}$ with $\sigma_k(\hat{\mathbf{L}}_{\mathbf{S}}) \leq \epsilon$

Define $r = n - k$
Compute the SVD decomposition of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$
Take $\mathbf{F}^{(0)} = \mathbf{U}_{k+1:n}\mathbf{\Sigma}^{1/2}_{k+1:n}$
Take $\mathbf{G}^{(0)} = \mathbf{\Sigma}^{1/2}_{k+1:n}\mathbf{V}^T_{k+1:n}$
while $\sigma_k(\hat{\mathbf{L}}_{\mathbf{S}}) > \epsilon$ **do**
    $\hat{\mathbf{S}}^{(t)} = \underset{\mathbf{S}>0,\ \mathbf{S1}=\mathbf{1}}{\operatorname{argmin}} \|\mathbf{S} - \mathbf{A}\|_F^2 + \lambda\|\mathbf{S} - \mathbf{I} + \mathbf{F}^{(t-1)}\mathbf{G}^{(t-1)}\|_F^2$
    $\hat{\mathbf{F}}^{(t)} = \underset{\mathbf{F}\in\mathbb{R}^{n\times r}}{\operatorname{argmin}} \|\mathbf{S}^{(t-1)} - \mathbf{A}\|_F^2 + \lambda\|\mathbf{S}^{(t-1)} - \mathbf{I} + \mathbf{F}\mathbf{G}^{(t-1)}\|_F^2$
    $\hat{\mathbf{G}}^{(t)} = \underset{\mathbf{G}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \|\mathbf{S}^{(t-1)} - \mathbf{A}\|_F^2 + \lambda\|\mathbf{S}^{(t-1)} - \mathbf{I} + \mathbf{F}^{(t-1)}\mathbf{G}\|_F^2$
    Estimate step size $\alpha^{(t)}$
    Take the steps

$$\mathbf{S}^{(t+1)} = \mathbf{S}^{(t)} + \alpha^{(t)}(\hat{\mathbf{S}}^{(t)} - \mathbf{S}^{(t)})$$
$$\mathbf{F}^{(t+1)} = \mathbf{F}^{(t)} + \alpha^{(t)}(\hat{\mathbf{F}}^{(t)} - \mathbf{F}^{(t)})$$
$$\mathbf{G}^{(t+1)} = \mathbf{G}^{(t)} + \alpha^{(t)}(\hat{\mathbf{G}}^{(t)} - \mathbf{G}^{(t)})$$

end while

Algorithm 6 Cyclic Block Coordinate Descent

Input: $\mathbf{A} \in \mathbb{R}^{n \times n}$, number of clusters k, a large enough λ, stopping criteria ϵ
Output: $\mathbf{S} \in \mathbb{R}^{n \times n}$ with $\sigma_k(\hat{\mathbf{L}}_{\mathbf{S}}) \leq \epsilon$

Define $r = n - k$
Compute the SVD decomposition of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$
Take $\mathbf{F}^{(0)} = \mathbf{U}_{k+1:n}\mathbf{\Sigma}^{1/2}_{k+1:n}$
Take $\mathbf{G}^{(0)} = \mathbf{\Sigma}^{1/2}_{k+1:n}\mathbf{V}^T_{k+1:n}$
while $\sigma_k(\hat{\mathbf{L}}_{\mathbf{S}}) > \epsilon$ **do**
    $\hat{\mathbf{S}}^{(t)} = \underset{\mathbf{S}>0,\ \mathbf{S1}=\mathbf{1}}{\operatorname{argmin}} \|\mathbf{S} - \mathbf{A}\|_F^2 + \lambda\|\mathbf{S} - \mathbf{I} + \mathbf{F}^{(t-1)}\mathbf{G}^{(t-1)}\|_F^2$
    Estimate step size $\alpha^{(t)}$
    Take the step $\mathbf{S}^{(t)} = \mathbf{S}^{(t-1)} + \alpha^{(t)}(\hat{\mathbf{S}}^{(t)} - \mathbf{S}^{(t-1)})$
    $\hat{\mathbf{F}}^{(t)} = \underset{\mathbf{F}\in\mathbb{R}^{n\times r}}{\operatorname{argmin}} \|\mathbf{S}^{(t)} - \mathbf{A}\|_F^2 + \lambda\|\mathbf{S}^{(t)} - \mathbf{I} + \mathbf{F}\mathbf{G}^{(t-1)}\|_F^2$
    Estimate step size $\alpha^{(t)}$
    Take the step $\mathbf{F}^{(t)} = \mathbf{F}^{(t-1)} + \alpha^{(t)}(\hat{\mathbf{F}}^{(t)} - \mathbf{F}^{(t-1)})$
    $\hat{\mathbf{G}}^{(t)} = \underset{\mathbf{G}\in\mathbb{R}^{r\times n}}{\operatorname{argmin}} \|\mathbf{S}^{(t)} - \mathbf{A}\|_F^2 + \lambda\|\mathbf{S}^{(t)} - \mathbf{I} + \mathbf{F}^{(t)}\mathbf{G}\|_F^2$
    Estimate step size $\alpha^{(t)}$
    Take the step $\mathbf{G}^{(t)} = \mathbf{G}^{(t-1)} + \alpha^{(t)}(\hat{\mathbf{G}}^{(t)} - \mathbf{G}^{(t-1)})$
end while

Both methods are very similar only differentiate on the fact that the updates in algorithm 5 can be taken in parallel and that we only have to estimate one step size α^t per iteration.

3.2.3 Log Determinant Heuristic

This method uses the log determinant as a smooth surrogate for the rank function. Assuming that $\mathbf{X}$ is positive definite, i.e, $\mathbf{X} \in \mathbb{S}^n_{++}$, we have,

$$\log\det(\mathbf{X}) = \log\prod_i^n \lambda_i = \sum_i^n \log(\lambda_i) \tag{3.20}$$

Then, if we wanted to minimize the rank of a matrix $\mathbf{X}$ given some constraints we could solve the following problem,

$$\begin{aligned} \underset{\mathbf{X}}{\text{minimize}} \quad & \log\det(\mathbf{X} + \delta\mathbf{I}) \\ \text{subject to} \quad & \mathbf{X} \in \mathcal{C} \\ & \mathbf{X} \in \mathbf{S}^n_+ \end{aligned} \tag{3.21}$$

where $\mathcal{C}$ is a convex set. However, the $\log\det(\mathbf{X}+\delta\mathbf{I})$ is not convex, it is actually concave. To overcome this problem, an iterative linearization scheme is proposed. If we take the first-order series expansion of $\log\det(\mathbf{X}+\delta\mathbf{I})$ around $\mathbf{X}^{(k)}$ we have,

$$\log\det(\mathbf{X}+\delta\mathbf{I}) \approx \log\det(\mathbf{X}^{(k)}+\delta\mathbf{I}) + \mathrm{Tr}((\mathbf{X}^{(k)}+\delta\mathbf{I})^{-1}(\mathbf{X}-\mathbf{X}^{(k)})) \quad (3.22)$$

Now, at each step k we have to minimize the trace which is linear,

$$\begin{aligned} \underset{\mathbf{X}}{\text{minimize}} \quad & \mathrm{Tr}((\mathbf{X}^{(k)}+\delta\mathbf{I})^{-1}\mathbf{X}) \\ \text{subject to} \quad & \mathbf{X} \in \mathcal{C} \\ & \mathbf{X} \in \mathbf{S}_+^n \end{aligned} \quad (3.23)$$

Unfortunately, this method as it is, is not applicable to our problem as our optimization variable $\mathbf{S}$ is not symmetric and hence not positive semidefinite. Then, we resort to the semidefinite embedding lemma,

Proposition 4. *Let $X \in \mathbb{R}^{m\times n}$ be a given matrix. Then $rank(\mathbf{X}) \leq r$ if and only if there exist matrices $\mathbf{Y} = \mathbf{Y}^T \in \mathbb{R}^{m\times m}$ and $\mathbf{Z} = \mathbf{Z}^T \in \mathbb{R}^{n\times n}$ such that,*

$$\begin{bmatrix} \mathbf{Y} & \mathbf{X} \\ \mathbf{X}^T & \mathbf{Z} \end{bmatrix} \in \mathbb{S}_+^n, \quad rank(\mathbf{Y}) + rank(\mathbf{Z}) \leq 2r$$

Then, instead of minimizing the $\log\det(\mathbf{X})$ we have to minimize the $\log\det(\text{blkdiag}(\mathbf{Y},\mathbf{Z}))$, where again we will have to apply the first-order taylor approximation.

Finally, we can use all these results for our low rank unsymmetric graph laplacian estimation,

$$\begin{aligned} \underset{\mathbf{S},\mathbf{Y},\mathbf{Z}}{\text{minimize}} \quad & \mathrm{Tr}(\text{blkdiag}((\mathbf{Y}^{(k)},\mathbf{Z}^{(k)})+\delta\mathbf{I})^{-1}\text{blkdiag}(\mathbf{Y},\mathbf{Z})) + \lambda\|\mathbf{S}-\mathbf{A}\|_F^2 \\ \text{subject to} \quad & \begin{bmatrix} \mathbf{Y} & \mathbf{I}-\mathbf{S} \\ (\mathbf{I}-\mathbf{S})^T & \mathbf{Z} \end{bmatrix} \succeq 0 \\ & \mathbf{s}_{ij} \geq 0 \\ & \sum_j s_{ij} = 1 \\ & \mathbf{S},\mathbf{Y},\mathbf{Z} \in \mathbb{R}^{n\times n} \end{aligned} \quad (3.24)$$

The algorithm can be described as follows,

Algorithm 7 Log det heuristic

Input: $\mathbf{A} \in \mathbb{R}^{n \times n}$, number of clusters k, a large enough λ, stopping criteria ϵ
Output: $\mathbf{S} \in \mathbb{R}^{n \times n}$ with $\sigma_k(\hat{\mathbf{L}}_\mathbf{S}) \leq \epsilon$

1: Define $r = n - k$
2: Compute the SVD decomposition of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$
3: Take $\mathbf{Y}^{(0)} = \mathbf{I}$ and $\mathbf{Z}^{(0)} = \mathbf{I}$
4: **while** $\sigma_k(\hat{\mathbf{L}}_\mathbf{S}) > \epsilon$ **do**
5: Update $\mathbf{Y}$ and $\mathbf{Z}$ by solving (3.24)
6: **end while**
7: Retrieve $\mathbf{S}$ from the last step.

This method has two drawbacks, first of all we cannot include the exact desired rank in the algorithm, we just have to hope that singular values will converge go to 0 in order, and secondly and most important, it involves solving a Semidefinite Programming (SDP) problem. SDP is the most general of the convex problems, Linear Programming (LP), Quadratic Programming (QP) and Second Order Cone programming (SOCP) can be rewritten as SDPs, but not the other way round. This level of generality comes at expenses of performance and computational cost. The computational complexity of a QP is at worst $O(n^3)$ (where n is the number of decision variables), whilst for the SDP is $O(n^6)$. Apart from the computational complexity there is the space complexity, i.e how much RAM this algorithms need, also in this case the requirements of the SDP are way higher than that's of a QP.

