ERASMUS UNIVERSITY ROTTERDAM

ERASMUS SCHOOL OF ECONOMICS

BACHELOR THESIS ECONOMETRICS

Majorization methods for solving the convex clustering problem

Author:
Tom VAN DEN BERG,
432879tb

Supervisor:
P.J.F. GROENEN

Second assessor:
S.I. BIRBIL

July 7, 2018

Abstract

The convex clustering problem is a convex relaxation of the problem of finding similar groups in data. Solving this yields a global minimum that can be used to construct a solution path or a network that shows how data points cluster. This research looks at majorization methods to solve the l_2 convex clustering problem and compares it with gradient descent in runtime. First, a proximal distance method is used and improved to get a majorization algorithm, next a norm marorization technique is used that gives a fast algorithm that involves solving a system of equations. Finally, another round of majorization gives a method based on eigenvalues, which bypasses solving a linear system and is thus faster for larger problems. The norm majorization method finds the l_2 solution path in the same runtime on average as the gradient descent method, which suggests that implementing this norm method in a low level programming language gives a fast algorithm to solve the convex clustering problem.

1 Introduction

In marketing data, often groups are present in the data structure. For example when reviewing respondents to a marketing campaign groups of people with the same characteristics can be formed. In this research, we study one of these algorithms that finds homogeneous groups in the data. This unsupervised learning algorithm is called *clusterpath* and is developed by Hocking, Joulin, Bach, and Vert (2011). The clusterpath algorithm finds clusters in the data in an agglomerative fashion, were all the observations start in a separate cluster and end in one cluster that contains all the observations. Where other methods like average linkage use a heuristic algorithm to find the clusters, clusterpath uses a convex relaxation of the clustering problem.

This convex relaxation results in cost function (1) for the data matrix $\boldsymbol{X} \in \mathbb{R}^{n\times d}$ with n observations and d features. The centroids of the clusters are represented by the matrix $\boldsymbol{A} \in \mathbb{R}^{n\times d}$, so that each unique row of $\boldsymbol{A}$ represents a centroid of a cluster. If multiple rows of $\boldsymbol{X}$ form a cluster, then the same rows of $\boldsymbol{A}$ are equal to the centroid of this cluster. The cost function is

$$f_q(\boldsymbol{A}, \boldsymbol{X}) = \frac{1}{2} \left\| \boldsymbol{A} - \boldsymbol{X} \right\|_F^2 + \lambda \sum_{i<j} w_{ij} \left\| \boldsymbol{a}_i - \boldsymbol{a}_j \right\|_q , \tag{1}$$

where $\left\|\cdot\right\|_q$, $q \geq 1$ is the l_q-norm and $\left\|\cdot\right\|_F^2$ is the squared Frobenius norm, with the Frobenius norm of a matrix $\boldsymbol{X}$ defined as $\left\|\boldsymbol{X}\right\|_F^2 = \sum_i \sum_j |x_{ij}|^2$.

Hocking et al. (2011) use the weights $w_{ij} = \exp(-\gamma \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_2^2)$. For $\gamma = 0$ this is interpeted as equal weights between the observations and $\gamma = 1$ means that observations closer to each other have a larger weight so local density in the data is accounted for. These weights are called decreasing because the weight for two points decreases as the distance between these points increases. It is important to note that the weights are influenced by the number of features d in the data. A remedy for this is to divide γ by d.

The main advantages of the convex clustering approach is that is converges to a global minimum, where other heuristic methods often get trapped in a local minimum. Another advantage is that the amount of clusters does not have to be known in advance, which is the case with k-means clustering. Solving the convex clustering problem for different values of λ reveals a solution path or a tree like structure of the data points that can be interpreted to find clusters. Figure 1 gives an example for two dimensions.

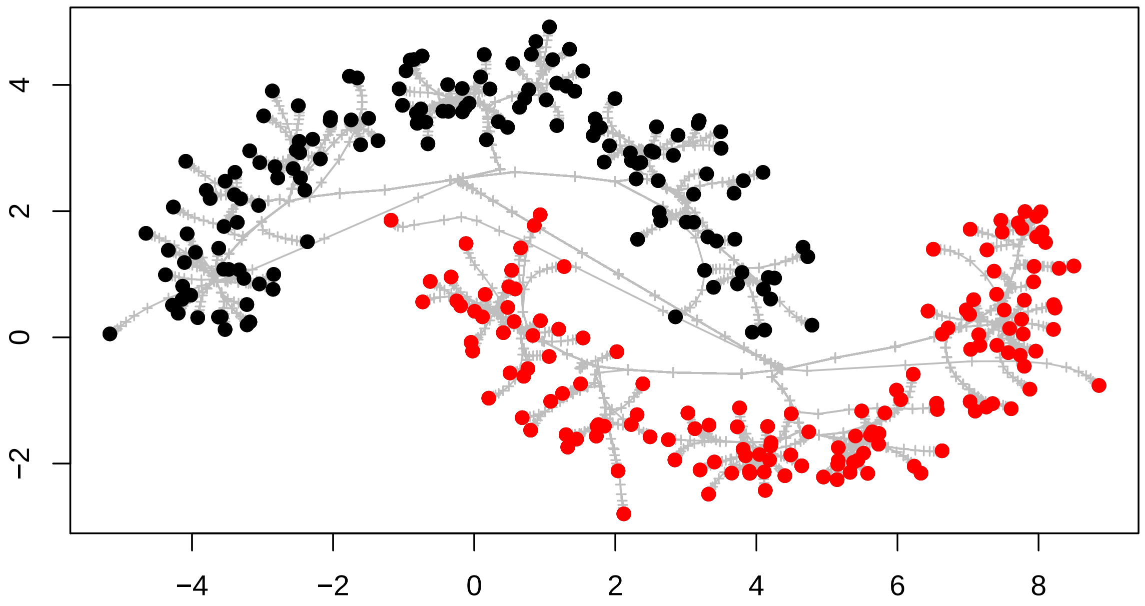


Figure 1: l_2 Solution path of the convex clustering problem of the interlocking Moons data with $n = 300$ and $\gamma = 4$. The solution path correctly identifies two moon-shaped clusters.

This research will focus on the l_2-norm, because this problem can not be split on dimensions like the l_1-norm (Hocking et al., 2011), so it will take the whole observation into account when clustering. In the literature, several methods have been developed to solve the l_2 convex clustering problem.

Hocking et al. (2011) use a gradient descent algorithm for minimizing the cost function for varying values of λ. They found that gradient descent uses many iterations before converging to the optimal solution. Also, they use that if two cluster centers get close to each other, they fuse. They conjecture that for decreasing weights, once clusters are fused, they can not split for larger λ's.

Chi and Lange (2015) propose a alternating direction method of multipliers (ADMM) and a alternating minimization algorithm (AMA) method for finding the solution path. Both methods split the variables in variables describing the cluster centers and variables describing the difference between cluster centers. The advantage of these methods is that they can be parallelized, which means that the problem can be split in similar smaller problems for fast calculation on multiple cores. They find that both methods are faster in runtime than gradient descent with AMA being the fastest. A disadvantage is a quadratic memory requirement, because of using the dual formulation and variable splitting. They solve this by using only the nearest k neighbors for defining the weights w_{ij}, which makes the variables with zero weights redundant.

Chen, Chi, Ranola, and Lange (2015) show that the convex clustering problem can also be solved by proximal distance methods from Lange and Keys (2015). This method also uses variable splitting, but they use the MM (majorization-minimization) principle to guarantee a descent step each iteration. They find that this method takes long to find the solution path.

A coordinate descent method can also be used, because the cost function is convex. Duckworth (2013) developed this method, he states that the algorithm is quick in finding a solution path but because it uses the dual formulation and variable splitting the number of variables grow quadratically and this lays a burden on memory.

It can be seen from the above literature review that most methods use the variable splitting for solving the convex clustering problem, which increases memory usage. The only method not using variable splitting is the gradient descent from Hocking et al. (2011), which has linear variable growth. However, this method uses many iterations to find the optimal solution. This research aims to improve the gradient descent method by finding an algorithm that uses majorization for finding the l_2 solution path. We will focus on the following research question:

Can majorization techniques be used to solve the l_2 convex clustering problem in shorter runtime than the gradient descent method?

To answer the research question, we develop majorization methods and implement and improve the proximal distance method to solve the convex clustering problem. Then we compare these majorization methods with the gradient descent method from Hocking et al. (2011) in runtime and precision. Like Hocking et al. (2011), we do this for two different generated datasets, the Moons Data and the Gaussian Clusters dataset.

This proposal is organized as follows: chapter 2 discusses the methodology used to solve the research problem and chapter 3 shows the data. Chapter 4 shows the comparison and results and chapter 5 gives the discussion and conclusion. The appendix contains the R code used in this research.

2 Methodology

This section is laid out as follows. First, we give the notation used. Second, we describe the methods in the literature, namely average linkage and k-means. Third, we give the different methods used to solve the convex clustering problem. Finally, we describe the adjusted Rand index.

We consider the following notation. $\boldsymbol{X}$ is a $n \times d$ data matrix consisting of n observations with d features. $\boldsymbol{x}_i$ represents row i of matrix $\boldsymbol{X}$. Let $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ be a partitioning of $\boldsymbol{X}$ into k clusters. The following properties hold: $C_i \subseteq \{1, \dots, n\}$, $C_i \cap C_j = \emptyset$ and $\cup_{i=1}^{k} C_i = \{1, \dots, n\}$ for $i, j = 1, \dots, k$ and $i \neq j$. The content of cluster C_i is a list of integers corresponding to rows of $\boldsymbol{X}$ that form a cluster.

To replicate the results of Hocking et al. (2011) we compare k-means, clusterpath and average linkage in their accuracy and runtime. The accuracy is measured with the normalized Rand index discussed later in this section. We first discuss average linkage and k-means. Groenen and Dalmeijer (2016b) explains these clustering methods in detail.

Average linkage clustering is a agglomerative clustering method, which means that the observations each start in a separate cluster. Let $\mathcal{C}$ be the set of k clusters, $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$. Each observation $\boldsymbol{x}_i$, $i = 1, \dots, n$, starts in a single cluster C_i, so we have $k = n$ and $C_i = \{i\}$.

In the first iteration of the algorithm, the dissimilarities between the clusters is defined as the dissimilarities between the observations $\boldsymbol{x}_i$. Next, in each iteration, the two clusters with the smallest dissimilarity between them are merged into one cluster. If clusters C_i and C_j are the clusters with the smallest dissimilarity, they are combined to cluster $C_t = C_i \cup C_j$. The set of clusters $\mathcal{C}$ is then updated by $\mathcal{C}_{new} = \mathcal{C}_{old} \setminus C_i \setminus C_j \cup C_t$, with $\setminus$ the set difference operator. The dissimilarity matrix is updated with respect to the other clusters $C_q \in \mathcal{C}_{new} \setminus C_t$ as follows:

$$\text{diss}_{C_t C_q} = \frac{|C_i|\text{diss}_{C_i C_q} + |C_j|\text{diss}_{C_j C_q}}{|C_i|+|C_j|}, \tag{2}$$

were *diss* is the function to calculate dissimilarities, for example the Euclidean distance. The algorithm stops when there is one cluster left.

