Smacof Meets Newton

Jan de Leeuw - University of California Los Angeles

Started September 02 2023, Version of October 06, 2023

Abstract

We present theory, algorithms, and R code for a version of the **smacof** method for multidimensional scaling that (1) allows for possibly different linear constraints on different dimensions and that (2) uses second derivatives to study and possibly accelerate convergence. Several examples are analyzed and discussed.

Contents

Note: This is a working paper which will be expanded/updated frequently. All suggestions for improvement are welcome. The Rmd file, the pdf, all R files, a LaTeX version and so on are available at https://github.com/deleeuw/mSmacof.

Notation

Conventions

Since we only work in finite dimensional vector spaces, and since our emphasis is on computation, we adopt the following conventions.

- A vector *is* a matrix with one column.
- A row-vector *is* a matrix with one row.
- Derivatives *are* matrices.

Notations

The length of vectors and the dimension of matrices will generally be clear from the context.

- e_i unit vector (element i is one, other elements zero).
- e vector with all elements one.
- E matrix with all elements one.
- 0 real number zero, also vector or matrix with all elements 0.
- I identity matrix.
- $J = I - \frac{ee'}{e'e}$ centering matrix.
- $A \otimes B$ Kronecker product of matrices A and B.

- $X \oplus Y$ direct sum of matrices X and Y.
- $X \times Y$ elementwise (Hadamard) product of matrices X and Y.
- vec(X) matrix X to vector (columns on top of each other).
- vecr(X) elements below diagonal of matrix X to vector (columns on top of each other).
- X' transpose of matrix X.
- X^+ Moore-Penrose inverse of matrix X.
- X^{-T} inverse of the transpose X' (and transpose of the inverse).
- $X \gtrsim Y$ Loewner order of symmetric matrices ($X - Y$ is positive semi-definite).
- $X \lesssim Y$ Loewner order of symmetric matrices ($Y - X$ is positive semi-definite).
- $:=$ definition.
- $X \times Y$ Cartesian product of sets X and Y.
- (x, y) is an ordered pair, i.e. an element of $X \times Y$.
- x_{is} or $\{X\}_{is}$ element (i, s) of matrix X.
- $a_{\bullet s}$ column s of matrix A.
- $a_{i\bullet}$ row i of matrix A.
- $[A]_{is}$ submatrix (i, s) of block-matrix A.
- $A^{[p]}$ direct sum of p copies of matrix A.
- $a^{(k)}$ the k^{th} element of the sequence $\{a\} = a^{(1)}, \cdots, a^{(k)}, \cdots$.
- $\mathbb{R}^n$ space of all vectors of length n.
- $\overline{\mathbb{R}}^n$ space of all centered vectors of length n (i.e. $x'e = 0$).
- $\mathbb{R}^{n\times p}$ space of all $n \times p$ matrices.
- $\overline{\mathbb{R}}^{n\times p}$ space of all column-centered $n \times p$ matrices (i.e. with $X'e = 0$).
- $f : X \Rightarrow Y$ function with arguments in X and values in Y.
- If $f : X \times Y \Rightarrow Z$ then $f(\bullet, y) : X \Rightarrow Z$ and $f(x, \bullet) : Y \Rightarrow Z$.
- $x'y$ inner product in $\mathbb{R}^n$.
- tr $X'Y$ inner product in $\mathbb{R}^{n\times p}$.
- $\|x\| = \sqrt{x'x}$ Euclidean norm of $x \in \mathbb{R}^n$.
- $\|X\| = \sqrt{\text{tr } X'X}$ Euclidean norm of $X \in \mathbb{R}^{n\times p}$.
- $\mathcal{D}f(x)$ derivative of f at x.
- $\mathcal{D}^2 f(x)$ second derivative of f at x.
- $\mathcal{D}_s f(x) = \{\mathcal{D}f(x)\}_s$ partial derivative with respect to x_s at x.
- $\mathcal{D}_{st} f(x) = \{\mathcal{D}^2 f(x)\}_{st}$ second partial with respect to x_s and x_t at x.

1 Introduction

2 The Problem

2.1 General Problem

The most general problem we will study in this report is minimization over all n-element vectors x of a least squares loss function of the form

$$\sigma(\theta) := \frac{1}{2}\sum_{k=1}^{K} w_k(\delta_k - \sqrt{\theta' A_k \theta})^2, \tag{1}$$

where the A_k are arbitrary positive semi-definite matrices. Different types of MDS problems in this framework are characterized by the fact that they have different sequences A_k. We can consequently say that a particular MDS problem is of type $(A_1, \cdots, A_k)$.

In definition (1) the w_k are positive *weights*, the δ_k are non-negative *dissimilarities*, the $d_k(\theta) := \sqrt{\theta' A_k \theta}$ are *distances*, and the objective function is called stress (following Kruskal (1964a), Kruskal (1964b)). For general A_k the $d_k(\theta)$ are just semi-norms, and they may not be actual distances, but we'll stick with the standard MDS terminology using distance anyway.

Minimizing (1) is an unconstrained least squares problem with a non-convex and non-smooth objective function. It can, of course, be tackled with general purpose methods for optimization of functions of this type. For an excellent overview of the various solvers that can be used in R we refer to the task view of Schwendinger and Borchers (1923). In this report, however, we develop special purpose algorithms that depend on the specific properties of our objective function.

First, some notation and terminology that applies to all MDS problems that minimize a function of the form (1). Convenient notation for standard MDS problems was introduced in De Leeuw (1977) and De Leeuw and Heiser (1977). Since then it has been used broadly, for example in the vignettes De Leeuw and Mair (2009) and Mair, Groenen, and De Leeuw (2022), in the review article Groenen and Van de Velden (2016), as well as in the comprehensive textbook of Borg and Groenen (2005), and in the textbook *in statu nascendi* of De Leeuw (2021). We generalize this notation here to the problem of minimizing (1).

We assume, without loss of generality, that

$$\sum_{k=1}^K w_k = 1, \tag{2}$$

$$\sum_{k=1}^K w_k \delta_k^2 = 1. \tag{3}$$

Multiplying the w_k by a positive constant does not change the solution of the minimization problem, only the scale of stress. Multiplying the δ_k by a positive constant changes both the scale of both stress and of the solution of the minimization problem, but the scale change is easily undone after the solution has been found.

Define, following De Leeuw (1977),

$$\rho(\theta) := \sum_{k=1}^K w_k \delta_k \sqrt{\theta' A_k \theta}, \tag{4}$$

$$\eta^2(\theta) := \sum_{k=1}^K w_k \theta' A_k \theta. \tag{5}$$

The function ρ is convex and homogenous of order one. It is not differentiable at those θ for which there is a k such that $w_k \delta_k > 0$ and $\delta_k(\theta) = 0$. The function η^2 is a convex quadratic, homogeneous of order two, and everywhere differentiable.

If

$$V := \sum_{k=1}^{K} w_k A_k \tag{6}$$

then $\eta^2(\theta) = \theta' V \theta$. Also define the matrix-valued function $B(\theta)$ by

$$B(\theta) := \sum_{k=1}^{K} w_k r_k(\theta) A_k, \tag{7}$$

with

$$r_k(\theta) := \begin{cases} \frac{\delta_k}{d_k(\theta)} & \text{if } d_k(\theta) > 0, \\ 0 & \text{if } d_k(\theta) = 0. \end{cases} \tag{8}$$

The $r_k(\theta)$ can be interpreted as residuals, where a residual is larger the further it deviates from one. Note that V is a constant matrix, but B is a matrix-valued function.

Now

$$\sigma(\theta) = \frac{1}{2}\left\{1 - 2\,\rho(\theta) + \eta^2(\theta)\right\} = \frac{1}{2}\left\{1 - 2\,\theta' B(\theta)\theta + \theta' V \theta\right\}. \tag{9}$$

Thus σ is a difference of two convex functions, also known as a DC-function. Clearly MDS problems of the general type we discuss here are also characterized by their B-function and their V-matrix.