3.2.4 Rank Constraint via Convex Iteration

This method is based on the work from [7], and they considered the following feasibility problem,

$$\begin{aligned} \underset{\mathbf{X}}{\text{find}} \quad & \mathbf{X} \\ \text{subject to} \quad & \mathbf{X} \in \mathcal{C} \\ & \mathbf{X} \in \mathbf{S}_+^n \\ & rank(\mathbf{X}) \leq r \end{aligned} \tag{3.25}$$

And they proposed to solve it via iteratively solving the following two problems,

$$\begin{aligned} \underset{\mathbf{X}}{\text{minimize}} \quad & Tr(\mathbf{W}^*\mathbf{X}) \\ \text{subject to} \quad & \mathbf{X} \in \mathcal{C} \\ & \mathbf{X} \in \mathbf{S}_+^n \end{aligned} \tag{3.26}$$

$$\begin{aligned} \underset{\mathbf{W}\in\mathbb{R}^{n\times n}}{\text{minimize}} \quad & Tr(\mathbf{W}\mathbf{X}^*) \\ \text{subject to} \quad & 0 \preceq \mathbf{W} \preceq \mathbf{I} \\ & Tr(\mathbf{W}) = n - r \end{aligned} \tag{3.27}$$

where $\mathbf{W}^*$ is the optimal solution of the second problem and $\mathbf{X}^*$ is the optimal solution of the first problem.

If we take the spectral decomposition of $\mathbf{X}^* = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T$, the second problem can be solved in close form solution, $\mathbf{W}^* = \mathbf{U}^*\mathbf{U}^{*^T}$ where $\mathbf{U}^*$ has per columns the eigenvectors associated to the $k = n - r$ smallest eigenvalues of $\mathbf{X}^*$, i.e $\mathbf{U}^* = \mathbf{Q}_{1:k} \in \mathbb{R}^{n\times k}$. We will not know show the proof, however, we will show that this solution indeed meets the constraints,

$$\mathbf{W}^* = \mathbf{U}^*\mathbf{U}^{*T} = \mathbf{U}^*\text{diag}(\mathbf{1}_k)\mathbf{U}*^T = [\mathbf{U}^* \;\; \mathbf{0}_{n\times r}]\text{diag}([\mathbf{1}_k, \mathbf{0}_r])[\mathbf{U}^* \;\; \mathbf{0}_{n\times r}]^T \tag{3.28}$$

Then, the eigenvalues of the matrix $\mathbf{W}^* = \mathbf{U}^*\mathbf{U}^{*T}$ are either 0 or 1, hence the constraint $0 \preceq \mathbf{W} \preceq \mathbf{I}$ is met. For the second constraint, we have to note that $\mathbf{X}^*$ is symmetric therefore its eigenvectors are orthogonal, then,

$$\text{Tr}(\mathbf{W}^*) = \text{Tr}(\mathbf{U}^*\mathbf{U}^{*T}) = \text{Tr}(\mathbf{U}^{*T}\mathbf{U}^*) = \text{Tr}(\mathbf{I}_k) = k = n - r \tag{3.29}$$

Besides, the objective function at the optimal value is,

$$\begin{aligned} \text{Tr}(\mathbf{W}^*\mathbf{X}^*) &= \text{Tr}(\mathbf{Q}_{1:k}\mathbf{Q}_{1:k}^T\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T) \\ &= \text{Tr}(\mathbf{Q}_{1:k}^T\mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T\mathbf{Q}_{1:k}) \\ &= \text{Tr}([\mathbf{I}_k \;\; \mathbf{0}_{k\times r}]\mathbf{\Lambda}[\mathbf{I}_k \;\; \mathbf{0}_{k\times r}]^T) \\ &= \text{Tr}(\mathbf{\Lambda}\text{diag}([\mathbf{1}_k, \mathbf{0}_r])) \\ &= \sum_i^k \lambda_i(\mathbf{X}) \end{aligned}$$

Therefore, when we are optimizing with respect to $\mathbf{X}$ in the first problem we are minimizing the sum of k-th smallest eigenvalues of $\mathbf{X}$ in that particular iteration of the algorithm.

If at convergence, $\text{Tr}(\mathbf{W}^*\mathbf{X}) = 0$, then, rank$(\mathbf{X}^*) \leq r$ and so we will have solved the initial feasibility problem (3.30). However, this algorithm is not guaranteed to converge to a solution, even if the problem (3.30) is feasible.

Now, we need to adapt this approach for our case. As our laplacian is not symmetric we have to invoke the Semidefinite embedding lemma again. The

equivalent feasibility problem would be,

$$\begin{aligned} \underset{\mathbf{S},\mathbf{Y},\mathbf{Z}}{\text{find}} \quad & \mathbf{S},\mathbf{Y},\mathbf{Z} \\ \text{subject to} \quad & \begin{bmatrix} \mathbf{Y} & \mathbf{I}-\mathbf{S} \\ \mathbf{I}-\mathbf{S}^T & \mathbf{Z} \end{bmatrix} \succeq 0, \\ & \sum_j s_{ij} = 1 \;\; i = 1,\ldots,n \\ & s_{ij} \geq 0. \\ & \|\mathbf{S}-\mathbf{A}\|_F^2 \leq \lambda \\ & rank(\begin{bmatrix} \mathbf{Y} & \mathbf{0} \\ \mathbf{0} & \mathbf{Z} \end{bmatrix}) \leq 2r \end{aligned} \tag{3.30}$$

Also note, by the Schur complement, that if,

$$\begin{bmatrix} \mathbf{Y} & \mathbf{I}-\mathbf{S} \\ \mathbf{I}-\mathbf{S}^T & \mathbf{Z} \end{bmatrix} \succeq 0$$

Then,

$$\mathbf{Y} \succeq 0 \quad , \quad \mathbf{Z} \succeq 0$$

And obviously,

$$\begin{bmatrix} \mathbf{Y} & \mathbf{0} \\ \mathbf{0} & \mathbf{Z} \end{bmatrix} \succeq 0$$

Hence, the formulation is exactly the same as in [7]. Then the updates will be given by the problems,

$$\begin{aligned} \underset{\mathbf{S},\mathbf{Y},\mathbf{Z}}{\text{minimize}} \quad & \text{Tr}(\mathbf{W}^*\text{blkdiag}(\mathbf{Y},\mathbf{Z})) + \lambda\|\mathbf{S}-\mathbf{A}\|_F^2 \\ \text{subject to} \quad & \begin{bmatrix} \mathbf{Y} & \mathbf{I}-\mathbf{S} \\ \mathbf{I}-\mathbf{S}^T & \mathbf{Z} \end{bmatrix} \succeq 0, \\ & \sum_j s_{ij} = 1 \;\; i = 1,\ldots,n \quad , \\ & s_{ij} \geq 0. \end{aligned} \tag{3.31}$$

$$\begin{aligned} \underset{\mathbf{W}}{\text{minimize}} \quad & \text{Tr}(\mathbf{W}\text{blkdiag}(\mathbf{Y}^*,\mathbf{Z}^*)) \\ \text{subject to} \quad & 0 \preceq \mathbf{W} \preceq \mathbf{I}, \\ & \text{Tr}(\mathbf{W}) = 2(n-r) \end{aligned} \tag{3.32}$$

where we have moved the constraint $\|\mathbf{S}-\mathbf{A}\|_F^2 \leq \lambda$ to the objective function for convenience.

Finally the pseudocode of the algorithm,

Algorithm 8 Rank Constraint via convex iteration

Input: $\mathbf{A} \in \mathbb{R}^{n \times n}$, number of clusters k, a large enough λ, stopping criteria ϵ
Output: $\mathbf{S} \in \mathbb{R}^{n \times n}$ with $\sigma_k(\hat{\mathbf{L}}_{\mathbf{S}}) \leq \epsilon$

1: Take $\mathbf{W}^{*(0)} = \mathbf{I}$
2: **while** $\sigma_k(\hat{\mathbf{L}}_{\mathbf{S}}) > \epsilon$ **do**
3: Update $\mathbf{S}$, $\mathbf{Y}$ and $\mathbf{Z}$ by solving (3.31)
4: Update $\mathbf{W}$ by solving (3.32)
5: **end while**

With this approach we have solved one of the inconvenience of the previous method (log determinat heuristic) that was the we cannot not precisely encode the desired rank in the algorithm. However, we still have the problem that it requires solving a SDP, problem (3.31). In order to try to speed up this algorithm, we explored the dual problem of (3.31) with the hope of finding an algorithm with simpler and faster updates, just as it happens with the water filling algorithm. We did not manage to come up with an easier algorithm to solve the SDP, however, we leave the analysis done here for future references.

Dual Barrier method

In order to derive a simple and efficient tailored implementation we consider the dual problem.

$$\mathcal{L}(\mathbf{S}, \mathbf{Y}, \mathbf{Z}, \mathbf{\Lambda}, \mathbf{\Psi}, \boldsymbol{\mu}) = Tr(\mathbf{W}^* \text{blkdiag}(\mathbf{Y}, \mathbf{Z})) + \lambda \|\mathbf{S} - \mathbf{A}\|_F^2 - Tr(\mathbf{\Psi}\mathbf{K}) - \boldsymbol{\mu}^T(\mathbf{S}\mathbf{1} - \mathbf{1}) - \sum_{i,j} \mathbf{\Lambda}_{ij}\mathbf{S}_{ij}$$

where $\mathbf{\Psi}$ is the dual variable associated to the PSD constraint, $\mathbf{\Lambda}$ is the dual variable associated to the non-negative orthan contraint, $\boldsymbol{\mu}$ is the dual associated to the equality constraint and $\mathbf{K} =$ is given by,

$$\mathbf{K} = \begin{bmatrix} \mathbf{Y} & \mathbf{I} - \mathbf{S} \\ \mathbf{I} - \mathbf{S}^T & \mathbf{Z} \end{bmatrix}$$

Now to construct the dual problem we have to take the infimum with respect to primal variables. In order to do that let's first consider the following block matrices,

$$\mathbf{\Psi} = \begin{bmatrix} \mathbf{\Psi}_{11} & \mathbf{\Psi}_{12} \\ \mathbf{\Psi}_{21} & \mathbf{\Psi}_{22} \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} \mathbf{W}_{11} & \mathbf{W}_{12} \\ \mathbf{W}_{21} & \mathbf{W}_{22} \end{bmatrix}$$

Also note that,

$$Tr(\mathbf{W}^* \text{blkdiag}(\mathbf{Y}, \mathbf{Z})) = Tr\big[(\mathbf{W}_{11} + \mathbf{W}_{21})\mathbf{Y} + (\mathbf{W}_{12} + \mathbf{W}_{22})\mathbf{Z})\big]$$

$$\|\mathbf{S} - \mathbf{A}\|_F^2 = Tr\big[(\mathbf{S} - \mathbf{A})^T(\mathbf{S} - \mathbf{A})\big]$$

$$\sum_{ij} \mathbf{\Lambda}_{ij}\mathbf{S}_{ij} = Tr(\mathbf{\Lambda}^T\mathbf{S})$$

$$\boldsymbol{\mu}^T(\mathbf{S}\mathbf{1} - \mathbf{1}) = Tr(\boldsymbol{\mu}^T\mathbf{S}\mathbf{1}) - \boldsymbol{\mu}^T\mathbf{1} = Tr(\mathbf{1}\boldsymbol{\mu}^T\mathbf{S}) - \boldsymbol{\mu}^T\mathbf{1}$$

$$\begin{aligned} Tr(\mathbf{\Psi}\mathbf{K}) = Tr\big[&(\mathbf{\Psi}_{11} + \mathbf{\Psi}_{21})\mathbf{Y} + (\mathbf{\Psi}_{12} + \mathbf{\Psi}_{22})\mathbf{Z} \\ &- (\mathbf{\Psi}_{11} + \mathbf{\Psi}_{21})\mathbf{S} - (\mathbf{\Psi}_{12} + \mathbf{\Psi}_{22})\mathbf{S}^T + (\mathbf{\Psi}_{11} + \mathbf{\Psi}_{12} + \mathbf{\Psi}_{21} + \mathbf{\Psi}_{22})\mathbf{I}\big] \end{aligned}$$