The advantages of this algorithm are that it is computationally efficient. The interpetation is that clusters that are close to each other are merged. The algorithm has a tendency to produce long, stringy clusters and non convex cluster shapes.

Another popular method is k-means, it minimizes the within cluster distance by choosing k appropriate centroids and partitions of the data. The number of clusters k is known in advance. It minimizes the cost function

$$\text{ESS} = \|\boldsymbol{X} - \boldsymbol{U}\boldsymbol{M}\|_F^2, \tag{3}$$

where $\|\cdot\|_F^2$ is the squared Frobenius norm, $\boldsymbol{X}$ is the $n \times d$ data matrix and $\boldsymbol{M}$ is a $k \times d$ matrix of the k cluster centroids. Also, $\boldsymbol{U}$ is a $n \times k$ matrix with $u_{ij} = 1$ if observation i belongs to cluster j and 0 otherwise, for $1 \leq i \leq n$ and $1 \leq j \leq k$.

Hartigan and Wong (1979) describe a fast algorithm to minimize the cost function. The basic idea is to assign each point to its closest cluster center, updating the cluster centers and then transferring points between clusters so that the within cluster distance gets smaller. Repeating this procedure results in a local minimum. To increase the probability of finding the global minimum Groenen and Dalmeijer (2016b) advice to use many random starts for the cluster centers.

In contrast, the convex clustering problem has a global minimum for different values of λ. Finding these reveals a solution path that minimize (1) for different values of λ, we call this the *clusterpath*. For $\lambda = 0$, the solution is $\boldsymbol{A} = \boldsymbol{X}$ and for some large λ, the rows in $\boldsymbol{A}$ are all equal to the mean of $\boldsymbol{X}$ because the distance between the rows is set to zero. For the λ's in between, rows of $\boldsymbol{A}$ move towards each other and towards the center of $\boldsymbol{X}$. As this happens, rows of $\boldsymbol{A}$ can get close to each other and cluster which means that the rows describe the same cluster centroid.

This can be exploited by using $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{M}$, in the same fashion as the k-means cost function. Note however, that the number of clusters does not have to be set in advance as is the case with k-means, because as λ increases, more clusters fuse together and $\boldsymbol{U}$ and $\boldsymbol{M}$ shrink with the number of clusters.

Revealing the clusterpath can be done iteratively by starting with a low value for λ and then using that solution as a warm restart for the minimization problem with a larger λ. This works because the l_2 clusterpath is continuous (Hocking et al., 2011). Algorithm 1 describes the clusterpath.

Algorithm 1: Clusterpath-L2

```
Input: data $\boldsymbol{X} \in \mathbb{R}^{n\times d}$, weights matrix $\boldsymbol{W} \in \mathbb{R}^{n\times n}$, with $w_{ij} = w_{ji} \geq 0, w_{ii} = 0$
$t \leftarrow 0$;                                                              1
$\boldsymbol{M}_t, \boldsymbol{U}_t \leftarrow$ Detect-cluster-fusion($\boldsymbol{X}, \boldsymbol{I}_n$);                      2
while $|clusters| > 1$ do                                                      3
    $\boldsymbol{M}_{t+1}, \boldsymbol{U}_{t+1} \leftarrow$ Solve-L2($\boldsymbol{U}_t, \boldsymbol{M}_t, \lambda_t$);               4
    $\lambda_{t+1} \leftarrow \lambda_t \times 1.5$;                                          5
    $t \leftarrow t + 1$;                                                      6
end                                                                            7
return Optimal $\boldsymbol{A}_t = \boldsymbol{U}_t\boldsymbol{M}_t$ with corresponding $\lambda_t$ for all $t$           8
```

Where Hocking et al. (2011) pass the solution matrix $\boldsymbol{A}$ to `Solve-L2`, we pass $\boldsymbol{U}$ and $\boldsymbol{M}$. This is possible, because the rows of $\boldsymbol{A}$ that are clustered are identical. This trick of splitting $\boldsymbol{A}$ is important, because it makes the method `Solve-L2` faster. This is because the method now only has to update the cluster centroids in $\boldsymbol{M}$ instead of all rows of $\boldsymbol{A}$. When more rows of $\boldsymbol{A}$ get clustered, $\boldsymbol{M}$ gets less rows and can be updated quicker.

Note however, that once clusters have fused, they can not unfuse. This can be problematic for finding the optimal solution, Hocking et al. (2011) even found an example in which the optimal solution path contained a cluster split. Using this trick thus means that the optimal solution might not be found. They conjecture that the clusterpath does not split for the weights $w_{ij} = \exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2)$, but this has not been proven. Hocking et al. (2011) did not observe cluster splits in their calculated solution paths for identity weights and neither did we.

For clustering, Hocking et al. (2011) use a method called `Detect-cluster-fusion`. This method fuses clusters together when the distance between their centroids is small. Two clusters $C_1, C_2 \in \mathcal{C}$ fuse when

$$\|\boldsymbol{m}_{C_1} - \boldsymbol{m}_{C_2}\|_2 < \delta \min_{1\leq i\neq j\leq n} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 \, . \tag{4}$$

δ is a fraction that determines how close the cluster centroids have to be related to distance between the two closest point in the data matrix $\boldsymbol{X}$. Using this makes the threshold for clustering dependent on the dataset used. This is important because this prevents data points from clustering in early iterations and it makes sure that the clusterpath is not dependent on the scale of the data. When clusters C_1 and C_2 fuse, the centroid of the new cluster C becomes

$$\boldsymbol{m}_C = \frac{|C_1|\boldsymbol{m}_{C_1} + |C_2|\boldsymbol{m}_{C_2}}{|C_1|+|C_2|}. \tag{5}$$

The method then returns the new clustering $\boldsymbol{U}$ and the updated cluster centroids $\boldsymbol{M}$. Algorithm

2 gives a detailed description of the fusion algorithm.

```
Algorithm 2: Detect-cluster-fusion
Input: $\boldsymbol{M}$, $\boldsymbol{U}$ and cluster threshold
if $rank(\boldsymbol{U}) = 1$ then                                    1
    return $\boldsymbol{M}$ and $\boldsymbol{U}$                      2
end                                                                   3
$i,j \leftarrow \text{argmin}_{i<j} \|\boldsymbol{m}_i - \boldsymbol{m}_j\|_2$;    4
if $\|\boldsymbol{m}_i - \boldsymbol{m}_j\|_2 < threshold$ then       5
    combine rows $i$ and $j$ of $\boldsymbol{M}$ with (5);            6
    combine columns $i$ and $j$ of $\boldsymbol{U}$ to $\boldsymbol{u}_{\cdot i} + \boldsymbol{u}_{\cdot j}$;    7
end                                                                   8
return $\boldsymbol{M}$ and $\boldsymbol{U}$                          9
```

Before finding the solution path, we apply a few rounds of cluster fusions with a small threshold. This is to cluster points that lie on top of each other and to prevent points starting with a distance of zero to each other, which result in problems. For example, the calculation of the gradient has division by the distance between points and if this distance is zero numerical problems arise.

Next, we give different methods for solving the convex clustering problem for a given value of λ. First we describe the gradient descent method used by Hocking et al. (2011) to solve the l_2 convex clustering problem. By looking at the first order conditions of (1), they find the following sufficient condition for an optimal $\boldsymbol{A}$:

$$\boldsymbol{0} = \boldsymbol{a}_i - \boldsymbol{x}_i + \lambda \sum_{\substack{j \neq i \\ \boldsymbol{a}_j \neq \boldsymbol{a}_i}} w_{ij} \frac{\boldsymbol{a}_i - \boldsymbol{a}_j}{\|\boldsymbol{a}_i - \boldsymbol{a}_j\|_2} + \lambda \sum_{\substack{j \neq i \\ \boldsymbol{a}_j = \boldsymbol{a}_i}} w_{ij} \boldsymbol{\beta}_{ij} \tag{6}$$

with $\boldsymbol{\beta}_{ij} \in \mathbb{R}^d$, $\|\boldsymbol{\beta}_{ij}\|_2 \leq 1$ and $\boldsymbol{\beta}_{ij} = -\boldsymbol{\beta}_{ji}$. To get the gradient for the cluster $C = \{i : \boldsymbol{a}_i = \boldsymbol{a}_C\}$, they sum over all $i \in C$ to get

$$\boldsymbol{g}_C = \boldsymbol{a}_C - \bar{\boldsymbol{x}}_C + \frac{\lambda}{|C|} \sum_{j \notin C} w_{jC} \frac{\boldsymbol{a}_C - \boldsymbol{a}_j}{\|\boldsymbol{a}_C - \boldsymbol{a}_j\|_2}, \tag{7}$$

where $\bar{\boldsymbol{x}}_C = \sum_{i \in C} \boldsymbol{x}_i / |C|$ and $w_{jC} = \sum_{i \in C} w_{ij}$. This gradient is used in the gradient descent step. The gradient descent method is given in algorithm 3.

```
Algorithm 3: Solve-L2-gradient
Input: initial guess $\boldsymbol{M}$, initial clustering $\boldsymbol{U}$, data $\boldsymbol{X}$, weights matrix $\boldsymbol{W}$, parameter $\lambda$
$\boldsymbol{G} \leftarrow$ Subgradient-L2($\cdot$);                  1
while $\frac{1}{n} \|\boldsymbol{G}\|_F^2 > eps$ do                   2
    $\boldsymbol{M} \leftarrow$ Subgradient-step($\cdot$);            3
    $\boldsymbol{M}, \boldsymbol{U} \leftarrow$ Detect-cluster-fusion($\cdot$);    4
    $\boldsymbol{G} \leftarrow$ Subgradient-L2($\cdot$);              5
end                                                                   6
return $\boldsymbol{M}, \boldsymbol{U}$                               7
```

In algorithm 3, `Subgradient-L2` calculates the gradients per cluster and returns a matrix $\boldsymbol{G}$, with on row i the gradient belonging to cluster i using (7). In `Subgradient-step` the matrix

$\boldsymbol{M}$ gets updated. Because the method takes many steps to converge to the optimal solution, Hocking et al. (2011) alternate every step with a decreasing step or a line search. The decreasing step is the update $\boldsymbol{M} \leftarrow \boldsymbol{M} - r\boldsymbol{G}$, with $r = 1/\textit{iteration}$. For the line search method we use the golden section method, because this method requires a small number of function evaluations.