2.2 Basic MDS

2.2.1 Configuration Space

In Metric, Euclidean, Least Squares MDS we minimize

$$\sigma(X) := \frac{1}{2} \sum_{1 \leq j < i \leq n} \sum w_{ij}(\delta_{ij} - d_{ij}(X))^2, \tag{10}$$

over *configurations* X in $\mathbb{R}^{n \times p}$, the space of all $n \times p$ matrices. The weights $W = \{wij\}$ are assumed to be non-negative, so they can be zero (for missing dissimilarities or for unfolding, for example). The $d_{ij}(X)$ are Euclidean distances between the n *objects*, represented numerically by the n rows of X and geometrically by the n corresponding *points* in $\mathbb{R}^p$. Thus

$$d_{ij}(X) := \sqrt{\sum_{s=1}^{p} (x_{is} - x_{js})^2}. \tag{11}$$

In addition we assume that W is *irreducible*, which means there is no permutation P such that $P'WP$ is the direct sum of two smaller matrices. In other words: irreducibility means the n objects cannot be partitioned into groups in such a way that all between-group weights are zero. If W is reducible the MDS problem decomposes into two or more smaller MDS problems, so assuming irreducibility causes no loss of generality (De Leeuw (1977)).

Define for all $j > i$ the matrices

$$E_{ij} := (e_i - e_j)(e_i - e_j)', \tag{12}$$

where e_i is the i^{th} unit vector, i.e. the vector of length n with all elements equal to zero, except for element i, which is one. The E_{ij} are symmetric, positive semi-definite, of rank one, and doubly-centered, i.e. with rows and columns adding up to zero. Using E_{ij} we can define distance by $d_{ij}(X) = \sqrt{\text{tr } X'E_{ij}X}$.

Now define $x = \text{vec}(X)$ and

$$A_{ij} := I_p \otimes E_{ij} = \underbrace{E_{ij} \oplus \cdots \oplus E_{ij}}_{p \text{ times}}$$

Here I_p is the identity matrix of order p. Thus A_{ij} is block-diagonal with p copies of E_{ij} in the diagonal blocks. The distance as function of x is $d_{ij}(x) := \sqrt{x'A_{ij}x}$. We also define for $s = 1, \cdots, p$

$$V_s := \sum_{1 \leq j < i \leq n} \sum w_{ij} E_{ij}, \tag{13}$$

$$B_s(x) := \sum_{1 \leq j < i \leq n} \sum w_{ij} r_{ij}(x) E_{ij}\cdot, \tag{14}$$

with $r_{ij}(x)$ defined as in (8).

For basic MDS all V_s are the same, and so are all $B_s(X)$. Thus the block-diagonal matrices

$$V := V_1 \oplus \cdots \oplus V_p, \tag{15}$$

$$B(x) := B_1(x) \oplus \cdots \oplus B_p(x), \tag{16}$$

have equal blocks.

Because the E_{ij} are symmetric, doubly-centered, block-diagonal, and positive semi-definite, so are the non-negative linear combinations V_s and the $B_s(X)$. By irreducibility each of the p identical blocks V_s has rank $p(n-1)$, with only the vectors proportional to e, the vector with all elements equal to one, in its null space (De Leeuw (1977)). In the important special case that $w_{ij} = 1$ for all $j < i$ the V_s are $nI - ee' = nJ$, with J the centering matrix.

With this new notation stress becomes

$$\sigma(x) := \frac{1}{2} \sum_{k=1}^{K} w_k (\delta_k - \sqrt{x'A_k x})^2 = \frac{1}{2} \left\{ 1 - 2\text{tr } x'B(x)x + x'Vx \right\}. \tag{17}$$

We have now written stress as a function of the np-vector x, and we have gotten rid of the double indexing by given each of the pairs (i, j) with $j > i$ and $w_{ij} > 0$ a number from $\mathcal{K} = \{1, \cdots, K\}$, with different pairs getting different numbers. Clearly (17) is of the form (1).

2.2.2 Full Basis

First we introduce Y_s as p matrices with dimension $n \times (n-1)$, with the columns of each of the Y_s forming a basis for the subspace of all centered vectors in $\mathbb{R}^n$. Column s of X will be written as $Y_s\theta_s$, so that X, and consequently stress, becomes a function of the θ_s.

The Y_s need not be the same, but we assume, without loss of generality, that $Y_s'V_sY_s = I$ for all s. If V_s has eigen-decomposition $V_s = K\Lambda^2K'$, with Λ^2 the diagonal matrix with the $n-1$ non-zero eigenvalues, then the singular value decompositions of the Y_s are $Y_s = K\Lambda^{-1}L_s'$. Thus different choices of Y_s differ by at most a rotation. Also $Y_sY_s' = V_s^+$, the Moore-Penrose inverse of V_s.

Define

$$Y := Y_1 \oplus \cdots \oplus Y_p, \tag{18}$$

and

$$\theta := \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_p \end{bmatrix}. \tag{19}$$

Then $\text{vec}(X) = Y\theta$. If

$$B_s(\theta) := Y_s'B_s(x)Y_s \tag{20}$$

with

$$d_{ij}(\theta) = \sqrt{\theta'Y'A_{ij}Y\theta}, \tag{21}$$

with the same provision for $d_{ij}(\theta) = 0$.

We can now recast stress as a function of $\theta \in \mathbb{R}^{p(n-1)}$.

$$\sigma(\theta) = \frac{1}{2}\left\{1 - 2\theta'B(\theta)\theta + \theta'\theta\right\}. \tag{22}$$

Of course if $x = \text{vec}(X) = Y\theta$ then $\sigma(\theta)$ from (22) is equal to $\sigma(x)$ from … and to $\sigma(X)$ from (17).

Our MDS task has been transformed into from minimization of stress over $\mathbb{R}^{n\times p}$ to minimization over *coefficient space* $\theta \in \mathbb{R}^m$. If we are done we can transform back to configuration space with $x_{\bullet s} = Y_s\theta_s$, or equivalently $\text{vec}(X) = Y\theta$.

Equivalent to $X = Y\Theta$.

2.2.3 Reduced Basis

Working in $\mathbb{S}^n$ gets rid of the translational indeterminacy in the MDS problem, but not the rotational indeterminacy.

Alternatively we can choose the Y_s in such a way that they each span the $n-s$-dimensional space of column-centered vectors with their first $s-1$ elements are zero. Thus the Y_s are dfferent and the θ_s have length $n-1, \cdots, n-p$. This defines the *reduced basis*. Using this reduced basis eliminates both translational and rotational indeterminacy from the original MDS problem. The reduced basis has dimension $np - \sum_{s=1}^p s = np - \frac{1}{2}p(p+1)$.

2.3 DCDD

In the terminology of Takane, Kiers, and De Leeuw (1995) choosing different bases for different s is called DCDD (Different sets of Constraints on Different Dimensions). Although it is not within

the scope of this report, DCDD also makes it possible to incorporate the various linear constraints on the configuration discussed in detail in De Leeuw and Heiser (1980). A proper choice of bases can be used to cover fitting simplexes, circumplexes, unique variances, and the rectangles of Borg and Leutner (1983).

We give one example. Basic scaling with unique variances (Bentler and Weeks (1978)) has p standard dimensions, for which we can use the full basis, and n additional dimensions equal to the e_i, reprensenting uniquenesses. Thus there are $p+n>n$ dimensions, and $p+n$ matrices Y_s. The first p are $n\times(n-1)$, the last n are the one-dimensional vectors e_i. There are problems with this parametrization, very much like the factor indeterminacy problem, but the algorithm does not care and picks one of the solutions.

2.4 Matrix Basis

An important special case of DCDD imposes the constraint

$$X=\sum_{v=1}^{r}\theta_v G_v, \tag{23}$$

where the G_s are $n\times p$ matrices. To see that this is indeed a special case of DCDD define Y_s as the matrix collecting the s^{th} columns of all G_v. Thus there are r of these $n\times r$ matrices Y_s. Now $\vec{(}X)=Y\theta$, with Y the direct sum of the Y_s, as usual, and

$$\theta=\left.\begin{bmatrix}\theta\\ \vdots\\ \theta\end{bmatrix}\right\}r\text{ times}. \tag{24}$$

The distinguishing DCDD characteristic in using the *matrix basis* (23) is that all r pieces of θ in (24) must be equal.

