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

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 Eucliden distance between the rows of $\mathbf{X}$ is approximately equat 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).

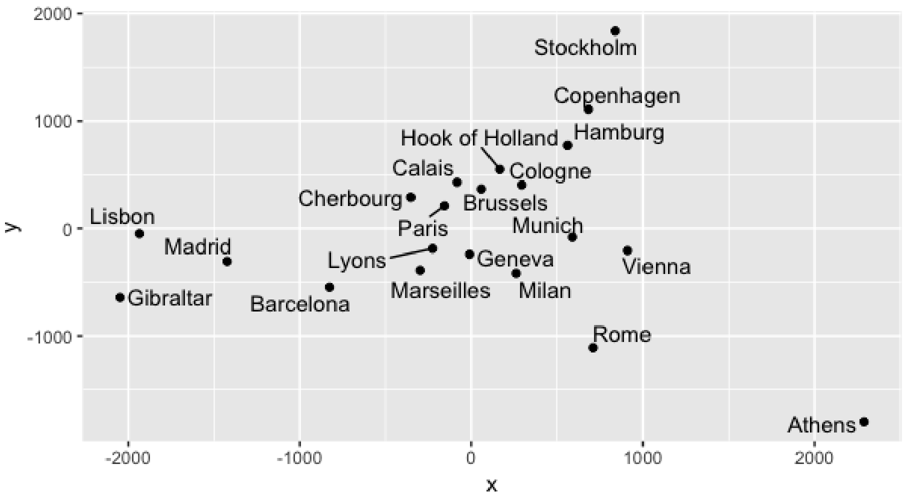


Figure 1.1: MDS on the Eurepean cities.

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:

$$\widetilde{\mathbf{X}} = \Big(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}'\Big)\mathbf{X} = \mathbf{P}\mathbf{X},$$

where

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

This new matrix, $\widetilde{\mathbf{X}}$, has the same dimensions as the orginial 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 $\widetilde{\mathbf{X}}'\widetilde{\mathbf{X}}/n$ and the cross-prodructs matrix $Q = \widetilde{\mathbf{X}}\widetilde{\mathbf{X}}'$. The last matrix can be interpreted as a similarity matrix between the n elements. The term ij is obtained as follows:

$$q_{ij} = \sum_{s=1}^{p} x_{is}x_{js} = \mathbf{x_i'}\mathbf{x_j} \tag{1.1}$$

where $\mathbf{x_i}'$ is the i-th row from $\widetilde{\mathbf{X}}$.

Given the scalar product formula, $\mathbf{x_i'}\mathbf{x_j} =\mid \mathbf{x_i} \mid\mid \mathbf{x_i} \mid \cos\theta_{ij}$, 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, $\widetilde{\mathbf{X}}\widetilde{\mathbf{X}}'$ can be interpreted as the similarity matrix between the elements.

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

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

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

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

We have just seen that, given the matrix $\widetilde{\mathbf{X}}$, it is possible to get the similarity matrix $\mathbf{Q} = \widetilde{\mathbf{X}}\widetilde{\mathbf{X}}'$ and from it, to get the distance matrix $\mathbf{D}$. Let $\mathrm{diag}(\mathbf{Q})$ 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

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

The problem we are dealing with goes in the opposite direction. We want to rebuild $\widetilde{\mathbf{X}}$ from a square distance matrix $\mathbf{D}$, with elements d_{ij}^2. The first step is to obtain $\mathbf{Q}$ and afterwards, to get $\widetilde{\mathbf{X}}$. 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:

$$d_{ij}^2 = \sum_{s=1}^{p}(x_{is} - x_{js})^2 = \sum_{s=1}^{p}[(x_{is} - \overline{x_s}) - (x_{js} - \overline{x_s})]^2. \tag{1.4}$$

The previous condition means that we are lookig for a matrix $\widetilde{\mathbf{X}}$ such that $\widetilde{\mathbf{X}}'\mathbf{1} = 0$. It also means that $\mathbf{Q1} = 0$, 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 constrains, we sum up (1.3) at row level:

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

where $t = \sum_{i=1}^{n} q_{ii} = \mathrm{Trace}(\mathbf{Q})$, and we have used that the condition $\mathbf{Q1} = 0$ implies $\sum_{i=1}^{n} q_{ij} = 0$. Summing up (1.3) at column level:

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

Summing up (1.5) we obtain:

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

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

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

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

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

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

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

Finally, from (1.9) we get the expression:

$$q_{ij} = -\frac{1}{2}(d^2_{ij} - d^2_{i.} - d^2_{.j} + d^2_{..}). \tag{1.10}$$

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

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

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

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

Getting

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

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

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

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 unchaged. Therefore, the solution is not unique.

1.3 Building principal coordinates

In general, the distance matrix is not compatible with an Eucliden metric but usually the similarity matrix obtained from it has p positive eigenvalues which are greater than the other ones. If the rest $n - p$ of not null eigenvalues are much less than the other ones, it is possible to obtain an (approximated) representation using the p eigenvectors associated with the first p eigenvalues of the similarity matrix.

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

1. Build the matrix $\mathbf{Q} = -\frac{\mathbf{1}}{\mathbf{2}}\mathbf{PDP}$ of cross-products.

2. Obtain the eigenvalues of $\mathbf{Q}$. Take the r greatest eigenvalues. Since $\mathbf{P1} = 0$, where $\mathbf{1}$ is a vector of ones, 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 λ_i is an eigenvalue of $\mathbf{Q}$ and $\mathbf{v_i}$ is the associated unitary eigenvector. This implies that $\mathbf{Q}$ is apporximated 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*, is characterized by the following properties (s_{ij} denotes the similarity between element i and j):

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

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

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

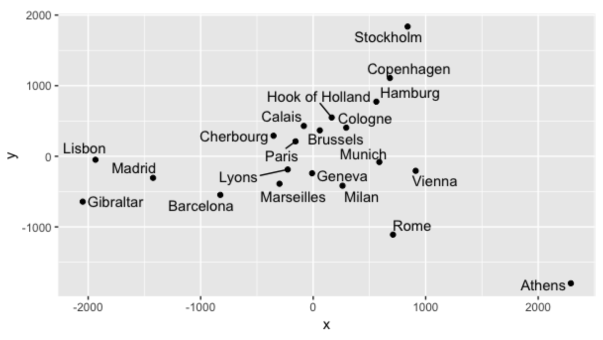
and it is easy to see that $\sqrt{2(1 - q_{ij})}$ is a distance and it verifies the triangle inequality.

1.4 Procrustes transformation

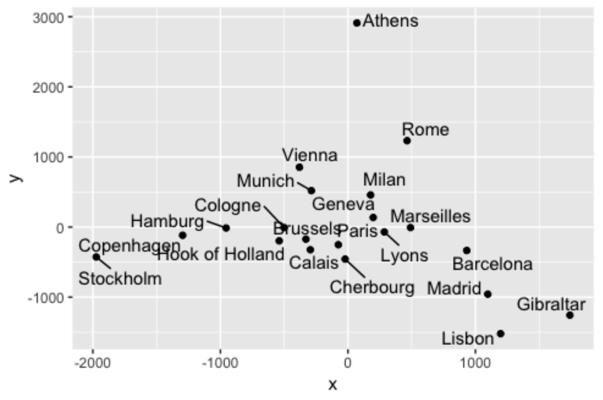
As we have mentioned before, the MDS solution is not unique. One example of it is shown in Figure 1.2.

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? *Align both solutions* (or multiple ones) means to find a common coordinate system for all the 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. 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.



(a)



(b)

Figure 1.2: Two different solutions of MDS.

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 repeatdly used tool, we briefly summarise it.

Let $\mathbf{A}$ and $\mathbf{B}$ be two different MDS configurations of dimensions $n \times t$ 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 $s \in \mathbb{R}$, $\mathbf{T} \in \mathrm{M}_{r\times r}(\mathbb{R})$ and $\mathbf{t} \in \mathbb{R}^r$ 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.