The golden section line search method finds the minimum of a scalar valued function $f(x)$ in an interval $[a, b]$. The method tries to use as few function evaluations as possible, by iteratively decreasing interval by the golden ratio $\phi = (1 + \sqrt{5})/2$, hence the method is named the golden section method. We give an algorithm of the method in appendix A.

In our implementation, we use $f(r) = f_2(\boldsymbol{U}(\boldsymbol{M} - r\boldsymbol{G}), \boldsymbol{X})$, with f_2 being the cost function (1). The choice for the interval is $[a, b] = [0, 1]$ and for the accuracy $\epsilon = 0.001$. Because the interval shrinks with $1/\phi$ each iteration golden section takes $\log\frac{\epsilon}{b-a}/\log(\frac{1}{\phi})$ function evaluations to complete (round up to nearest integer). In this case golden section takes 15 function evaluations to converge.

A disadvantage of the gradient descent method is that it takes many iterations before finding the optimum, even with the use of linesearch techniques. This research tries to solve this issue, by using majorization methods to solve the convex clustering problem. Let us first look at what majorization is exactly.

Majorization Minimization (Hunter & Lange, 2004) is a technique used to find a minimum of an objective function $f(x)$ with respect to x. The main idea of majorization is to replace a difficult optimization problem by a simple problem. Then the optimal solution of the simple problem can be used to make a step towards the optimum of the difficult problem. This is done iteratively until the process finds the optimum of the difficult problem. The cost function of the simple problem is called a surrogate function, and it *majorizes* $f(x)$.

Lange and Keys (2015) explain that if a surrogate function $g(x \mid x_n)$ majorizes $f(x)$, two properties hold: the tangency condition $g(x_n \mid x_n) = f(x_n)$ and the domination condition $g(x \mid x_n) \geq f(x)$. These mean that $g(x \mid x_n)$ lies above $f(x)$ for all x and $g(x \mid x_n)$ touches $f(x)$ at $x = x_n$. Now, to get a step closer to the minimum of $f(x)$, we can use the following: if x^* is the minimum of $g(x \mid x_n)$ with respect to x, then

$$f(x^*) \leq g(x^* \mid x_n) \leq g(x_n \mid x_n) = f(x_n). \tag{8}$$

This shows that minimizing $g(x \mid x_n)$ with respect to x results in a lower objective function $f(x)$. Below we give a general algorithm for majorization (from Groenen (2017)).

Algorithm 4: Majorization algorithm

```
Input: some initial $\boldsymbol{x}_0 \in \mathbb{R}^n$
Set $k \leftarrow 1$;                                                1
while $k = 1$ or $|f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1})| > \epsilon$ do          2
    $\boldsymbol{x}_{k+1} \leftarrow \text{argmin}_{\boldsymbol{x}}\, g(\boldsymbol{x}|\boldsymbol{x}_k)$;            3
    $k \leftarrow k + 1$;                                            4
end                                                                  5
return $\boldsymbol{x}_k$                                                 6
```

The majorization algorithm stops if the change in the objective function due to the updates gets small. This is the case when the optimization is in a local minimum. Thus the majorization algorithm converges to a stationary point of the objective function. The difficulty of majorization is finding a surrogate function that majorizes the objective function. But if it can be shown that a surrogate function exists and the minimum of this surrogate can be found easily, then majorization is a fast method to find the minimum of the objective function.

A method that uses majorization is the proximal distance algorithm, which is used to solve constrained optimization problems (Lange & Keys, 2015). The idea of this method is to minimize $f(x)$ constrained by $x \in \mathcal{R}$, with $\mathcal{R}$ a set of constraints forming a closed set. Clarke (1990) replace the constraint problem by $f(x) + \rho \operatorname{dist}(x, \mathcal{R})$, where $\operatorname{dist}(x, \mathcal{R}) = \inf\{|r - x| : r \in \mathcal{R}\}$ represents the distance from x to $\mathcal{R}$. If x adheres to the constraints in $\mathcal{R}$, then the distance from x to $\mathcal{R}$ becomes small. A solution that minimizes $f(x) + \rho \operatorname{dist}(x, \mathcal{R})$ is a solution that has a low value for $f(x)$ while still adhering to the restrictions. For larger ρ's, the restrictions get imposed more on the solution.

For solving the constraint problem, Lange and Keys (2015) use an approximate objective function. They replace $f(x) + \rho \operatorname{dist}(x, \mathcal{R})$ by $f(x) + \rho \sqrt{\operatorname{dist}(x, \mathcal{R})^2 + \epsilon}$. This is an approximation, but if epsilon tends to zero, the two functions become equal. This approximate function can be majorized in two steps. Lange and Keys (2015) propose

$$\operatorname{dist}(x, \mathcal{R}) \leq \|x - P_{\mathcal{R}}(x_k)\|_2 \,, \tag{9}$$

where $P_{\mathcal{R}}(x_k)$ is the projection of the current iterate x_k on set $\mathcal{R}$, and the Taylor expansion of the square root

$$\sqrt{t + \epsilon} \leq \sqrt{t_k + \epsilon} + \frac{1}{2\sqrt{t_k + \epsilon}}(t - t_k). \tag{10}$$

Applying these to majorizations to the approximate objective function results in the surrogate function

$$\begin{aligned} g(\boldsymbol{x} \mid \boldsymbol{x}_n) &= f(\boldsymbol{x}) + \frac{w_n}{2} \|\boldsymbol{x} - P_{\mathcal{R}}(\boldsymbol{x}_n)\|^2 \\ w_n &= \frac{\rho}{\sqrt{\|\boldsymbol{x}_n - P_{\mathcal{R}}(\boldsymbol{x}_n)\|^2 + \epsilon}}. \end{aligned} \tag{11}$$

Solving this function with respect to $\boldsymbol{x}$ results in a majorization update. This procedure is called the proximal distance method.

Chen et al. (2015) use this method to solve the convex clustering problem. We will show their steps and expand upon them and improve them. Let us first look at how they used the proximal distance algorithm on the convex clustering problem.

First, to introduce constraints in the convex clustering problem, they replace (1) by:

$$f_2(\boldsymbol{A}, \boldsymbol{X}) = \frac{1}{2} \|\boldsymbol{A} - \boldsymbol{X}\|_F^2 + \lambda \sum_{i<j} w_{ij} \|\boldsymbol{v}_{ij}\|_2 \,, \tag{12}$$

where $\boldsymbol{v}_{ij}$ is equal to $\boldsymbol{a}_i - \boldsymbol{a}_j$, $\|\cdot\|_F^2$ is the squared Frobenius norm and $\|\cdot\|_2$ is the l_2-norm or Euclidean norm. Now, by applying the proximal distance method to this cost function, we get the approximate objective function

$$h[(\boldsymbol{A},\boldsymbol{V})|(\boldsymbol{A}_k,\boldsymbol{V}_k)] = \frac{1}{2}\left\|\boldsymbol{A}-\boldsymbol{X}\right\|_F^2 + \lambda\sum_{i<j} w_{ij}\left\|\boldsymbol{v}_{ij}\right\|_2 + \frac{\rho}{2d_k}\left\|\begin{pmatrix}\boldsymbol{A}\\ \boldsymbol{V}\end{pmatrix} - P_{\mathcal{R}}\begin{pmatrix}\boldsymbol{A}_k\\ \boldsymbol{V}_k\end{pmatrix}\right\|_F^2 \tag{13}$$

$$d_k = \sqrt{\left\|\begin{pmatrix}\boldsymbol{A}_k\\ \boldsymbol{V}_k\end{pmatrix} - P_{\mathcal{R}}\begin{pmatrix}\boldsymbol{A}_k\\ \boldsymbol{V}_k\end{pmatrix}\right\|_F^2 + \epsilon}.$$

In this surrogate function, $\boldsymbol{A}_k$ and $\boldsymbol{V}_k$ are the matrices of the current iterate. The optimal solution of the surrogate function can be found by looking at the first order conditions. For the update of $\boldsymbol{A}$ we solve and get

$$\frac{\partial h[(\boldsymbol{A},\boldsymbol{V})|(\boldsymbol{A}_k,\boldsymbol{V}_k)]}{\partial\,(\text{vec}\,\boldsymbol{A})'} = \boldsymbol{A}-\boldsymbol{X}+\frac{\rho}{d_k}(\boldsymbol{A}-\boldsymbol{A}_k^p) = \boldsymbol{0}$$

$$\implies \boldsymbol{A}_{k+1} = \frac{d_k}{d_k+\rho}\boldsymbol{X} + \frac{\rho}{d_k+\rho}\boldsymbol{A}_k^p \tag{14}$$

with $\boldsymbol{A}_k^p$ the part of the projection pertaining to $\boldsymbol{A}_k$. Chen et al. (2015) found the update for $\boldsymbol{v}_{ij}$ by minimizing $\lambda w_{ij}\left\|\boldsymbol{v}_{ij}\right\|_2 + \frac{\rho}{2d_k}\left\|\boldsymbol{v}_{ij}-\boldsymbol{v}_{k,ij}^p\right\|^2$. This results in

$$\boldsymbol{v}_{k+1,ij} = \max\left\{\left(1-\frac{\lambda w_{ij}d_k}{\rho\left\|\boldsymbol{v}_{k,ij}^p\right\|}\right),0\right\}\boldsymbol{v}_{k,ij}^p. \tag{15}$$

Here, $\boldsymbol{v}_{k,ij}^p$ is the part of the projection pertaining to $\boldsymbol{v}_{k,ij}$.

