

Universitat Politècnica de Catalunya  
Facultat de Matemàtiques i Estadística

Master thesis

# Multidimensional Scaling for Big Data

Cristian Pachón García

Advisor: Pedro Delicado Useros

Dept. d'Estadística i Investigació Operativa

# Contents

<b>1</b>	<b>Classical Multidimensional Scaling</b>	<b>2</b>
1.1	Introduction to Multidimensional Scaling . . . . .	2
1.2	Principal coordinates . . . . .	3
1.3	Building principal coordinates . . . . .	5
1.4	Procrustes transformation . . . . .	6
1.5	Multidimensional Scaling with R . . . . .	7
<b>2</b>	<b>Algorithms for Multidimensional Scaling with Big Data</b>	<b>8</b>
2.1	Why is it needed? . . . . .	8
2.2	Divde and Conquer Multidimensional Scaling . . . . .	9
2.2.1	Algorithm . . . . .	9
2.2.2	Some indicators about the goodness of fit . . . . .	10
2.3	Fast Multidimensional Scaling . . . . .	11
2.3.1	Algorithm . . . . .	11
2.3.2	Some indicators about the goodness of fit . . . . .	12
2.4	Multidimensional Scaling based on Gower interpolation . . . . .	12
2.4.1	Algorithm . . . . .	13
2.4.2	Some indicators about the goodness of fit . . . . .	13
2.5	Comparison of the algorithms . . . . .	14
2.6	Output of the algorithms . . . . .	15
<b>3</b>	<b>Simulation study</b>	<b>16</b>
<b>4</b>	<b>Conclusions</b>	<b>20</b>
	<b>Bibliography</b>	<b>21</b>
<b>A</b>	<b>Code</b>	<b>22</b>
<b>B</b>	<b>Problems encountered during the development</b>	<b>23</b>

# Chapter 1

## Classical Multidimensional Scaling

### 1.1 Introduction to Multidimensional Scaling

Multidimensional Scaling (MDS) is a family of methods that represents measurements of dissimilarity (or similarity) among pairs of objects as Euclidean distances between points of a low-dimensional space. The data, for example, may be correlations among intelligence tests and the MDS representation is a plane that shows the tests as points. The graphical display of the correlations provided by MDS enables the data analyst to literally “look” at the data and to explore their structure visually. This often shows regularities that remain hidden when studying arrays of numbers.

Given a square matrix  $\mathbf{D}$   $n \times n$ , the goal of MDS is to obtain a configuration matrix  $\mathbf{X}$   $n \times p$  with orthogonal columns that can be interpreted as the matrix of  $p$  variables for the  $n$  observations, where the Euclidean distance between the rows of  $\mathbf{X}$  is approximately equal to  $\mathbf{D}$ . The columns of  $\mathbf{X}$  are called *principal coordinates*.

This approach arises two questions: is it (always) possible to find this low dimensional configuration  $\mathbf{X}$ ? How is it obtained? In general, it is not possible to find a set of  $p$  variables that reproduces *exactly* the initial distance. However, it is possible to find a set of variables which distance is approximately the initial distance matrix  $\mathbf{D}$ .

As classical example, consider the distances between European cities as in the Table (1.1). One would like to get a representation in a 2-dimensional space such that the distances would be almost the same as in the Table (1.1). The representation of these coordinates are displayed in Figure (1.1).

	Athens	Barcelona	Brussels	Calais	Cherbourg	...
Athens	0	3313	2963	3175	3339	...
Barcelona	3313	0	1318	1326	1294	...
Brussels	2963	1318	0	204	583	...
Calais	3175	1326	204	0	460	...
Cherbourg	3339	1294	583	460	0	...
:	:	:	:	:	:	...

Table 1.1: Distances between European cities (just 5 of them are shown).

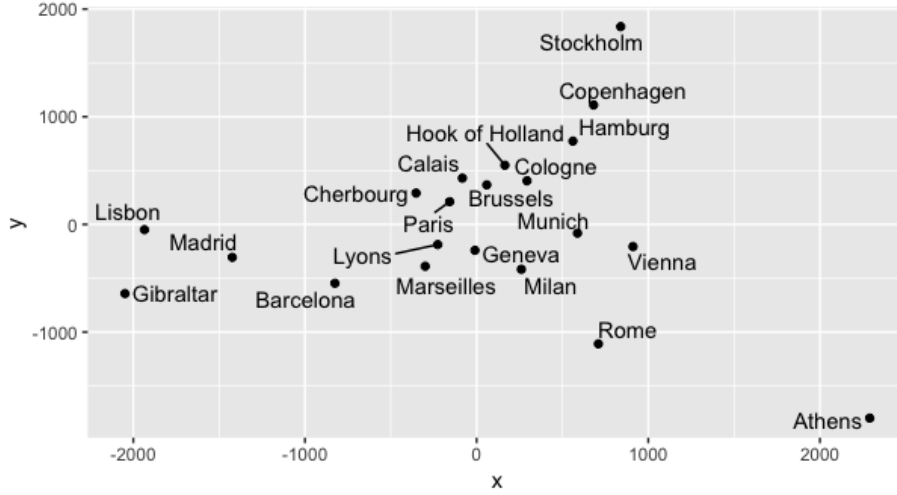


Figure 1.1: MDS on the European cities.

## 1.2 Principal coordinates

Given a matrix  $\mathbf{X}$   $n \times p$ , the matrix of  $n$  individuals over  $p$  variables, it is possible to obtain a new one with mean equal to 0 by column from the previous one:

$$\tilde{\mathbf{X}} = \left( \mathbf{I} - \frac{1}{n} \mathbf{1}\mathbf{1}' \right) \mathbf{X} = \mathbf{P}\mathbf{X},$$

where

$$\mathbf{P} = \left( \mathbf{I} - \frac{1}{n} \mathbf{1}\mathbf{1}' \right).$$

This new matrix,  $\tilde{\mathbf{X}}$ , has the same dimensions as the original one but its columns mean is  $\mathbf{0}$ . From this matrix, it is possible to build two square semi-positive definite matrices: the covariance matrix  $\mathbf{S}$ , defined as  $\tilde{\mathbf{X}}'\tilde{\mathbf{X}}/n$  and the cross-products matrix  $\mathbf{Q} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}'$ . The last matrix can be interpreted as a similarity matrix between the  $n$  elements. The term  $ij$  is obtained as follows:

$$q_{ij} = \sum_{s=1}^p x_{is}x_{js} = \mathbf{x}'_i\mathbf{x}_j \quad (1.1)$$

where  $\mathbf{x}'_i$  is the  $i$ -th row from  $\tilde{\mathbf{X}}$ . Given the scalar product formula,  $\mathbf{x}'_i\mathbf{x}_j = |\mathbf{x}_i| |\mathbf{x}_j| \cos \theta_{ij}$ , if the elements  $i$  and  $j$  have similar coordinates, then  $\cos \theta_{ij} \simeq 1$  and  $q_{ij}$  will be large. On the contrary, if the elements are very different, then  $\cos \theta_{ij} \simeq 0$  and  $q_{ij}$  will be small. So,  $\tilde{\mathbf{X}}\tilde{\mathbf{X}}'$  can be interpreted as the similarity matrix between the elements.