1.5 Multidimensional Scaling with R

All the algorithms have been coded in R. 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 a list of two elements:
  - The first r principal coordinates for the individuals, i.e, the low-dimensional configuration for the data.
  - 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 a list of three elements:
  - The dilation coeficient 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.

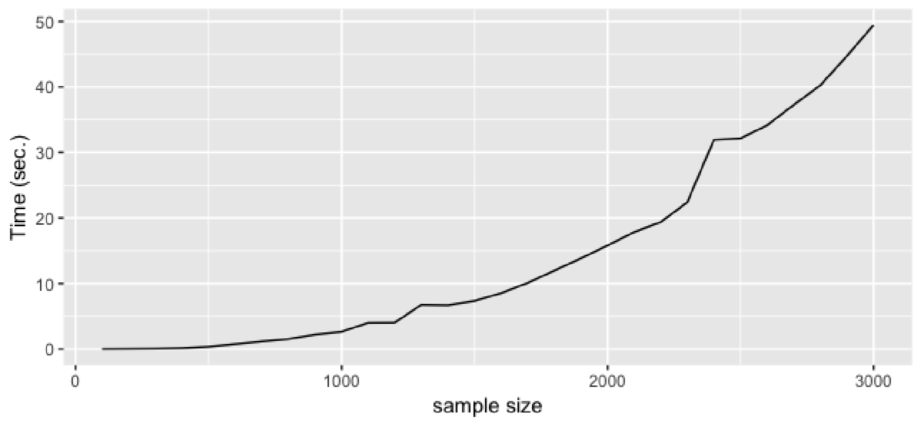


Figure 2.1: Elapsed time to compute MDS.

Figure 2.1 shows the time needed to compute MDS as a function of the sample size. As we can see, the time grows considerably as the sample size increases when using cmdscale function. 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 considered 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 dividng 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.

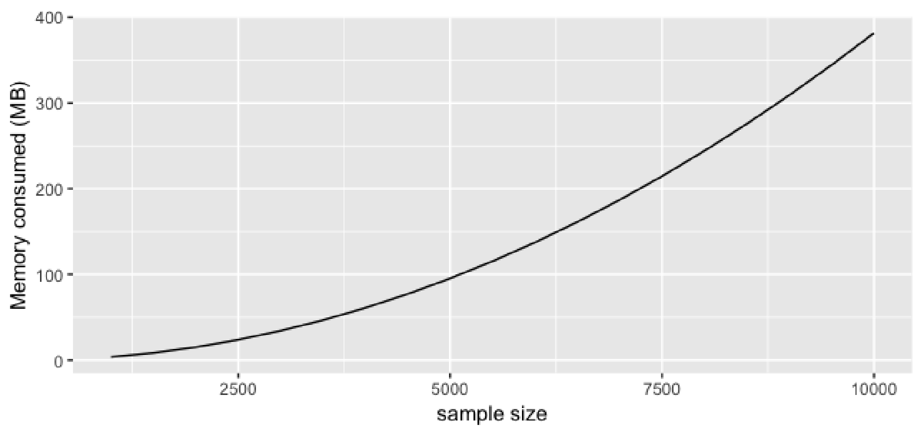


Figure 2.2: Memory consumed to compute distance.

- *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. For further details see, for instance, the appendix in (Gower and Hand 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 B.

2.2 Divide and Conquer MDS

2.2.1 Algorithm

- The first step is to divide the original dataset into p partitions: $\mathbf{X_1}, \ldots, \mathbf{X_p}$. 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**: **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

$$\mathbf{cum\text{-}mds}\Big|_{k-1} \approx s\mathbf{MDS_{k,k-1}}\Big|_{k-1}\mathbf{T} + \mathbf{1t}'.$$

– Take the rows of the partition k from $\mathbf{MDS_{k,k-1}}$: $\mathbf{MDS_{k,k-1}}\Big|_{k}$.

– Use the previous Procrustes parameters to add the rows of $\mathbf{MDS_{k,k-1}}\Big|_{k}$ to **cum-mds**:

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

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

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

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 size of the largest matrix that allows to run MDS eficiently, i.e, in a reasonable amount of time. If n is the number of rows of $\mathbf{X}$, then p is $\frac{2n}{l}$.

2.2.2 Some indicators about the performance of the algorithm

The aim of this section is to show some indicators about the performance of the algorithm. What we want to do here is to show some visual results. In Section 3.1 we provide more details.

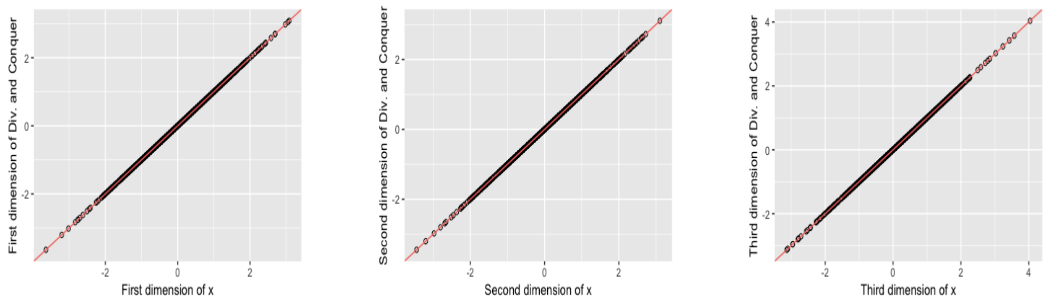
We have generated a matrix $\mathbf{X}$ with 3 independent *Normal* distributions ($\mu = 0$ and $\sigma = 1$) and 1000 rows with l equal to 500. Afterwards, we have run the algorithm. We have requiered the algorithm to return 3 columns. So, a new matrix with 3 columns and 1000 rows ($\mathbf{MDS_{Div}}$) 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 preserved.

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 $\mathbf{MDS_{Div}}$, $i \in \{1, 2, 3\}$.

As we can see, the algorithm is able to caputure the dimensions of the original matrix. We do not show cross-dimensions (i.e dimesion i of $\mathbf{X}$ against dimesion j of $\mathbf{MDS_{Div}}$ $i \neq j$), but Table 2.1 contains the cross-correlation matrix. The results show that dimension i of $\mathbf{MDS_{Div}}$ caputures dimension i of $\mathbf{X}$ and just dimension i. So, it seems that the algorithm, for this particular case, has a good performance.

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

Table 2.1: Cross-correlation of $\mathbf{X}$ and $\mathbf{MDS_{Div}}$.



(a) (b) (c)

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

2.3 Fast MDS

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 $\mathbf{X_1}, \ldots, \mathbf{X_p}$.
- Compute MDS for each $\mathbf{X_i}$: $\mathbf{MDS_1}, \ldots, \mathbf{MDS_p}$. 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 $\mathbf{M_{align}}$ of size $sp \times sp$. In principle, s should be at least 1 plus the estimated dimensionality of the dataset. In practice, they oversample by a factor of 2 or more.
- MDS is run on $\mathbf{M_{align}}$. After this, it is obtained **mMDS**. Given a sampled point, there are two solutions of MDS: one from $\mathbf{X_i}$ and another one from $\mathbf{M_{align}}$.
- The next step is to compute the Procrustes transformation to line these two sets of solutions up in a common coordinate system:

  $$\mathbf{mMDS_i} = s_i \mathbf{dMDS_i T_i} + \mathbf{1 t_i}'$$

  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 **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,

  $$\mathbf{MDS_i}^a := s_i \mathbf{MDS_i T_i} + \mathbf{1 t_i'}.$$

- The last step is to join $\mathbf{MDS_1}^a, \dots, \mathbf{MDS_p}^a$ all together, i.e,

