Multidimensional Scaling for Big Data

Student: Cristian Pachón García
Advisor: Pedro Delicado Useros

January 11, 2019

What is MDS?

- 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.
- Given a square matrix $\mathbf{D}$ $n \times n$, the goal of MDS is to obtain a configuration matrix $\mathbf{X}$ $n \times p$ satisfying:
  - Columns are orthogonal.
  - The Euclidean distance between the rows of $\mathbf{X}$ is approximately equal to $\mathbf{D}$.

- $\mathbf{X}$ can be interpreted as the matrix of p variables for the n observations.
- The columns of $\mathbf{X}$ are called *principal coordinates*.

Example

Consider the distance between some cities of Europe, as shown in the following matrix:

	Athens	Barcelona	Brussels	Calais	Cherbourg	$\cdots$
Athens	0	3313	2963	3175	3339	$\cdots$
Barcelona	3313	0	1318	1326	1294	$\cdots$
Brussels	2963	1318	0	204	583	$\cdots$
Calais	3175	1326	204	0	460	$\cdots$
Cherbourg	3339	1294	583	460	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

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

Example

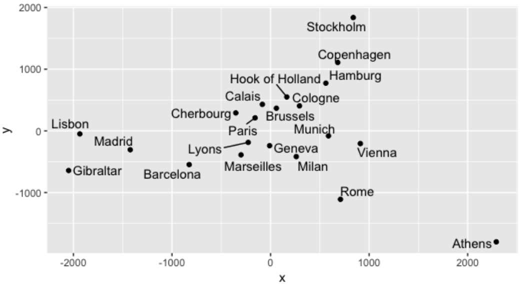

Figure: MDS configuration for European cities

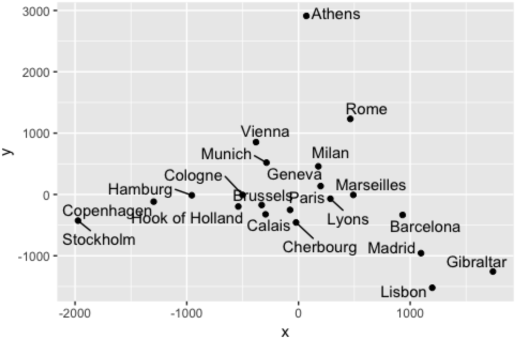

Figure: (Another) MDS configuration for European cities

Given a MDS configuration, any rotation, reflection or translation is a valid MDS configuration, since they preserve the distance. So, the solution is not unique.

Procrustes transformation

- The Procrustes problem is concerned with fitting a configuration (testee) to another (target) as closely as possible.
- Under orthogonal transformations, the testee can be fitted to the target. In addition to such rigid motions, one may also allow for dilations and for shifts.

- Let **A** and **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 **A** and the testee is **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 **T** is an orthogonal matrix.
- In Borg and Groenen (2005) are all the details needed to estimate these parameters.

Why is it needed?

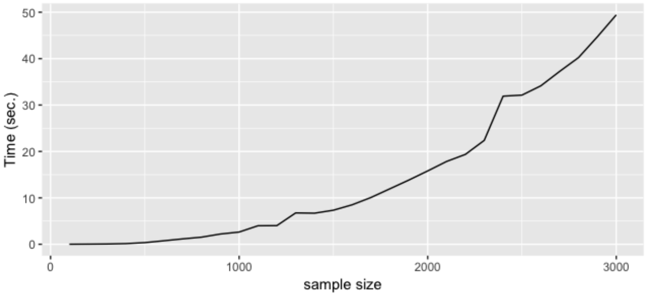

Figure: Elapsed time to compute MDS.