For finding the projection $\boldsymbol{A}^p$ and $\boldsymbol{V}^p$ of $P_{\mathcal{R}}\binom{\boldsymbol{A}}{\boldsymbol{V}}$ we minimize

$$m(\boldsymbol{A}^p) = \frac{1}{2}\sum_{i=1}^{n}\left\|\boldsymbol{a}_i^p-\boldsymbol{a}_i\right\|_2^2 + \frac{1}{2}\sum_{i\neq j}\left\|\boldsymbol{a}_i^p-\boldsymbol{a}_j^p-\boldsymbol{v}_{ij}\right\|_2^2. \tag{16}$$

Chen et al. (2015) use a block update approach, but we found a way to solve it explicitly. This can be done by writing function m as

$$m(\boldsymbol{A}) = \frac{1}{2}\text{Tr}\left((\boldsymbol{A}^p-\boldsymbol{A})(\boldsymbol{A}^p-\boldsymbol{A})'\right) + \frac{1}{2}\text{Tr}\left(\boldsymbol{A}^{p\,\prime}\boldsymbol{V}_1\boldsymbol{A}^p\right) - \text{Tr}\left(\boldsymbol{A}^{p\,\prime}\boldsymbol{V}_2\right) + \sum_{i\neq j}\text{Tr}\left(\boldsymbol{v}_{ij}\boldsymbol{v}_{ij}'\right),$$

$$\boldsymbol{V}_1 = \sum_{i\neq j}\boldsymbol{E}_{ij},\quad \boldsymbol{E}_{ij} = (\boldsymbol{e}_i-\boldsymbol{e}_j)(\boldsymbol{e}_i-\boldsymbol{e}_j)',$$

$$\boldsymbol{V}_2 = \sum_{i\neq j}(\boldsymbol{e}_i-\boldsymbol{e}_j)\boldsymbol{v}_{ij}.$$

Solving the first order conditions yield

$$\frac{\partial m(\boldsymbol{A}^p)}{\partial\,(\text{vec}\,\boldsymbol{A}^p)'} = \boldsymbol{A}^p-\boldsymbol{A}+\boldsymbol{V}_1'\boldsymbol{A}^p-\boldsymbol{V}_2 = \boldsymbol{0}$$

$$\implies \boldsymbol{A}^p = (\boldsymbol{I}_n+\boldsymbol{V}_1)^{-1}(\boldsymbol{A}+\boldsymbol{V}_2). \tag{17}$$

Here, $\boldsymbol{I}_n$ is the $n\times n$ identity matrix. We can explicitly solve $(\boldsymbol{I}_n+\boldsymbol{V}_1)^{-1}$, because $\boldsymbol{V}_1$ is a matrix with -2 on the off diagonal elements and $2(n-1)$ on the diagonal elements. We can use the Sherman Morrison identity (Bartlett, 1951)

$$(\boldsymbol{Q}+\boldsymbol{x}\boldsymbol{y}')^{-1} = \boldsymbol{Q}^{-1} - \frac{\boldsymbol{Q}^{-1}\boldsymbol{x}\boldsymbol{y}'\boldsymbol{Q}^{-1}}{1+\boldsymbol{y}'\boldsymbol{Q}^{-1}\boldsymbol{x}} \tag{18}$$

to get the inverse of $\boldsymbol{I}_n + \boldsymbol{V}_1$. We use $\boldsymbol{Q}$ with $2n+1$ as diagonal elements and zero on the off diagonal elements, $\boldsymbol{x} = \iota$ and $\boldsymbol{y} = -2\iota$. ι is a vector with only ones. This results in that $(\boldsymbol{I}_n + \boldsymbol{V}_1)^{-1}$ is a matrix with $3/(2n+1)$ on the diagonal elements and $2/(2n+1)$ on the off diagonal elements. To get the projection $\boldsymbol{V}^p$ we use $\boldsymbol{v}_{ij}^p = \boldsymbol{a}_i^p - \boldsymbol{a}_j^p$.

We can use the clustering of $\boldsymbol{A}$ with $\boldsymbol{A} = \boldsymbol{UM}$ to get the following update

$$\begin{aligned}\frac{\partial h[(\boldsymbol{UM}, \boldsymbol{V})|(\boldsymbol{A}_k, \boldsymbol{V}_k)]}{\partial\,(\mathrm{vec}\,\boldsymbol{M})'} &= \boldsymbol{U}'\boldsymbol{UM} - \boldsymbol{U}'\boldsymbol{X} + \frac{\rho}{d_k}(\boldsymbol{U}'\boldsymbol{UM} - \boldsymbol{U}'\boldsymbol{A}_k^p) = \boldsymbol{0} \\ \implies \boldsymbol{M}_{k+1} &= (\boldsymbol{U}'\boldsymbol{U})^{-1}\boldsymbol{U}'\left(\frac{d_k}{d_k+\rho}\boldsymbol{X} + \frac{\rho}{d_k+\rho}\boldsymbol{A}_k^p\right). \end{aligned} \tag{19}$$

This trick does unfortunately not use the clustering, as the update is the mean of the clusters of the update of $\boldsymbol{A}_{k+1}$.

Lange and Keys (2015) propose to start with ρ small to emphasize minimization of the cost function in early iterations. They also advice to gradually decrease ϵ to avoid the risk enforcing the constraints too strongly in early iterations. They suggest the sequences $\rho_k = \min(\alpha^k\rho_0, \rho_{\max})$ and $\epsilon_k = \min(\beta^{-k}\epsilon_0, \epsilon_{\min})$ with α and β larger than 1 and $\rho_0 = \epsilon_0 = 1$. They say that on many problems aggresive choices for α and β are possible. In our implementation we use $\alpha = 1.5, \beta = 1.5, \rho_0 = 1, \epsilon_0 = 10^{-3}, \rho_{\max} = 10^3$ and $\epsilon_{\min} = 10^{-6}$. This choice speeds up the convergence without sacrificing too much precision.

These updates result in a majorization algorithm. Here, ρ and ϵ also influence the value of the approximate objective function (13). Because we update these values each iteration, the cost function changes. This becomes problematic when increasing ρ, because this increases the approximate objective function and then the stop condition of majorization algorithm 4 is not valid anymore. To solve this, we change the stop condition to $\frac{|f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1})|}{|f(\boldsymbol{x}_{k-1})|} \leq \epsilon$, with f the objective function and $\boldsymbol{x}_k$ the kth iterate. Now, the algorithm stops when the approximate objective function stabilizes. The proximal distance algorithm is described by algorithm 5.

Algorithm 5: Solve-proximal

```
Input: initial guess $M$, initial clustering $U$, data $X$, weights matrix $W$, parameter $\lambda$
initialize $A \leftarrow UM$;                                     1
initialize $V$ with $v_{ij} \leftarrow a_i - a_j$;                2
while stopping condition is not reached do                        3
    $A_p \leftarrow$ find-projection($A$, $V$);                   4
    update $\rho$ and $\epsilon$;                                  5
    update $M$ and $V$;                                            6
    $M, U \leftarrow$ Detect-cluster-fusion($\cdot$);              7
    $A \leftarrow UM$;                                             8
end                                                               9
return $M, U$                                                     10
```

Another way to majorize (1) is by using the following trick: let $\boldsymbol{x}$ and $\boldsymbol{y}$ be vectors in $\mathbb{R}^n$ space and $\|\cdot\|_q$ be the l_q-norm. Then the following inequality holds

$$\begin{aligned}(\|\boldsymbol{x}\|_q - \|\boldsymbol{y}\|_q)^2 \geq 0 &\implies \|\boldsymbol{x}\|_q^2 - 2\|\boldsymbol{x}\|_q\|\boldsymbol{y}\|_q + \|\boldsymbol{y}\|_q^2 \geq 0 \\ &\implies 2\|\boldsymbol{x}\|_q\|\boldsymbol{y}\|_q \leq \|\boldsymbol{x}\|_q^2 + \|\boldsymbol{y}\|_q^2 \\ &\implies \|\boldsymbol{x}\|_q \leq \frac{1}{2}\frac{\|\boldsymbol{x}\|_q^2}{\|\boldsymbol{y}\|_q} + \frac{1}{2}\|\boldsymbol{y}\|_q.\end{aligned}$$

If we use this trick on the cost function, we get

$$\begin{aligned}f_q(\boldsymbol{A},\boldsymbol{X}) &= \frac{1}{2}\|\boldsymbol{A}-\boldsymbol{X}\|_F^2 + \lambda\sum_{i<j} w_{ij}\|\boldsymbol{a}_i - \boldsymbol{a}_j\|_q \\ &\leq \frac{1}{2}\|\boldsymbol{A}-\boldsymbol{X}\|_F^2 + \frac{1}{2}\lambda\sum_{i<j} w_{ij}\frac{\|\boldsymbol{a}_i - \boldsymbol{a}_j\|_q^2}{\|\boldsymbol{a}_{k,i} - \boldsymbol{a}_{k,j}\|_q} + \frac{1}{2}\lambda\sum_{i<j} w_{ij}\|\boldsymbol{a}_{k,i} - \boldsymbol{a}_{k,j}\|_q \\ &= g_q(\boldsymbol{A}|\boldsymbol{A}_k).\end{aligned}$$

This gives the surrogate function $g_q(\boldsymbol{A}|\boldsymbol{A}_k)$ to use in majorization. For $q = 2$, we can write it as

$$g_2(\boldsymbol{A}|\boldsymbol{A}_k) = \frac{1}{2}\mathrm{Tr}\left((\boldsymbol{X}-\boldsymbol{A})'(\boldsymbol{X}-\boldsymbol{A})\right) + \frac{1}{2}\lambda\mathrm{Tr}\left(\boldsymbol{A}'\boldsymbol{V}_k\boldsymbol{A}\right) + \frac{1}{2}\lambda\sum_{i<j} w_{ij}\|\boldsymbol{a}_{k,i} - \boldsymbol{a}_{k,j}\|_2 \tag{20}$$

$$\boldsymbol{V}_k = \sum_{i<j}\left[\frac{w_{ij}}{\|\boldsymbol{a}_{k,i} - \boldsymbol{a}_{k,j}\|_2}\boldsymbol{E}_{ij}\right], \quad \boldsymbol{E}_{ij} = (\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)'.$$

To get the minimum of $g_2(\boldsymbol{A}|\boldsymbol{A}_k)$ with respect to $\boldsymbol{A}$, we look at the first order conditions

$$\begin{aligned}\frac{\partial g_2(\boldsymbol{A}|\boldsymbol{A}_k)}{\partial\,(\mathrm{vec}\ \boldsymbol{A})'} &= \boldsymbol{A} - \boldsymbol{X} + \lambda\boldsymbol{V}_k\boldsymbol{A} = \boldsymbol{0} \\ \implies \boldsymbol{A}_{k+1} &= (\boldsymbol{I} + \lambda\boldsymbol{V}_k)^{-1}\boldsymbol{X}. \end{aligned} \tag{21}$$

By solving this system of equations, we get the next iteration of the majorization algorithm.