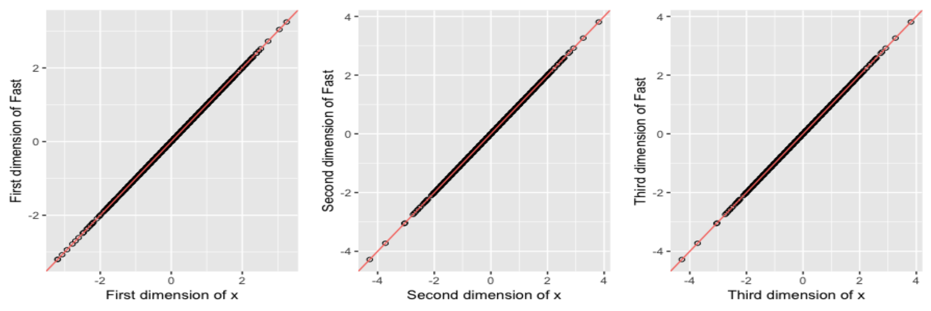
$$\mathbf{MDS_X} := \mathbf{MDS_1}^a \cup \dots \cup \mathbf{MDS_p}^a.$$

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 $\mathbf{M_{align}}$ 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 performance of the algorithm

As we have done in Section 2.2.2, we present some visual results of this algorithm. The data used are the same as in Section 2.2.2. We call $\mathbf{MDS_{Fast}}$ the result that provides the previous algorithm.

Figure 2.4 shows that, for this particular case, the algorithm captures quite well the dimesions of the original data, providing a good performance. In addition, dimesion i of $\mathbf{MDS_{Fast}}$ fits perfectly the same dimesions i of $\mathbf{X}$ and just this one, as we can see in Table 2.2.



(a) (b) (c)

Figure 2.4: Dimesion 1,2 and 3 of $\mathbf{X}$ against dimensions 1,2 and 3 of $\mathbf{MDS_{Fast}}$. 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: Cross-correlation of $\mathbf{X}$ and $\mathbf{MDS_{Fast}}$.

2.4 MDS based on Gower interpolation

Gower interpolation formula (see appendix of (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 $\mathbf{X_{new}}$ $m \times p$, one wants to add these new m rows to the existing MDS configuration. So, after adding this new rows, the MDS configuration will have $n + m$ rows and c columns. We briefly summarise how to do so:

- Obtain $\mathbf{J} = \mathbf{I_n} - \frac{1}{n}\mathbf{11'}$, 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 $\mathbf{G} = -\frac{1}{2}\mathbf{J\Delta J'}$
- Let $\mathbf{g}$ be the diagonal of $\mathbf{G}$, i.e, $\mathbf{g} = \text{diag}(\mathbf{G})$.
- Let $\mathbf{A}$ be the distance matrix between the rows of $\mathbf{X}$ and the rows of $\mathbf{X_{new}}$. $\mathbf{A}$ 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 interpolated coordinates for the new m observations are given by

$$\frac{1}{2n}(\mathbf{1g'} - \mathbf{A}^2)\mathbf{MS}^{-1}. \tag{2.1}$$

The resulting MDS for the m observations of $\mathbf{X_{new}}$ 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 size of the largest distance matrix that a computer can calculate efficently, i.e, in a reasonable amount of time. The value of p is set as n/p.

2.4.2 Some indicators about the performance of the algorithm

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

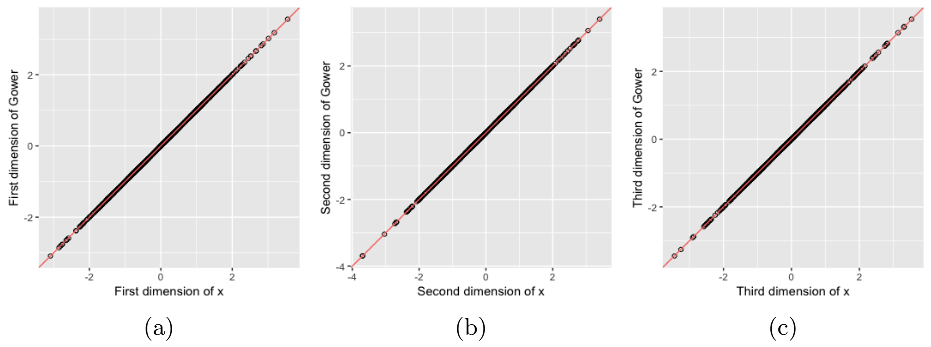


Figure 2.5: Dimesion 1,2 and 3 of $\mathbf{X}$ against dimensions 1,2 and 3 of $\mathbf{MDS_{Gower}}$. In red, the line $x = y$.

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

Table 2.3: Cross-correlation of $\mathbf{X}$ and $\mathbf{MDS_{Gower}}$.

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 MDS* 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 MDS* does not use a guide an it uses a set of subsamples to find the Procrustes parameters.
- *Fast MDS* is based on recursive programming. It divides until a manageable dimensionality is found. However, *Divide and Conquer MDS* finds the number of partitions without applying recursive programming.
- *MDS based on Gower interpolation* does not need any Procrustes transformation.

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

- Are these algorithms able to capture the data dimensionaly 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 Section 3.1.

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 configuration 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 paramenter is a list of eigenvalues. This list is built as follows:

- All the algorithms divide the initial data into a set of p partitions.
- Given a partition i, a distance matrix of dimensions $m_i \times m_i$ is calculated: $\mathbf{D_i}$.
- Over $\mathbf{D_i}$ a singular value decomposition is performed, providing a list of length m_i that contains all the eigenvalues of the previous decomposition: $list_i$.
- Let $norm\_eigenvalues_i$ be $list_i/m_i$, i.e, each eigenvalue is divided by the number of rows of $\mathbf{D_i}$.
- The algorithms return $norm\_eigenvalues_1 \cup \cdots \cup norm\_eigenvalues_p$. We refer to this union as the *normalized eigenvalues*.

Chapter 3

Simulation study

3.1 Design of the simulation

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 main dimensions of the original dataset, i.e, the columns with the highest variance. We set 5 combinations for this variable, which are:
  - All the columns with the same values of λ: $\lambda_1 = \cdots = \lambda_k = 1$.
  - One main dimension with $\lambda_1 = 15$ and $\lambda_2 = \cdots = \lambda_k = 1$.
  - Two main dimensions of the same value λ: $\lambda_1 = \lambda_2 = 15$, $\lambda_3 = \cdots \lambda_k = 1$.
  - Two main dimensions of different values λ: $\lambda_1 = 15$, $\lambda_2 = 10$, $\lambda_3 = \cdots \lambda_k = 1$.
  - Four main dimensions of the same value λ: $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 15$, $\lambda_5 = \cdots \lambda_k = 1$.
- As distribution, it is 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 values of λ'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. So, a total of 6000 simulations are carried out.

Table 3.1 shows the configuration of each scenario. Given a scenario, *scenario_id* identifes it. We refer to a scenario by its *scenario_id*.

	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

Note that scenarios 1, 2, 2000, 2001, 4000, 4001, 6000, 6001, 20000, 20001, 30000, 30001 are pure noise. We refer to them as *noisy scenarios*.

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 *normalized eigenvalues*.
- 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 "correcty". 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 *normalized eigenvalues*, we expect that its square root is approximately 15 or 10 when the scenarios are not the *noisy scenrarios*.

For the time of the algorithms, we have done some tests and it seems that *MDS based on Gower interpolation* seems to be the fastest. So, it will be tested.