If we group terms according to the three primal variables $(\mathbf{S}, \mathbf{Y}, \mathbf{Z})$,

$$\mathbf{Y}: \quad Tr\big[(\mathbf{W}_{11} + \mathbf{W}_{21} - (\mathbf{\Psi}_{11} + \mathbf{\Psi}_{21}))\mathbf{Y}\big]$$

$$\mathbf{Z}: \quad Tr\big[(\mathbf{W}_{12} + \mathbf{W}_{22} - (\mathbf{\Psi}_{12} + \mathbf{\Psi}_{22}))\mathbf{Z}\big]$$

$$\mathbf{S}: \quad Tr\big[(\mathbf{S}-\mathbf{A})^T(\mathbf{S}-\mathbf{A}) + (\mathbf{\Psi}_{11}+\mathbf{\Psi}_{21})\mathbf{S} + (\mathbf{\Psi}_{12}+\mathbf{\Psi}_{22})\mathbf{S}^T - \mathbf{\Lambda}^T\mathbf{S} - \mathbf{1}\boldsymbol{\mu}^T\mathbf{S}\big]$$

The terms associated to $\mathbf{Y}$ and $\mathbf{Z}$ are unbonded below unless the following constraints are met,

$$\mathbf{W}_{11} + \mathbf{W}_{21} = \mathbf{\Psi}_{11} + \mathbf{\Psi}_{21}$$

$$\mathbf{W}_{12} + \mathbf{W}_{22} = \mathbf{\Psi}_{12} + \mathbf{\Psi}_{22}$$

For the terms associated to $\mathbf{S}$ we have to find the minimum with respect to $\mathbf{S}$ and for that we take the gradient,

$$\nabla_{\mathbf{S}}\mathcal{L}(\mathbf{S}) = 2\mathbf{S}^T - \mathbf{A} - \mathbf{A}^T - \mathbf{\Lambda} + (\mathbf{\Psi}_{11} + \mathbf{\Psi}_{12} + \mathbf{\Psi}_{21} + \mathbf{\Psi}_{22})^T - \boldsymbol{\mu}\mathbf{1}^T$$

$$\mathbf{S}^T = \frac{1}{2}(\mathbf{A} + \mathbf{A}^T + \mathbf{\Lambda} + \boldsymbol{\mu}\mathbf{1}^T - (\mathbf{\Psi}_{11} + \mathbf{\Psi}_{12} + \mathbf{\Psi}_{21} + \mathbf{\Psi}_{22})^T)$$

At this point, we lost hope of being able to find any idea that lead us to an efficient and simpler algorithm.

3.2.5 Laplacian Optimizaion

In this section, we present an approach very different to the previous ones. The idea is the following, instead of minimizing the distance between affinity matrices, we directly minimize the distance between laplacian matrices. That is, instead of considering $\|\mathbf{S} - \mathbf{A}\|_F$, we now consider $\|\hat{\mathbf{L}}_\mathbf{S} - \hat{\mathbf{L}}_\mathbf{A}\|_F$. Recall that $\hat{\mathbf{L}}_\mathbf{S} = \hat{\mathbf{D}}_\mathbf{S} - \mathbf{S}$. For convenience, we will now drop the notation $\hat{\mathbf{L}}$ to indicate non-symmetrized laplacians and will consider $\mathbf{L}$ the non-symmetrized laplacian. These two distances are not exactly the same but they convey the same information, that is that both graphs should be closer in the Frobenous norm. The advantage of considering the laplacian matrices directly is that we can impose the rank constraint directly into this variable. More precisely the formulation is the following,

$$\begin{aligned}
\underset{\mathbf{L}_\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{L}_\mathbf{S} - \mathbf{L}_\mathbf{A}\|_F^2 \\
\text{subject to} \quad & \mathbf{L}_\mathbf{S}\mathbf{1} = 0 \\
& \mathbf{L}_{\mathbf{S}_{ij}} \leq 0 \quad \forall i \neq j. \\
& \mathbf{L}_{\mathbf{S}ii} = 1 \\
& \text{rank}[\mathbf{L}_\mathbf{S}] = n - k
\end{aligned}$$

where the first two constraints are to impose laplacian structure. i.e row sum equal 0 and negative off-diagonal terms. The third constraint is to keep the constraint on the affinity matrix, $\sum_j^n s_{ij} = 1$ that we have in the other methods.

This problem is not convex, and to tackle it we propose to different approximations (the second approximation we will discuss it in the next section). Both relaxations first introduce an additional variables to decouple the non convex

constraint from the convex ones. We define then the equivalent problem,

$$\begin{aligned} \underset{\mathbf{L_S},\tilde{\mathbf{L}}_\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{L_S} - \mathbf{L_A}\|_F^2 \\ \text{subject to} \quad & \mathbf{L_S}\mathbf{1} = 0 \\ & \mathbf{L}_{\mathbf{S}_{ij}} \leq 0 \quad \forall i \neq j \\ & \mathbf{L}_{\mathbf{S}ii} = 1 \\ & \text{rank}[\tilde{\mathbf{L}}_\mathbf{S}] = n - k \\ & \mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S} = 0 \end{aligned} \tag{3.33}$$

Note that the laplacian constraints (convex) only apply to $\mathbf{L_S}$ and the rank constraints (non-convex) only apply to $\tilde{\mathbf{L}}_\mathbf{S}$. Also note that we could have defined the constraints the other way, i.e, laplacian constraints applying only to $\tilde{\mathbf{L}}_\mathbf{S}$ and rank constraint to $\mathbf{L_S}$. And in this case both definitions are equivalent because $\mathbf{L_S} = \tilde{\mathbf{L}}_\mathbf{S}$.

The first relaxation consists on moving the equality constraint $\mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S} = 0$ to the objective function with a parameter λ,

$$\begin{aligned} \underset{\mathbf{L_S},\tilde{\mathbf{L}}_\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{L_S} - \mathbf{L_A}\|_F^2 + \lambda\|\mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S}\|_F^2 \\ \text{subject to} \quad & \mathbf{L_S}\mathbf{1} = 0 \\ & \mathbf{L}_{\mathbf{S}_{ij}} \leq 0 \quad \forall i \neq j \\ & \mathbf{L}_{\mathbf{S}ii} = 1 \\ & \text{rank}[\tilde{\mathbf{L}}_\mathbf{S}] = n - k \end{aligned} \tag{3.34}$$

In this case, swaping the constraints between $\mathbf{L_S}$ and $\tilde{\mathbf{L}}_\mathbf{S}$ does not yield the same problem. We choose the first method (as it is in (3.34)) as in the objective we are then comparing two laplacian matrices.

The problem is still not convex on $\mathbf{L_S}$ and $\tilde{\mathbf{L}}_\mathbf{S}$, however, when we fix any of the two variables the problem can be solved.

- If $\tilde{\mathbf{L}}_\mathbf{S}$ is fixed the problem becomes,

$$\begin{aligned} \underset{\mathbf{L_S}}{\text{minimize}} \quad & \|\mathbf{L_S} - \mathbf{L_A}\|_F^2 + \lambda\|\mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S}\|_F^2 \\ \text{subject to} \quad & \mathbf{L_S}\mathbf{1} = 0 \\ & \mathbf{L}_{\mathbf{S}_{ij}} \leq 0 \quad \forall i \neq j \\ & \mathbf{L}_{\mathbf{S}ii} = 1 \end{aligned}$$

which can be rewritten as follows,

$$\begin{aligned} \underset{\mathbf{L_S}}{\text{minimize}} \quad & \|\mathbf{L_S} - \frac{\mathbf{L_A} + \lambda\tilde{\mathbf{L}}_\mathbf{S}}{1+\lambda}\|_F^2 \\ \text{subject to} \quad & \mathbf{L_S}\mathbf{1} = 0 \\ & \mathbf{L_S}_{ij} \leq 0 \quad \forall i \neq j \\ & \mathbf{L_S}_{ii} = 1 \end{aligned} \tag{3.35}$$

This problem can be decoupled for each row $\mathbf{l}_i^s$,

$$\begin{aligned} \underset{\mathbf{l}_\mathbf{i}^s}{\text{minimize}} \quad & \|\mathbf{l}_\mathbf{i}^s - \frac{\mathbf{l}_i^a + \lambda\tilde{\mathbf{l}}_i^s}{1+\lambda}\|_F^2 \\ \text{subject to} \quad & \mathbf{l}_i^s\mathbf{1} = 0 \\ & \mathbf{l}_{ij}^s \leq 0 \quad \forall i \neq j \\ & \mathbf{l}_{ii}^s = 1 \end{aligned} \tag{3.36}$$

Again, the solution is given by the water filling algorithm, for details we refer the reader to section 2.2.6.

- If $\mathbf{L_S}$ is fixed the problem becomes,

$$\begin{aligned} \underset{\tilde{\mathbf{L}}_\mathbf{S}}{\text{minimize}} \quad & \|\tilde{\mathbf{L}}_\mathbf{S} - \mathbf{L_S}\|_F^2 \\ \text{subject to} \quad & \text{rank}[\tilde{\mathbf{L}}_\mathbf{S}] = n - k \end{aligned}$$

  The solution to this problem is given by the truncated SVD decomposition of $\mathbf{L_S}$. If $\mathbf{L_S} = \mathbf{U\Sigma V}^T$, then $\tilde{\mathbf{L}}_\mathbf{S} = \mathbf{U\Sigma}_{k+1:n}\mathbf{V}^T$ i.e, setting the k smallest singular values to 0. From now on, we will use the notation $\text{SVD}_{i:j}(X)$ to refer to the truncated SVD of $\mathbf{X}$, taking the singular values from i to j

If we had considered the constraints reversed the analysis would have been analogous.

The pseudocode of the algorithm,

Algorithm 9 Laplacian optimization

Input: $\mathbf{A} \in \mathbb{R}^{n \times n}$, number of clusters k, a large enough λ, stopping criteria ϵ
Output: $\mathbf{S} \in \mathbb{R}^{n \times n}$ with $\sigma_k(\hat{\mathbf{L}}_\mathbf{S}) \leq \epsilon$

```
Compute L_A = D_A − A
Take = SVD_{k+1:n}(L_A)
while σ_k(L̂_S) > ε do
    Update L_S by solving (3.35)
    Set L̃_S = SVD_{k+1:n}(L_S)
end while
```

This algorithm despite being fundamentally different to the Constrained rank laplacian algorithm in [12] (see2.2.6), has a very similar optimization algorithm. In [12], they require computing the eigenvectors of the symmetrized laplacian matrix, in here we require singular vectors of the unsymmetrized laplacian matrix, the second update is done through the water filling algorithm in both approaches.

3.2.6 ADMM

In this case we will also optimize directly the laplacian matrix $\mathbf{L_S}$, however, now we will make use of the Alternating Directions Method of Multipliers (ADMM) [4]. In a nutshell, the ADMM is an algorithm that combines the best of two well known algorithms, Dual Ascent and Method of Multipliers. Dual Ascent is an optimization method that solves convex problems by iteratively optimizing the lagrangian and the dual function. However, the requirements that ensure convergence are very hard. The Method of Multipliers is and algorithm very similar to the dual ascent but it manages to relax this strong requirements to achieve convergence, however, this improvement comes at expenses of destroying the decomposability that a problem may have. ADMM uses the same principles as the Method of Multipliers but manages to keep the decomposability of the problem.

For convex problems ADMM is guaranteed to converge to the optimal solutions, for non-convex problems neither optimality nor convergence to a feasible solution is guaranteed. However, if convergence is met, we will have a matrix that is exactly both laplacian and has the desired low rank. We will show later why it may be important to achieve a solution with the exact desired rank.