One problem with this update is the calculation of matrix $\boldsymbol{V}_k$, because if rows i and j of $\boldsymbol{A}$ get close to each other, $\|\boldsymbol{a}_{k,i} - \boldsymbol{a}_{k,j}\|_2$ tends to zero and then $\boldsymbol{V}_k$ can not be calculated because of division by zero. This can be prevented by ensuring that rows can not get closer to each other than a certain distance. Another solution is to again use the trick that $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{M}$. Let us first look at the derivation of the update of $\boldsymbol{M}$ by examining the first order condition

$$\frac{\partial\,\mathrm{vec}\ g_2(\boldsymbol{U}\boldsymbol{M}|\boldsymbol{A}_k)}{\partial\,(\mathrm{vec}\ \boldsymbol{M})'} = \boldsymbol{M}'\boldsymbol{U}'\boldsymbol{U} - \boldsymbol{X}'\boldsymbol{U} + \lambda\boldsymbol{M}'\boldsymbol{U}'\boldsymbol{V}_k\boldsymbol{U} = \boldsymbol{0} \tag{22}$$

$$\implies \boldsymbol{M}_{k+1} = (\boldsymbol{U}'\boldsymbol{U} + \lambda\boldsymbol{U}'\boldsymbol{V}_k\boldsymbol{U})^{-1}\boldsymbol{U}'\boldsymbol{X}. \tag{23}$$

Now, the matrix $\boldsymbol{V}_k$ is transformed by matrix $\boldsymbol{U}$, which results in that the elements that tend to infinity get subtracted out. An example of this is given in the proof of Theorem 1. The trick also reduces the computation time by a great deal, because in the update a smaller matrix has to be inverted. This makes the update faster to calculate in later iterations. Algorithm 6 describes the norm majorization algorithm.

```
Algorithm 6: Solve-norm-majorization
Input: initial guess $M$, initial clustering $U$, data $X$, weights matrix $W$, parameter $\lambda$
calculate $G$ with (22);                                  1
while $\frac{1}{n} \|G\|_F^2 > eps$ do                     2
    update $M$ with (23);                                  3
    $M, U \leftarrow$ Detect-cluster-fusion($\cdot$);      4
    calculate $G$ with (22);                               5
end                                                        6
return $M, U$                                              7
```

In the previous method solving a linear system is needed to get the majorization update. For large problems, say $n > 1000$, this will take a toll on the computation time. In (20) the part $\mathrm{Tr}\,(\boldsymbol{A}'\boldsymbol{V}_k\boldsymbol{A})$ is the culprit that makes matrix solving a system of equations necessary for finding the update. The remedy is another round of majorization. For a similar problem, Groenen, Heiser, and Meulman (1999) propose to use the inequality

$$\begin{aligned} &(\boldsymbol{x}-\boldsymbol{y})'(\boldsymbol{V}-\lambda\boldsymbol{I})(\boldsymbol{x}-\boldsymbol{y}) \leq 0 \\ \Longrightarrow\ &\boldsymbol{x}'\boldsymbol{V}\boldsymbol{x} \leq \lambda\boldsymbol{x}'\boldsymbol{x} - 2\boldsymbol{x}(\lambda\boldsymbol{I}-\boldsymbol{V})\boldsymbol{y} + \boldsymbol{y}'(\lambda\boldsymbol{I}-\boldsymbol{V})\boldsymbol{y} \end{aligned}$$

from Heiser (1987). Here $\boldsymbol{x}, \boldsymbol{y}$ are $\mathbb{R}^n$ vectors, $\boldsymbol{V}$ is a $n \times n$ matrix and λ is larger than or equal to the largest eigenvalue of $\boldsymbol{V}$.

This trick can be extended to a positive semidefinite matrix. This is a matrix whose eigenvalues are all equal or larger than zero. Let $\boldsymbol{Q}$ be such a positive semidefinite matrix, because it is positive semidefinite it follows that $\mathrm{Tr}\,(\boldsymbol{Q}) \geq 0$ and $\boldsymbol{B}'\boldsymbol{Q}\boldsymbol{B}$, with $\boldsymbol{B}$ a $n \times d$ matrix with rank d, is also positive semidefinite (Petersen & Pedersen, 2012). It then follows with $\boldsymbol{B} = \boldsymbol{A} - \boldsymbol{A}_k$ and $\boldsymbol{Q} = \boldsymbol{L} - \boldsymbol{V}_k$ that

$$\begin{aligned} &\mathrm{Tr}\left((\boldsymbol{A}-\boldsymbol{A}_k)'(\boldsymbol{L}-\boldsymbol{V}_k)(\boldsymbol{A}-\boldsymbol{A}_k)\right) \geq 0 \\ \Longrightarrow\ &\mathrm{Tr}\left(\boldsymbol{A}'\boldsymbol{V}_k\boldsymbol{A}\right) \leq \mathrm{Tr}\left(\boldsymbol{A}'\boldsymbol{L}\boldsymbol{A}\right) - 2\mathrm{Tr}\left(\boldsymbol{A}'(\boldsymbol{L}-\boldsymbol{V}_k)\boldsymbol{A}_k\right) + \mathrm{Tr}\left(\boldsymbol{A}_k'(\boldsymbol{L}-\boldsymbol{V}_k)\boldsymbol{A}_k\right), \end{aligned}$$

with $\boldsymbol{L}$ a zero matrix with upper bounds of the eigenvalues of $\boldsymbol{V}_k$ as diagonal elements.

It is now left to prove that $\boldsymbol{Q} = \boldsymbol{L} - \boldsymbol{V}_k$ is positive semidefinite. This can be done using the eigendecomposition of $\boldsymbol{V}_k$ as $\boldsymbol{P\Lambda P}'$. We can show that $\boldsymbol{L} - \boldsymbol{V}_k = \boldsymbol{PLP}' - \boldsymbol{P\Lambda P}' = \boldsymbol{P}(\boldsymbol{L}-\boldsymbol{\Lambda})\boldsymbol{P}'$ because $\boldsymbol{PP}' = \boldsymbol{I}$ and $\boldsymbol{PLP}' = \boldsymbol{LPP}'$. With $\boldsymbol{L} - \boldsymbol{\Lambda} \geq \boldsymbol{0}$, all eigenvalues of $\boldsymbol{L} - \boldsymbol{V}_k$ are positive, and thus it is a positive semidefinite matrix.

Applying this majorization on (20) results in the following surrogate function

$$
\begin{aligned}
f_2(\boldsymbol{A},\boldsymbol{X}) &= \frac{1}{2}\left\|\boldsymbol{A}-\boldsymbol{X}\right\|_F^2 + \lambda\sum_{i<j} w_{ij}\left\|\boldsymbol{a}_i-\boldsymbol{a}_j\right\|_q \\
&\leq \frac{1}{2}\mathrm{Tr}\left((\boldsymbol{X}-\boldsymbol{A})'(\boldsymbol{X}-\boldsymbol{A})\right) + \frac{1}{2}\lambda\mathrm{Tr}\left(\boldsymbol{A}'\boldsymbol{V}_k\boldsymbol{A}\right) + \frac{1}{2}\lambda\sum_{i<j} w_{ij}\left\|\boldsymbol{a}_{k,i}-\boldsymbol{a}_{k,j}\right\|_2 \\
&\leq \frac{1}{2}\mathrm{Tr}\left((\boldsymbol{X}-\boldsymbol{A})'(\boldsymbol{X}-\boldsymbol{A})\right) + \frac{1}{2}\lambda\mathrm{Tr}\left(\boldsymbol{A}'\boldsymbol{L}\boldsymbol{A}\right) - \lambda\mathrm{Tr}\left(\boldsymbol{A}'(\boldsymbol{L}-\boldsymbol{V}_k)\boldsymbol{A}_k\right) \\
&+ \frac{1}{2}\lambda\mathrm{Tr}\left(\boldsymbol{A}_k'(\boldsymbol{L}-\boldsymbol{V}_k)\boldsymbol{A}_k\right) + \frac{1}{2}\lambda\sum_{i<j} w_{ij}\left\|\boldsymbol{a}_{k,i}-\boldsymbol{a}_{k,j}\right\|_2 \\
&= g_2(\boldsymbol{A}|\boldsymbol{A}_k).
\end{aligned}
$$

Because the complexity of calculating the eigenvalues of a matrix is the same as solving a linear system, namely $O(n^3)$, we use another trick to get an upper bound on the largest eigenvalue of $\boldsymbol{V}_k$. This can be done with Gerschgorin's Theorem (Richard, 1989) that states that the eigenvalues of a matrix $\boldsymbol{V}$ fall in the intervals $[v_{ii}-\sum_{j\neq i}|v_{ij}|, v_{ii}+\sum_{j\neq i}|v_{ij}|]$. Here, the diagonal elements of $\boldsymbol{L}$ are a upper bound on the eigenvalues of $\boldsymbol{V}_k$. $\boldsymbol{L}$ can be calculated easily because of the structure of $\boldsymbol{V}_k$, as the diagonal elements of $\boldsymbol{V}_k$, $v_{k,ii}$, are equal to the absolute sum of the ith row of $\boldsymbol{V}_k$. Also, the off diagonal elements in $\boldsymbol{V}_k$ are negative. This means that the upper bound on the eigenvalues of $\boldsymbol{V}_k$ is equal to $l_{ii} = v_{k,ii} + \sum_{j=1}^{n}|v_{k,ij}| = 2v_{k,ii}$.

Now the update can be derived by looking at the first order conditions using $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{M}$

$$
\begin{aligned}
\frac{\partial \text{ vec } g_2(\boldsymbol{U}\boldsymbol{M}|\boldsymbol{A}_k)}{\partial\, (\text{vec } \boldsymbol{M})'} &= \boldsymbol{U}'\boldsymbol{U}\boldsymbol{M} - \boldsymbol{U}'\boldsymbol{X} + \lambda\boldsymbol{U}'\boldsymbol{L}\boldsymbol{U}\boldsymbol{M} - \lambda\boldsymbol{U}'(\boldsymbol{L}-\boldsymbol{V}_k)\boldsymbol{A}_k = \boldsymbol{0} \\
\implies \boldsymbol{M}_{k+1} &= (\boldsymbol{U}'\boldsymbol{U} + \lambda\boldsymbol{U}'\boldsymbol{L}\boldsymbol{U})^{-1}(\boldsymbol{U}'\boldsymbol{X} + \lambda\boldsymbol{U}'(\boldsymbol{L}-\boldsymbol{V}_k)\boldsymbol{A}_k). \qquad (24)
\end{aligned}
$$

The advantage of this update is that no difficult matrix inverses have to be calculated. Because $\boldsymbol{U}'\boldsymbol{U}+\lambda\boldsymbol{U}'\boldsymbol{L}\boldsymbol{U}$ is a matrix with zeros on the off-diagonal elements, its inverse can be calculated quickly.

However, This update does have a problem. If two rows of $\boldsymbol{A}_k$ get a small distance, the matrix $\boldsymbol{V}_k$ gets a large value in the corresponding row. This means that one of the eigenvalues of $\boldsymbol{V}_k$ will tend to infinity and then the update for the majorization can not be calculated. To solve this problem we replace $\mathrm{Tr}\,(\boldsymbol{A}'\boldsymbol{V}_k\boldsymbol{A})$ in (20) with $\mathrm{Tr}\,(\boldsymbol{M}'\boldsymbol{U}'\boldsymbol{V}_k\boldsymbol{U}\boldsymbol{M})$. On the matrix $\boldsymbol{C}_k = \boldsymbol{U}'\boldsymbol{V}_k\boldsymbol{U}$ smaller upper bounds on its eigenvalues can be computed which solves the problem. Also, this matrix has special properties, as it looks just like matrix $\boldsymbol{V}_k$.