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

1. Generate the data according to the scenario.
2. Run *divide and conquer* algorithm and get the time and the *normalized eigenvalues.*
3. Align $\mathbf{MDS_{Div}}$ and $\mathbf{X}$ using Procrustes.
4. Get the correlation between the main dimensions of $\mathbf{MDS_{Div}}$ and $\mathbf{X}$.
5. Run *fast* algorithm and get the time and the *normalized eigenvalues.*
6. Align $\mathbf{MDS_{Fast}}$ and $\mathbf{X}$ using Procrustes.
7. Get the correlation between the main dimensions of $\mathbf{MDS_{Fast}}$ and $\mathbf{X}$.
8. Run *MDS based on Gower interpolation* algorithm and get the time and the *normalized eigenvalues.*
9. Align $\mathbf{MDS_{Gower}}$ and $\mathbf{X}$ using Procrustes.
10. Get the correlation between the main dimensions of $\mathbf{MDS_{Gower}}$ 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 runnig the algorithms, we ask for as many columns as the original data has, i.e k.

For the *normalized eigenvalues*, we just get 6 eigenvalues instead of the full list of eigenvalues (orherwise we would get n eigenvalues).

For Procruster we dot 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, i.e it is one of the *noisy scenarios*,we just select 4 columns.

To avoid memory problems with the alignment when n is greater or equal to 10^5, Procrustes is done in the following way:

1. Create p partitions of $\mathbf{X}$ and the result of a given MDS algorithm ($\mathbf{MDS_{alg}}$). Both set of partitios contains exctaly the same observations.
2. For each partition get the Procrustes parameters without dilations.
3. Accumulate the paramenters 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 $\mathbf{MDS_{alg}}$ so that $\mathbf{X}$ and $\mathbf{MDS_{alg}}$ are in the same coordinate system and they can be compared.

The algorithms have as input values a set of variables. The input matrix is already explained, but there is another parameter that has been used during the description of the algorithms: l. The meaning of l is a little bit different in each algorithm, but for simplicity we set this value equals to 500.

Fas MDS has an extra parameter: the amplification parameter. It is used the value that they used to test this algorithm, i.e, a value of 3. So, for each partition, it is taken 30 (when the orginial matrix has 10 columns) or 300 (when the orginial matrix has 10 columns) points for every partition to buid $\mathbf{M_{align}}$.

Finally, there is an extra "parameter" to take into account: the machine used to do the simulations. Since a total of 6000 simulations are performed and some of them include big datasets, we use *Amazon Web Services* (AWS) to carry out the simulations. 10 servers of he same type are used: *c5n.4xlarge*. It has 16 cores and 42 GB of memory RAM. This kind of server is designed for applications like batch and log processing, distributed and or real-time analytics.

3.2 Correlation coeficients

In this section we provide some results for the correlation coeficients based on the simulations. Given a scenario and its dataset $\mathbf{X}$ $n \times k$, the correlation matrix between the main dimensions of $\mathbf{X}$ and the main dimensions of $\mathbf{MDS_{alg}}$ is computed. We are interested in the diagonal of the correlation matrix, expecting that the values are close to 1.

The length of the diagonal can be:

- 1, when $\lambda_1 = 15$ and $\lambda_2 = \cdots = \lambda_k = 1$.
- 2, when$\lambda_1 = \lambda_2 = 15$, $\lambda_3 = \cdots \lambda_k = 1$ or $\lambda_1 = 15$, $\lambda_2 = 10$, $\lambda_3 = \cdots \lambda_k = 1$.
- 4, when $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 15$, $\lambda_5 = \cdots \lambda_k = 1$.
- 0, when $\lambda_1 = \cdots = \lambda_k = 1$, i.e, *noisy scenrarios.*

3.2.1 Divide and Conquer MDS

Figure 3.1 shows some boxplots of the correlation coeficients between the main dimensions of the data $\mathbf{X}$ and the main dimensions of $\mathbf{MDS_{Div}}$. As we can see, all the values are close to 1, showing high goodness of the algorithm.

Note that before calculating the correlation matrix, Procrustes transformation is performed (dilations are not allowed).

Figure 3.1 just shows the correlation coeficients for $n = 1000$. The remaining figures are in Appendix A.1. All of them have the same shape no matter what scenario it is: there is always high correlation between the data and the MDS.

Figure 3.2 shows the boxplot for the *noisy scenarios*. It has been taken 4 dimesions to do the alignment, instead of all the dimensions (10 or 100). Because of this, we expect a random distribution between them within [0,1] interval. If we had taken all the dimensions, we would be seeing higher correlation values.

The correlation is so high that it is not needed any hypothesis test to test if the value is 1 or not.

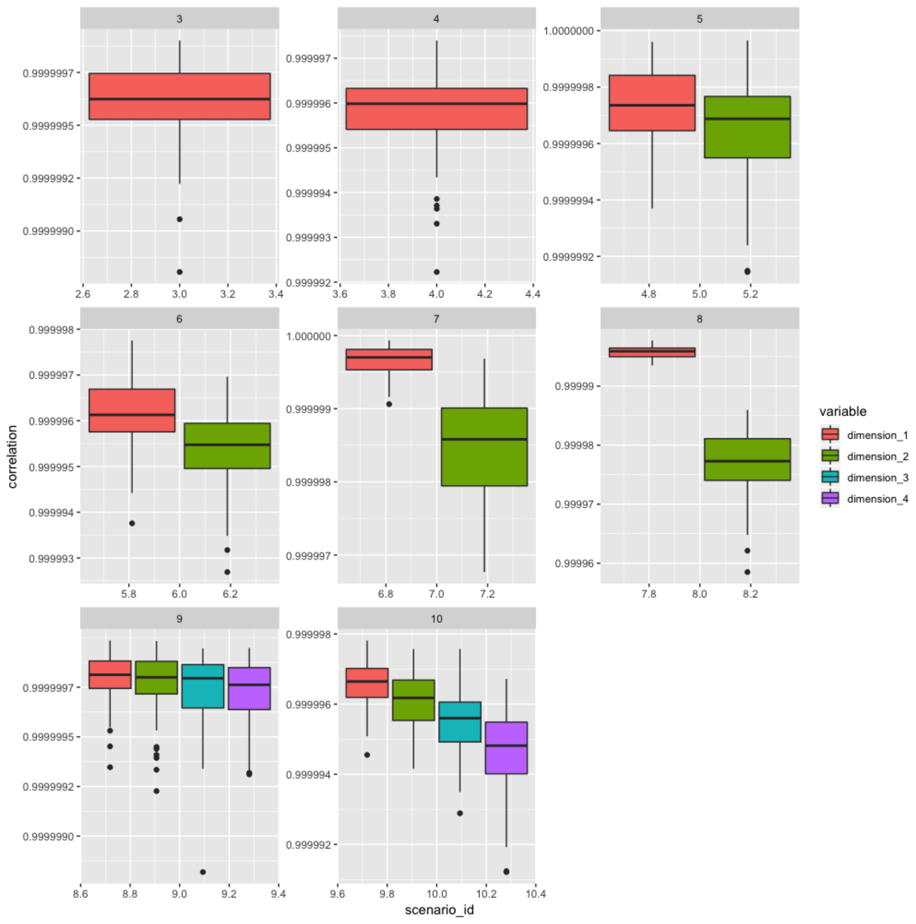


Figure 3.1: Correlation for $n = 1000$ and MDS Divide and conquer algorithm

3.2.2 Fast MDS

We do the same for this algorithm. Figure 3.3 shows the boxplot of the correlation coeficients between the data and the MDS of this algorithm. Again, there is high correlation between them. Figure 3.4 shows the boxplot fot the *noisy scenarios*.

The remaining plots are in Appendix A.2.

3.2.3 MDS based on Gower interpolation

We do the same for this algorithm. Figure 3.5 shows the boxplot of the correlation coeficients between the data and the MDS of this algorithm. Again, there is high correlation between them. Figure 3.6 shows the boxplot fot the *noisy scenarios*.

The remaining plots are in Appendix A.3.