The distances between elements can be deduced from the similarity matrix. The Euclidean distance between two elements is calculated in the following way:

$$d_{ij}^2 = \sum_{s=1}^p (x_{is} - x_{js})^2 = \sum_{s=1}^p x_{is}^2 + \sum_{s=1}^p x_{js}^2 - 2 \sum_{s=1}^p x_{is}x_{js}. \quad (1.2)$$

This expression can be obtained directly from the matrix  $\mathbf{Q}$ :

$$d_{ij}^2 = q_{ii} + q_{jj} - 2q_{ij}. \quad (1.3)$$

We have just seen that, given the matrix  $\tilde{\mathbf{X}}$ , it is possible to get the similarity matrix  $\mathbf{Q} = \tilde{\mathbf{X}}\tilde{\mathbf{X}}'$  and from it, to get the distance matrix  $\mathbf{D}$ . Let  $\text{diag}(\mathbf{Q})$  be the vector that contains the diagonal terms of  $\mathbf{Q}$  and  $\mathbf{1}$  be the vector of ones, the matrix  $\mathbf{D}$  is given by

$$\mathbf{D} = \text{diag}(\mathbf{Q})\mathbf{1}' + \mathbf{1}\text{diag}(\mathbf{Q})' - 2\mathbf{Q}.$$

The problem we are dealing with goes in the opposite direction. We want to rebuild  $\tilde{\mathbf{X}}$  from a square distance matrix  $\mathbf{D}$ , with elements  $d_{ij}^2$ . The first step is to obtain  $\mathbf{Q}$  and afterwards, to get  $\tilde{\mathbf{X}}$ . The theory needed to get the solution can be found in (Peña 2002). Here, we summarise it.

The first step is to find out a way to obtain the matrix  $\mathbf{Q}$  given  $\mathbf{D}$ . We can assume without loss of generality that the mean of the variables is equal to 0. This is a consequence of the fact that the distance between two points remains the same if the variables are expressed in terms of the mean:

$$d_{ij}^2 = \sum_{s=1}^p (x_{is} - x_{js})^2 = \sum_{s=1}^p [(x_{is} - \bar{x}_s) - (x_{js} - \bar{x}_s)]^2. \quad (1.4)$$

The previous condition means that we are looking for a matrix  $\tilde{\mathbf{X}}$  such that  $\tilde{\mathbf{X}}'\mathbf{1} = 0$ . It also means that  $\mathbf{Q}\mathbf{1} = 0$ , i.e, the sum of all the elements of a column of  $\mathbf{Q}$  is 0. Since the matrix is symmetric, the previous condition should state for the rows as well.

To establish these constraints, we sum up (1.3) at row level:

$$\sum_{i=1}^n d_{ij}^2 = \sum_{i=1}^n q_{ii} + nq_{jj} = t + nq_{jj} \quad (1.5)$$

where  $t = \sum_{i=1}^n q_{ii} = \text{Trace}(\mathbf{Q})$ , and we have used that the condition  $\mathbf{Q}\mathbf{1} = 0$  implies  $\sum_{i=1}^n q_{ij} = 0$ . Summing up (1.3) at column level:

$$\sum_{j=1}^n d_{ij}^2 = t + nq_{ii}. \quad (1.6)$$

Summing up (1.5) we obtain:

$$\sum_{i=1}^n \sum_{j=1}^n d_{ij}^2 = 2nt \quad (1.7)$$

Replacing in (1.3)  $q_{jj}$  obtained in (1.5) and  $q_{ii}$  obtained in (1.6), we have the following expression:

$$d_{ij}^2 = \frac{1}{n} \sum_{i=1}^n d_{ij}^2 - \frac{t}{n} + \frac{1}{n} \sum_{j=1}^n d_{ij}^2 - \frac{t}{n} - 2q_{ij} \quad (1.8)$$

Let  $d_{i.}^2 = \frac{1}{n} \sum_{j=1}^n d_{ij}^2$  and  $d_{.j}^2 = \frac{1}{n} \sum_{i=1}^n d_{ij}^2$  be the row-mean and column-mean of the elements of  $\mathbf{D}$ . Using (1.7), we have that

$$d_{ij}^2 = d_{i.}^2 + d_{.j}^2 - d_{..}^2 - 2q_{ij} \quad (1.9)$$

where  $d_{..}$  is the mean of all the elements of  $\mathbf{D}$ , given by

$$d_{..}^2 = \frac{1}{n^2} \sum \sum d_{ij}^2.$$

Finally, from (1.9) we get the expression:

$$q_{ij} = -\frac{1}{2}(d_{ij}^2 - d_{i.}^2 - d_{.j}^2 + d_{..}^2). \quad (1.10)$$

The previous expression shows how to build the matrix of similarities  $\mathbf{Q}$  from the distance matrix  $\mathbf{D}$ .

The next step is to obtain the matrix  $\mathbf{X}$  given the matrix  $\mathbf{Q}$ . Let's assume that the similarity matrix is positive definite of range  $p$ . Therefore, it can be represented as

$$\mathbf{Q} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}'$$

where  $\mathbf{V}$  is a  $n \times p$  matrix that contains the eigenvectors with not nulls eigenvalues of  $\mathbf{Q}$ .  $\mathbf{\Lambda}$  is a diagonal matrix  $p \times p$  that contains the eigenvalues.