One important application of the matrix basis is finding the optimal step size in iterative procedures, or, more generally, finding optimal weights in multistep procedures. For the steepest descent method, for example, we choose $G_1=X$ and $G_2=\nabla\sigma(X)$.

$$\{C_{ij}\}_{vw}=\text{tr }G_v'A_{ij}G_w$$

Thus

$$\{V_s\}_{vw}=\text{tr }G_v'V_0G_w$$
$$\{B_s(\theta)\}_{vw}=\text{tr }G_v'B_0(\theta)G_w$$

3 Properties of Stress

It is convenient, following De Leeuw (1977), to define

$$\rho(X):=\sum_{1\leq j<i\leq n}\sum w_{ij}\delta_{ij}d_{ij}(X)=\text{tr }X'B(X)X, \tag{25}$$

and

$$\eta^2(X) := \sum_{1\leq j<i\leq n}\sum w_{ij}d_{ij}^2(X) = \text{tr } X'VX, \tag{26}$$

Both ρ and η^2 are convex functions on configuration space, which means that

$$\sigma(X) = \frac{1}{2}\left\{1 - 2\rho(X) + \eta^2(X)\right\} \tag{27}$$

is a difference of two convex functions (a.k.a. a DC-function).

$$\sigma(\lambda\theta) = \frac{1}{2}\left\{1 - 2|\lambda|\rho(\theta) + \lambda^2\eta^2(\theta)\right\}$$

This shows that stress is a convex quadratic on any ray emanating from the origin. On the ray $\lambda\theta$ the unique minimum is attained at $\lambda = \rho(\theta)/eta^2(\theta)$ and the minimum is

$$\min_{\lambda\geq 0}\sigma(\lambda\theta) = \frac{1}{2}\left\{1 - \frac{\rho^2(\theta)}{\eta^2(\theta)}\right\}$$

At a local minimum differentiable

At a local minimum norm

One local maximum

4 Derivatives

The relevant general result for going from $x \in \mathbb{R}^{np}$ to $\theta \in \mathbb{R}^m$ is that if $g(\theta) := f(Y\theta)$ then $\mathcal{D}g(\theta) = Y'\mathcal{D}f(x)$ and $\mathcal{D}^2g(\theta) = Y'\mathcal{D}^2f(x)Y$, where $x = Y\theta$.

4.1 First

$$\mathcal{D}_s d_{ij}(\theta) = \frac{1}{d_{ij}(\theta)}Y_s'A_{ij}Y_s\theta_s, \tag{28}$$

thus

$$\mathcal{D}\sigma(\theta) = Y'(V - B^{[p]}(\theta))Y\theta = Y'(V - B^{[p]}(\theta))x. \tag{29}$$

Since $Y'VY = I$ we can also write this as

$$\mathcal{D}\sigma(\theta) = \theta - Y'B^{[p]}(\theta))Y\theta. \tag{30}$$

Thus the stationary equation $\mathcal{D}\sigma(\theta) = 0$ says that $Y'B^{[p]}(\theta))Y\theta = \theta$, i.e. $Y'B^{[p]}(\theta))Y$ has an eigenvalue equal to one, with eigenvector θ.

4.2 Second

Differentiating equation (28) again gives

$$\mathcal{D}_{st}d_{ij}(\theta)=\delta^{st}\frac{1}{d_{ij}(\theta)}Y_s'A_{ij}Y_s-\frac{1}{d_{ij}^3(\theta)}Y_s'A_{ij}Y_s\theta_s\theta_t'Y_t'A_{ij}Y_t. \tag{31}$$

Using

$$A_{ij}Y_s\theta_s=(e_i-e_j)(e_i-e_j)'x_s=(e_i-e_j)(x_{is}-x_{js}), \tag{32}$$

we have

$$\mathcal{D}_{st}d_{ij}(\theta)=\delta^{st}\frac{1}{d_{ij}(\theta)}Y_s'A_{ij}Y_s-\frac{1}{d_{ij}^3(\theta)}(x_{is}-x_{js})(x_{it}-x_{jt})Y_s'A_{ij}Y_t, \tag{33}$$

or

$$\mathcal{D}^2\sigma(\theta)=Y'\left\{\delta^{st}(V^{[p]}-B^{[p]}(\theta))+H(\theta)\right\}Y, \tag{34}$$

where $H(\theta)$ is a partitioned matrix with

$$[H]_{st}(\theta)=\sum_{1\leq j<i\leq n}\sum w_{ij}\frac{\delta_{ij}}{d_{ij}^3(\theta)}(x_{is}-x_{js})(x_{it}-x_{jt})A_{ij}. \tag{35}$$

Using the Loewner order, we see that

$$B^{[p]}(\theta)\gtrsim H(\theta)\gtrsim 0, \tag{36}$$

which implies

$$\mathcal{D}^2\sigma(\theta)\lesssim I. \tag{37}$$

Thus all eigenvalues of the Hessian are less than or equal to one. This does not mean that the Hessian is bounded, because it can have arbitrarily large negative eigenvalues. At a local minimum all eigenvalues are non-negative, at an isolated local minimum they are all positive. If the Hessian at the stationary value has a negative eigenvalue, then it is a saddle point. If the smallest eigenvalue is zero, then the stationary value can be either a local minimum or a saddle point, depending on the values of the higher derivatives. Stress has no local maxima, except at the origin $\theta=0$ where its value is one.

Note that if we use the full basis then the Hessian has $\frac{1}{2}p(p-1)$ zero eigenvalues. Although we have eliminated translational indeterminacy, we still have rotational indeterminacy. The eigenvectors corresponding with the zero eigenvalues can be constructed from θ by interchanging a pair θ_s and θ_t and changing the sign of one of the two (De Leeuw (1988)). The reduced basis generally leads to a non-singular Hessian. Both for the full and reduced basis the Hessian has one eigenvalue equal to one (its largest eigenvalue), corresponding with eigenvector θ.

4.3 Third

Why look at third derivatives ? One reason is that if we have a solution with gradient equal to zero and Hessian both positive semi-definite and singular then we do not know if this solution is a saddle

point or local minimum. The third derivatives can help us in deciding this. A second reason could be, although we do not exploit this possibility here, that third derivatives could be used in some form of cubic regularization to stabilize the **newton** method (Nesterov and Polyak (2006)).

In order to not get involved right away in large scale manipulation of indices we look at the more general problem, where

$$d(\theta) = \sqrt{\theta' A \theta}, \tag{38}$$

with A any positive semi-definite matrix. In **smacof** A is one of the $Y' A_{ij}^{[p]} Y$.

Now

$$\mathcal{D}_s d(\theta) = \frac{1}{d(\theta)} \theta' a_{s\bullet}, \tag{39}$$

and

$$\mathcal{D}_{st} d(\theta) = \frac{1}{d(\theta)} a_{st} - \frac{1}{d^3(\theta)} (\theta' a_{s\bullet} \times \theta' a_{t\bullet}) \tag{40}$$

and

$$\mathcal{D}_{stu} d(\theta) = \frac{1}{d^3(\theta)} (\theta' a_{s\bullet} \times \theta' a_{t\bullet} \times \theta' a_{u\bullet}) - \frac{1}{d^5(\theta)} (a_{st} \times \theta' a_{u\bullet} + a_{su} \times \theta' a_{t\bullet} + a_{tu} \times \theta' a_{s\bullet}) \tag{41}$$

Of course the quadratic part of stress has third derivative equal to zero. Thus

$$\mathcal{D}_{stu} \sigma(\theta) = -\sum_{1 \leq j < i \leq n} \sum w_{ij} \delta_{ij} \mathcal{D}_{stu} d_{ij}(\theta) \tag{42}$$

Combined with equation (41) this gives us enough information to compute the third partials, although directly using the rows and elements of $Y' A_{ij}^{[p]} Y$ is computationally horribly inefficient. The R function *deriv123()* in *mSmacof.R* computes the three-dimensional array of partials using this inefficient technique. It will be improved over time.