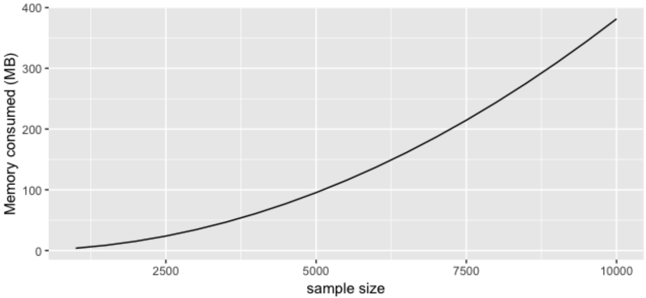

Figure: Memory consumed to compute the distance matrix.

Three algorithms for MDS with Big Data

- *Divide and Conquer MDS:*
  - First approach of this thesis.
- *Fast MDS:*
  - It uses recursive programming.
  - Developed by (Tynia, Jinze, Leonard, and Wei 2006).
- *MDS based on Gower interpolation:*
  - It adds a new set of points to an existing MDS configuration.
  - See Appendix of (Gower and Hand 1995).

Divide and Conquer MDS

- Let p be n/l, where l is the the size of the largest matrix that allows to run MDS efficiently, i.e, in a reasonable amount of time.
- Divide the original dataset into p partitions: $\mathbf{X_1}, \ldots, \mathbf{X_p}$.
- Calculate the MDS for the first partition: **MDS(1)**. This solution will be used as a guide to align the MDS configuration for the remaining partitions.

- Define **cum-mds** equals to **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**: $\textbf{cum-mds}\Big|_{k-1}$.
- Apply Procrustes to obtain the set of parameters that makes $\mathbf{MDS_{k,k-1}}\Big|_{k-1}$ be aligned with $\textbf{cum-mds}\Big|_{k-1}$.
- Use this set of parameters with $\mathbf{MDS_{k,k-1}}\Big|_{k}$
- Once $\mathbf{MDS_{k,k-1}}\Big|_{k}$ is in the same coordinate system as **cum-mds**, add to **cum-mds**.

Fast MDS

- It also partitions the original dataset, $\mathbf{X}$ into p subsets.
- The alignment is done using a set of sample points, instead of the full partition as the previous algorithm.
- For each partition, it is taken $s \cdot k$ points, where s is the estimated dimensionality and k is an amplification factor.
- Therefore, if l is the largest matrix that allows to run MDS efficiently, then $p = \frac{l}{ks}$ and $\frac{n}{p} < l$.

- Let $p = \frac{l}{s \cdot k}$.
- Divide $\mathbf{X}$ into $\mathbf{X_1}, \ldots, \mathbf{X_p}$.
- If $n/p > l$, call (recursively) *Fast MDS*.
- Otherwise, compute MDS over all the partitions.

- Get *sk* points from each $\mathbf{X_i}$ and put them into an alignment matrix $\mathbf{M_{align}}$.
- Run MDS over $\mathbf{M_{align}}$.
- The next step is to compute Procrustes transformation to line these two sets of solutions up in a common coordinate system.

MDS based on Gower interpolation

- Gower interpolation formula (see Appendix of (Gower and Hand 1995)) 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 the new rows, the MDS configuration has $n + m$ rows and c columns.

- Partition **X** into p submatrices. Again, $p = n/l$, being l the the size of the largest distance matrix that a computer can calculate efficiently.
- Compute MDS on the first partition.
- For the each of the remaining partitions, use *Gower interpolation formula* to compute a MDS configuration.

MDS based on Gower interpolation

- We generate a matrix $\mathbf{X}$ with 3 independent *Normal* distributions ($\mu = 0$ and $\sigma = 1$) and 10^3 rows.
- Afterwards, we run the algorithm, obtaining $\mathbf{MDS}_{\mathbf{alg}}$, setting l equals to 500. We require the algorithm to return 3 columns.
- Procrustes is performed over $\mathbf{X}$ and $\mathbf{MDS}_{\mathbf{alg}}$, not allowing dilations.