Re-writing the previous expression, we obtain

$$\mathbf{Q} = (\mathbf{V}\mathbf{\Lambda}^{1/2})(\mathbf{\Lambda}^{1/2}\mathbf{V}'). \quad (1.11)$$

Getting

$$\mathbf{Y} = \mathbf{V}\mathbf{\Lambda}^{1/2}.$$

We have obtained a matrix with dimensions  $n \times p$  with  $p$  uncorrelated variables that reproduce the initial metric. It is important to notice that if one starts from  $\mathbf{X}$  (i.e  $\mathbf{X}$  is known) and calculates from these variables the distance matrix in (1.2) and after that it is applied the method explained, the matrix obtained is not the same as  $\mathbf{X}$ , but its principal components. This happens since the distance between the rows in a matrix does not change if:


- The row-mean values are modified by adding the same row vector to all the rows in  $\mathbf{X}$ .
- Rows are rotated, i.e,  $\mathbf{X}$  is postmultiplied by an orthogonal matrix.

By (1.3), the distance is a function of the terms of the similarity matrix  $\mathbf{Q}$  and this matrix is invariant given any rotation, reflexion or translation of the variables

$$\mathbf{Q} = \widetilde{\mathbf{X}}\widetilde{\mathbf{X}}' = \widetilde{\mathbf{X}}\mathbf{A}\mathbf{A}'\widetilde{\mathbf{X}}'$$

for any orthogonal  $\mathbf{A}$  matrix. The matrix  $\mathbf{Q}$  only contains information about the space generated by the variables  $\mathbf{X}$ . Any rotation, reflexion or translation keeps the distance unchanged. Therefore, the solution is not unique.

### 1.3 Building principal coordinates

Let  $\mathbf{D}$  be a square distance matrix. The process to obtain the *principal coordinates* is: 

1. Build the matrix  $\mathbf{Q} = -\frac{1}{2}\mathbf{P}\mathbf{D}\mathbf{P}$  of cross-products.
2. Obtain the eigenvalues of  $\mathbf{Q}$ . Take the  $r$  greatest eigenvalues. Since  $\mathbf{P}\mathbf{1} = 0$ , where  $\mathbf{1}$  is a vector of ones,  $\text{range}(\mathbf{Q}) = n - 1$ , being the vector  $\mathbf{1}$  an eigenvector with eigenvalue 0.

3. Obtain the coordinates of the rows in the variables  $\mathbf{v}_i\sqrt{\lambda_i}$ , where  $\lambda_i$  is an eigenvalue of  $\mathbf{Q}$  and  $\mathbf{v}_i$  is the associated unitary eigenvector. This implies that  $\mathbf{Q}$  is approximated by:

$$\mathbf{Q} \approx (\mathbf{V}_r \mathbf{\Lambda}^{1/2})(\mathbf{\Lambda}_r^{1/2} \mathbf{V}_r').$$

4. Take as coordinates of the points the following variables:

$$\mathbf{Y}_r = \mathbf{V}_r \mathbf{\Lambda}_r^{1/2}.$$

The method can also be applied if the initial information is not a distance matrix but a similarity matrix. A *similarity function*,  $s_{ij}$ , between two element  $i$  and  $j$  is defined as:

- $s_{ii} = 1$ .
- $0 \leq s_{ij} \leq 1$ .
- $s_{ij} = s_{ji}$ .

If the initial information is  $\mathbf{Q}$ , a similarity matrix, then  $q_{ii} = 1$ ,  $q_{ij} = q_{ji}$  and  $0 \leq q_{ij} \leq 1$ . The associated distance matrix is

$$d_{ij}^2 = q_{ii} + q_{jj} - 2q_{qij} = 2(1 - q_{ij}),$$

and it is easy to see that  $\sqrt{2(1 - q_{ij})}$  is a distance and it verifies the triangle inequality.

## 1.4 Procrustes transformation

As we have mentioned before, the MDS solution is not unique. One example of it is shown in Figure (1.2).

Since rotations, translations and reflections are distance-preserving functions, one can find two different MDS configurations for the same set of data. How is it possible to align both solutions? This problem is solved by means of *Procrustes transformations*.

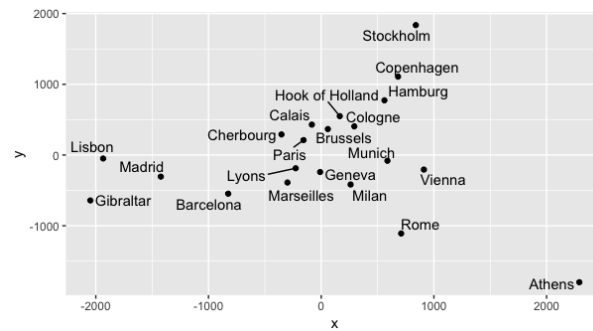
The Procrustes problem is concern with fitting a configuration (testee) to another (target) as closely as possible. In the simple case, both configurations have the same dimensionality and the same number of points, which can be brought into 1-1 correspondence by substantive considerations. Under orthogonal transformations, the testee can be fitted it to the target. In addition to such rigid motions, one may also allow for dilations and for shifts.

All the details are developed in (Borg and Groenen 2005). This is out of the scope of this thesis. However, since it has been a repeatedly used tool, we briefly summarise it.

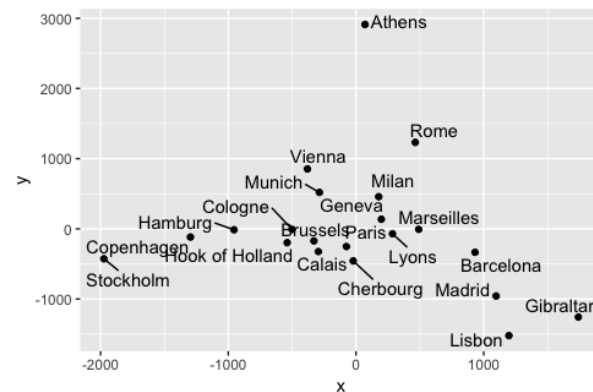
Let  $\mathbf{A}$  and  $\mathbf{B}$  be two different MDS configurations for the same set of data. Without loss of generality, let's assume that the target is  $\mathbf{A}$  and the testee is  $\mathbf{B}$ . One wants to obtain  $s$ ,  $\mathbf{T}$  and  $\mathbf{t}$  such that

$$\mathbf{A} = s\mathbf{B}\mathbf{T} + \mathbf{1}\mathbf{t}'$$

where  $\mathbf{T}$  is an orthogonal matrix as mentioned before, in (Borg and Groenen 2005) are all the details needed to estimate these parameters.



(a)



(b)

Figure 1.2: Two different solutions of MDS.