Theorem 1. *Let $\boldsymbol{V}$ be a $n\times n$ symmetric matrix with negative values as off-diagonal elements and the absolute sum of each row as diagonal elements. Let $\boldsymbol{U}$ be a $n\times k$ matrix with one element per row equal to 1 and the other elements equal to zero. Then $\boldsymbol{C} = \boldsymbol{U}'\boldsymbol{V}\boldsymbol{U}$ is a symmetric matrix with negative off-diagonal elements and the absolute sum of each row as diagonal elements.*

Proof. $\boldsymbol{V}$ Looks as follows

$$
\boldsymbol{V} = \begin{pmatrix} \sum_{j\neq 1} v_{1j} & -v_{12} & \cdots & -v_{1n} \\ -v_{21} & \sum_{j\neq 2} v_{2j} & \cdots & -v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -v_{n1} & -v_{n2} & \cdots & \sum_{j\neq n} v_{nj} \end{pmatrix}, \qquad (25)
$$

with $v_{ij} = v_{ji}$ for $1 \leq i, j \leq n$. Because $\boldsymbol{U}$ contains only ones and zeros, multiplying by it from left and right means that rows and columns are summed together. If for example $\boldsymbol{U}$ contains a 1 in the same position for the first and second row, then these rows get summed together, and then the first and second column get summed together. This means that the new off-diagonal elements of the first row and column become $-(v_{1j} + v_{1j})$ for $j = 3, \ldots, n$ and the diagonal element becomes $\sum_{j=3}^{n}(v_{1j} + v_{1j})$. This results in a matrix with the same characteristics as $\boldsymbol{V}$, thus repeated operations of summing rows and columns together results in matrix $\boldsymbol{C}$ with the property that the diagonal elements remain the sum of the off-diagonal elements of the corresponding row. $\square$

Theorem 1 can be applied on $\boldsymbol{V}_k$ defined by (20), because it has $-w_{ij}/\left\|\boldsymbol{a}_{k,i} - \boldsymbol{a}_{k,j}\right\|$ as off-diagonal elements and the sum of absolute elements in a row as diagonal elements. Because a norm and the weights are all positive, the off diagonal elements of $\boldsymbol{V}_k$ are all negative. Also, the weights and the distances between rows of a matrix are symmetric, thus $\boldsymbol{V}_k$ is also symmetric. The resulting matrix $\boldsymbol{C}_k$ has the elements that resulted in large eigenvalues (v_{12} in the example of the proof) removed by subtracting them out, because the rows with large values are clustered. This is important, because now Gerschgorin's Theorem (Richard, 1989) can be applied for finding smaller upper bounds of the eigenvalues of $\boldsymbol{C}_k$, namely $l_{ii}^c = 2c_{k,ii}$.

The surrogate function now becomes

$$\begin{aligned} g_2(\boldsymbol{UM}|\boldsymbol{A}_k) = &\frac{1}{2}\mathrm{Tr}\left((\boldsymbol{X} - \boldsymbol{UM})'(\boldsymbol{X} - \boldsymbol{UM})\right) + \frac{1}{2}\lambda\mathrm{Tr}\left(\boldsymbol{M}'\boldsymbol{L}^c\boldsymbol{M}\right) \\ &- \lambda\mathrm{Tr}\left(\boldsymbol{M}'(\boldsymbol{L}^c - \boldsymbol{C}_k)\boldsymbol{M}_k\right) + c, \end{aligned}$$

with c an irrelevant constant. The update becomes

$$\begin{aligned} \frac{\partial \text{ vec } g_2(\boldsymbol{UM}|\boldsymbol{A}_k)}{\partial \text{ (vec } \boldsymbol{M})'} &= \boldsymbol{U}'\boldsymbol{UM} - \boldsymbol{U}'\boldsymbol{X} + \lambda\boldsymbol{L}^c\boldsymbol{M} - \lambda(\boldsymbol{L}^c - \boldsymbol{C}_k)\boldsymbol{M}_k = \boldsymbol{0} \\ \implies \boldsymbol{M}_{k+1} &= (\boldsymbol{U}'\boldsymbol{U} + \lambda\boldsymbol{L}^c)^{-1}(\boldsymbol{U}'\boldsymbol{X} + \lambda(\boldsymbol{L}^c - \boldsymbol{C}_k)\boldsymbol{M}_k). \end{aligned} \tag{26}$$

An advantage of this update, other than that is has no large eigenvalue problems, is that the matrix $\boldsymbol{C}_k$ gets smaller as rows clusters and thus the update can be calculated quicker. The algorithm for the eigenvalue majorization method is the same as for the norm majorization method, but now $\boldsymbol{M}$ gets the update from (26).

For comparing the clustering performance of different clustering methods, we use the Adjusted Rand Index from Hubert and Arabie (1985). This index is an improvement of the Rand index from Rand (1971), as it corrects the index for chance. Let $\mathcal{S} = \{1, 2, \ldots, n\}$ be the set of n observations of data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times d}$. We compare two clusterings if there are k clusters, the correct one $\mathcal{U} = \{U_1, U_2, \ldots, U_k\}$ and the cluster result of a method $\mathcal{C} = \{C_1, C_2, \ldots, C_t\}$ that partitions the data in t clusters.

Let $n_{ij} = |U_i \cap C_j|, 1 \leq i \leq k, 1 \leq j \leq t$ be the amount of elements in cluster U_i and C_j. The nomralized Rand index is defined as

$$\mathrm{R} = \frac{\sum_{i,j}\binom{n_{ij}}{2} - \sum_i\binom{|U_i|}{2}\sum_j\binom{|C_j|}{2}/\binom{n}{2}}{\frac{1}{2}\left[\sum_i\binom{|U_i|}{2} + \sum_j\binom{|C_j|}{2}\right] - \sum_i\binom{|U_i|}{2}\sum_j\binom{|C_j|}{2}/\binom{n}{2}}. \tag{27}$$

A Rand index of 0 means that the clustering $\mathcal{C}$ is not better than a random clustering of the data. A Rand index of 1 means that $\mathcal{C}$ clusters the data correctly which means that $\mathcal{C} = \mathcal{U}$.

3 Data

For comparing the different methods to find the solution path of the convex clustering problem we use generated datasets. This ensures that the methods can be compared for different number of observations n. The main goal of the datasets is to compare the different implementations in run time and performance. We use the following datasets:

Table 1: Datasets used for comparing clustering methods

name	n	d	k	reference
Moons Data	300	2	2	Hocking et al. (2011)
Gaussian Clusters	200	2	4	

n is the number of observations, d the number of features and k the number of clusters in the dataset.

The Moons data is generated so that two half moon-shaped clusters that interlock are present in the data. The Gaussian cluster data is also generated and has 4 Gaussian groups in a 2×2 grid. Both datasets have two dimensions so that the solution path can be plotted and inspected. In appendix B code for generating the datasets can be found.

Normally one takes z-scores of each feature in the dataset, because then each feature is equal in importance when clustering. For our datasets, we do not take z-scores because the datasets are generated in such a way that each feature is equally important in clustering. No transformations are made on the data before applying the clustering methods.

4 Results

For comparing the different methods of clustering and solving the convex clustering problem we tested each method on the generated Moons dataset and compared them in accuracy and runtime. The results are below in table 2 below. For all clustering methods, we used $k = 2$ because k-means requires the amount of cluster to be known in advance and the hierarchy of the other methods had to be cut at some point to get the clustering. The resulting clustering is then used to calculate the adjusted rand index as described in the methodology.

The method of Hocking et al. (2011) is also implemented using their R package, so that it can be compared with our methods. Their method did however not converge in 1000 iterations, which gave an error, so we removed the maximum iteration limit. Furthermore, we also changed the stopping condition for the gradient decent method from Hocking et al. (2011) from $\sum_{i=1}^{k} \|\boldsymbol{g}_i\|_2 < eps$ to $\frac{1}{n}\|\boldsymbol{G}\|_F^2 < eps$, here $\boldsymbol{G}$ is the gradient with rows $\boldsymbol{g}_i$, $1 \leq i \leq k$ and n the number of observations. This way it has the same stopping condition as the norm majorization and the eigenvalue majorization method, so these can be compared in runtime.

Table 2: Runtime and accuracy of clustering methods on Moons data

	Rand	SD	seconds	SD	min	max
gradient descent Hocking	0.80	0.42	23.12	13.50	9.03	46.69
gradient descent	0.80	0.42	107.64	64.78	49.29	226.55
norm majorization	0.80	0.42	17.32	2.53	13.71	21.20
eigenvalue majorization	0.80	0.42	25.88	5.52	17.98	34.17
proximal distance method	-	-	-	-	-	-
k-means	0.24	0.05	0.00	0.00	0.00	0.00
average linkage	0.32	0.08	0.00	0.00	0.00	0.00

The different methods tested on 10 simulations of the Moons data with $n = 300$. Rand gives the average adjusted Rand index of the clustering found and the seconds are the average runtime of the algorithm for finding the l_2 solution path. Min and max give the minimum and maximum value for the runtime of each method. For calculating the weights, $\gamma = 10$ is used and as joining threshold $\delta = 0.5$ is used in (4). The proximal distance method is not shown because it did not converge within reasonable time.

We can draw two conclusions from table 2. First, the solution path cut off at $k = 2$ clusters gives a better clustering than k-means and average linkage, because of a larger Rand index. This is the same as Hocking et al. (2011) found. Second, the gradient descent from Hocking et al. (2011) is slower on average than the norm majorization, but it is sometimes faster. To see if this holds for other datasets we look at the results of applying the methods on the Gaussian clusters data in table 3.

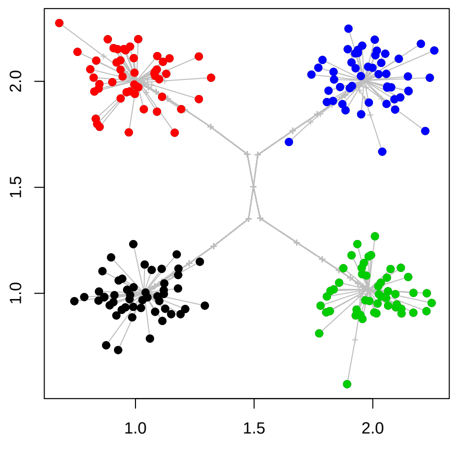
Table 3: Runtime and accuracy of clustering methods on Gaussian Clusters data

	Rand	SD	seconds	SD	min	max
gradient descent Hocking	1.00	0.00	1.13	0.80	0.70	3.28
gradient descent	1.00	0.00	10.63	3.99	7.99	21.32
norm majorization	1.00	0.00	0.88	0.13	0.70	1.09
eigenvalue majorization	1.00	0.00	1.63	0.52	1.14	2.89
proximal distance method	-	-	21.28	4.01	17.75	29.77
k-means	0.92	0.13	0.00	0.00	0.00	0.00
average linkage	0.92	0.13	0.00	0.00	0.00	0.00

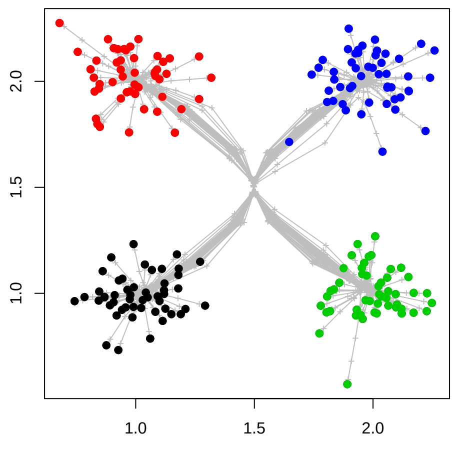
The different methods tested on 10 simulations of the Gaussian clusters data with $n = 200$. Rand gives the average adjusted Rand index of the clustering found and the seconds are the average runtime of the algorithm for finding the l_2 solution path. Min and max give the minimum and maximum value for the runtime of each method. For calculating the weights, $\gamma = 4$ is used and as joining threshold $\delta = 0.5$ is used in (4). The proximal distance method did not cluster points correctly, so its adjusted Rand index is not shown.