5 Convergence

6 Algorithms

In this report we calculate with $\theta \in \mathbb{R}^m$, and not with $x \in \mathbb{R}^{np}$ or with $X \in \mathbb{R}^{n \times p}$.

6.1 Smacof

Following De Leeuw and Heiser (1980) we define the *Guttman transform* of θ as

$$\Gamma(\theta) := Y' B^{[p]}(\theta) Y \theta. \tag{43}$$

Then

$$\sigma(\theta) = 1 - \theta' \Gamma(\theta) + \frac{1}{2} \theta' \theta, \tag{44}$$

which we can also write, by completing the square and using the Euclidean norm, as

$$\sigma(\theta) = 1 + \frac{1}{2}\|\theta - \Gamma(\theta))\|^2 - \frac{1}{2}\|\Gamma(\theta)\|^2, \tag{45}$$

Now suppose θ and ξ are two vectors in $\mathbb{R}^m$ and that $d_{ij}(\xi) > 0$. Then, by Cauchy-Schwartz,

$$d_{ij}(\theta) \geq \frac{1}{d_{ij}(\xi)}\theta' Y' A_{ij}^{[p]} Y \xi, \tag{46}$$

and thus, using weighted summation on both sides of equation (46),

$$\theta'\Gamma(\theta) \geq \theta'\Gamma(\xi). \tag{47}$$

This implies that if we define

$$\tau(\theta, \xi) := 1 + \frac{1}{2}\|\theta - \Gamma(\xi)\|^2 - \frac{1}{2}\|\Gamma(\xi)\|^2 \tag{48}$$

then

$$\sigma(\theta) \leq \tau(\theta, \xi) \qquad \forall (\theta, \xi) \in \mathbb{R}^m \times \mathbb{R}^m, \tag{49}$$

$$\sigma(\theta) = \tau(\theta, \theta) \qquad \forall \theta \in \mathbb{R}^m. \tag{50}$$

Thus $\tau : \mathbb{R}^m \times \mathbb{R}^m \Rightarrow \mathbb{R}^+$ is a *majorization function* for $\sigma : \mathbb{R}^m \Rightarrow \mathbb{R}^+$ in the sense of De Leeuw (1977). It follows that

$$\sigma(\Gamma(\theta)) \leq \tau(\Gamma(\theta), \theta) \leq \tau(\theta, \theta) = \sigma(\theta). \tag{51}$$

In the majorization literature (51) is called the *sandwich inequality*. Alternatively, we could call it the *MM chain*, because the first inequality follows from majorization, and the second from minimization (see Lange (2016), for an excellent general overview of theory and applications of MM algorithms). We have equality in the second inequality if and only if $\theta = \Gamma(\theta)$. A necessary and sufficient conditions for equality in the first inequality is equality in all Cauchy-Schwartz inequalities (46) with $w_{ij}\delta_{ij} > 0$, which is the case if and only if for those index pairs

$$\frac{x_{i\bullet} - x_{j\bullet}}{d_{ij}(X)} = \pm\frac{z_{i\bullet} - z_{j\bullet}}{d_{ij}(X)}, \tag{52}$$

where $\text{vec}(X) = Y\theta$ and $\text{vec}(Z) = Y\xi$. It is clearly sufficient for equality that θ is proportional to ξ.

The iterations of the **smacof** algorithm are

$$\theta_s^{(k+1)} = \Gamma(\theta^{(k)}). \tag{53}$$

If Y_s has rank $n-1$, which means there are no constraints, then $Y_s Y_s'$ is equal to the Moore-Penrose inverse V^+, and thus (53) becomes

$$x_{\bullet s}^{(k+1)} = V^+ B(\theta^{(k)}) x_{\bullet s}^{(k)}, \tag{54}$$

which is the usual **smacof** update.

Guttman (1968) already observed that we can also write (53) as a gradient algorithm with unit step size.

$$\theta_s^{(k+1)} = \theta^{(k)} - \mathcal{D}\sigma(\theta^{(k)}). \tag{55}$$

As pointed out by De Leeuw (1977) a slightly more elegant points of view is to write

$$\theta_s^{(k+1)} = \theta^{(k)} - \gamma^{(k)}, \tag{56}$$

where $\gamma^{(k)} \in \partial\sigma(\theta^{(k)})$, the subgradient of σ at $\theta^{(k)}$. This makes **smacof** a subgradient algorithm with unit step size.

Using subgradients instead of gradients makes it unnecessary to single out the case where $d_{ij}(\theta) = 0$ for one or more index pairs, for example in the (14), because even if the distance is zero the subgradient continues to exist. Computationally the gradient and subgradient formulations are basically the same, because if $w_{ij}\delta_{ij} > 0$ for all pairs then stress is differentiable at a local minimum (De Leeuw (1984)).

The usual qualitative and quantitative convergence analysis (De Leeuw and Heiser (1980), De Leeuw (1988)) also applies to our **smacof** inn $\mathbb{R}^m$. First the qualitative results.

1. $\{\sigma(\theta^{(k)})\}$ is a boundex decreasing sequence converging to, say, σ_∞.
2. The sequence $\{\theta^{(k)}\}$ is bounded and has one or more accumulation points.
3. Accumulation points are fixed points of the Guttman transform, i.e. $\theta = \Gamma(\theta)$.
4. All accumulation points have stress value σ_∞.
5. The sequence $\{\theta^{(k)}\}$ is asymptotically regular, i.e. the sequence $\{\|\theta^{(k)} - \theta^{(k-1)}\}$ converges to zero.
6. Either there is a single accumulation point, and the sequence $\{\theta^{(k)}\}$ converges, or the accumulation points form a continuum (a set which is not the union of disjoint closed sets).

The main quantitative convergence result for **smacof** is that the sequence $\{\mu^{(k)}\}$ of empirical convergence rates

$$\mu^{(k)} := \frac{\|\theta^{(k+1)} - \theta^{(k)}\|}{\|\theta^{(k)} - \theta^{(k-1)}\|}, \tag{57}$$

converges to the largest eigenvalue of $\mathcal{D}\Gamma$ at the solution, with

$$\mathcal{D}\Gamma(\theta) = Y'(B^{[p]}(\theta) - H(\theta))Y. \tag{58}$$

It follows that $\mathcal{D}\Gamma(\theta) \gtrsim 0$, with one eigenvalue equal to zero, corresponding with the eigenvector θ. We assume in this section that σ is two times differentiable. De Leeuw (1984) shows that if $w_{ij}\delta_{ij} > 0$ for all $j < i$ then this is always the case near a local minimum.

Thisa is also the place to say something about *full-dimensional smacof*. It is defined simply as **smacof** with $p = n - 1$. If we define

$$C := \sum_{s=1}^{p} Y_s\theta_s\theta_s'Y_s', \tag{59}$$

then we can write

$$\sigma(C) := 1 - \sum_{1\leq j<i\leq n}\sum w_{ij}\delta_{ij}\sqrt{\text{tr } A_{ij}C} + \text{tr } C.$$

By ... C has rank p, and **smacof** is equivalent to minimizing stress over all C with rank$(C) \leq p$. Now stress is defined on the set $\mathbb{C}_p$ of all positive semi-definite matrices of rank less than or equal to p. Moreover $\sqrt{\text{tr } A_{ij}C}$ is concave on $\mathbb{C}_p$ and tr C is linear, and thus stress is convex on $\mathbb{C}_p$. Now $\mathbb{C}_p$ is a rather nasty set, except when $p = n-1$, in which case it is the cone of positive semi-definite matrices. Thus full-dimensional scaling minimizes a convex function over a convex set, and is consequently a convex programming problem in which all local minima are global. In fact stress is strictly convex and the global minimum is unique.