## 1.5 Multidimensional Scaling with R

All the algorithms have been coded in R, since it has a widely statistics packages already implemented. We have used two packages for developing our MDS approaches:

- Package: `stats`. From this one we have used the function `cmdscale` to do the MDS. The output of this function is:
  - The `new` coordinates for the individuals.
  - All the eigenvalues found.
- Package: `MCMCpack`. From this one we have used the function `procrustes` to do the Procrustes transformation. The output of this function is:
  - The dilation coefficient  $s$ .
  - The orthogonal matrix  $\mathbf{T}$ .
  - The translation vector  $\mathbf{t}$ .



## Chapter 2

# Algorithms for Multidimensional Scaling with Big Data

### 2.1 Why is it needed?

In this chapter we present the algorithms developed so that MDS can be applied when we are dealing with large datasets. The natural question here is why we need them if there are already some implementations. To answer this question, let's take a look at the Figures (2.1) and (2.2).

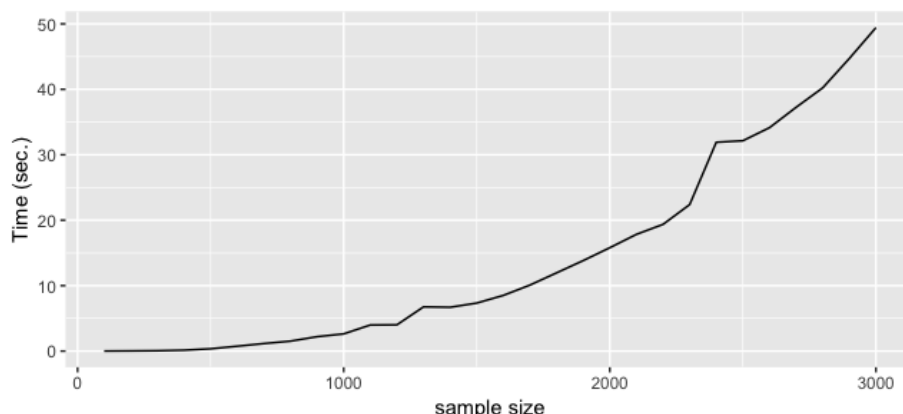


Figure 2.1: Elapsed time to compute MDS.

Figure (2.1) shows the time needed to compute MDS as a function of the sample size. As we can see, the time grows exponentially. Apart of the time issue, there is another one related to the memory needed to compute the distance matrix. Figure (2.2) points out that it is required at least 400MB to store the distance matrix when the dataset is close to 10000 observations.

In order to solve these problems, we have developed three algorithms:

- *Divide and Conquer MDS*: Before this thesis, *Pedro Delicado* had done some work about MDS with big datasets and he had already created a first approach, which is this one. The algorithm is based on the idea of dividing and conquering. Given a big dataset, it is divided into  $p$  partitions. After that, MDS is performed over all the partitions and, finally, the  $p$  solutions are stitched so that all the points lie on the same coordinate system.

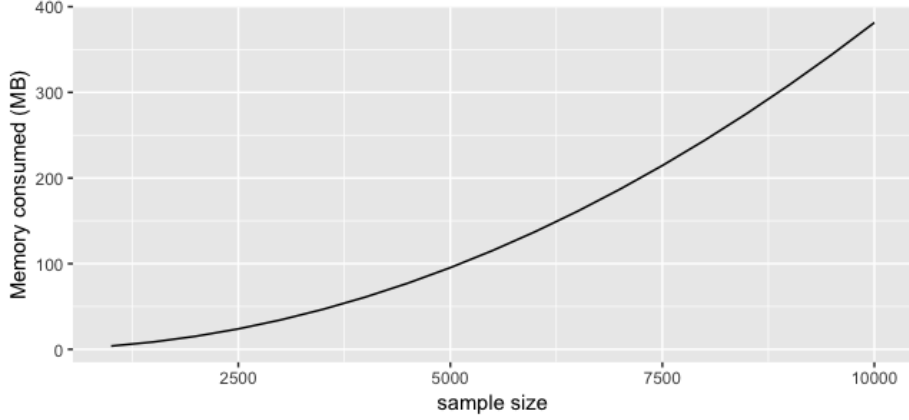


Figure 2.2: Memory consumed to compute distance.

- *Fast MDS*: during the phase of gathering information, we found an article that solved the problem of scalability (Tynia, Jinze, Leonard, and Wei 2006). The authors use a divide and conquer idea together with recursive programming.
- *MDS based on Gower interpolation formula*: this algorithm uses Gower interpolation formula which allows to add a new set of points to an existing MDS configuration (Handl 1996).

In the next sections we provide a description of the algorithms. If further details about the implementation are needed, the code is provided in Appendix A.

## 2.2 Divide and Conquer Multidimensional Scaling

### 2.2.1 Algorithm

- The first step is to divide the original dataset into  $p$  partitions:  $\mathbf{X}_1, \dots, \mathbf{X}_p$ . The number of partitions,  $p$ , is also the number of steps needed to compute the algorithm.
- Calculate the MDS for the first partition:  $\mathbf{MDS}(1)$ . This solution will be used as a guide to align the MDS for the remaining partitions. We use a new variable, **cum-mds**, that will be growing as long as new partitions are used. Before adding a new MDS, it is aligned and, after that, added.
- Define **cum-mds** equal to  $\mathbf{MDS}(1)$  and start iterating until the last partition is reached.
- Given a step  $k$ ,  $1 < k \leq p$ , partitions  $k$  and  $k-1$  are joint, i.e.,  $\mathbf{X}_k \cup \mathbf{X}_{k-1}$ . MDS is calculated on this union, obtaining  $\mathbf{MDS}_{k,k-1}$ . In order to add the rows of the  $k$ -th partition to **cum-mds**, the following steps are performed:
  - Take the rows of the partition  $k-1$  from  $\mathbf{MDS}_{k,k-1}$ :  $\mathbf{MDS}_{k,k-1} \Big|_{k-1}$ .
  - Take the rows of the partition  $k-1$  from **cum-mds**:  $\mathbf{cum-mds} \Big|_{k-1}$ .