The problem that we consider is the following,

$$\begin{aligned} \underset{\mathbf{L_S}, \tilde{\mathbf{L}}_\mathbf{S}}{\text{minimize}} \quad & \|\mathbf{L_S} - \mathbf{L_A}\|_F^2 \\ \text{subject to} \quad & \text{rank}[\tilde{\mathbf{L}}_\mathbf{S}] = n - k \\ & \mathbf{L_S}\mathbf{1} = 0 \\ & \mathbf{L_S}_{ij} \leq 0 \quad \forall i \neq j \\ & \mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S} = 0. \end{aligned}$$

If at convergence the constraint $\mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S} = 0$ is met we will have a matrix that is both laplacian and low-rank. The ADMM framework goes as follow, we first consider the lagrangian on the constraint $\mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S} = 0$, given by,

$$\mathcal{L}(\mathbf{L_S}, \tilde{\mathbf{L}}_\mathbf{S}, \mathbf{U}) = \|\mathbf{L_S} - \mathbf{L_A}\|_F^2 + \text{Tr}(\mathbf{U}^T(\mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S}))$$

Where, $\mathbf{U} \in \mathbb{R}^{n \times n}$ is the dual variable associated to the constraint $\mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S} = 0$. Now, we consider the augmented lagrangian, where we basically include the norm of the residuals associated the constraint into the lagrangian,

$$\mathcal{L}(\mathbf{L_S}, \tilde{\mathbf{L}}_\mathbf{S}, \mathbf{U}) = \|\mathbf{L_S} - \mathbf{L_A}\|_F^2 + \text{Tr}(\mathbf{U}^T(\mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S})) + \frac{\rho}{2}\|\mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S}\|_F^2$$

where $\rho > 0$ is a design parameter.

Now, we can make the following transformation $\mathbf{W} = \frac{1}{\rho}\mathbf{U}$, and we get the scaled form. Denoting $\mathbf{Z} = \mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S}$ to ease notation we have,

$$\begin{aligned} \rho\,\text{Tr}(\mathbf{W}^T\mathbf{Z}) + \frac{\rho}{2}\|\mathbf{Z}\|_F^2 &= \frac{\rho}{2}\,\text{Tr}(\mathbf{W}^T\mathbf{Z}) + \frac{\rho}{2}\,\text{Tr}(\mathbf{Z}^T\mathbf{W}) + \frac{\rho}{2}\|\mathbf{Z}\|_F^2 \\ &= \frac{\rho}{2}\,\text{Tr}(\mathbf{W}^T\mathbf{Z} + \mathbf{Z}^T\mathbf{W} + \mathbf{Z}^T\mathbf{Z}) \\ &= \frac{\rho}{2}\,\text{Tr}(\mathbf{W}^T\mathbf{Z} + \mathbf{Z}^T\mathbf{W} + \mathbf{Z}^T\mathbf{Z} + \mathbf{W}^T\mathbf{W} - \mathbf{W}^T\mathbf{W}) \\ &= \frac{\rho}{2}\,\text{Tr}((\mathbf{Z} + \mathbf{W})^T(\mathbf{Z} + \mathbf{W})) - \frac{\rho}{2}\,\text{Tr}(\mathbf{W}^T\mathbf{W}) \\ &= \frac{\rho}{2}\|\mathbf{Z} + \mathbf{W}\|_F^2 - \frac{\rho}{2}\|\mathbf{W}\|_F^2 \end{aligned}$$

Then, we can rewrite the augmented lagrangian in the scaled form as,

$$\mathcal{L}(\mathbf{L_S}, \tilde{\mathbf{L}}_\mathbf{S}, \mathbf{W}) = \|\mathbf{L_S} - \mathbf{L_A}\|_F^2 + \frac{\rho}{2}\|\mathbf{L_S} - \tilde{\mathbf{L}}_\mathbf{S} + \mathbf{W}\|_F^2 - \frac{\rho}{2}\|\mathbf{W}\|_F^2$$

Finally, the ADMM updates are,

- $\mathbf{L_S}^{k+1} = \underset{\mathbf{L_S} \in \mathcal{C}}{\operatorname{argmin}}\ \mathcal{L}(\mathbf{L_S}, \tilde{\mathbf{L}}_\mathbf{S}^k, \mathbf{W}^k)$

  where $\mathcal{C}$ is the set of laplacian matrices.

  Ignoring all the terms that do not depend on $\mathbf{L_S}$ and defining $\mathbf{C}^k = \tilde{\mathbf{L}}_\mathbf{S}^k - \mathbf{W}^k$ to ease notation, the objective function can be rewritten as,

$$\|\mathbf{L_S} - \mathbf{L_A}\|_F^2 + \frac{\rho}{2}\|\mathbf{L_S} - \mathbf{C}^k\|_F^2 \equiv \|\mathbf{L_S} - \frac{\mathbf{L_A} + \frac{\rho}{2}\mathbf{C}^k}{1 + \frac{\rho}{2}}\|_F^2$$

  Then the final optimization problem is,

$$\begin{aligned} \underset{\mathbf{L_S}}{\text{minimize}} \quad & \|\mathbf{L_S} - \frac{\mathbf{L_A} + \frac{\rho}{2}\mathbf{C}^k}{1 + \frac{\rho}{2}}\|_F^2 \\ \text{subject to} \quad & \mathbf{L_S}\mathbf{1} = 0 \\ & \mathbf{L_S}_{ij} \leq 0 \quad \forall i \neq j \end{aligned}$$

  which can be obtained in close form solution by using waterfilling solution.

- $\tilde{\mathbf{L}}_\mathbf{S}^{k+1} = \underset{\tilde{\mathbf{L}}_\mathbf{S} \in \mathcal{R}_{n-k}}{\operatorname{argmin}}\ \mathcal{L}(\mathbf{L_S}^{k+1}, \tilde{\mathbf{L}}_\mathbf{S}, \mathbf{W}^k)$

  where $\mathcal{R}_{n-k}$ is the set of matrices with rank $n-k$.

  Ignoring all the terms that do not depend on $\tilde{\mathbf{L}}_\mathbf{S}$, the final optimization problem

$$\begin{aligned} \underset{\tilde{\mathbf{L}}_\mathbf{S}}{\text{minimize}} \quad & \|\tilde{\mathbf{L}}_\mathbf{S} - (\mathbf{L_S}^{k+1} + \mathbf{W}^k)\|_F^2 \\ \text{subject to} \quad & \text{rank}[\tilde{\mathbf{L}}_\mathbf{S}] = n - k \end{aligned}$$

  Despite being non-convex, this problem can be solved in closed form solution by means of the truncated SVD.

- $\mathbf{W}^{k+1} = \mathbf{W}^k + \mathbf{L_S}^{k+1} - \tilde{\mathbf{L}}_\mathbf{S}^{k+1}$ (which follows from Dual Ascent algorithm)

3.3 Implementation

In this section we will dive into the details of each algorithm in terms of actual implementation. Firs thing to note is that all the presented algorithms require

an input parameter λ which controls the "low rankness" of the solution. If we set it too low we will not get a solution with the desired rank and hence we will not be able to obtain the clusters, and if we set it too high, the algorithm will ignore the input matrix and will just focus on obtaining a low rank matrix that will not maintain the desired connections. Therefore, the parameter λ has to be sensibly chosen. To speed up this tuning parameter though the authors in [12] propose the following adaptative $\lambda(t)$,

$$\lambda(t) = \begin{cases} \frac{\lambda(t-1)}{\alpha} & \text{if \# zero eigenvalues} > k \\ \alpha\lambda(t-1) & \text{if \# zero eigenvalues} < k \end{cases}$$

However, with this procedure another question arises, how we define if an eigenvalue is 0. We will see in the experiments section that for some algorithms we cannot make the eigenvalues arbitrarily low. Then, we have to set a threshold at which we consider the eigenvalues are 0.

Once the algorithm has converged to some $\mathbf{S}^*$ or $\mathbf{L}^*_{\mathbf{S}}$ we have to retrieve its connected components or clusters. Then we have to decide which connectivity are we interested in. For the algorithm in [12], they optimize with respect to the symmetric laplacian, hence they are implicitly considering the undirected graph, therefore, to find the connected components they take $\hat{\mathbf{S}} = (\mathbf{S}^* + \mathbf{S}^{*T})/2$ and find the connected component there. As the algorithm does not obtain a matrix with the exactly desired rank we have to threshold on the matrix $\hat{\mathbf{S}}$ in order to obtain a solution. In the experiments section we will see what role it plays.

For the rest of algorithms that use the non-symmetric laplacian we can consider both connectivies, strong or weak. If we were considering weak connectivity we could just do like in [12] and take the average of the edges, on the other hand, if we were considering strong connectivity we would take the minimum of both edges $\hat{\mathbf{S}} = \min(\mathbf{S}^*, \mathbf{S}^{*T})$. In any case threshold is also required, we will talk more about the role of the threshold in the experiment section.

Chapter 4

Results and Discussion

In this chapter the performed experiments are explained. In addition, we discuss and compare results between the different proposed methods as well as base line approaches. Prior to that, the datasets and metrics used for evaluating the results are also described.

4.1 Datasets

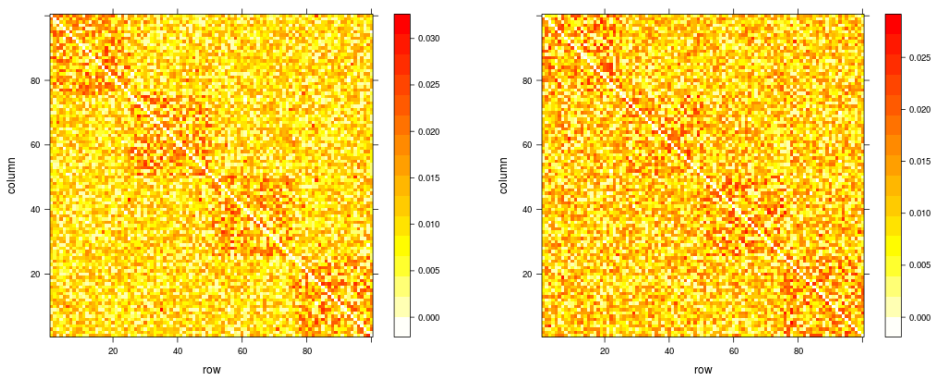
Two different datasets have been used in this thesis. The first a synthetic dataset which consists on a noisy block diagonal matrix and the second one the *Yeast* [2] dataset.

4.1.1 Noisy Block Diagonal Synthetic Dataset

We consider a block diagonal matrix of 100×100 with 4 blocks of 25×25, with the following properties,

- Entries in each block are sampled from *unif*$(0, 1)$
- Entries not belonging to any block are sampled from *unif*$(0, c)$ where $c = \{\ 0.6,\ 0.7,\ 0.8\}$
- 25 samples not belonging to any block are taken at random and set to 1

This noisy block diagonal matrix represents the affinity matrix of a graph we could have build using any of the methods explained in the previous chapters (ϵ-neighborhood graph, k-nearest neighbor graph, ...). The 4 blocks of 25x25 represent each clusters (or connected component) and the entries outside this



(a) $c = 0.7$ (b) $c = 0.8$

Figure 4.1: Two realizations of the Noisy Block diagonal dataset

blocks are edges that connect two blocks which should be removed by a sensible algorithm. The reason for considering this dataset is two-fold. First we can assess performance of the presented algorithm independent of how we build the initial affinity matrix and second we can tune how difficult the task is by raising or decreasing the noise level.