The necessary and sufficient conditions for C to be a solution of the full-dimensional **smacof** are

$$C \geq 0, \tag{60}$$

$$V - B(C) \geq 0, \tag{61}$$

$$\text{tr } C(V - B(C)) = 0. \tag{62}$$

It is easy to see that θ is a solution of the **smacof** problem on $\mathbb{R}^m$ with $p = n - 1$ if and only C defined by (59) solves the full-dimensional problem on $\mathbb{C}_p$. A **smacof** solution in any dimension p is the full-dimensional solution if $V^+B(\theta) \lesssim 1$, i.e. if the p unit eigenvalues of $B(\theta)$ at the solution are actually the largest ones.

Empirically we find that the optimal C usually has rank strictly less than $p - 1$. The rank of the optimal C is called the *Gower rank* of the dissimilarity matrix (or, more precisely, the Gower rank of the pair (Δ, W)). One plausible alternative method of metric multidimensional scaling in dimension p is computing the full-dimensional solution X and then using its first p principal components as the p-dimensional solution. This is similar to classical scaling and may provide at least a good (although rather expensive) initial estimate. Full-dimensional scaling has also been used as a method to find the global minimum of stress for $p < n - 1$ (De Leeuw (2019)).

6.2 Relax !

De Leeuw and Heiser (1980) were the first to propose a technique to accelerate convergence of the **smacof** iterations. They considered the family of updates

$$\Gamma_\alpha(\theta) := (1 + \alpha)\Gamma(\theta) - \alpha\theta \tag{63}$$

For $\alpha > 0$ this over-relaxes and for $\alpha < 0$ it under-relaxes the Guttman transform. For $\alpha = 0$ we recover the Guttman transform and for $\alpha = -1$ we have the identity transform. By (48)

$$\eta(\Gamma_\alpha(\theta), \theta) = 1 + \frac{1}{2}\alpha^2\|\theta - \Gamma(\theta)\|^2 - \frac{1}{2}\|\Gamma(\theta)\|^2 \tag{64}$$

Thus if $-1 < \alpha < +1$ and $\theta \neq \Gamma(\theta)$ we have $\eta(\Gamma_\alpha(\theta), \theta) < \sigma(\theta)$. The sandwich inequality still applies, and we have global convergence of the stable algorithm

$$\theta^{(k+1)} = \Gamma_\alpha(\theta^{(k)}). \tag{65}$$

For **smacof**, with either basis, the largest eigenvalue of $\mathcal{D}^2\sigma(\theta)$ is $0 < \lambda < 1$ and the smallest eigenvalue is zero. The convergence rate of the relaxed update (65) is

$$\lambda(\alpha) := \max(|(1+\alpha)\lambda - \alpha|, |\alpha|). \tag{66}$$

The minimum of (66) is attained at $\lambda/(2-\lambda)$ and is also equal to $\lambda/(2-\lambda)$. Note that for $0 < \lambda < 1$

$$0 < \frac{\lambda}{2-\lambda} < \lambda < 1. \tag{67}$$

If $\lambda = 1 - \epsilon$, with ϵ a small positive number, then

$$\frac{\lambda}{2-\lambda} = \frac{1-\epsilon}{1+\epsilon} \approx (1-\epsilon)^2 = \lambda^2 \tag{68}$$

and thus the rate of convergence is approximately squared and the number of iterations to attain a given precision is approximately halved.

This derivation assume we know the largest eigenvalue of $\mathcal{D}\Gamma(\theta)$ at the solution from the start, which of course we don't. Thus we use an adaptive scheme, where λ is approximated by the empirical convergence rate at iteration k. This still produces the desired improvement (De Leeuw (2006)).

The original analysis in De Leeuw and Heiser (1980) suggested, perhaps, to define a related update as $2\Gamma(\theta) - \theta$. The reasoning was to go as far as possible in the right direction, while maintaining global convergence. The choice $\alpha = 1$ was implemented in some early versions of the **smacof** program. It is not a good choice, however. For this relaxed update (64) shows $\eta(\Gamma_\alpha(\theta), \theta) = \eta(\theta, \theta)$. The sandwich inequality partially breaks down and we do not have a decrease of stress if $\Gamma_\alpha(\theta)$ is proportional to θ. As a consequence the sequence $\{\theta^{(k)}\}$ of solutions has multiple accumulation points and does not converge. The change of θ from one iteration to the next does not converge to zero. The reported minimum value of stress will not be the correct value. Two simple changes correct this problem with $\alpha = 1$. The first is to normalize $\Gamma_1(\theta)$ in each iteration. The second is to define the update as $\Gamma(\Gamma_1(\theta))$. See De Leeuw (2006) for details. In this report we prefer the adaptive strategy that estimates λ by using the empirical convergence rate.

6.3 Newton

$$\theta^{(k+1)} = \theta^{(k)} - \{\}^{-1}(\theta^{(k)} - \Gamma(\theta^{(k)}))$$

Safeguard ?

6.4 Combines

The arguments *eps3* and *strategy* regulate the trade-off between **smacof** and **newton** iterations. Remember that **smacof** generates a decreasing and converging sequence of stress values. The vector sequence of θ values converges to a local minimum (from almost all starting points). Convergence is linear and can be very slow. The sequence of stress values generated by the **newton** method, on the other hand, may not be monotone and may not converge. The corresponding θ sequence may

not converge either. But if started close enough to a local minimum convergence is quadratic and very fast.

newton updates are computationally much more expensive than **smacof** updates.

We will try out three combinations of **smacof** and **newton** in this report, chosen by setting the *eps3* and *strategy* parameters.

1. The first strategy is to start with **smacof** iterations (the burn-in) before switching to **newton**. We switch when $\|\theta^{(k)} - \theta^{(k-1)}\|$ is less than some small ϵ. This strategy may switch back to **smacof**, but eventually it will be **newton**. If ϵ is zero, we never switch to **newton** and it's smacof all the way, when ϵ is large we switch to **newton** in the first iteration and it's **newton** all the way.
2. The second strategy is to see if a **newton** iteration lowers the current best function value. If it does, keep it. If it does not, use the **smacof** iteration.
3. The third strategy updates using either the **smacof** update or the **newton** update, depending on which has the lowest function value.

The second and third strategies produce a decreasing sequence of stress values. In all three strategies eventually **newton** wil take over. But "eventually" make take a long time. In the second and third strategies we compute both the smacof and newton updates in each step. This is expensive. The first strategy may produce a non-monotone sequence of stress values, but it will tend to be much faster because with the default settings the percentage of **newton** iterations will be small.

6.5 Matrix Based

Generalizes relax.

7 Software

7.1 General

IO in configuration (X) space, calcultate in θ space.

triangular storage

extended precision

Computationally we also use the fact that any symmetric matrix of the form

$$R = \sum_{1\leq j<i\leq n}\sum r_{ij}A_{ij} \tag{69}$$

can be computed by setting the off-diagonal elements of R equal to $-r_{ij}$ and then filling the diagonal elements such that rows and columns sum to zero. No multiplications are required. Although the (very sparse) matrices A_{ij} appear in the definitions of V, $B_s(\theta)$, and $H_{st}(\theta)$, they do not appear in the actual computations.

7.2 mSmacof

The arguments for the *mSmacof* function (with their defaults) are

```
mSmacof <- function(delta,
                    w = oneDist(attr(delta, "Size")),
                    p = 2L,
                    xold = NULL,
                    basis = "B",
                    itmax = 100000L,
                    relax = TRUE,
                    strategy = 1L,
                    eps1 = 15L,
                    eps2 = 10L,
                    eps3 = 15L,
                    verbose = FALSE
)
```

Let's go over these briefly. Arguments *delta* and *w* are obvious, and *xold* is the starting configuration for the iterations. Previous results have shown that the most effective method for avoiding non-global local minima is to start with as good an initial estimate as possible. We use the classical MDS solution from Torgerson (1958) and Gower (1966). The basis we choose is a list of matrices Y, one Y_s for each of the p dimensions. Default is the full basis, with all Y_s the same matrix spanning $\overline{\mathbb{R}}^n$.

itmax is the maximum number of iterations, and *eps1* and *eps2* are cut-offs to decide when to stop iterating. We stop when the change in stress value from one iteration to the next is less than 10^{-eps1} and when the change in θ from one iteration to the next, measured by the least squares norm of the difference, is less that 10^{-eps2}.

relax allows us to choose between the regular **smacof** update and the *relaxed update* (De Leeuw and Heiser (1980)).