3.3 Eigenvalues

3.4 Time to compute MDS

In this section we test if there exists an algorithm that is faster than the other ones. Our intuition is that *MDS based on Gower interpolation* is the fastest algorithm. Out

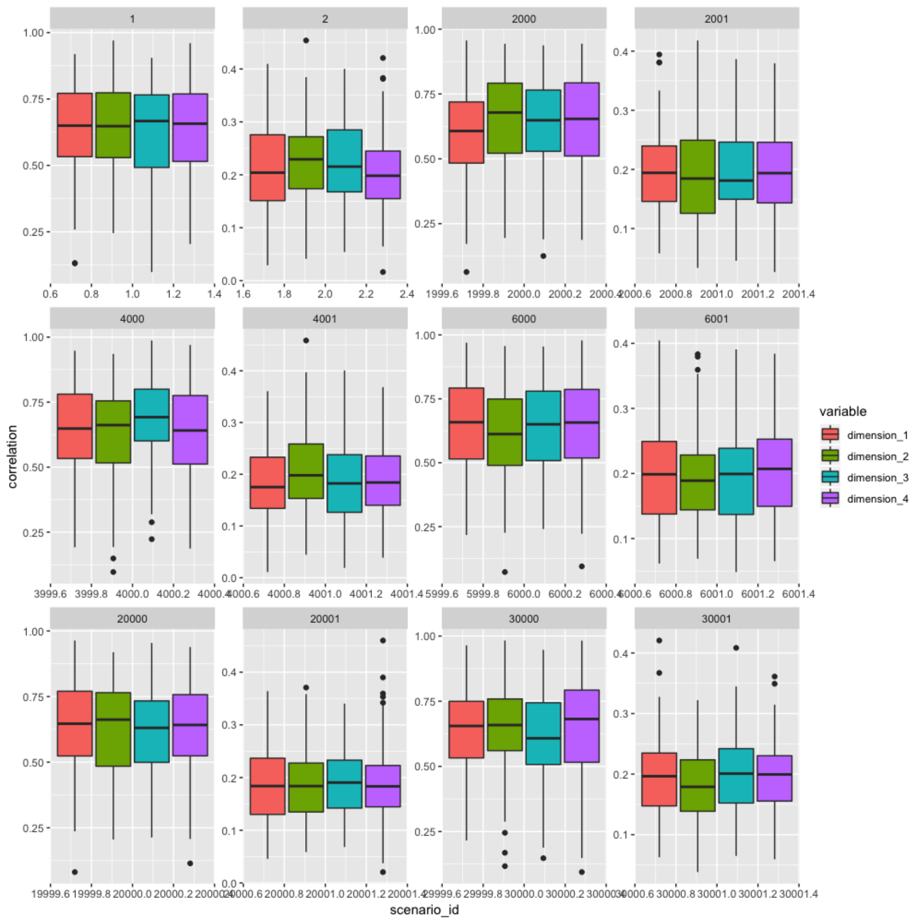


Figure 3.2: Correlation for *noisy scenarios* and MDS Divide and conquer algorithm

intuition is based than is has the lowest mean value, as we can see in Table 3.2.

Table 3.2 provides a rank between the methods: it seems that *MDS based on Gower interpolation* is the fastest one. In second position it would be *Fast MDS* and finally *Divide and Conquer*.

We do an hypthesis test to check it. We do an ANOVA test using two factors: the sample size (which has 6 levels) and the number of dimensions (which has 2 levels). Instead of using the *elapsed time* variable, we use its logarithm.

Given the results of the ANOVA test, which are in Table 3.3, we can reject the null hypothesis. So, there exists τ_{ij} such that $\tau_{ij} \neq 0$.

Now, we should look for such a level, expecting that $\tau_{ij} \neq 0 \forall i, j$. Instead, we fit a linear regression with these variables and see the magnitude of the coeficients. In addition, we plot the distribution of log(elpased_time) for all the algorithms.

Table 3.4 contains the value of the coefficients. As long as either the sample size or the data dimensions increases the coefficient does the same (and so the time needed). This is what we expect: the bigger the data the more time needed. Looking at the algorithm values, it seems that *MDS base on Gower interpolation* is the fastest, being the coefficient equal to 0. On the other hand, *Divide and Conquer* is the slowest one.

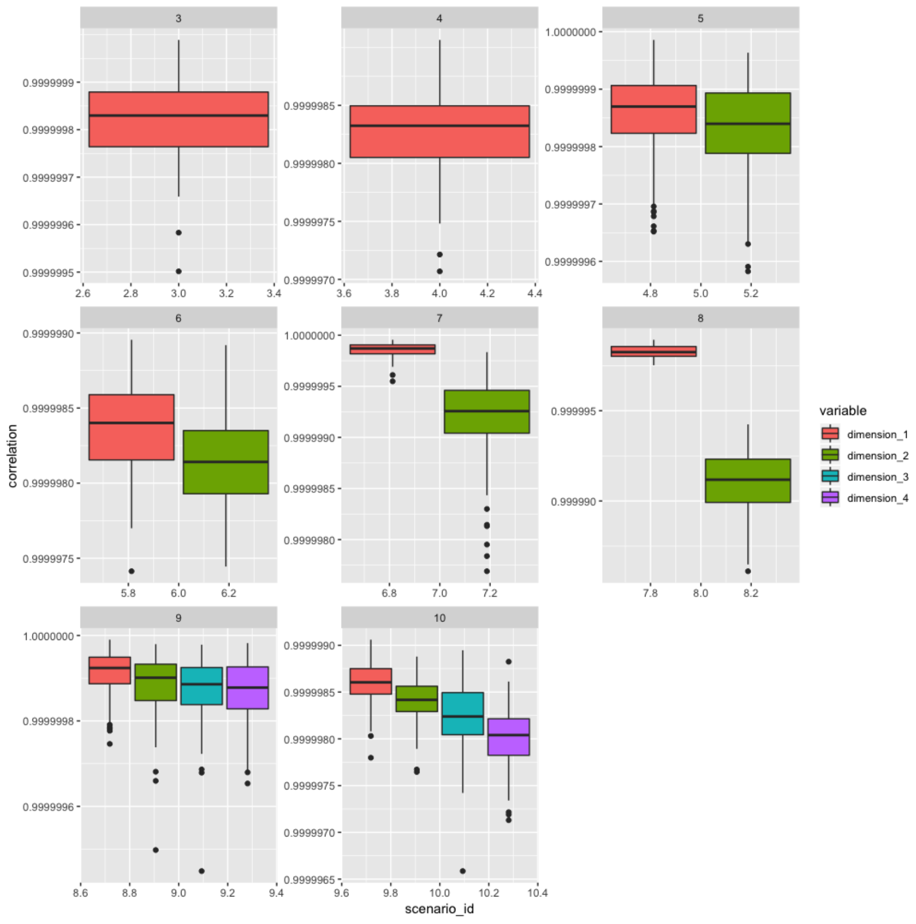


Figure 3.3: Correlation for $n = 1000$ and MDS Fast algorithm.

	sample_size	n_dimensions	mean_divide_conquer	mean_fast	mean_gower
1	1000	10	0.27	0.14	0.10
2	1000	100	0.78	0.69	0.28
3	3000	10	0.78	0.32	0.16
4	3000	100	2.50	3.14	0.52
5	5000	10	1.37	0.54	0.20
6	5000	100	4.25	5.69	0.84
7	10000	10	2.60	1.81	0.31
8	10000	100	8.85	11.79	1.37
9	100000	10	28.10	11.46	2.44
10	100000	100	106.30	116.46	18.02
11	1000000	10	420.29	106.59	53.15
12	1000000	100	2365.46	1070.19	813.15

Table 3.2: Mean of elapsed time (in seconds) to compute each algorithm.