4.1.2 Yeast dataset

This is a dataset from the Center from Machine Learning and Intelligent System in the University of California, Irvine and consists of 1484 data points and there are 10 annotated clusters. The samples have dimensions 8, i.e $\mathbf{x_i} \in \mathbb{R}^8$.

4.2 Evaluation

To measure the performance of our algorithms we need some ground truth clusters or classes $C = \{c_1, \ldots, c_K\}$ to compare our solution to $\Omega = \{\omega_1, \ldots, \omega_K\}$, where c_i is the set of data point that belong to class i and ω_i the set of points that have been assigned to cluster i by the algorithm under study. This ground truth or gold standard is normally produced by human judges with a good level of inter-judge agreement. This section introduces four external criteria of clustering quality

4.2.1 Purity

$$\textit{Purity}(\Omega, C) = \frac{1}{n}\sum_{k}\max_{j}|\omega_k \cap c_j|$$

where n is the number of data points. Purity ranges from 0 to 1, where 1 is perfect clustering. It has the disadvantage that it is not normalized to number of clusters, then it is easy to achieve high purity when the number of clusters is high. In fact, purity 1 is achieved when each sample has its own cluster.

4.2.2 Normalized Mutual Information

From the information theory point of view, a clustering technique is good if the uncertainty that we have about the real classes decreases once we are told what the clusters are. It can be measured by the mutual information of both random variables,

$$\begin{aligned} I(\Omega, C) &= \sum_{k}\sum_{j} P(\omega_k \cap c_j)\log\frac{P(\omega_k \cap c_j)}{P(\omega_k)P(c_j))} \\ &= \sum_{k}\sum_{j}\frac{|\omega_k \cap c_j|}{n}\log\frac{n|\omega_k \cap c_j|}{|\omega_k||\omega_k|} \end{aligned}$$

.

However, this metric alone has the same problem as Purity as it is not normalized to the number of outputed clusters. And again, if each sample has its own cluster the NMI will be 1. That is why, the mutual information is normalized by average entropy of each random variable,

$$\textit{NMI}(\Omega, C) = \frac{I(\Omega, C)}{[H(\Omega) + H(C)]/2}$$

There exists several variants of this normalization. Instead of taking the average of entropies one can take the geometric mean, the maximum, the minimum or the joint entropy.

4.2.3 Rand Index

The Rand index (RI) considers all $\binom{N}{2}$ pairs of points and arranges them into one of the four sets:

- **TP**: A pair of points is considered **TP** if their classes are the same and they belong to same computed cluster.

- **TN**: A pair of points is considered **TN** if their classes are different and they belong to different computed cluster.
- **FP**: A pair of points is considered **FP** if their classes are different and they belong to the same cluster
- **FN**: A pair of points is considered **FN** if their classes are the same and they belong to different clusters.

Then, the RI is computed as follows,

$$RI = \frac{TP + TN}{TP + TN + FP + FN}$$

In some papers this metric is referred to Accuracy.

4.2.4 Maximum Matching

Maximum matching considers the pairwise matching between clusters and classes that maximizes the sum of intersections,

$$MM = \max_{align} \frac{1}{N} \sum_{i=1}^{k} |\omega_i \cap c_i|$$

The maximum alignment can be easily computed with the Hungarian algorithm [9].

4.3 Experiments

In this section we analyze the performance of the proposed algorithms both in terms of clustering and speed and compare it to the method in [12]. All the algorithms have been implemented in the programming language R with the help of additional packages such as OSQP solver for solving QP problems and CVXR for solving SDP problems.

Before starting let us recall all the presented method as well as the subrutines necessaries to solvem them, the name is parenthesis is how we will refer to them from now on,

- Constrained Rank Laplacian [12] (CRL): It requires eigenvectors and water filling algorithm
- Matrix Factorization QP (QP): It requires solving two QP

- Matrix Factorization Block Coordinate Descent (BCD): It requires water filling algorithm, and solving two linear system of equations
- Log determinant (logdet): It requires solving a SDP
- Convex iteration (ConvIte): It requires solving a SDP and eigenvectors
- Laplacian Optimization (Lopt): It requires water filling algorithm and singular value decomposition
- ADMM: It requires water filling algorithm and singular value decomposition

4.3.1 Analysis of the CRL

λ effect

Let's start by analyzing the baseline approach [12]. For this case we will consider the Yeast dataset and will analyze the effect of the tuning parameter λ to see how it effects convergence. Remember that the Yeast dataset have 10 clusters, therefore, we want the 10-th smallest eigen values to go 0 and the next eigenvalues to be "big" enough. In the following figure we depict the evolution of the first 11-th smallest eigenvalues of the symmetrized laplacian for the algorithm [12],

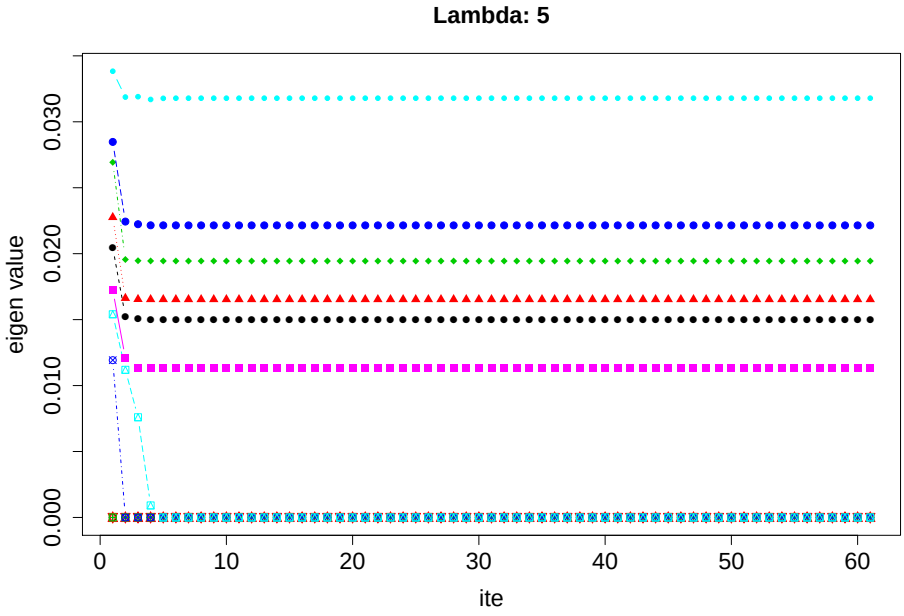


Figure 4.2: Eigenvalues evolution for the CRL

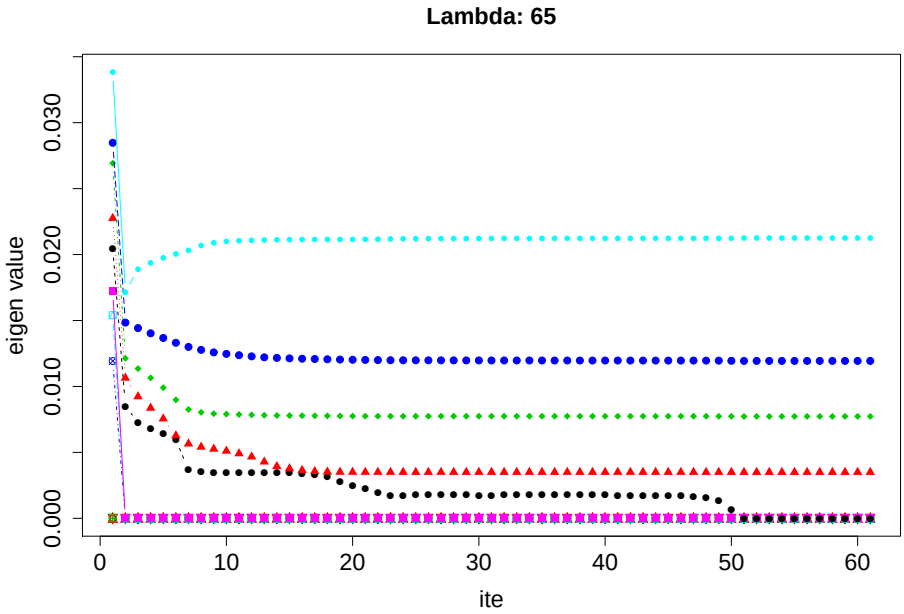


Figure 4.3: Eigenvalues evolution for the CRL

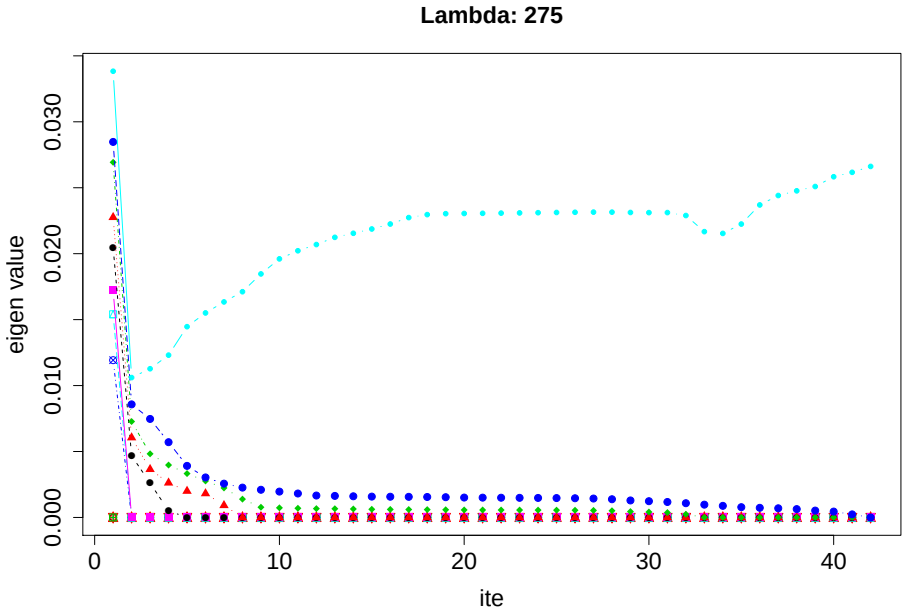


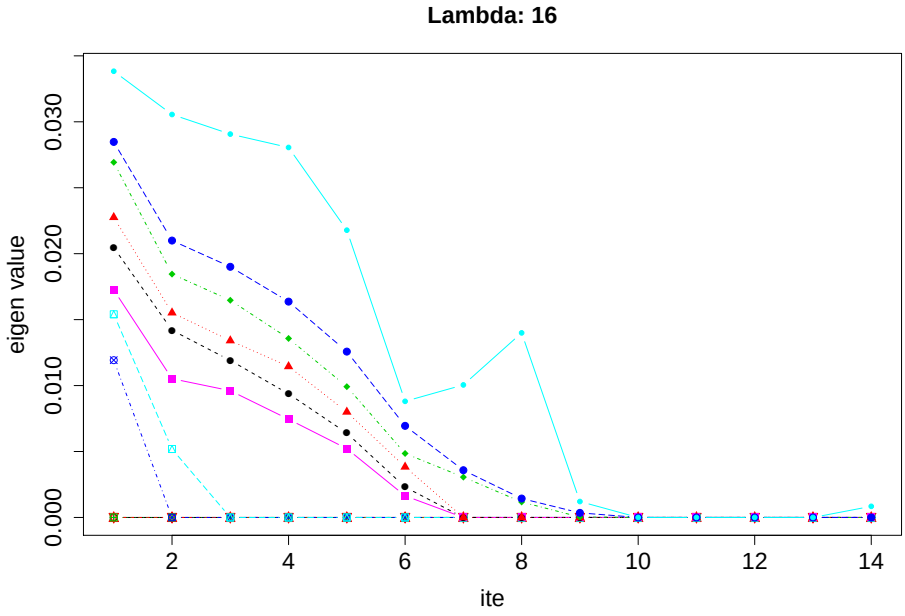
Figure 4.4: Eigenvalues evolution for the CRL