If *verbose* is TRUE *mSmacof* print intermediate results for each iteration (iteration counter, stress, stress improvement in this iteration, the norm of the change in θ for this iteration, and the ratio of this change to the previous change in θ).

The return value of *mSmacof* is a list with fifteen elements

```
    return(
      list(
        xmat = xnew,
        bmat = bmat,
        dmat = dnew,
        itel = itel,
        loss = snew,
        thet = tnew,
        grax = grax,
        grat = grat,
```

```
        hesx = hesx,
        hest = hest,
        evlt = evlt,
        evlx = evlx,
        evlb = evlb,
        erat = erat,
        trat = trat
      )
    )
```

xmat is the configuration matrix X, *bmat* is $B(X)$, and *dmat* is the matrix $d(X)$. *itel* is the number of iterations until convergence, or *itmax* if there is no convergence. *loss* is the minimum value stress obtained, and *thet* is θ as a list of p vectors. *gradx* is the gradient at X as a matrix and *gradt* the gradient at θ as llist of vectors, *hessx* is the hessian at X as a partioned matrix and *hesst* the Hessian at θ as a partitioned matrix. Partitioned matrices are lists of lists, *evlt* and *evlx* are the eigenvalues of these two Hessians, and *evlb* are the eigenvalues of $V^+B(\theta)$. The *erat* is the observed convergence rate from equation (57) and *trat* is the theoretical convergence rate, the largest non-trivial eigenvalue of $I - \mathcal{D}^2\sigma(\theta) = B(\theta) - H(\theta)$. For a convergent **smacof** sequence *erat* and *trat* will be equal, for a convergent **newton** sequence *erat* will rapidly converge to zero.

Note that at a solution the matrix $V^+B(\theta)$ will have p eigenvalues equal to one. If these are the largest eigenvalues, then the Gower rank is p, and we have found the global minimum of stress (De Leeuw (2014)). In MDS practice, however, a Gower rank of p is exceedingly rare, and the p unit eigenvalues are usually among the smaller eigenvalues of $V^+B(\theta)$.

7.3 vSmacof

vSmacof minimizes stress using the matrix basis

7.4 sSmacof

In sSmacof we combine mSmacof and vSmacof steps.

8 Examples

In this section we give the results of running *mSmacof*, with the various strategies, on seven small to medium examples. All runs use the classical Torgerson-Gower MDS solution as the initial estimate, with a multiplicative scaling to minimize stress (Malone, Tarazaga, and Trosset (2002)). All weights are equal to one.

Iteration control parameters of the program are set to the default, which means *eps1* = 15 and *eps2* = 10, and *itmax* = 1,000,000. This, of course, is requiring a ridiculous precision for almost all MDS applications I am aware of. But comparing the various strategies, and deciding on the nature of a stationary point, is easier if we use enhanced precision. Moreover, it is a well-known claim (Kearsley, Tapia, and Trosset (1994)) that the standard **smacof** program "stops too soon". This

probably means that the R version from De Leeuw and Mair (2009), with the default stopping criteria, may stop in an area where stress is merely flat, and possibly still far from a local minimumn. By requiring high precision we prevent this from happening in our examples (at the cost of many more iterations, of course).

To compare the full and the reduced basis we use both bases for every strategy. Default is the full basis, the runs with the reduced basis are coded by adding an *a* at the end of their names.

We run strategy 1 (and strategy 1a) with *eps3* equal to *eps1*. This means **smacof** all the way. strategy1 and strategy1a are also run with *eps3* equal to -1, which means **newton** all the way. In order to prevent computing 1,000,000 **newton** iterations we set *itmax* equal to 250 for **newton** and **newtona**. For all examples we do run **strategy2** and **strategy2a**, as well as **strategy3** and **strategy3a**, as well as **strategy1** with *eps3* equal to 2, 4, and 6.

The table of results for each example is generated automatically by the R program *exampleRun()* in the appendix. It shows *itel*, the number of iterations, *stress*, the loss at the stopping point, *emprate*, the empirical rate of convergence, *theorate*, the theoretical rate, *minhess*, the smallest eigenvalue of the Hessian, and *time*. The time is computed with the microbenchmark package (Mersmann (2023)). Microbenchmark is made to repeat each run eleven times and report the median running time as *time*. For the other strategies, that generally end with **newton** iterations, the *emprate* should be close to zero. For the strategies that use the full basis the *minhess* should be zero, for the reduced basis the *minhess* should be positive (for a local minimum). For *smacofa* the *minhess* and *emprate* add up to one. Using the full basis runs the risk of sublinear convergence, because of zero or negative eigenvalues of the Hessian.

8.1 Trosset/Mathar

In Trosset and Mathar (1997) the authors set themselves the task of finding a non-global local minimum in MDS (and the task of showing that the standard **smacof** interpretation stops too soon and that consequently **newton** is to be preferred). Their two examples both have $n = 4$.

```
trosmat <- as.dist(matrix(c(0,1,sqrt(2),1,1,0,1,sqrt(2),sqrt(2),1,0,1,1,sqrt(2),1,0),4,
dlmat <- as.dist(1 - diag(4))
xtria <- rbind(0, apply(matrix(c(0,1,1/2,0,0,sqrt(3)/2), 3, 2), 2, function(x) x - mean
xsqua <- matrix(c(-1,1,1,-1,-1,-1,1,1), 4, 2) # correct square
xstar <- matrix(c(-1,1,-1,1,-1,-1,1,1), 4, 2) # incorrect square
```

The first example, also analyzed by De Leeuw (1988), has all six dissimilarities equal to one. This is fitted perfectly by a regular tetrahedron, and consequently the Gower rank is three. It is generally assumed that the global minimizer in two dimensions is a square, with the four points in the four corners. In De Leeuw (1988) a second stationary point was found, with three points in the vertices of an equilateral triangle and the fourth point in the centroid of the triangle. De Leeuw claimed that this stationary point was a non-isolated local minimum, which is a confusing claim because all local minima are non-isolated because of translation and rotation. The reason for the confusion is that the triangle + centroid solution has a positive-semidefinite Hessian with a zero non-trivial eigenvalue. Trosset and Mathar (1997) point out that this means the solution fails the second derivative test, and by finding a descent direction they actually show we have found a saddle point.

8.1.1 Results

For the Trosset/Mathar first example, with all dissimilarities equal to one, the Torgerson initial configuration turns out to be the triangle+centroid, which is a stationary point. Thus all 13 strategies stop after the first iteration.

```
h <- mSmacof(dlmat, basis = "A")
mPrint(h$evlt, digits = 6)
```

```
## [1] +1.000000 +0.767949 +0.500000 +0.000000 +0.000000
```

```
mPrint(h$evlx, digits = 6)
```

```
## [1] +0.666667 +0.511966 +0.511966 +0.000000 +0.000000 +0.000000 +0.000000
## [8] -0.000000
```

```
htrosmat <- exampleRun(trosmat)
```