When we compare the methods on the Gaussian clusters data, we see that the norm majorization and the gradient descent from Hocking et al. (2011) are both the fastest methods on average. Also, the k-means performs better than on the Moons data and almost identifies the clustering correctly. This is because k-means works well for detecting clusters that are bell-shaped.

The proximal distance method did converge on this data. However, it is the slowest method, and it did not converge in reasonable time on the moons data. This method is not fast enough for finding the optimal solution path and due to its approximative nature its solutions differ from the other methods and a clustering can not be found because the cluster centroids do not get close enough to fuse. This can be seen in figure 2. We conclude that the proximal distance method is not the best method for solving the convex clustering problem.



(a) Norm majorization method.



(b) Proximal distance method.

Figure 2: The l_2 solution paths of the norm majorization method and the proximal distance method run on the same Gaussian Clusters data with $n = 200$, $\gamma = 4$ and the joining threshold δ set to 0.5.

It can be seen from comparing tables 2 and 3 that the norm majorization is faster on average than the eigenvalue majorization method. However, for large n the eigenvalue method should be faster because its update does not involve solving a system of equations. In figure 3 we compare both methods in runtime for finding the solution in different steps of the solution path.

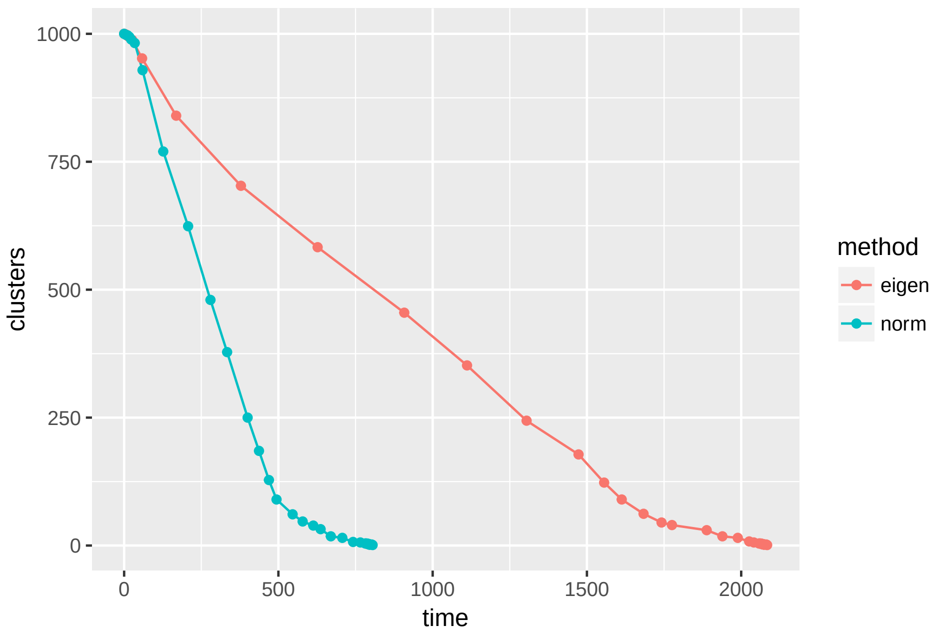


Figure 3: Runtime of the eigenvalue and norm majorization compared to the number of clusters in the l_2 solution path. The run starts with all the observations each in its own cluster and ends with all the observations in one single cluster. As the solver progresses, observations are grouped together in clusters and the number of clusters decreases. The run is done on the Moons data with $n = 1000$, $\gamma = 10$ and $\delta = 0.5$.

Figure 3 shows that for a large problem, eigenvalue majorization is faster than norm majorization in the first six steps of the solution path. After that, the norm majorization is faster. This means that it is quicker to use eigenvalue majorization in the first few steps, until the problem becomes small enough for solving the system of equations.

5 Conclusion

We used majorization techniques and information about the clustering of the data to create methods to solve the convex clustering problem. First, the proximal distance method was applied on the clustering problem with explicit solutions for the projection instead of block descent approaches. Then, the majorization of the norm was used to get an update that involved solving a system of equations. Because this is slow for large problems, another round of majorization was used to get an update that uses upper bounds on eigenvalues instead. These methods were compared in runtime and clustering accuracy on the same problems.

Timing the methods on finding the l_2 solution path on two different datasets reveals that both gradient descent and norm majorization have the shortest runtime on average. Eigenvalue majorization is on average $1.1 - 1.5\times$ slower than gradient descent and the proximal distance method has the largest runtime and is approximately $18\times$ slower than gradient descent.

Gradient descent, norm majorization and eigenvalue majorization did not use variable splitting, which makes the memory required linear in problem size. Also, this makes it possible to use a hard clustering approach which means that once rows are clustered, they remain clustered. Like Hocking et al. (2011), we found that this improves the speed of the algorithm, especially in later iterations once observations have fused to only a few clusters.

It is important to note however, that this hard clustering can result in not finding the global minimum to the convex clustering problem. Hocking et al. (2011) found such an example problem in which using hard clustering did not find the optimal solution, but they also found that problems with decreasing weights do not have cluster splits. Therefore they conjecture that for decreasing weights observations can not split once they have fused, meaning that hard clustering finds the global minimum.

Another limitation of our methods is that they were programmed in R instead of C++ as Hocking et al. (2011) uses. Also using C++ for the majorization methods might speed up the majorization algorithms, as C++ is usually faster than R.

We found two unexpected findings about the proximal distance method. First, it was slow compared to what Chen et al. (2015) found, which is because they used parallelization techniques to increase the speed of the algorithm, and we did not. Second, the clustering results of the proximal distance method were poor, because the cluster centroids did not get close enough to be fused by our fusion method. However, clusters were visible in its solution path, thus the solution path has to be interpreted differently for getting a correct clustering result.

We also investigated for what problem size eigenvalue majorization is quicker than norm majorization. For a problem with 1000 observations eigenvalue majorization was faster than norm majorization in only the first six steps, which suggests that eigenvalue majorization should be used for very large problems, until the problem is small enough for norm majorization.

In summary, our main finding is that the use of majorization can improve the runtime of solving the convex clustering problem compared to gradient descent. This means that norm majorization, our fastest method, could be used to solve the l_2 convex clustering problem in short runtime. This will make the work flow of convex clustering more efficient, and hopefully more popular.

Next steps for building on this research can be to investigate using sparse weights for the convex clustering problem, which can increase the speed of the norm majorization algorithm because then it solves a system of equations with many zeros which is faster. This is in line with what Chen et al. (2015) and Chi and Lange (2015) found for their methods. They suggest to use the nearest k-neighbors for defining the weights, which introduces sparsity.

Other steps include investigations of step doubling in majorization, where the majorization step is doubled each iteration. This is possible because the objective function is convex, so in the worst case the majorization update results in the same objective value. This technique can possible double the convergence speed of the majorization methods, but they have to be examined further.

References

Bartlett, M. S. (1951). An inverse matrix adjustment arising in discriminant analysis. *The Annals of Mathematical Statistics*, *22*(1), 107–111. Retrieved from `http://www.jstor.org/stable/2236707`

Chen, G. K., Chi, E. C., Ranola, J. M. O., & Lange, K. (2015, 05). Convex clustering: An attractive alternative to hierarchical clustering. *PLOS Computational Biology*, *11*(5), 1-31. doi: 10.1371/journal.pcbi.1004228

Chi, E. C., & Lange, K. (2015). Splitting methods for convex clustering. *Journal of Computational and Graphical Statistics*, *24*(4), 994-1013. doi: 10.1080/10618600.2014.948181

Clarke, F. H. (1990). *Optimization and nonsmooth analysis* (Vol. 5). Siam.

Duckworth, D. (2013). *Coordinate ascent for convex clustering.* `http://www.stronglyconvex.com/`. Retrieved 2018-05-07, from `http://www.stronglyconvex.com/blog/coordinate-ascent-convex-clustering.html`

Groenen, P. J. F. (2017). *Feb22006-16: Nonlinear optimization, week 4: Majorization lecture notes.* Retrieved from `https://bb-app02.ict.eur.nl/bbcswebdav/pid-147880-dt-content-rid-435334_1/courses/FEB22006-16/Lecture_4_students_large.pdf` (PowerPoint slides)

Groenen, P. J. F., & Dalmeijer, K. (2016a). *Feb22006-16: Nonlinear optimization, week 3: Derivative-free optimization lecture notes.* Retrieved from `https://bb-app02.ict.eur.nl/bbcswebdav/pid-145555-dt-content-rid-427975_1/courses/FEB22006-16/Lecture_3_students_large.pdf` (PowerPoint slides)

Groenen, P. J. F., & Dalmeijer, K. (2016b). *Statistical methods, week 7: Cluster analysis lecture notes.*

Groenen, P. J. F., Heiser, W. J., & Meulman, J. J. (1999, Jul 01). Global optimization in least-squares multidimensional scaling by distance smoothing. *Journal of Classification*, *16*(2), 225–254. doi: 10.1007/s003579900055

Hartigan, J. A., & Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, *28*(1), 100–108. doi: 10.2307/2346830

Heiser, W. J. (1987). Correspondence analysis with least absolute residuals. *Computational Statistics & Data Analysis*, *5*(4), 337–356. doi: 10.1016/0167-9473(87)90057-0

Hocking, T. D., Joulin, A., Bach, F., & Vert, J.-P. (2011, June). Clusterpath An Algorithm for Clustering using Convex Fusion Penalties. In *28th international conference on machine learning* (p. 1). United States.

Hubert, L., & Arabie, P. (1985, Dec 01). Comparing partitions. *Journal of Classification*, *2*(1), 193–218. doi: 10.1007/BF01908075

Hunter, D. R., & Lange, K. (2004). A tutorial on mm algorithms. *The American Statistician*, *58*(1), 30-37. doi: 10.1198/0003130042836

Lange, K., & Keys, K. L. (2015). The proximal distance algorithm. *arXiv preprint arXiv:1507.07598*.