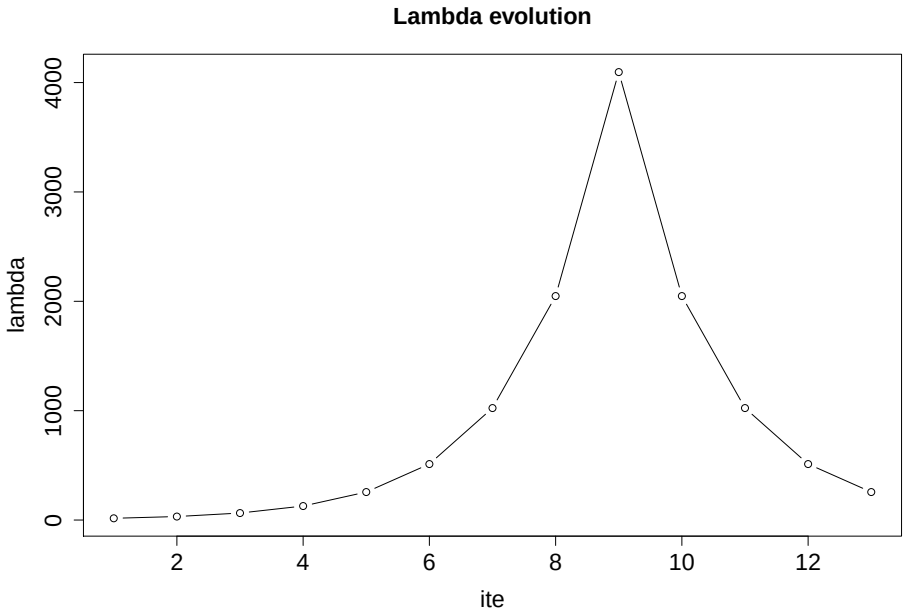
We can see how for a big enough value of λ, the algorithm converges

Adaptive λ

Now we consider an adaptive $\lambda(t)$ as described in section (3) with $\lambda(0) = 16$.

$$\lambda(t) = \begin{cases} \frac{\lambda(t-1)}{2} & \text{if \# zero eigenvalues} > k \\ 2\lambda(t-1) & \text{if \# zero eigenvalues} < k \end{cases}$$





As we can see, when the $k+1$ eigenvalue reaches a value close to 0 (at iteration 10), λ is decreased, until the $k+1$ reaches a "big" enough value. We can also see that using this adaptive λ convergence is achieved much faster 14 iterations vs 40 iterations.

Performance on Noisy Block Diagonal dataset

Some realizations of the The resulting graphs after optimizing $\mathbf{S}^*$ for the noisy block diagonal dataset with noise level $c = 0.7$ are the following,

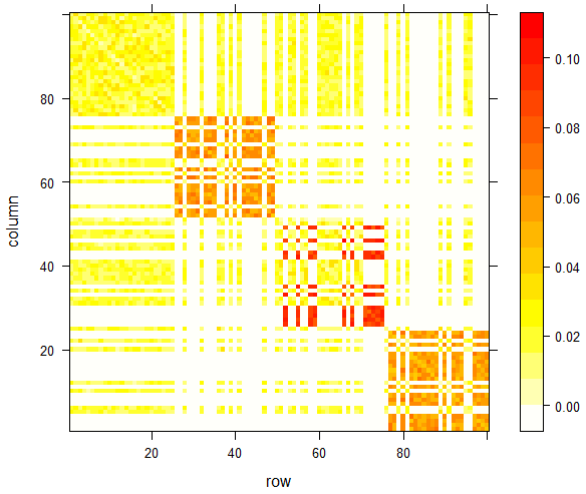
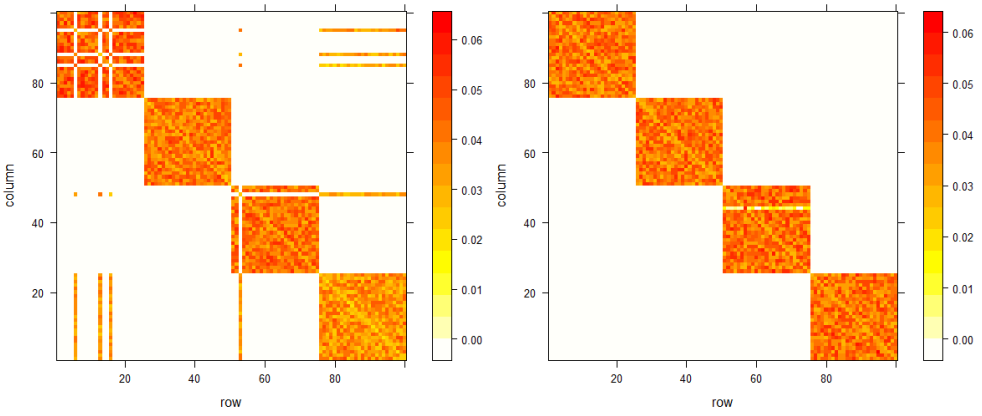


Figure 4.5: Solution $\mathbf{S}^*$ with poor performance MM = 0.71



(a) Clustering with some errors MM = 0.96 (b) Ideal clustering MM = 1

Figure 4.6: Solution $\mathbf{S}$ for two different noise realizations

4.3.2 Analysis of the non-symmetric laplacian based algorithms

First of all, we must say that we have not considered the algorithms that require SDP, i.e log determinant (3.2.3) and Convex Iteration (3.2.4). The reason is that for the problem size considered (graphs of 100 nodes), the SDP program takes more than 16GB of RAM which was the amount of RAM available. We could have considered smaller graphs, however we thought that 100 nodes was small enough and that there was no point in considering this algorithm if it cannot only handle such small problems.

Analysis of QP Method

Again, we consider the noisy block diagonal dataset (noise level c=0.7). This dataset has 4 clusters, hence we want the first 4 singular values to go to 0. If we take a look at the evolution of the first 5 singular values,

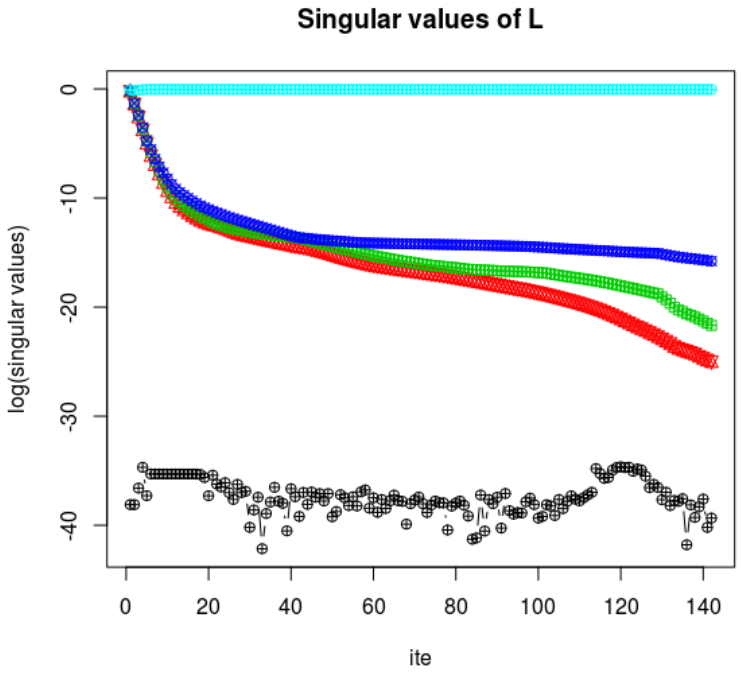


Figure 4.7: Singular values evolution for QP method

We can see that the singular values converge (it is not a value as close to 0 as in the CRL, but they are at the order of 1e-7). However, we now face the

problem of recovering a clustering. If we consider weak connectivity, we can symmetrize by the mean and look for connected components. However, when we do that we see that in order to get a valid solution (i.e k-connected components) the threshold necessary in some cases is quite high (1e-2), compared with the CRL (1e-6).

One of the reasons we thought for that behaviour, was that the algorithm did not reach singular values small enough and that was why we had to set such a big threshold. However, we will see later that that may not be the case.

We must note though, that this thresholding technique (minimum threshold such that we get k-components) always works. That means, by thresholding we can always get a solution with k-connected components. Hence, we can assess the performance of such algorithm,

Metric	Purity	NMI	Max-Matching
CRL	0.9570	0.9193	0.9563
QP	0.9920	0.9769	0.9920
Threshold	0.2906	0.07450	0.2817

Table 4.1: Performance of different methods for the noisy block diagonal matrix

We have also added as a comparison a technique that consists on thresholding directly the noisy matrix, (sometimes it cannot get a valid solution), to show that the thresholding done in the QP is not making our method improve it is just helping get a valid solution. We must also note that the proposed algorithm improves the CRL on all metrics.

Analysis of BCD Method

This method is very similar in nature to the QP, with the exception that the each iteration is much faster, (2 QPs vs 2 system of Linear equations). The averaged times are the following,

Method	Time secs
CLR	0.25
QP	4903
BCD	67

We see that BCD is much faster than QP, but also CRL is much faster than BCD. The good thing with CRL is that it converges much faster.

For the BCD we also compared the parallel update vs the cyclic update, as well as different step sizes estimations. As expected the cyclic update performs better than the parallel, but remember that the parallel method as it name says can be parallelized and if the resources are available be three times faster. For the different step sizes we did not notice any difference between the 3 proposed methods in 3.2.2.

We also run more experiments to test performance,

Metric	Purity	NMI	Max-Matching
CRL	0.6810	0.5831	0.6784
BCD	0.9609	0.9023	0.9609

Table 4.2: Performance of CRL and BCD for the noisy block diagonal matrix c = 0.75

Metric	Purity	NMI	Max-Matching
CRL	0.4195	0.2212	0.4135
BCD	0.7930	0.5796	0.7915

Table 4.3: Performance of CRL and BCD for the noisy block diagonal matrix c = 0.8

Again we see a big difference in performance with respect to the baseline CRL. This method also requires of the thresholding technique

Analysis of the Laplacian Optimization algorithm

The performance is very similar to the two previous approaches, and what we test know is the convergence speed of this method and the BCD,