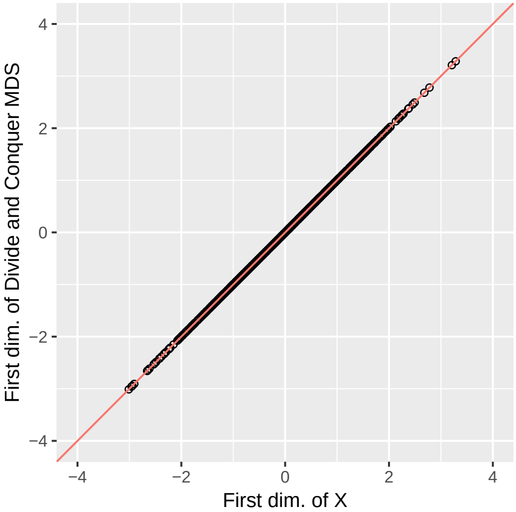

Figure: Dimension 1 $\mathbf{X}$ against dimension 1 of $\mathbf{MDS_{Div}}$. In red, the line $x = y$.

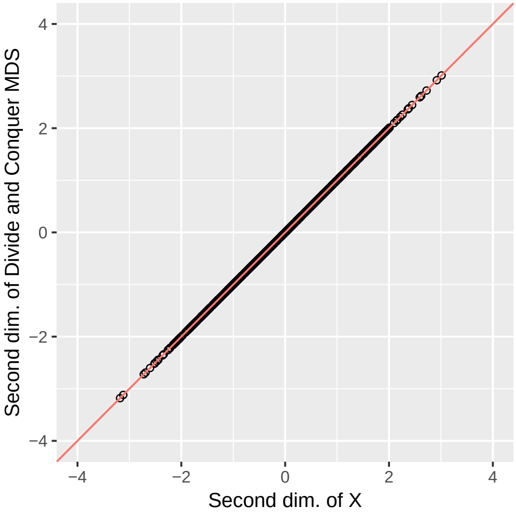

Figure: Dimension 2 **X** against dimension 2 of $\mathbf{MDS_{Div}}$. In red, the line $x = y$.

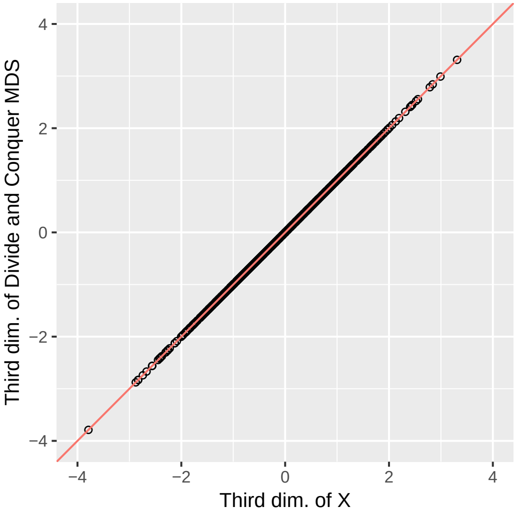

Figure: Dimension 3 of $\mathbf{X}$ against dimension 3 of $\mathbf{MDS_{Div}}$. In red, the line $x = y$.

	X_1	X_2	X_3
MDS_{Div1}	1	0.02	-0.04
MDS_{Div2}	0.02	1	0.02
MDS_{Div3}	-0.04	0.02	1

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

	X_1	X_2	X_3
MDS_{Fast1}	1	0.02	0
MDS_{Fast2}	0.02	1	0.02
MDS_{Fast3}	0	0.02	1

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

	X_1	X_2	X_3
MDS_{Gower1}	1	0	-0.04
MDS_{Gower2}	0	1	-0.0
MDS_{Gower3}	-0.04	-0.03	1

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

Output of the algorithms

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

- A MDS configuration for the initial dataset.
- The second parameter is a list of eigenvalues.

The list of eigenvalues 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 $\text{norm\_eigenvalues}_i$ be list_i / m_i, i.e, each eigenvalue is divided by the number of rows of $\mathbf{D_i}$.
- The algorithms return $\text{norm\_eigenvalues}_1 \cup \cdots \cup \text{norm\_eigenvalues}_p$. We refer to this union as the *normalized eigenvalues*.