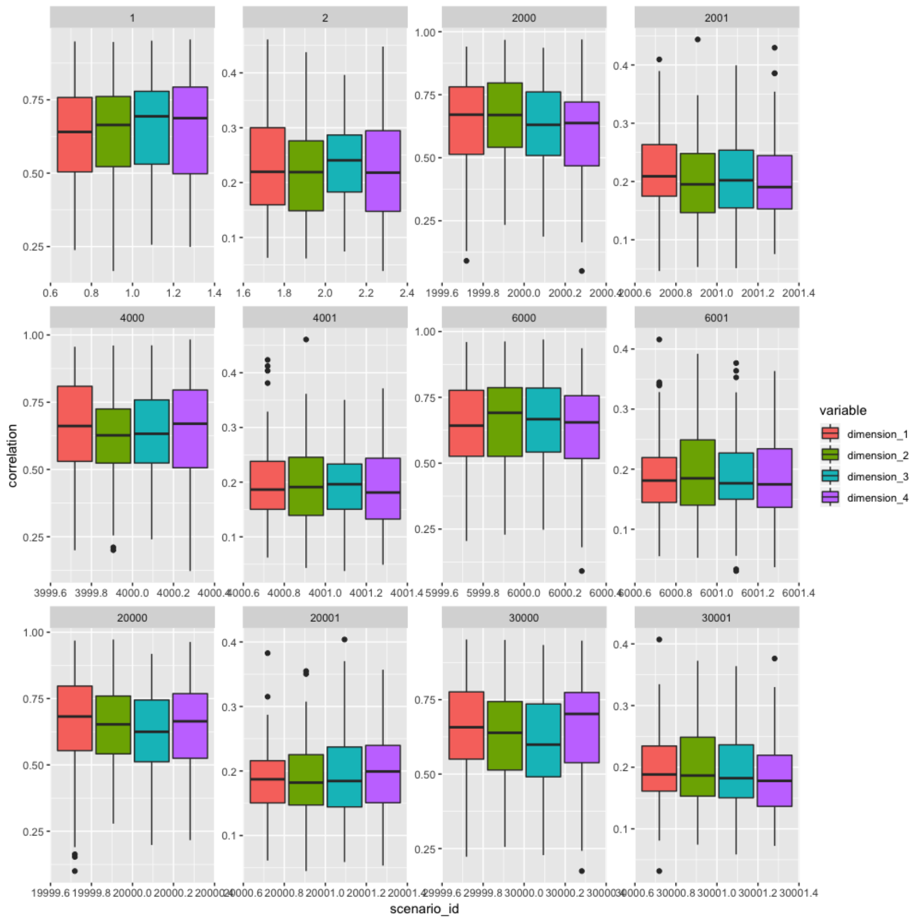


Figure 3.4: Correlation for *noisy scenarios* and MDS Fast algorithm.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
algorithm	2	9283.73	4641.86	32143.99	$< 2e-16$
sample_size	5	108572.93	21714.59	150369.26	$< 2e-16$
n_dimensions	1	12868.36	12868.36	89110.86	$< 2e-16$
Residuals	17991	2598.05	0.14		

Table 3.3: Results for ANOVA test for differences in log(elpased_time).

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-1.4058	0.0085	-165.44	$< 2e-16$
algorithmfast	-0.4313	0.0069	-62.17	$< 2e-16$
algorithmgower	-1.6926	0.0069	-243.96	$< 2e-16$
sample_size3000	0.9473	0.0098	96.54	$< 2e-16$
sample_size5000	1.4434	0.0098	147.10	$< 2e-16$
sample_size10000	2.1505	0.0098	219.17	$< 2e-16$
sample_size1e+05	4.4286	0.0098	451.35	$< 2e-16$
sample_size1e+06	7.2782	0.0098	741.78	$< 2e-16$
n_dimensions100	1.6910	0.0057	298.51	$< 2e-16$

Table 3.4: Linear model for response log(elpased_time).

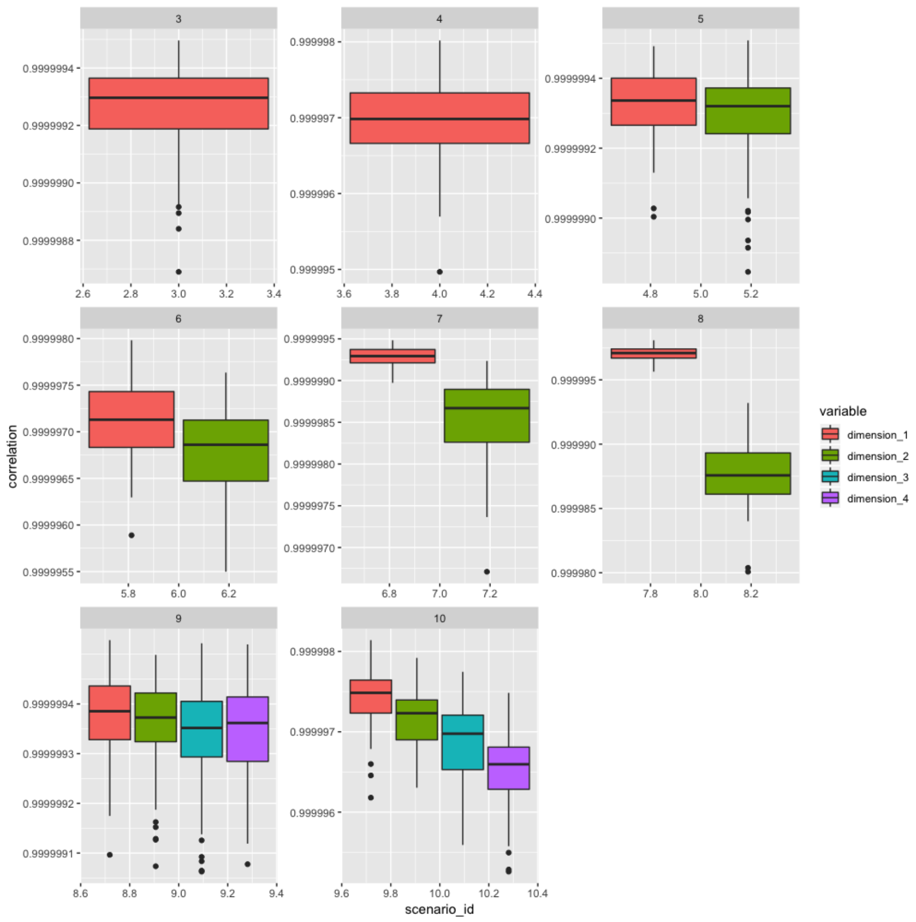


Figure 3.5: Correlation for $n = 1000$ and MDS based on Gower algorithm.