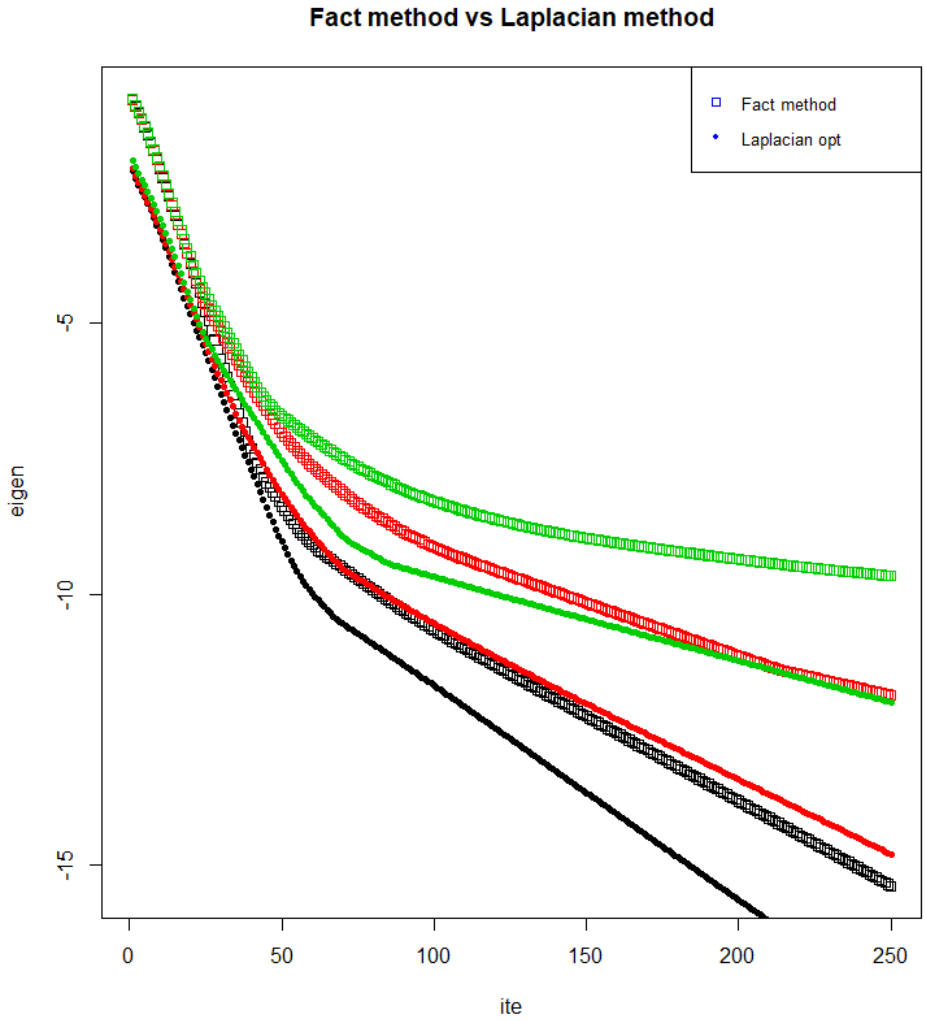
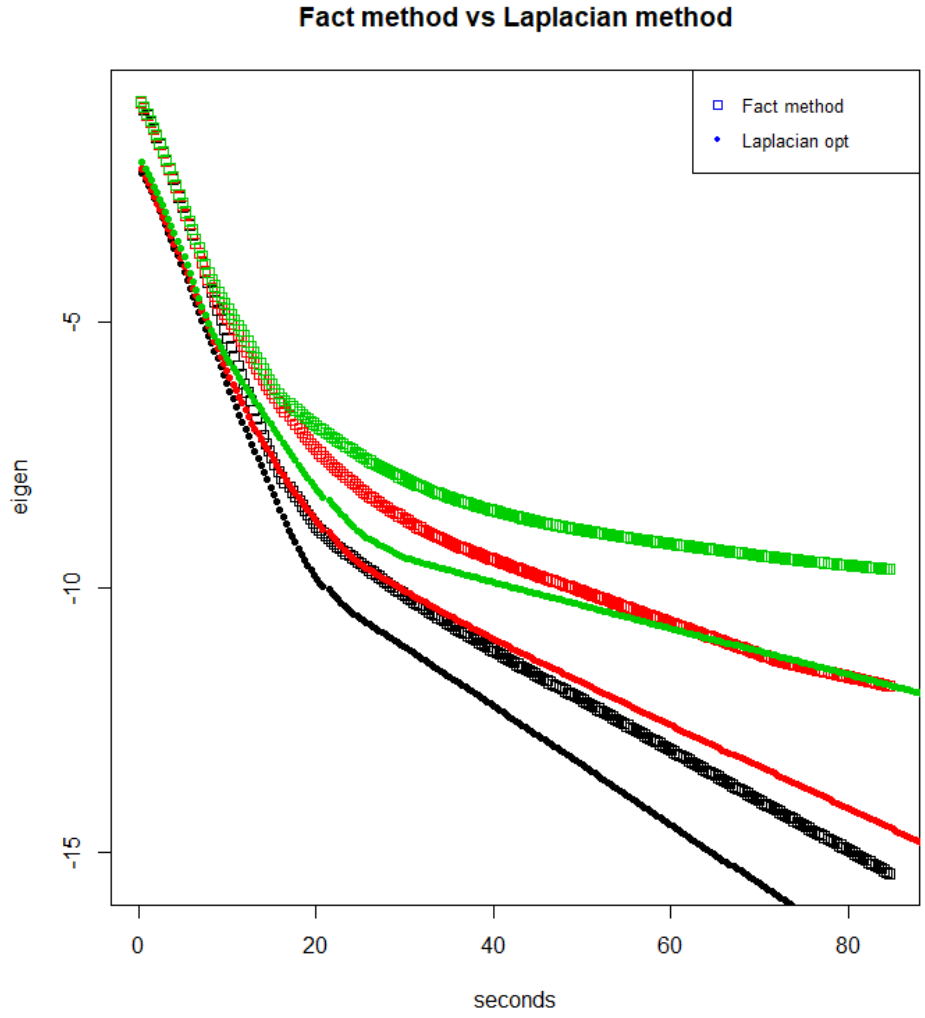
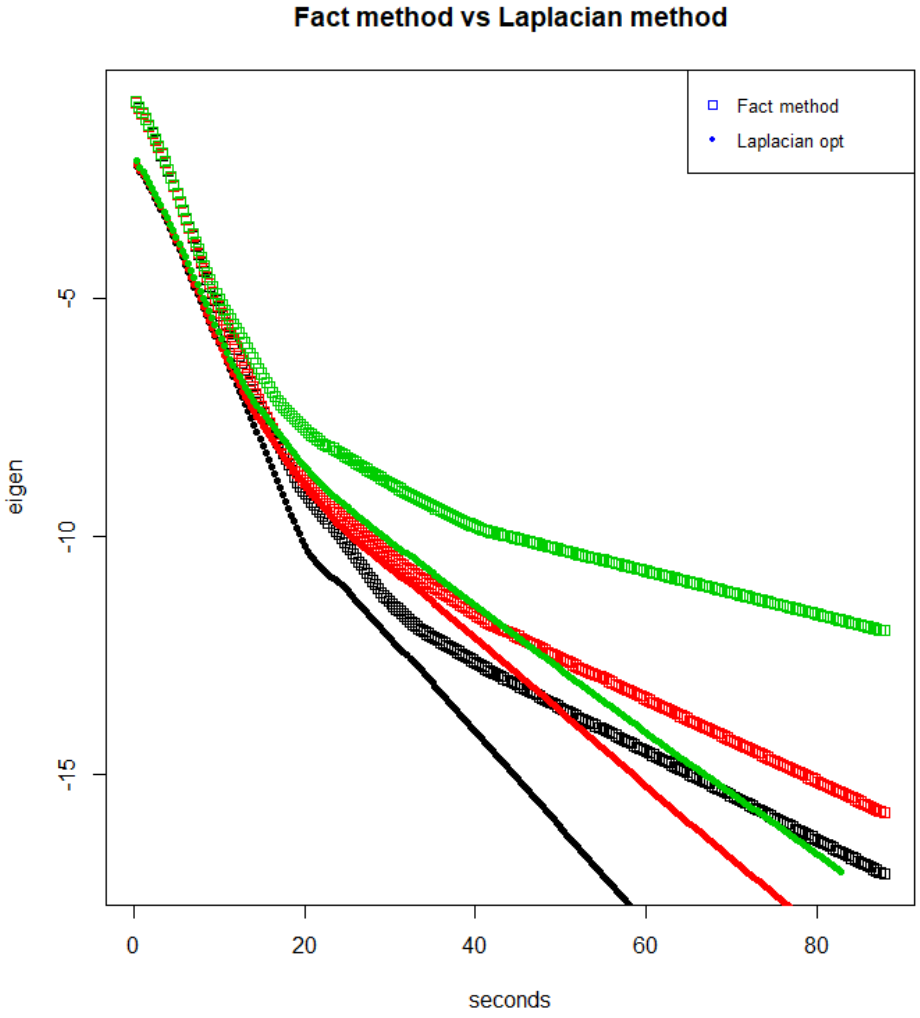


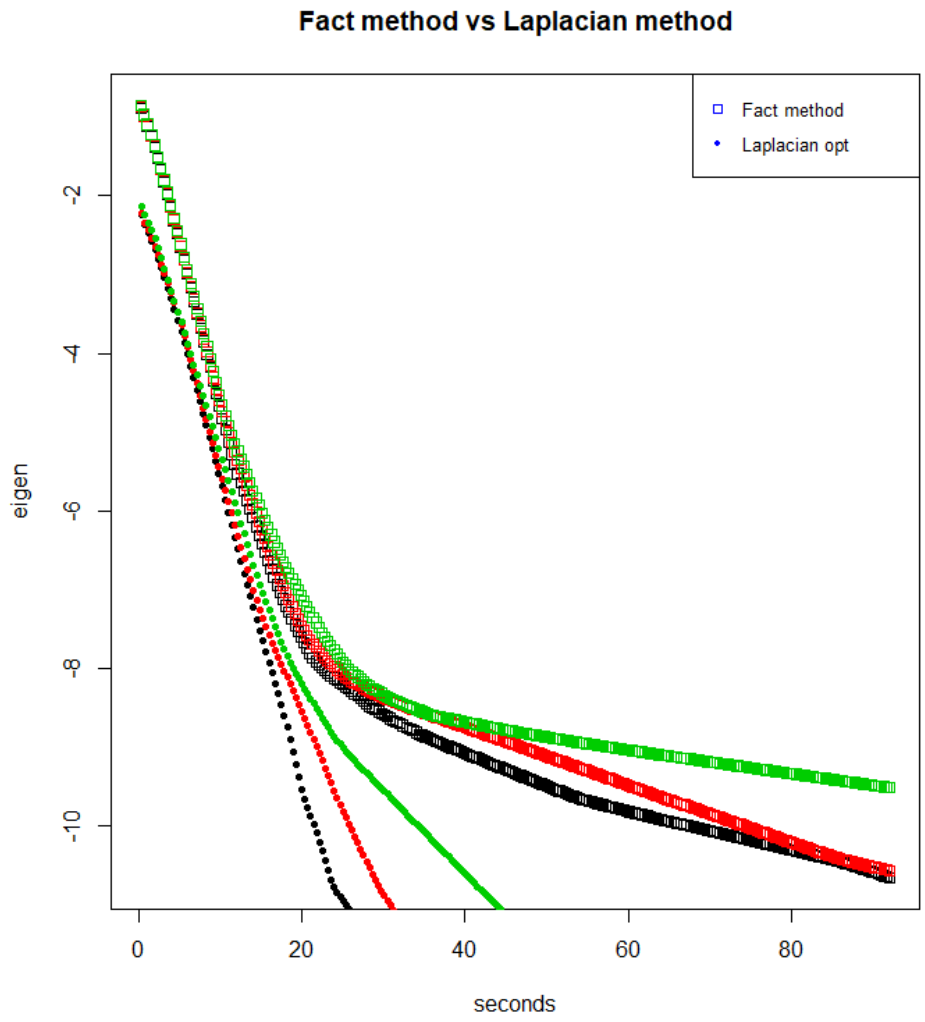
Figure 4.8: Singular value evolution for the BCD (Fact method) and the Laplacian opt

In this case we have not plotted the evolution of the first singular value as it 0 all the time by construction.