- Apply Procrustes to align both solutions. It means that a scalar number  $s$ , a vector  $\mathbf{t}$  and an orthogonal matrix  $\mathbf{T}$  are obtained so that


$$\mathbf{cum-mds} \Big|_{k-1} \stackrel{!}{=} s \mathbf{MDS}_{\mathbf{k}, \mathbf{k}-1} \Big|_{k-1} \mathbf{T} + \mathbf{1t}'.$$

- Take the rows of the partition  $k$  from  $\mathbf{MDS}_{\mathbf{k}, \mathbf{k}-1} : \mathbf{MDS}_{\mathbf{k}, \mathbf{k}-1} \Big|_k$ .
- Use the previous Procrustes parameters to add the rows of  $\mathbf{MDS}_{\mathbf{k}, \mathbf{k}-1} \Big|_k$  to **cum-mds**:


$$\mathbf{cum-mds}_k := s \mathbf{MDS}_{\mathbf{k}, \mathbf{k}-1} \Big|_k \mathbf{T} + \mathbf{1t}'.$$

- Add the previous dataset to **cum-mds**, i.e:

$$\mathbf{cum-mds} = \mathbf{cum-mds} \cup \mathbf{cum-mds}_k$$

As we have seen, the algorithm depends on  $p$ , the number of partitions. How many of them are needed? To answer this question, let  $l \times l$  be the largest matrix that allows to run MDS **efficiently**. If  $n$  is the number of rows of  $\mathbf{X}$ , then  $p$  is  $\frac{2n}{l}$ . 

### 2.2.2 Some indicators about the **goodness of fit**

 This section does not pretend to prove that the algorithm is **working**, this is done in chapter 3, where deeper analysis is done. What we want to do here is to show some visual results so that the reader can have some indicators about the **goodness of fit** of the algorithm.

We have generated a matrix  $\mathbf{X}$  with 3 independent *Normal* distributions ( $\mu = 0$  and  $\sigma = 1$ ) and 1000 rows. Afterwards, we have run the algorithm. We have required **the algorithm** to return 3 columns. So, a new matrix with 3 columns and 1000 rows ( $\mathbf{MDS}_{\mathbf{Div}}$ ) has been obtained. Both matrices should be “equal” with an exception of either a rotation, translation or reflection, but not a dilation. We have not allowed dilations to see that the distance is **conserved**.

To align the matrices we have performed a Procrustes transformation, but setting the the dilation parameter ( $s$ ) equal to 1. After that, we have compared the three columns (we refer to the columns as *dimensions*). Figure (2.3) shows the dimension  $i$  of  $\mathbf{X}$  against the dimension  $i$  of  $\mathbf{MDS}_{\mathbf{Div}}$ ,  $i \in \{1, 2, 3\}$ .

As we can see, the algorithm is able to capture the dimensions of the original matrix. We do not show cross-dimensions (i.e dimension  $i$  of  $\mathbf{X}$  against dimension  $j$  of  $\mathbf{MDS}_{\mathbf{Div}}$   $i \neq j$ ), but Table (2.1) contains the **correlation matrix**. The results show that dimension  $i$  of  $\mathbf{MDS}_{\mathbf{Div}}$  captures dimension  $i$  of  $\mathbf{X}$  and just dimension  $i$ . So, it seems that the algorithm, for this particular case, has a **high goodness of fit**.

	1	2	3
1	1	0.02	-0.04
2	0.02	1	0.02
3	-0.04	0.02	1

Table 2.1: **Correlation** of  $\mathbf{X}$  and  $\mathbf{MDS}_{\mathbf{Div}}$ .

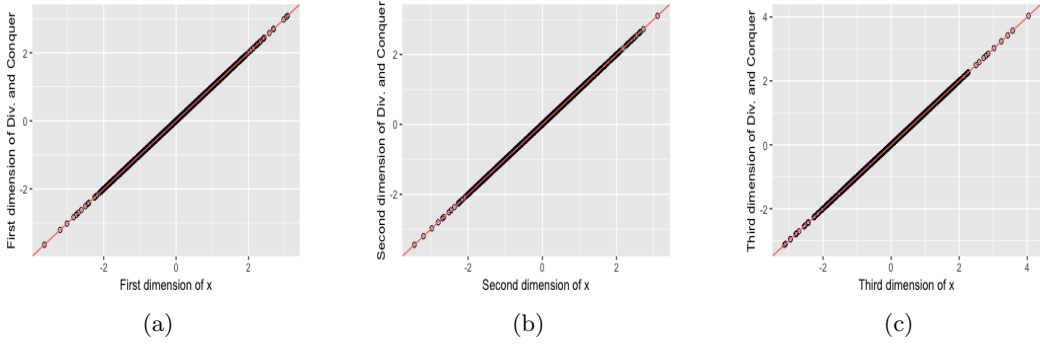




Figure 2.3: Dimesion 1,2 and 3 of  $\mathbf{X}$  against dimensions 1,2 and 3 of  $\mathbf{MDS}_{\text{Div}}$ . In red, the line  $x = y$ .

## 2.3 Fast Multidimensional Scaling

During the process of gathering information about work previously done around MDS with big datasets, we found that Tynia, Jinze, Leonard, and Wei (2006) already **did** an algorithm, they named it *Fast Multidimensional Scaling*.

### 2.3.1 Algorithm

- Divide  $\mathbf{X}$  into  $\mathbf{X}_1, \dots, \mathbf{X}_p$ .
- Compute MDS for each  $\mathbf{X}_i$ :  $\mathbf{MDS}_1, \dots, \mathbf{MDS}_p$   these individuals MDS solutions are stitched together by sampling  $s$  points (rows) from each submatrix  $\mathbf{X}_i$  and putting them into an alignment matrix  $\mathbf{M}_{\text{align}}$  of size  $sp \times sp$ . In principle,  $s$  should be at least 1  the estimated dimensionality of the dataset. In practice, they oversample by a factor of 2 or more.
- MDS is run on  $\mathbf{M}_{\text{align}}$ . After this, it is obtained  $\mathbf{mMDS}$ . Given a sampled point, there are two solutions of MDS: one from  $\mathbf{X}_i$  and another one from  $\mathbf{M}_{\text{align}}$ .
- The next step is to compute the Procrustes transformation to line these two sets of solutions up in a common coordinate system:

$$\mathbf{mMDS}_i = s_i \mathbf{dMDS}_i \mathbf{T}_i + \mathbf{1t}_i'$$

where:

- $\mathbf{dMDS}_i$  is  $\mathbf{MDS}_i$  but taking into account just the subset of the sample points that belongs to partition  $i$ .
- $\mathbf{mMDS}_i$  is  $\mathbf{mMDS}$  but taking into account just the subset of the sample points that belongs to partition  $i$
- After doing the previous part, we obtain a set of  $p$  Procrustes parameters  $(s_i, \mathbf{T}_i, \mathbf{t}_i)$ . So, the next step is to apply this set of parameters to each  $\mathbf{MDS}_i$ , i.e,

$$\mathbf{MDS}_i^a := s_i \mathbf{MDS}_i \mathbf{T}_i + \mathbf{1t}_i'.$$

- The last step is to join  $\mathbf{MDS}_1^a, \dots, \mathbf{MDS}_p^a$  all together, i.e,

$$\mathbf{MDS}_X := \mathbf{MDS}_1^a \cup \dots \cup \mathbf{MDS}_p^a.$$

They apply this process recursively until the size of  $\mathbf{X}_i$  is optimal to run MDS on. They find the stop condition as follows. Let  $l \times l$  be the largest matrix that allow MDS to be executed efficiently. There are two issues that impact the performance of the algorithm: the size of each submatrix after subdivision and the number of submatrices,  $p$ , that are stitched together at each step. Ideally, the size of each submatrix after division should be as large as possible without exceeding  $l$ . By the same token, the size of  $\mathbf{M}_{\text{align}}$  should be bounded by  $l$ . The number of submatrices to be stitched together,  $p$ , should be the largest number such that  $sp \leq l$ .