8.1.1.1 Smacof Meets Newton

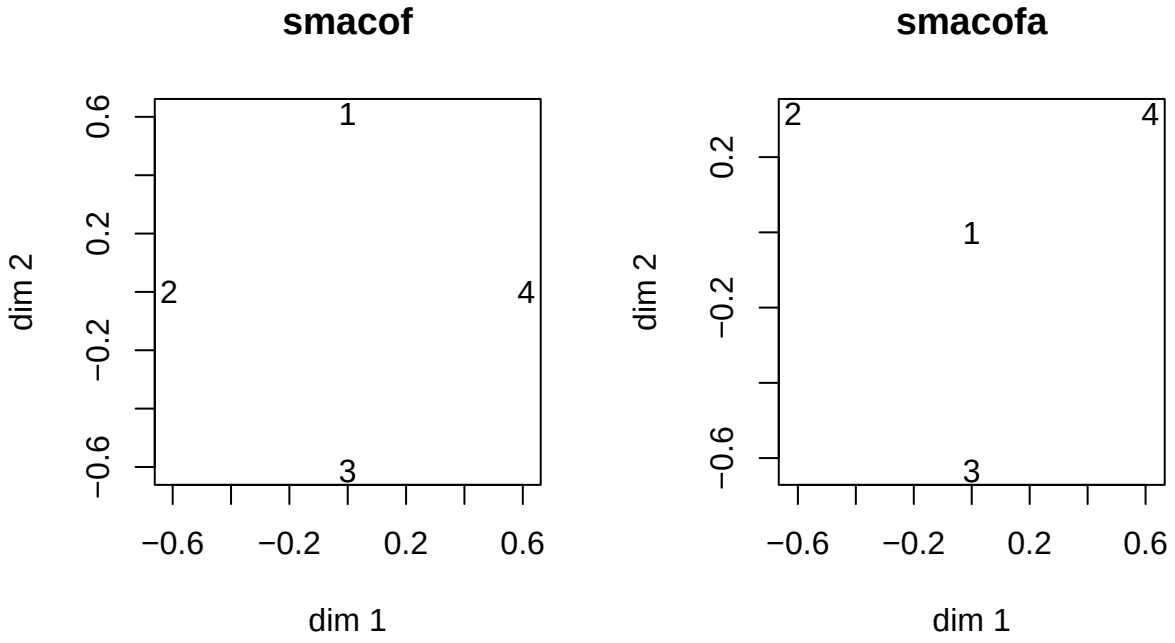
```
##               itel          stress         emprate          minhess     time
## smacof           2 3.184204175e-32 6.001718319e-16 -2.220446049e-16 1.689446
## smacofrelax      2 3.184204175e-32 6.001718319e-16 -2.220446049e-16 1.650496
## smacofa         28 4.597380272e-02 4.521024692e-01 -1.790436694e-01 2.673528
## smacofarelax    21 4.597380272e-02 3.426513990e-01 -1.790436694e-01 2.405183
## newton           2 4.724948130e-32 1.073606511e-15  4.440892099e-16 1.909944
## newtona          5 4.597380272e-02 4.822331845e-08 -1.790436694e-01 2.809402
## strategy2        2 3.184204175e-32 6.001718319e-16 -2.220446049e-16 1.907484
## strategy2a       5 4.597380272e-02 2.897930955e-08 -1.790436694e-01 2.797840
## strategy3        2 3.184204175e-32 6.001718319e-16 -2.220446049e-16 1.896168
## strategy3a       5 4.597380272e-02 2.897930955e-08 -1.790436694e-01 2.803211
## strategy1-2      2 3.184204175e-32 6.001718319e-16 -2.220446049e-16 1.635121
## strategy1-4      2 3.184204175e-32 6.001718319e-16 -2.220446049e-16 1.675670
## strategy1-6      2 3.184204175e-32 6.001718319e-16 -2.220446049e-16 1.649799
```

```
hstrosmat <- exampleRun2(trosmat)
print(hstrosmat$results, digits = 10)
```

8.1.1.2 Smacof Quickstep

```
##                            itel          stress     time
## otmax = 1,    nterms = 1    1 1.129878901e-32 0.284089
## otmax = 5,    nterms = 1    1 1.129878901e-32 0.282982
## otmax = 10,   nterms = 1    1 1.129878901e-32 0.281424
## otmax = 100,  nterms = 1    1 1.129878901e-32 0.282367
## otmax = 1,    nterms = 2    1 1.325039802e-31 0.324638
```

```
## otmax = 5,    nterms = 2    1 1.325039802e-31 0.321522
## otmax = 10,   nterms = 2    1 1.325039802e-31 0.325212
## otmax = 100,  nterms = 2    1 1.325039802e-31 0.320743
## otmax = 1,    nterms = 3    1 2.157041538e-32 0.367032
## otmax = 5,    nterms = 3    1 2.157041538e-32 0.361415
## otmax = 10,   nterms = 3    1 2.157041538e-32 0.363834
## otmax = 100,  nterms = 3    1 2.157041538e-32 0.362522
## otmax = 1,    nterms = 4    1 6.162975822e-33 0.403727
## otmax = 5,    nterms = 4    1 6.162975822e-33 0.405039
## otmax = 10,   nterms = 4    1 6.162975822e-33 0.401267
## otmax = 100,  nterms = 4    1 6.162975822e-33 0.400693
```



8.2 Ekman

We start with the canonical MDS example: average similarity judgments between 14 colors from Ekman (1954).

```
data(ekman, package = "smacof")
ekman <- (1 - ekman) ^ 3
hekman <- exampleRun(ekman)
```

Note that we use power three to compute dissimilarities from the similarities, because previous research has shown that for these transformed dissimilarities **smacof** in two dimensions computes the global minimum (De Leeuw (2019)). The ekman example is definitely atypical in this respect.

8.2.1 Results

```
print(hekman$results, digits = 10)
```

8.2.1.1 Smacof Meets Newton

```
##              itel         stress         emprate          minhess      time
## smacof         32 0.005512405968 5.357762267e-01  1.998401444e-15  3.772164
## smacofrelax    24 0.005512405968 4.124112181e-01  1.554312234e-15  3.154376
## smacofa       581 0.005512405968 9.645817777e-01  3.541831207e-02 33.523814
## smacofarelax  315 0.005512405968 9.335771478e-01  3.541831208e-02 19.714317
## newton          5 0.005512405968 4.879580532e-07  1.998401444e-15  4.095490
## newtona        10 0.020611324154 2.607932558e-05 -1.737420981e-02  6.258117
## strategy2       7 0.005512405968 1.114331611e-06  1.776356839e-15  4.996096
## strategy2a     76 0.005512405968 4.712405280e-07  3.541831208e-02 39.738102
## strategy3       6 0.005512405968 6.515656134e-07  1.998401444e-15  4.554034
## strategy3a     77 0.005512405968 2.216251231e-07  3.541831208e-02 40.248060
## strategy1-2     6 0.005512405968 1.937624704e-07  2.664535259e-15  3.751418
## strategy1-4    12 0.005512405968 1.482112273e-07  1.554312234e-15  3.623129
## strategy1-6    19 0.005512405968 1.276362317e-06  1.776356839e-15  3.600169
```

```
hsekman <- exampleRun2(ekman)
print(hsekman$results, digits = 10)
```

8.2.1.2 Smacof Quickstep

```
##                           itel         stress     time
## otmax = 1,    nterms = 1    5 0.005512428024 0.894743
## otmax = 5,    nterms = 1    5 0.005512408969 1.095766
## otmax = 10,   nterms = 1    5 0.005512408969 1.089534
## otmax = 100,  nterms = 1    5 0.005512408969 1.252345
## otmax = 1,    nterms = 2    4 0.005512409453 0.950872
## otmax = 5,    nterms = 2    3 0.005512414745 0.919753
## otmax = 10,   nterms = 2    3 0.005512414745 0.944066
## otmax = 100,  nterms = 2    3 0.005512414745 0.916637
## otmax = 1,    nterms = 3    4 0.005512406024 1.156610
## otmax = 5,    nterms = 3    3 0.005512406849 1.114298
## otmax = 10,   nterms = 3    3 0.005512406849 1.114052
## otmax = 100,  nterms = 3    3 0.005512406849 1.113191
## otmax = 1,    nterms = 4    3 0.005512406470 1.106180
## otmax = 5,    nterms = 4    3 0.005512406125 1.246195
## otmax = 10,   nterms = 4    3 0.005512406125 1.398428
## otmax = 100,  nterms = 4    3 0.005512406125 1.569603
```

8.2.2 Discussion

This small example has a good fit in two dimensions. It can be characterized as very well-behaved and very easy. All eight strategies give the same solution, which we shall show to be the global minimum. relaxed **smacof** takes the least time to converge. Both **newton**, **strategy 2**, and **strategy 3** are quite competative, being less than a second slower.