Petersen, K. B., & Pedersen, M. S. (2012). *The matrix cookbook.* Retrieved from `https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf`

Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, *66*(336), 846-850. doi: 10.1080/01621459.1971.10482356

Richard, B. (1989). *Schaum's outline of theory and problems of matrix operations.* New York: McGraw–Hill.

A Golden section algorithm

This algorithm was taken from Groenen and Dalmeijer (2016a).

Algorithm 7: Golden-Section

```
Input: function $f$, interval $[a, b]$ that contains minimum of $f$
$c \leftarrow b - (b - a)/\phi$;
$d \leftarrow a + (b - a)/\phi$;
$f_c \leftarrow f(c)$;
$f_d \leftarrow f(d)$;
while $|c - d| > eps$ do
    if $f_c < f_d$ then
        $b \leftarrow d$;
        $d \leftarrow c$;
        $c \leftarrow b - (b - a)/\phi$;
        $f_d \leftarrow f_c$;
        $f_c \leftarrow f(c)$;
    else
        $a \leftarrow c$;
        $c \leftarrow d$;
        $d \leftarrow a + (b - a)/\phi$;
        $f_c \leftarrow f_d$;
        $f_d \leftarrow f(d)$;
    end
end
return $(b - a)/2$
```

B Data generation

Below is the R code used for generating the Gaussian clusters data and the Moons data.

```r
# Generate Gaussian clusters
gaussian.data <- function(n=200) {
  k <- 4
  X <- NULL
  for (q in 1:2) {
    for (j in 1:2) {
      mu <- c(q, j)
      var <- rbind(c(0.015,0), c(0,0.015))
      X <- rbind(X, cbind(mvtnorm::rmvnorm(n/4, mean = mu,
              sigma = var), group=(q-1)*2+j))
    }
  }
  return(X)
}
```

```r
# From: https://rdrr.io/rforge/clusterpathRcpp/src/R/sim.R
# Edited by: Tom van den Berg
### simulate some data in the shape of 2 half-moons
halfmoon.cluster.data <- function(N=150,noise=0.5) {
  halfcircle <- function(r,center=c(0,0),class,sign){
    angle <- runif(N,0,pi)
    rad <- rnorm(N,r,noise)
    return(cbind(rad*cos(angle)+center[1],
            sign*rad*sin(angle)+center[2], class))
  }
  pts <- rbind(halfcircle(4,c(0,0),1,1),
               halfcircle(4,c(4,2),2,-1))
  return(pts)
}
```

C R code

Below is the R code used for solving the convex clustering problem for given λ using the gradient descent method.

```r
solve_gradient<- function(U, M, X, W, lambda, threshold) {
  iteration <- 1
  G <- subgradient_L2(U, M, X, W, lambda)
  f_k <- cost_function(U%*%M, X, lambda, W)
  n <- nrow(X)

  while (sum(G^2)/n > 0.001 && iteration < 1000) {

    if (iteration %% 2 == 0) {
      r <- 1/iteration
    } else {
      r <- golden_section(function(x) cost_function(U%*%(M-x*G), X,
            lambda, W), 0, 1)
    }

    # Subgradient step
    M <- M - r*G

    # Check for fusions
    o <- find_cluster_fusion(U, M, threshold)
    if (o$fusion) {
      U <- o$U
      M <- o$M
      iteration <- 0
    }
    # Subgradient
    G <- subgradient_L2(U, M, X, W, lambda)
    iteration <- iteration + 1
  }
  return(list("U" = U, "M" = M))
}
golden_section <- function(f, a, b, tol=0.001) {
  gr <- (1+sqrt(5))/2
  c <- b-(b-a)/gr
  d <- a+(b-a)/gr
  f_c <- f(c)
  f_d <- f(d)
  while(abs(c-d)>tol) {
    if (f_c < f_d) {
      b <- d
      d <- c
      f_d <- f_c
      c <- b-(b-a)/gr
      f_c <- f(c)
    } else {
      a <- c
      c <- d
      f_c <- f_d
      d <- a+(b-a)/gr
      f_d <- f(d)
    }
  }
  return((b+a)/2)
}
subgradient_L2 <- function(U, M, X, W, lambda) {
  a <- U%*%M
  a_c <- M
  X_c <- crossprod(U, X)/colSums(U)
  W_c <- crossprod(U, W) * t(1-U)
  s <- M
  for (i in 1:nrow(M)) {
    diff <- -sweep(a, 2, M[i,])
    sq <- sqrt(rowSums(diff^2))
    sq[sq==0] <- 1 # will be multiplied out by W_c
    sum_1 <- W_c[i,] %*% (diff / sq)
    s[i,] <- sum_1
  }
  G <- a_c - X_c + lambda / colSums(U) * s
  return(G)
}
```

Below is the R code used for solving the convex clustering problem for given λ using the proximal distance method.

```r
solve_proximal <- function(U, M, X, W, lambda, threshold) {
  # Init
  A <- U%*%M
  n <- nrow(A)
  d <- ncol(A)
  V <- array(0, dim=c(n, n, d))
  V[] <- apply(A, 2, function(x) {tt <- matrix(rep(x, n),
          nrow=n); t(tt)-tt}) # Difference array
  Vp <- V
  k <- 0
  f_k <- 0
  f_k_1 <- 1
  max_iterations <- 1000
  rho <- 1

  while (k < 1 || abs(f_k_1 - f_k)/abs(f_k_1) > 0.001
                  && k < max_iterations) {
    # Find projection
    Ap <- find_projection(A, V)
    Vp[] <- apply(Ap, 2, function(x) {tt <- matrix(rep(x, n),
            nrow=n); t(tt)-tt}) # Difference array

    ## Update eps and rho
    eps <- max(1e-3*1.5^(-k), 1e-6)
    rho <- min(1.5^(k), 1e3)
    mapdist <- sum((A-Ap)^2) + sum((V-Vp)^2)
    d_m <- sqrt(mapdist + eps)

    # Update M and V
    M <- (d_m/(d_m+rho)*crossprod(U,X)+rho/(d_m+rho)*
            crossprod(U,Ap))/colSums(U)

    Vp_norm <- as.matrix(dist(Ap))
    Vp_norm <- pmax(Vp_norm, 1e-8)
    S <- pmax(1-lambda*W*d_m/(rho*Vp_norm), 0)
    V[] <- apply(Vp, 3, function(R) S*R)

    # Check for fusion
    o <- find_cluster_fusion(U, M, threshold)
    if (o$fusion) {
      U <- o$U
      M <- o$M
    }

    A <- U%*%M
    f_k_1 <- f_k
    mapdist <- sum((A-Ap)^2) + sum((V-Vp)^2)
    f_k <- 0.5*sum((A-X)^2) + lambda*sum(W[lower.tri(W)]*
            S[lower.tri(S)]*dist(Ap)) + 0.5*rho*mapdist/d_m
    k <- k+1
  }

  return(list("U" = U, "M" = M))
}
find_projection <- function(A, V) {
  n <- nrow(A)
  d <- ncol(A)
  V2 <- t(apply(V, 2, colSums)-apply(V, 1, colSums))
  solve_V1 <- matrix(2/(2*n+1), n, n)
  diag(solve_V1) <- 3/(2*n+1)
  Ap <- crossprod(solve_V1, A+V2)
  return(Ap)
}
```

Below is the R code used for solving the convex clustering problem for given λ using the norm majorization method.

```
solve_norm <- function(U, M, X, W, lambda, threshold) {
  n <- nrow(X)
  eps <- 1e-8

  # Init
  k <- 1
  g <- 0

  while (k == 1 || g > 0.001) {
    k <- k+1

    ### Find minimum of majorization function
    # Caculate V
    D <- as.matrix(dist(U%*%M))
    D <- pmax(D, eps)
    V <- -W/D
    diag(V) <- -rowSums(V)

    # Solve system
    UX <- crossprod(U, X)
    UVU <- crossprod(U, crossprod(V, U))
    G <- crossprod(U,U%*%M)-UX+lambda*crossprod(UVU,M)
    M <- solve(crossprod(U)+lambda*UVU, UX)

    # Check for fusions
    repeat {
      o <- find_cluster_fusion(U, M, threshold)
      fusion <- o$fusion
      if (!o$fusion)
```

```
        break
      U <- o$U
      M <- o$M
    }

    g <- sum(G^2)/n
  }

  return(list("U" = U, "M" = M))
}
```

Below is the R code used for solving the convex clustering problem for given λ using the eigenvalue majorization method.

```
solve_eigen <- function(U, M, X, W, lambda, threshold) {
  n <- nrow(X)
  eps <- 1e-8

  # Init
  k <- 1
  g <- 0

  while (k == 1 || g > 0.001) {
    k <- k+1

    ### Find minimum of majorization function
    # Caculate V
    D <- as.matrix(dist(U%*%M))
    D <- pmax(D, eps)
    V <- -W/D
    diag(V) <- -rowSums(V)

    # Upper bound on eigen values of C
    C <- crossprod(U, V%*%U)
    l <- 2*(diag(C)+eps)
    diag(C) <- diag(C)-l
    C <- -C # Result is L-C

    # Gradient and update
    UX <- crossprod(U, X)
    L_CM <- C%*%M
    G <- crossprod(U,U%*%M)-UX+lambda*l*M-lambda*L_CM
    M <- (UX + lambda*L_CM)/(lambda*l+colSums(U))

    # Check for fusions
    repeat {
      o <- find_cluster_fusion(U, M, threshold)
      fusion <- o$fusion
```

```r
      if (!o$fusion)
        break
      U <- o$U
      M <- o$M
    }

    g <- sum(G^2)/n
  }

  return(list("U" = U, "M" = M))
}
```

Below the R code for cluster fusions and the cost function.

```r
find_cluster_fusion <- function(U, M, threshold) {

  if (ncol(U) == 1) {
    return(list("U" = U, "M" = M, "fusion" = FALSE))
  }

  diff <- as.matrix(dist(M))
  diff_min <- min(diff[diff>0])

  if (diff_min > threshold) {
    return(list("U" = U, "M" = M, "fusion" = FALSE))
  } else {
    idx <- which(diff == diff_min, arr.ind=TRUE)
    i <- idx[2, 1]
    j <- idx[2, 2]
    # it follows that i < j

    nC_i <- sum(U[,i])
    nC_j <- sum(U[,j])
    a_c <- (nC_i*M[i,]+nC_j*M[j,]) / (nC_i+nC_j)

    M[i, ] <- a_c
    M <- M[-j,,drop=F]
    U[,i] <- U[,i]+U[,j]
    U <- U[,-j,drop=F]
  }

  return(list("U" = U, "M" = M, "fusion" = TRUE))
}
cost_function <- function(A, X, lambda, W) {
  return(0.5*sum((A-X)^2)+lambda*sum(W[lower.tri(W)]*dist(A)))
}
```