### 2.3.2 Some indicators about the goodness of fit

As we have done in subsection 2.2.1, we present some visual results of this algorithm. The data used is the same as in subsection 2.2.2. We call  $\mathbf{MDS}_{\text{Fast}}$  the result that provides the previous algorithm.

Figure (2.4) shows that, for this particular case, the algorithm captures quite well the dimensions of the original data, providing a high goodness of fit. In addition, dimension  $i$  of  $\mathbf{MDS}_{\text{Fast}}$  fits perfectly the same dimensions  $i$  of  $\mathbf{X}$  and just this one, as we can see in Table (2.2).

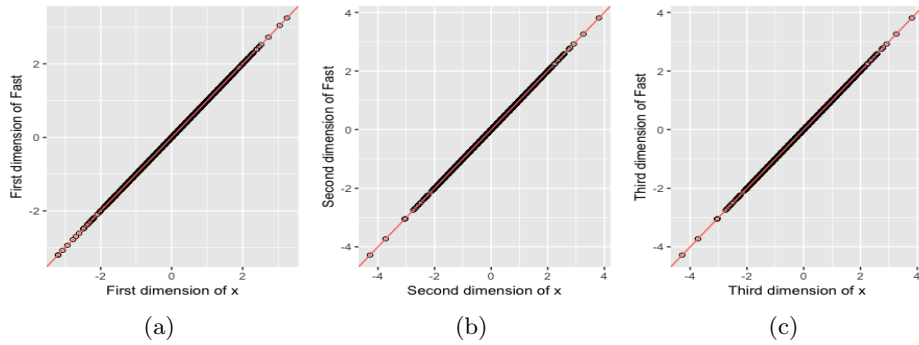


Figure 2.4: Dimesion 1,2 and 3 of  $\mathbf{X}$  against dimensions 1,2 and 3 of  $\mathbf{MDS}_{\text{Fast}}$ . In red, the line  $x = y$ .

	1	2	3
1	1	0.02	0
2	0.02	1	0.02
3	0	0.02	1

Table 2.2: Correlation of  $\mathbf{X}$  and  $\mathbf{MDS}_{\text{Fast}}$ .

## 2.4 Multidimensional Scaling based on Gower interpolation

Gower interpolation formula (Handl 1996) allows to add a new set of points to a given MDS configuration. Given a matrix  $\mathbf{X}$   $n \times p$ , a MDS configuration for this matrix

of dimension  $n \times c$  and a matrix  $\mathbf{X}_{\text{new}}$   $m \times p$ , one wants to add these new  $m$  rows to the existing MDS configuration. So, after adding this new rows, the MDS configuration will have  $m + n$  rows and  $c$  columns. We briefly summarise how to do so.

- Obtain  $\mathbf{J} = \mathbf{I}_n - \frac{1}{n}\mathbf{1}\mathbf{1}'$ , where  $\mathbf{I}_n$  is the identity matrix  $n \times n$ .
- Given the distance matrix  $\mathbf{D}$  of the rows of  $\mathbf{X}$ , calculate  $\mathbf{\Delta} = (\delta_{ij}^2)$ .
- Calculate  $\mathbf{G} = -\frac{1}{2}\mathbf{J}\mathbf{\Delta}\mathbf{J}'$
- Let  $\mathbf{g}$  be the diagonal of  $\mathbf{G}$ , i.e,  $\mathbf{g} = \text{diag}(\mathbf{G})$ .
- Let  $\mathbf{A}$  be the square distance matrix between the rows of  $\mathbf{X}$  and the rows of  $\mathbf{X}_{\text{new}}$ .  $\mathbf{A}$  has dimensions  $m \times n$ . Let  $\mathbf{A}^2$  be the matrix of the square elements of  $\mathbf{A}$ , i.e,  $\mathbf{A}^2 = (a_{ij}^2)$ .
- Let  $\mathbf{M}$  and  $\mathbf{S}$  be the MDS for  $\mathbf{X}$  and the variance-covariance matrix of the  $c$  columns of  $\mathbf{M}$ .
- The MDS for the new  $m$  observations is given by

$$\frac{1}{2n}(\mathbf{1}\mathbf{g} - \mathbf{A}^2)\mathbf{M}\mathbf{S}^{-1}. \quad (2.1)$$

The resulting MDS for the  $m$  observations of  $\mathbf{X}_{\text{new}}$  is in the same coordinate system as  $\mathbf{M}$ . So, here it is not needed to do any Procrustes transformation.

#### 2.4.1 Algorithm

- Divide  $\mathbf{X}$  into  $p$  partitions  $\mathbf{X}_1, \dots, \mathbf{X}_p$ .
- Calculate  $\mathbf{J}$ ,  $\mathbf{\Delta}$ ,  $\mathbf{G}$ ,  $\mathbf{g}$ ,  $\mathbf{A}$ ,  $\mathbf{M}$  and  $\mathbf{S}$  according to the above formulas.
- Obtain MDS for the first partition  $\mathbf{X}_1$ .
- Given a partition  $1 < k \leq p$ , do the following steps to get the related MDS:
  - Calculate the distance matrix between the rows of  $\mathbf{X}_1$  and  $\mathbf{X}_k$  and calculate the square of each element of this matrix. Let  $\mathbf{A}^2$  be this matrix (same as above).
  - Use Gower interpolation formula (2.1) to obtain MDS for partition  $k$ .
  - Accumulate this solution.

As in the previous two algorithms, there is a key parameter to choose:  $p$ , the number of partitions. For this algorithm,  $p$  is set in the following way. Let  $l \times l$  be the largest distance matrix that a computer can calculate efficiently. The value of  $p$  is set as  $n/p$ .