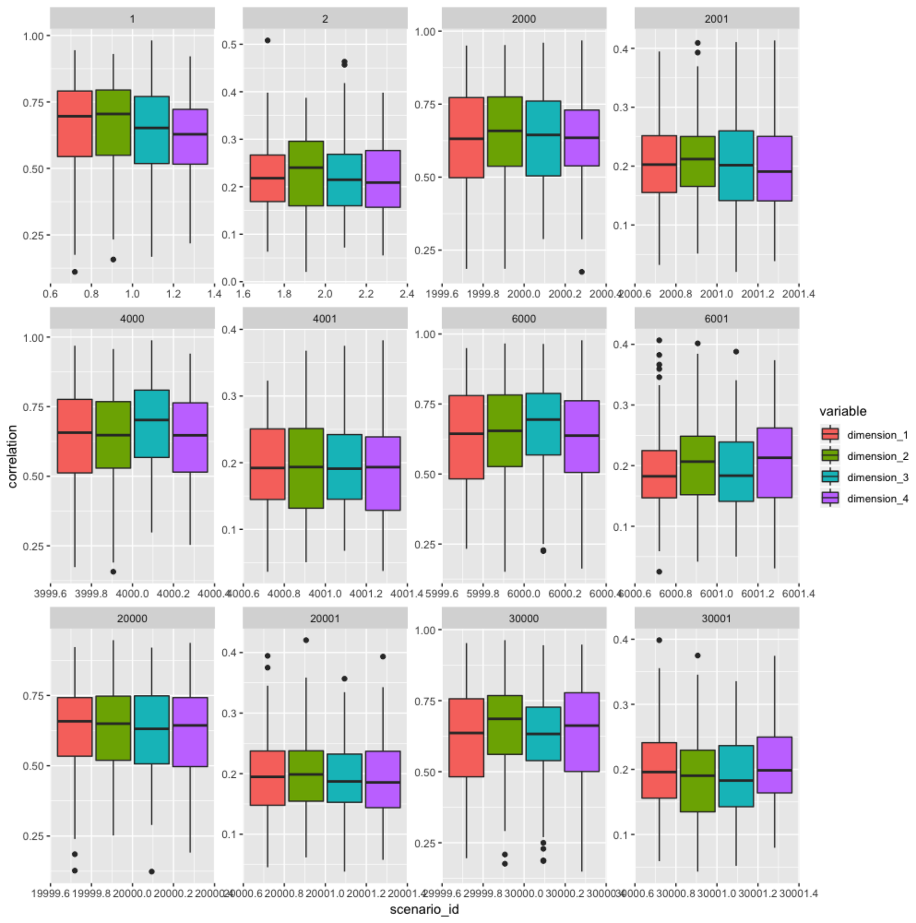


Figure 3.6: Correlation for *noisy scenarios* and MDS based on Gower algorithm.

Chapter 4

Conclusions

Bibliography

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

Gower, J. C. and D. J. Hand (1996). *Biplots* (1st ed ed.). London ; Melbourne : Chapman and Hall. Includes index and bibliographical references.

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

Correlation coeficients boxplots

A.1 Divide and Conquer MDS

A.2 Fast MDS

A.3 MDS based on Gower interpolation

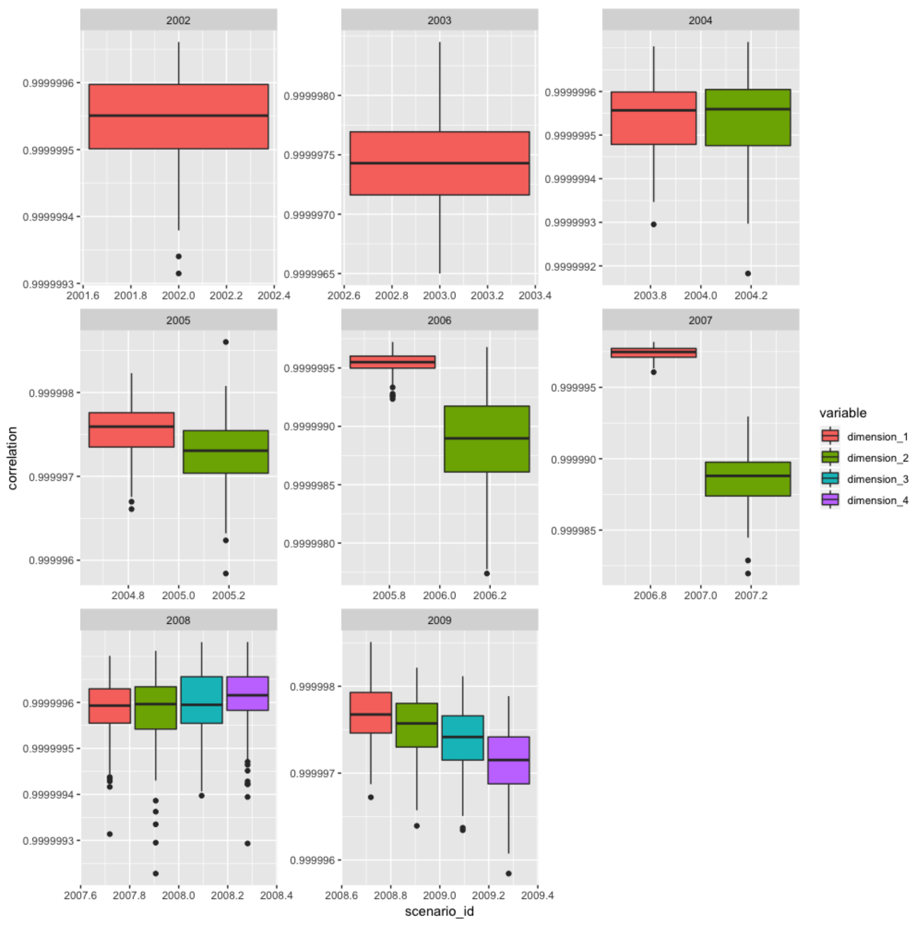


Figure A.1: Correlation for $n = 3000$ and MDS based on Gower interpolation algorithm.

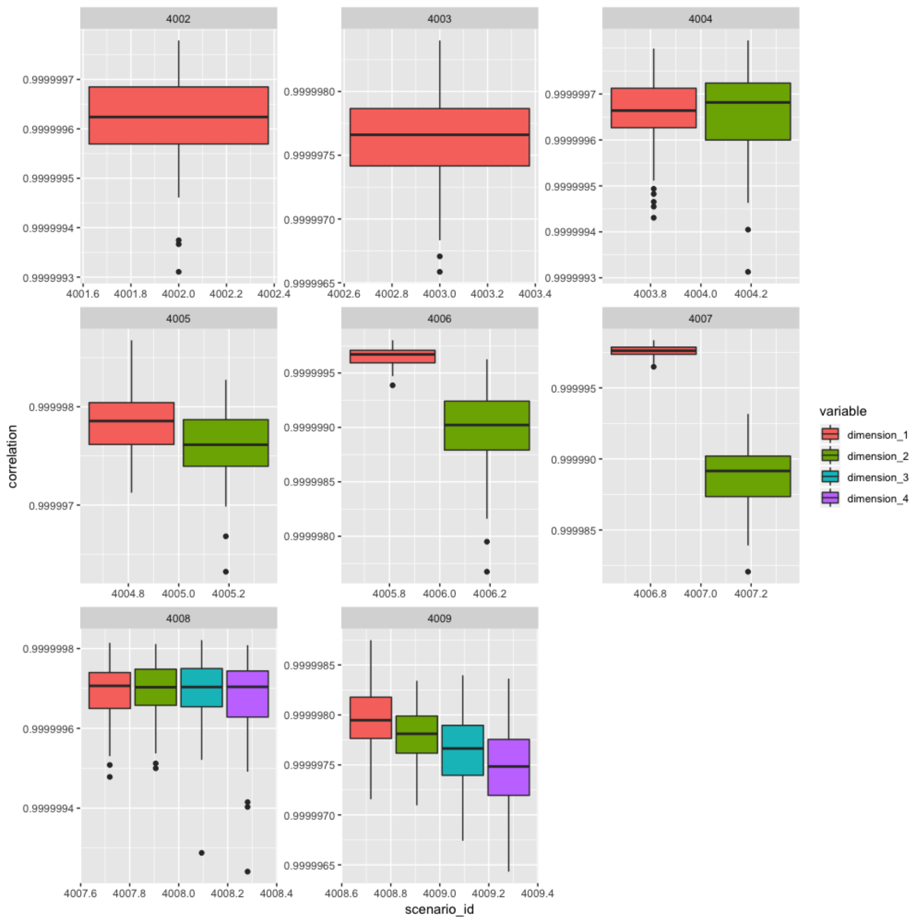


Figure A.2: Correlation for $n = 5000$ and MDS based on Gower interpolation algorithm.