However, the update operations are quite different (1 SVD decomposition vs 2 systems of linear equations), then it makes more sense to measure the singular value evolution vs time,


Fact method vs Laplacian method
Fact method
Laplacian opt
eigen
-5
-10
-15
0
20
40
60
80
seconds


Fact method vs Laplacian method
Fact method
Laplacian opt
eigen
-5
-10
-15
0
20
40
60
80
seconds



We can note then that the laplacian method is indeed faster than BCD. In terms of clustering performance it is the same. This method also requires the thresholding technique.

4.3.3 ADMM

The main idea for considering ADMM was to try to achieve an algorithm that converges to a solution with the exact desired rank. And for a certain values of the tuning parameter ρ we managed to do so,

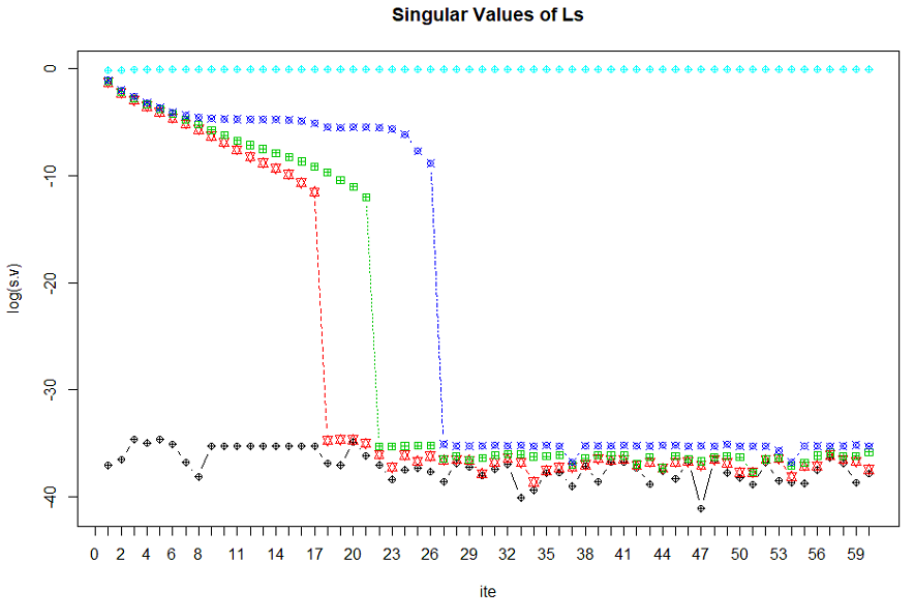


Figure 4.9: Singular value evolution for the ADMM algorithm

After achieving such a low singular values we were expecting to be able to get a solution with the desired number of components without using the thresholding technique. However, that was not the case. Even more, if we take a look at the laplacian of the symmetrized graph,

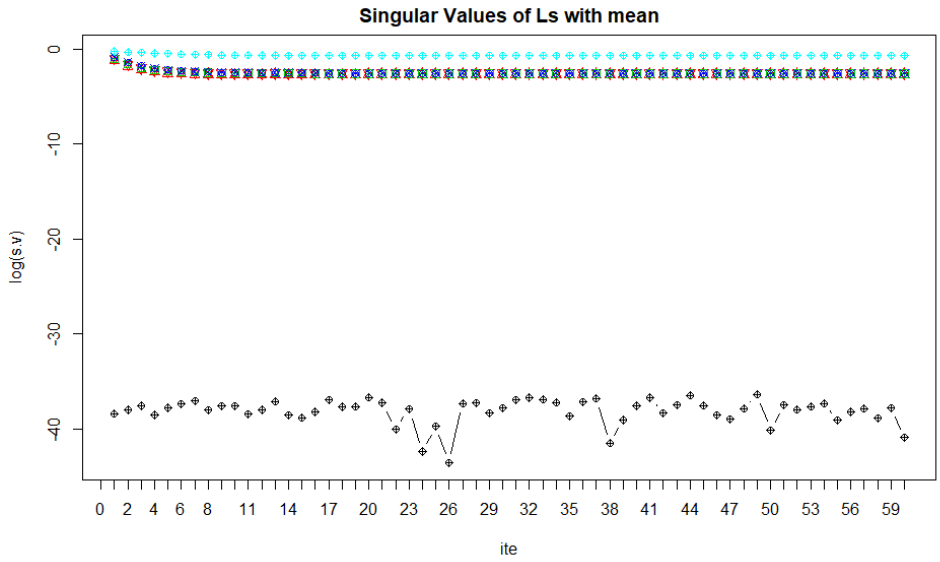


Figure 4.10: Singular value evolution of the mean symmetrized graph for the ADMM algorithm

we see that we are nowhere near converging.

At this point we decided to take a look at the strong connectivity instead of the weak connectivity,

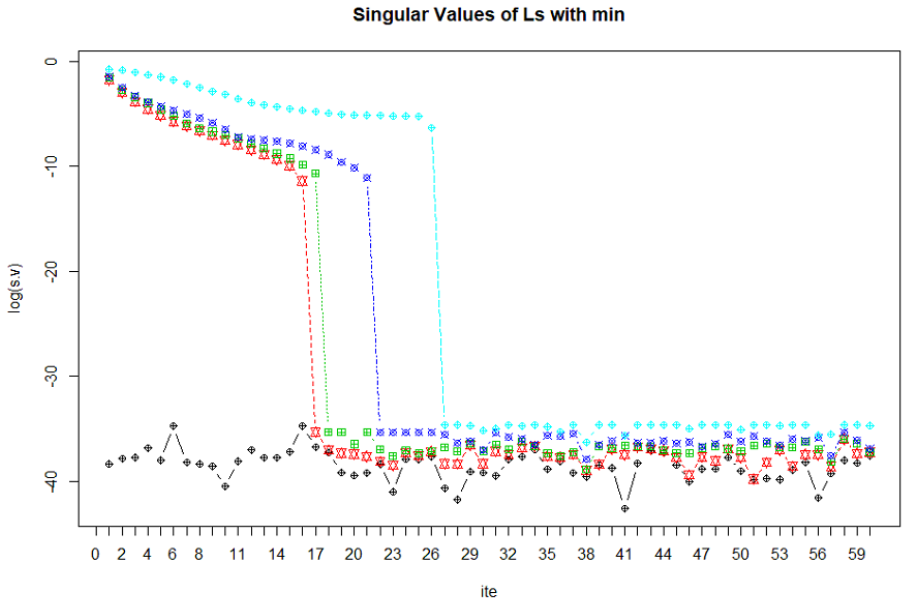


Figure 4.11: Singular value evolution of the min symmetrized graph for the ADMM algorithm

In this case we get convergence for the desired number of singular values (k) but only for some iterations (from iteration 23 to 27). After that point the $k+1$ singular value also converges and in this situation we cannot get a valid solution, we get more clusters than desired. It is quite strange also that the non symmetric laplacian (Figure (4.9)) converges at the desired number of components k just at the same time that in the min symmetrized laplacian (strongly connected laplacian) the $k+1$ singular value converges.

At this point we decided to go back to the literature [1]. And we noted that the rank of the laplacian of directed graph defined as $\mathbf{L} = \mathbf{D} - \mathbf{A}$ does not denote neither the number of weekly connected components nor the number of strongly connected components. Instead it denotes the number of connected components defining connectivity in a more involved way. That's why we had to resort to the thresholding in the first place. Despite that, the rank still encodes the notion of connectivity and that is why the algorithm perform well in practice

Chapter 5

Conclusions and future work

In this work, we have proposed a new method for graph based clustering. This method tackles the problem of estimating a directed graph using a low rank constraint on its non symmetric laplacian. We have proposed several algorithms to do this estimation and we have studied their properties. We have shown that our presented method outperforms state of the art results among similar algorithms.

The main novelty is the use of the non-symmetric laplacian to characterize the connectivity of a directed graph. The proposed algorithms are simple and efficient as they only involve standard linear algebra methods, such as spectral decomposition or singular value decomposition.

Results have shown the method proposed to be a promising baseline for further research. We have come up with some proposals by the end of the thesis that could be interesting to investigate in the future. We also think that further benchmarking with bigger and more challenging datasets should be done.

The proposed algorithm is based on the fact that the rank of non symmetric laplacian is directly related to its connectivity, however, we do not know how to properly characterize this connectivity in a way that it is directly useful to clustering, that is why we have to resort to thresholding to get the desired solution. Despite all that, we have shown that it is able to make use of the connectivity information embedded in the rank to perform clustering. In the same way that the we have a fundamental theorem that relates the rank laplacian of an undirected graph to the number of connected components, it would be desirable to have a theorem that relates strongly connected components to a laplacian matrix.

In addition, it would be of interest to formally characterize the information

lost when symmetrizing the laplacian matrix of a directed graph.

Bibliography

[1] Rafig Agaev and Pavel Chebotarev. “On the spectra of nonsymmetric Laplacian matrices”. In: *Linear Algebra and its Applications* 399 (2005), pp. 157–168.

[2] Arthur Asuncion and David Newman. *UCI machine learning repository*. 2007.

[3] Marwa Balti. “On the eigenvalues of weighted directed graphs”. In: *Complex analysis and operator theory* 11.6 (2017), pp. 1387–1406.

[4] Stephen Boyd et al. “Distributed optimization and statistical learning via the alternating direction method of multipliers”. In: *Foundations and Trends® in Machine learning* 3.1 (2011), pp. 1–122.

[5] Michael R Bussieck and Stefan Vigerske. “MINLP solver software”. In: *Wiley encyclopedia of operations research and management science* (2010).

[6] Fan RK Chung and Fan Chung Graham. *Spectral graph theory*. 92. American Mathematical Soc., 1997.

[7] Jon Dattorro. *Convex optimization & Euclidean distance geometry*. Lulu. com, 2010.

[8] Michael Grant and Stephen Boyd. *CVX: Matlab software for disciplined convex programming, version 2.1*. 2014.

[9] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.

[10] Bojan Mohar et al. “The Laplacian spectrum of graphs”. In: *Graph theory, combinatorics, and applications* 2.871-898 (1991), p. 12.

[11] Andrew Y Ng, Michael I Jordan, and Yair Weiss. “On spectral clustering: Analysis and an algorithm”. In: *Advances in neural information processing systems*. 2002, pp. 849–856.

[12] Feiping Nie et al. “The constrained laplacian rank algorithm for graph-based clustering”. In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016.

[13] Robert J Plemmons. “M-matrix characterizations. I—nonsingular M-matrices”. In: *Linear Algebra and its Applications* 18.2 (1977), pp. 175–188.

[14] Imre Polik, Tamas Terlaky, and Yuriy Zinchenko. “SeDuMi: a package for conic optimization”. In: *IMA workshop on Optimization and Control, Univ. Minnesota, Minneapolis*. Citeseer. 2007.

[15] Jianbo Shi and Jitendra Malik. “Normalized cuts and image segmentation”. In: *Departmental Papers (CIS)* (2000), p. 107.

[16] Bartolomeo Stellato et al. “OSQP: An operator splitting solver for quadratic programs”. In: *2018 UKACC 12th International Conference on Control (CONTROL)*. IEEE. 2018, pp. 339–339.

[17] Gilbert W Stewart. “Matrix perturbation theory”. In: (1990).

[18] Kim-Chuan Toh, Michael J Todd, and Reha H Tütüncü. “SDPT3—a MATLAB software package for semidefinite programming, version 1.3”. In: *Optimization methods and software* 11.1-4 (1999), pp. 545–581.

[19] Ulrike Von Luxburg. “A tutorial on spectral clustering”. In: *Statistics and computing* 17.4 (2007), pp. 395–416.

[20] Stephen J Wright. “Coordinate descent algorithms”. In: *Mathematical Programming* 151.1 (2015), pp. 3–34.