#### 2.4.2 Some indicators about the goodness of fit

We repeat the same as in subsection 2.2.2. Figure (2.5) and Table (2.3) shows that the algorithm, for this particular case, captures quite well the dimensions of the original data, providing a high goodness of fit.

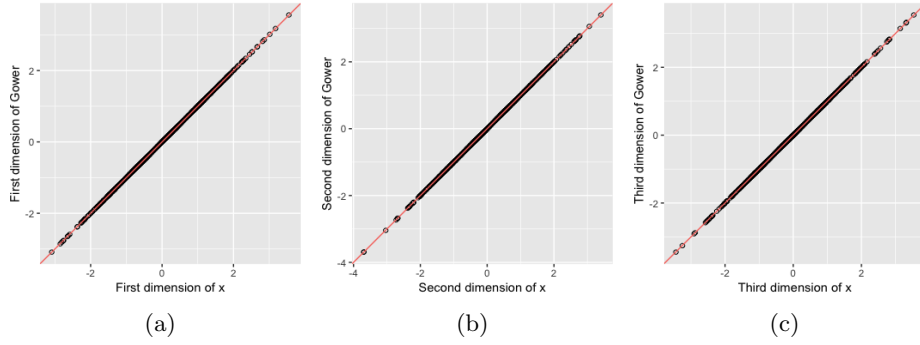


Figure 2.5: Dimesion 1,2 and 3 of  $\mathbf{X}$  against dimensions 1,2 and 3 of  $\text{MDS}_{\text{Gower}}$ . In red, the line  $x = y$ .

	1	2	3
1	1	0	-0.04
2	0	1	-0.0
3	-0.04	-0.03	1

Table 2.3: Correlation of  $\mathbf{X}$  and  $\text{MDS}_{\text{Gower}}$ .

## 2.5 Comparison of the algorithms

The three previous algorithms share the same goal: obtaining a MDS configuration for a given big dataset. However, there are some differences between the approaches that impact the performance of the algorithms. The main differences between them are:

- *Divide and Conquer* uses a guide (the first subset,  $\mathbf{X}_1$ ) to align the solutions as well as it uses the whole partition  $\mathbf{X}_i$  to find the Procrustes parameters. However, *Fast* does not use a guide an it uses a set of subsamples to find the Procrustes parameters.
- *Fast* is based on recursive programming. It divides until a manageable dimensionality is found. However, *Divide and Conquer* estimates such a dimensionality.
- *MDS based on Gower interpolation* does not need any Procrustes transformation.

The fact that we found three algorithms to compute MDS arises some questions that need to be answered:

- Are these algorithms able to capture the data dimensionality as good as calssical MDS does?
- Which is the fastest method?
- Can they deal with big datasets in a reasonable amount of time?
- How are they performing when dealing with big data sets?

All these questions and are answered in chapter 3.

## 2.6 Output of the algorithms

The three algorithms have the same type of output. It consists on a list of two parameters.

The first parameter is the **MDS** calculated by the algorithm. It is a matrix of  $n$  rows and  $c$  columns, where  $n$  is the number of rows of the input data and  $c$  is the number of dimensions the user has required.

The second parameter is a list of eigenvalues. When performing **MDS** a distance matrix is calculated and a singular value decomposition is performed over this distance matrix. Each eigenvalue is divided by the number of observations taken into account to obtain the distance matrix. We refer to this as the *normalized eigenvalues*. **The previous list is** returned.



## Chapter 3

# Simulation study

Given the three algorithms, we would like to know how they are performing. There are two issues to study:

- Goodness of fit of the algorithms: are they able to capture data dimensionality?
- Performance in terms of time: are they “fast” enough? Which one is the fastest?

To test the algorithm under different conditions, a simulation study has been carried out. The scenarios are obtained as a combination of:

- *Sample size*: we use different sample sizes, combining small data sets and big ones. A total of six sample sizes are used, which are:  $10^3, 3 \cdot 10^3, 5 \cdot 10^3, 10^4, 10^5, 10^6$ .
- *Data dimensionls*: we generate a matrix with two different number of columns: 10, 100.
- *Main dimensions*: given the data matrix  $\mathbf{X}$   $n \times k$ , it is postmultiplied by a diagonal matrix that contains  $k$  values,  $\lambda_1, \dots, \lambda_k$ . The first values are much higher than the rest. The idea of this is to see if the algorithms are able to capture the directions of the original dataset with the highest variance. We set 5 combinations for this variable, which are:
  - All the columns with the same values of  $\lambda$ :  $\lambda_1 = \dots = \lambda_k = 1$ .
  - One main dimension with  $\lambda_1 = 15$  and  $\lambda_2 = \dots = \lambda_k = 1$ .
  - Two main dimensions of the same value  $\lambda$ :  $\lambda_1 = \lambda_2 = 15, \lambda_3 = \dots \lambda_k = 1$ .
  - Two main dimensions of different values  $\lambda$ :  $\lambda_1 = 15, \lambda_2 = 10, \lambda_3 = \dots \lambda_k = 1$ .
  - Four main dimensions of the same value  $\lambda$ :  $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 15, \lambda_5 = \dots \lambda_k = 1$ .
- As distribution, it has been use a Normal one with  $\mu = 0$  and  $\sigma = 1$ . With this distribution, we generate a matrix of  $n$  observations and  $k$  columns, being the  $k$  columns independent. After generating the dataset  $\mathbf{X}$ , it is postmultiplied by the diagonal matrix that contains de value of the  $\lambda$ 's.

There is a total of 60 scenarios to simulate. Given a scenario, it is replicated 100 times. For every simulation, it is generated a dataset (according to the scenario), and all the algorithms are run using this dataset.