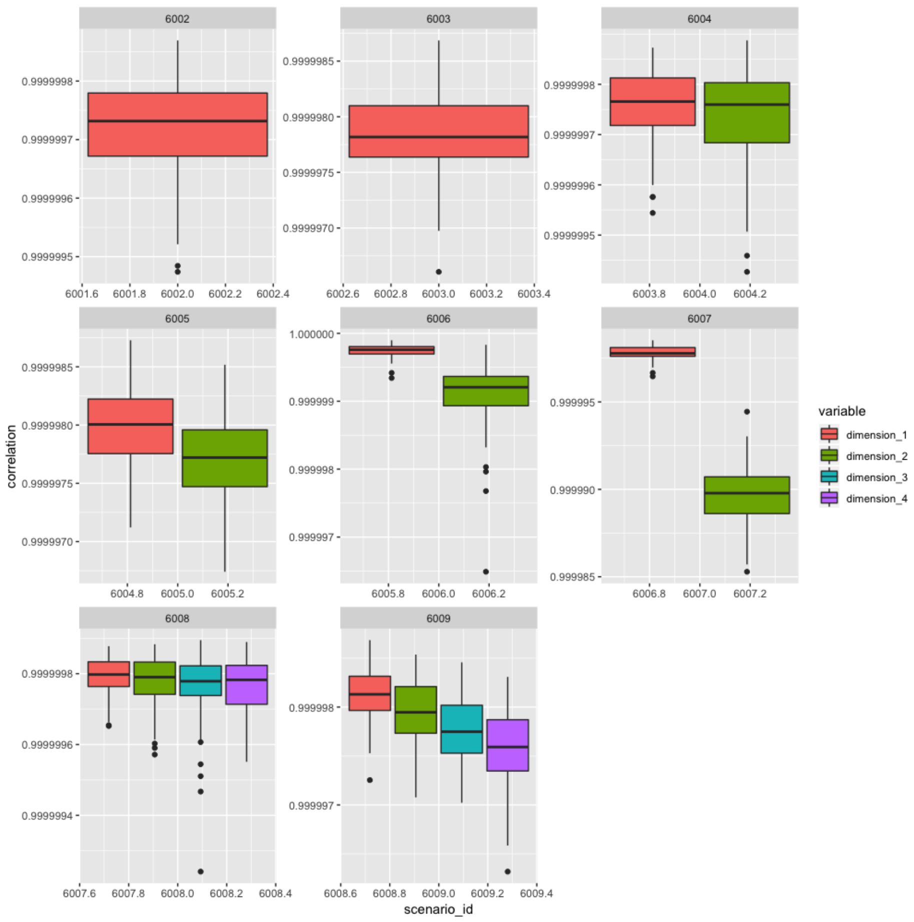


Figure A.3: Correlation for $n = 10000$ and MDS based on Gower interpolation algorithm.

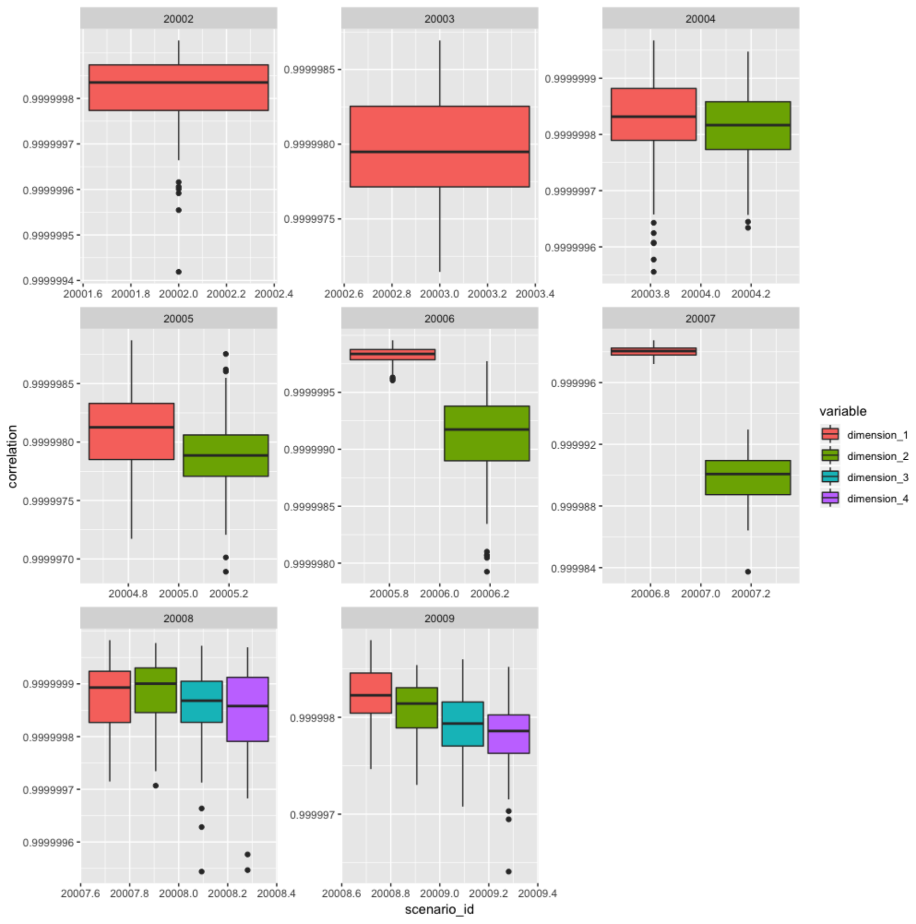


Figure A.4: Correlation for $n = 100000$ and MDS based on Gower interpolation algorithm.

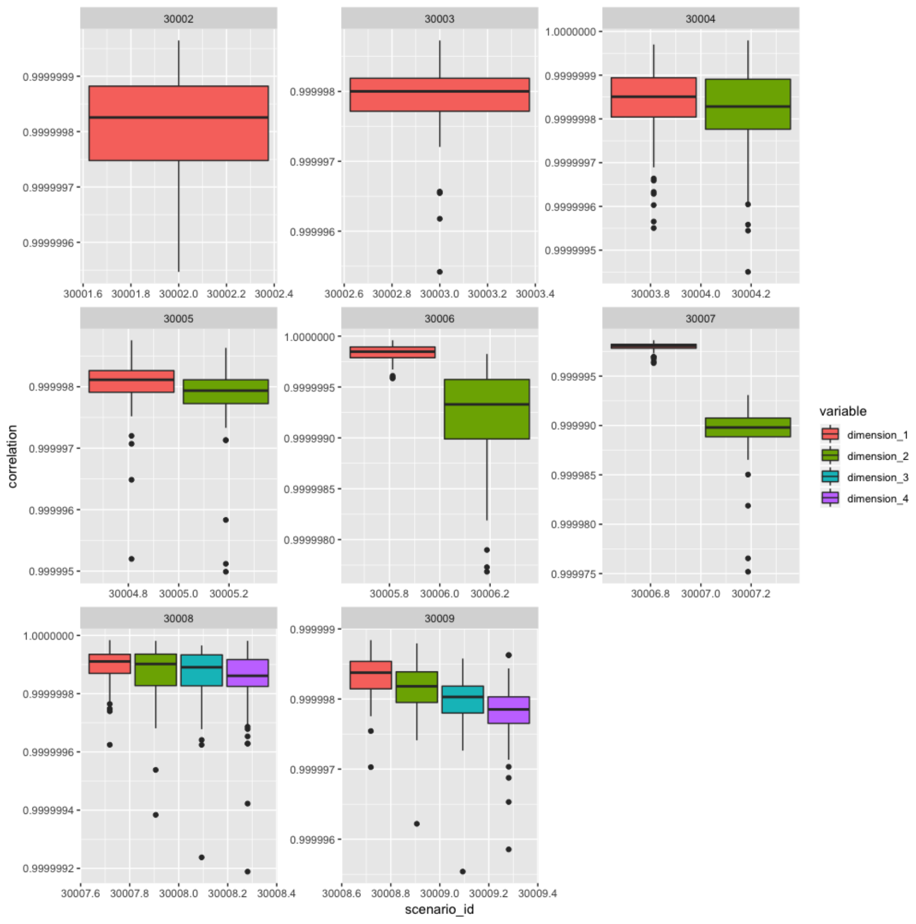


Figure A.5: Correlation for $n = 1000000$ and MDS based on Gower interpolation algorithm.

Appendix B

Code

Appendix C

Problems encountered during the development