Comparison of the algorithms

- *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 Procrustes parameters. However, *Fast MDS* does not use a guide an it uses a set of subsamples to find 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.
- When the number of rows is not large, $p = 1$ and the resulting algorithms are the classical one.

Simulation study

Given the three algorithms, we would like to explore their performance:

- Performance in terms of results quality: are they able to capture the right data dimensionality?
- Performance in terms of time: are they " fast" enough? Which one is the fastest?

Simulation study

- *Sample sizes*: we use different sample sizes, combining small datasets and large ones. A total of six sample sizes are used, which are:
  - Small sample sizes: $10^3, 3 \cdot 10^3, 5 \cdot 10^3$ and 10^4.
  - Large sample sizes: 10^5 and 10^6.

- *Data dimensions*: we generate a matrix with two different number of columns: 10 and 100.
- *Main dimensions*: given $\mathbf{X}$ $n \times k$, where $n \in \{10^3, 3 \cdot 10^3, 5 \cdot 10^3, 10^4, 10^5, 10^6\}$ and $k \in \{10, 100\}$, it is postmultiplied by a diagonal matrix that contains k values, $\lambda_1, \ldots, \lambda_k$.
  - All the columns with the same values of λ: $\lambda_1 = \cdots = \lambda_k = 1$ (*noisy scenarios*).
  - One main dimension with $\lambda_1 = 15$ and $\lambda_2 = \cdots = \lambda_k = 1$.
  - Two main dimensions of the same value λ: $\lambda_1 = \lambda_2 = 15$ and $\lambda_3 = \cdots = \lambda_k = 1$.
  - Two main dimensions of different values λ: $\lambda_1 = 15$, $\lambda_2 = 10$ and $\lambda_3 = \cdots = \lambda_k = 1$.
  - Four main dimensions of the same value λ: $\lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 15$ and $\lambda_5 = \cdots = \lambda_k = 1$.

- As a probabilistic model, we use a Normal distribution with $\mu = 0$ and $\sigma = 1$. With this distribution, we generate a matrix of n observations and k columns, being the k columns independent.

- 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.

1. Generate the dataset $\mathbf{X}$ according to the scenario.
2. For each algorithm, we do the following steps:
    1. Run the algorithm and get MDS configuration for the algorithm ($\mathbf{MDS_{alg}}$).
    2. Get the elapsed time to compute MDS configuration and store it.
    3. Get *normalized eigenvalues* and store them.
    4. Align $\mathbf{MDS_{alg}}$ and $\mathbf{X}$ using (Partition) Procrustes.
    5. Get the correlation coefficients between the main dimensions of $\mathbf{MDS_{alg}}$ and $\mathbf{X}$ and store them.

Important details:

- When running the algorithms, we ask for as many columns as the original data has.
- For Procrustes we dot not allow dilations, otherwise distance could not be preserved.
- To do the alignment, we select the main dimensions. If there is not any main dimension, (*noisy scenarios*), we just select 4 columns.
- Note that the original dataset, $\mathbf{X}$, is always available and it is already the MDS configuration, since we simulate independent columns with mean value equals to 0.

- Performance of results quality:
  - Correlation between the main dimensions of the data and the main dimensions after applying the algorithms. We get the diagonal of the correlation matrix.
  - *Normalized eigenvalues* as an approximation of the standard deviation of the variables of $\mathbf{X}$.
- Elapsed Time to get the MDS configuration.

Correlation coefficients

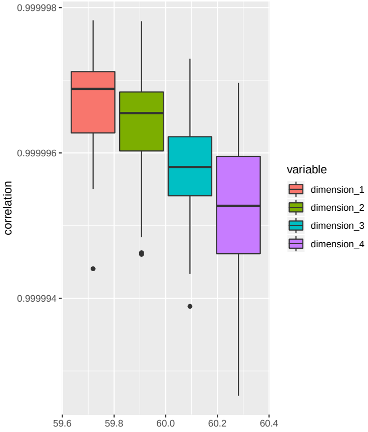

Figure: Correlation coefficients between the main dimensions of **X** and the main dimensions of $\mathbf{MDS_{Div}}$ for a scenario with the following configuration: $n = 10^6$, 100 columns and 4 main dimensions with $\lambda_i = 15$, $i \in \{1, 2, 3, 4\}$.