Table (3.1) shows the configuration of each scenario. Given a scenario, *scenario\_id* identifies it. We refer a scenario by its *scenario\_id*. So, a total of 6000 simulations are carried out.

	scenario_id	sample_size	n_dimensions	value_primary_dimensions
1	1	1000	10	NULL
2	2	1000	100	NULL
3	3	1000	10	15
4	4	1000	100	15
5	5	1000	10	c(15, 15)
6	6	1000	100	c(15, 15)
7	7	1000	10	c(15, 10)
8	8	1000	100	c(15, 10)
9	9	1000	10	c(15, 15, 15, 15)
10	10	1000	100	c(15, 15, 15, 15)
11	2000	3000	10	NULL
12	2001	3000	100	NULL
13	2002	3000	10	15
14	2003	3000	100	15
15	2004	3000	10	c(15, 15)
16	2005	3000	100	c(15, 15)
17	2006	3000	10	c(15, 10)
18	2007	3000	100	c(15, 10)
19	2008	3000	10	c(15, 15, 15, 15)
20	2009	3000	100	c(15, 15, 15, 15)
21	4000	5000	10	NULL
22	4001	5000	100	NULL
23	4002	5000	10	15
24	4003	5000	100	15
25	4004	5000	10	c(15, 15)
26	4005	5000	100	c(15, 15)
27	4006	5000	10	c(15, 10)
28	4007	5000	100	c(15, 10)
29	4008	5000	10	c(15, 15, 15, 15)
30	4009	5000	100	c(15, 15, 15, 15)
31	6000	10000	10	NULL
32	6001	10000	100	NULL
33	6002	10000	10	15
34	6003	10000	100	15
35	6004	10000	10	c(15, 15)
36	6005	10000	100	c(15, 15)
37	6006	10000	10	c(15, 10)
38	6007	10000	100	c(15, 10)
39	6008	10000	10	c(15, 15, 15, 15)
40	6009	10000	100	c(15, 15, 15, 15)
41	20000	100000	10	NULL
42	20001	100000	100	NULL

43	20002	100000	10	15
44	20003	100000	100	15
45	20004	100000	10	c(15, 15)
46	20005	100000	100	c(15, 15)
47	20006	100000	10	c(15, 10)
48	20007	100000	100	c(15, 10)
49	20008	100000	10	c(15, 15, 15, 15)
50	20009	100000	100	c(15, 15, 15, 15)
51	30000	1000000	10	NULL
52	30001	1000000	100	NULL
53	30002	1000000	10	15
54	30003	1000000	100	15
55	30004	1000000	10	c(15, 15)
56	30005	1000000	100	c(15, 15)
57	30006	1000000	10	c(15, 10)
58	30007	1000000	100	c(15, 10)
59	30008	1000000	10	c(15, 15, 15, 15)
60	30009	1000000	100	c(15, 15, 15, 15)

Table 3.1: Scenarios simulated

In order to test the goodness of fit and to check the time of the algorithms, some metrics are calculated.

- Goodness of fit:
  - We get the correlation between the main dimensions of the data and the main dimensions after applying the algorithms. We get just the diagonal, i.e, the correlation between dimension  $i$  of the data and the dimension  $i$  of the algorithm.
  - We get the value of the eigenvalue divided by the sample sized of the distance matrix used to compute the MDS.
- Time: Given an algorithm, the elapsed time to get the corresponding MDS configuration is stored.

We do it in this way because we want to check some hypothesis. We expect the three algorithms to behave “correctly”. By it, we mean that the behavior should be the same as if classical MDS were run. Therefore, we expect that the correlation between the main dimensions of the data and the main dimensions of the MDS of each algorithm to be close to 1.

In addition, the variance of the original data should be captured. So, given the highest eigenvalues, we expect that  $\sqrt{\text{eigenvalues}/r}$ <sup>1</sup> is approximately 15 or 10, where  $r$  is the number of rows of the distance matrix used to compute the eigenvalues. In case there is not any main dimension in the original data, i.e  $\lambda_1 = \dots = \lambda_k = 1$ , we expect this quocient to be approximately 1.

Given a scenario, the steps that we have perform to calculate all the data needed is:

---

<sup>1</sup> This is what we call *normalized eigenvalues* and it is returned by each algorithm.

1. Generate the data according to the scenario.
2. Run *divide and conquer* algorithm and get the time and the normalized eigenvalues.
3. Align  $\mathbf{MDS}_{\text{Div}}$  and  $\mathbf{X}$  using Procrustes.
4. Get the correlation between the main dimensions of  $\mathbf{MDS}_{\text{Div}}$  and  $\mathbf{X}$ .
5. Run *fast* algorithm and get the time and the normalized eigenvalues.
6. Align  $\mathbf{MDS}_{\text{Fast}}$  and  $\mathbf{X}$  using Procrustes.
7. Get the correlation between the main dimensions of  $\mathbf{MDS}_{\text{Fast}}$  and  $\mathbf{X}$ .
8. Run *MDS based on Gower interpolation* algorithm and get the time and the normalized eigenvalues.
9. Align  $\mathbf{MDS}_{\text{Gower}}$  and  $\mathbf{X}$  using Procrustes.
10. Get the correlation between the main dimensions of  $\mathbf{MDS}_{\text{Gower}}$  and  $\mathbf{X}$ .

There are some important details related to the previous process that we have taken into account to make the simulator process faster. When running the algorithms, we ask for as many columns as the original data has, i.e.  $k$ .

For the eigenvalues, we just get 6 eigenvalues instead of the full list of eigenvalues (otherwise we would get  $n$  eigenvalues).

For Procrustes we do not allow dilations, otherwise we would not know if the distance is preserved. In addition, we do not use all the columns to do the alignment, we select the main dimensions. If there is not any main dimension, we just select 5 columns. In addition to that, we have had problems to do the alignment when  $n$  was  $10^5$  or higher. To do the alignment we have done the following process:

1. Create  $p$  partitions of  $\mathbf{X}$  and the result of a given MDS algorithm ( $\mathbf{MDS}_{\text{alg}}$ ).
2. For each partition get the Procrustes parameters without dilations.
3. Accumulate the parameters iteration after iteration. So, at the end, we obtain  $\mathbf{R} = \sum_{i=1}^p \mathbf{R}_i$  and  $\mathbf{t} = \sum_{i=1}^p \mathbf{t}_i$ .
4.  $\mathbf{R} = \mathbf{R}/p$  and  $\mathbf{t} = \mathbf{t}/p$ .
5. Apply these parameters to align the matrices.

## Chapter 4

## Conclusions

# Bibliography

Borg, I. and P. Groenen (2005). *Modern Multidimensional Scaling: Theory and Applications*. Springer.

Handl, A. (1996, October). Biplots : J. C. Gower and D. J. Hand (1996) London: Chapman & Hall, ISBN 0-412-71630-5, [pound sign] 32.00, pp. 277. *Computational Statistics & Data Analysis* 22(6), 655–651.

Peña, D. (2002). *Análisis de datos multivariantes*. Madrid, Spain: McGraw Hill.

Tynia, Y., L. Jinze, M. Leonard, and W. Wei (2006). A fast approximation to multidimensional scaling.

# Appendix A

## Code

## Appendix B

# Problems encountered during the development