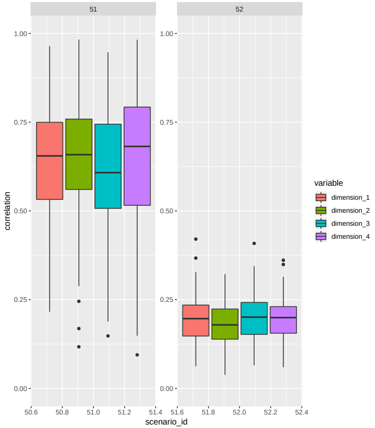

Figure: Correlation coefficients between **X** and $\mathbf{MDS_{Div}}$ for two different *noise scenarios*

Eigenvalues

- Since the original dataset, **X**, is postmultiplied by a diagonal matrix $k \times k$ that contains $\lambda_1, \dots, \lambda_k$, then $\mathrm{var}(X_i) = \lambda_i^2$ and $\mathrm{sd}(X_i) = \lambda_i$.
- Let $\phi_1, \dots, \phi_t$ be the *normalized eigenvalues* of the MDS configuration such that $\phi_1 > \phi_2 > \cdots > \phi_t$. The first highest *normalized eigenvalues* have to verify $\sqrt{\phi_j} \approx \lambda_j$.

$$\widehat{\text{bias}} = \frac{1}{100} \sum_{i=1}^{100} \sqrt{\phi_{ij}} - \lambda_j = \overline{\sqrt{\phi_j}} - \lambda_j.$$

$$\widehat{\text{MSE}} = \frac{1}{100} \sum_{i=1}^{100} (\lambda_j - \sqrt{\phi_{ij}})^2.$$

scenario_id	$\widehat{\sqrt{\phi_1}}$	$\widehat{\text{bias}_1}$	$\widehat{\text{MSE}_1}$
3	14.98	-0.02	0.03
4	15.03	0.03	0.11
13	15.00	-0.00	0.00
14	14.96	-0.04	0.16
23	14.99	-0.01	0.02
24	14.99	-0.01	0.01
33	14.99	-0.01	0.01
34	14.99	-0.01	0.00
43	14.99	-0.01	0.01
44	14.99	-0.01	0.01
53	14.98	-0.02	0.03
54	14.99	-0.01	0.01

Table: Estimator, $\widehat{\text{bias}}$ and $\widehat{\text{MSE}}$ for scenarios with one main dimension $\lambda_1 = 15$ for *Divide and Conquer MDS*.

scenario_id	$\widehat{\sqrt{\phi_1}}$	$\widehat{\text{bias}_1}$	$\widehat{\text{MSE}_1}$
3	14.85	-0.15	2.27
4	15.01	0.01	0.01
13	14.91	-0.09	0.76
14	15.10	0.10	0.93
23	14.96	-0.04	0.14
24	15.03	0.03	0.07
33	14.33	-0.67	44.82
34	15.09	0.09	0.76
43	15.00	-0.00	0.00
44	15.00	0.00	0.00
53	14.86	-0.14	1.88
54	14.90	-0.10	1.02

Table: Estimator, $\widehat{\text{bias}}$ and $\widehat{\text{MSE}}$ for scenarios with one main dimension $\lambda_1 = 15$ for *Fast MDS*.

scenario_id	$\sqrt{\phi_1}$	$\widehat{\text{bias}_1}$	$\widehat{\text{MSE}_1}$
3	15.05	0.05	0.22
4	15.02	0.02	0.04
13	14.94	-0.06	0.36
14	15.04	0.04	0.20
23	14.98	-0.02	0.04
24	15.02	0.02	0.05
33	14.99	-0.01	0.01
34	15.06	0.06	0.31
43	15.04	0.04	0.19
44	14.97	-0.03	0.07
53	14.98	-0.02	0.06
54	14.90	-0.10	1.07

Table: Estimator, $\widehat{\text{bias}}$ and $\widehat{\text{MSE}}$ for scenarios with one main dimension $\lambda_1 = 15$ for *MDS based on Gower interpolation*.

Time to compute MDS

sample_size	n_dim	mean_divide_conquer	mean_fast	mean_gower
10^3	10	0.27	0.14	0.10
10^3	100	0.78	0.69	0.28
$3 \cdot 10^3$	10	0.78	0.32	0.16
$3 \cdot 10^3$	100	2.50	3.14	0.52
$5 \cdot 10^3$	10	1.37	0.54	0.20
$5 \cdot 10^3$	100	4.25	5.69	0.84
10^4	10	2.60	1.81	0.31
10^4	100	8.85	11.79	1.37
10^5	10	28.10	11.46	2.44
10^5	100	106.30	116.46	18.02
10^6	10	420.29	106.59	53.15
10^6	100	2365.46	1070.19	813.15

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

- We do an ANOVA test using three factors:
  - The sample size, which has 6 levels.
  - The number of dimensions, which has 2 levels.
  - The algorithm, which has 3 levels.
- Instead of using the *elapsed time* variable, we use its logarithm.

	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: Results for ANOVA test for differences in log(elapsed_time) using algorithm, sample size and num. dimensions as factors.

	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: Linear model for response log(elapsed_time).

Time to compute MDS

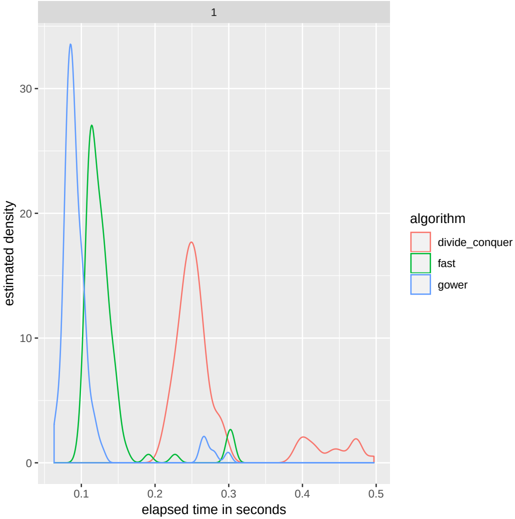

Figure: Estimated density of elapsed time (in sec.) for each algorithm and scenario with $n = 10^3$, 10 columns and $\lambda_i = 1$ $i \in \{1, \dots, 10\}$.

Time to compute MDS

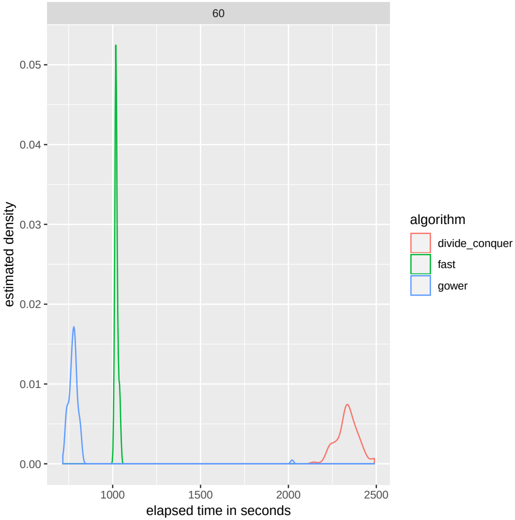

Figure: Estimated density of elapsed time (in sec.) for each algorithm and scenario with $n = 10^6$, 100 columns and $\lambda_i = 15$ $i \in \{1, 2, 3, 4\}$.

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

Gower, J. C. and D. J. Hand (1995).
Biplots, Volume 54.
CRC Press.

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

Thank You