The eigenvalues of $V^+B(\theta)$ at the **smacof** solution are

```
##  [1] +1.000000000023987 +0.999999999953508 +0.923497086335441 +0.907901212921802
##  [5] +0.862936584809353 +0.852692003044618 +0.829803620826543 +0.814556167660674
##  [9] +0.793238576326464 +0.791651722426952 +0.786442678063926 +0.747679475652772
## [13] +0.728268247392202 +0.000000000000001
```

The fact that the two largest eigenvalues are equal to one verifies that the Gower rank is two, and that the solution is indeed the two-dimensional global minimum (De Leeuw (2016)). Remember this is for the dissimilarities $\delta_{ij} = (1 - s_{ij})^3$. Powering dissimilarities generally decreases the Gower rank (De Leeuw (2023)).

The reduced basis leads to many more iterations, and a much more time-consuming run, compared to the full basis. It seems that **smacof** likes the additional freedom of the full basis, and does not like to be forced into a fixed rotation. We will find this throughout our examples, so there is no need to point this out every time.

8.3 Equal Distances

The next example has 15 objects. All dissimilarities are equal.

```
equi <- oneDist(15)
hequi <- exampleRun(equi)
```

This is a wordt-case example in most respects. The MDS problem is known to have many local minima, mainly because permuting the points of any local minimizer gives other local minimizers with the same stress. Given previous research (De Leeuw and Stoop (1984)) we think that the global minimum is attained with twelve points equally spaced on a circle and the remaining three points as corners of an equilateral triangle in the interior of the circle. Which of the 15 points ultimately occupy each of the fifteen possible positions on the circle and triangle is obviously arbitrary.

8.3.1 Results

8.3.1.1 Smacof Meets Newton

```
##                 itel          stress           emprate            minhess
## smacof        455929 0.0656151291922 9.99987233515e-01  2.99335468054e-16
## smacofrelax   297936 0.0656151291922 9.99957175521e-01  2.79627942445e-15
## smacofa      1000000 0.0656151299341 1.00000081107e+00 -8.12969537114e-07
## smacofarelax 1000000 0.0656151293147 9.99997800055e-01  9.41461513281e-07
## newton            37 0.0670111879007 9.86462255373e-07 -1.66991062323e-01
## newtona          222 0.0661927090396 1.31607436892e-06 -1.49242477209e-01
```

```
## strategy2      15728 0.0656151296469 4.53706668312e-05 -1.36764999048e-08
## strategy2a    100000 0.0656151293052 9.99999313529e-01  6.89711689832e-07
## strategy3     100000 0.0656151300845 1.00001247723e+00 -1.24786555312e-05
## strategy3a    100000 0.0656151292282 9.99998806956e-01  1.18985417551e-06
## strategy1-2      182 0.0659016350177 1.13567760436e-05 -4.88435314865e-02
## strategy1-4      381 0.0656151291922 6.71890462374e-04  2.69186786526e-16
## strategy1-6      706 0.0656151291922 9.22109094156e-05  2.67669872779e-16
##                      time
## smacof       27482.396678
## smacofrelax  18411.155985
## smacofa      60446.070687
## smacofarelax 62176.770149
## newton          19.562125
## newtona        115.297453
## strategy2     8815.855219
## strategy2a   53660.499716
## strategy3    55553.805465
## strategy3a   53600.590844
## strategy1-2     31.147495
## strategy1-4     31.108504
## strategy1-6     48.058929
```

```
hsequi <- exampleRun2(equi)
print(hsequi$results, digits = 10)
```

8.3.1.2 Smacof Quickstep

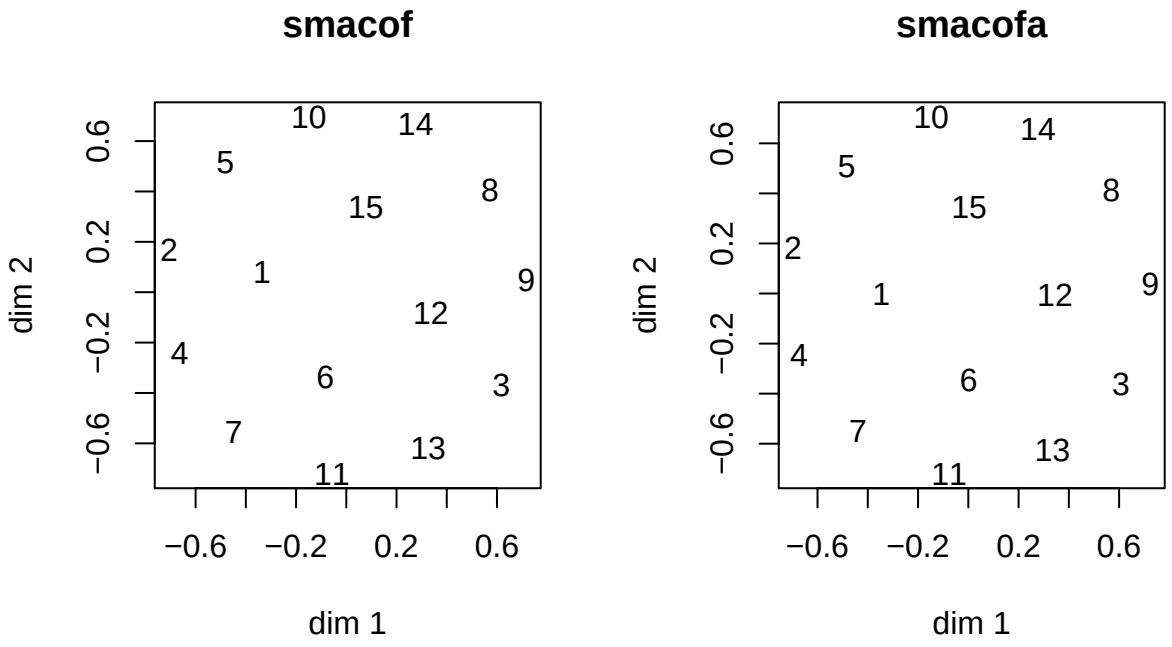
```
##                           itel        stress      time
## otmax = 1,    nterms = 1   94 0.06561750099 14.446596
## otmax = 5,    nterms = 1   64 0.06561691380 11.195009
## otmax = 10,   nterms = 1   62 0.06561692029 11.108909
## otmax = 100,  nterms = 1   62 0.06561691707 11.018176
## otmax = 1,    nterms = 2   91 0.06561594246 16.375769
## otmax = 5,    nterms = 2   76 0.06561592535 17.166044
## otmax = 10,   nterms = 2   67 0.06561595994 16.255803
## otmax = 100,  nterms = 2   68 0.06561591964 17.106922
## otmax = 1,    nterms = 3   75 0.06561555622 17.653821
## otmax = 5,    nterms = 3   64 0.06561554509 17.447304
## otmax = 10,   nterms = 3   63 0.06561552359 19.174675
## otmax = 100,  nterms = 3   63 0.06561552711 19.707798
## otmax = 1,    nterms = 4   70 0.06561539711 20.871255
## otmax = 5,    nterms = 4   66 0.06553507721 22.877713
## otmax = 10,   nterms = 4   38 0.06553491132 17.249643
## otmax = 100,  nterms = 4   32 0.06553492217 18.300514
```

8.3.2 Discussion

This example, though small, is tough, because it has factorial(15) local minima, all with the same function value. The Gower rank is 14, with the points in the optimal configuration in 14 dimensions on a regular simplex. The fact that all singular values of the full-dimensional solution are equal indicates there will be difficulties projecting into two dimensions (think of the power method with all eigenvalues equal).

Both **smacof** and **smacofa** have not converged in 100,000 iterations. To more clearly distinguish the two, we now set *itmax* equal to 1,000,000 and run **smacof** and **smacofa** again.

The final **smacof** configuration consists of eleven points equally spaced on a circle and four points in the corners of a square inside the circle. The final **smacofa** configuration is the presumed global minimum with twelve points on the circle and an equilateral triangle within the circle.



It is not impossible that if we continue iterating then in the distant future, after life on earth has long disappeared, eigenvalue 24 will be zero, and the convergence rate will consequently be one. The only way to be sure is to use study the two configurations using exact arithmetic to see if eigenvalue 24 is really zero.

Both **strategy2a** and **strategy2b** seem to converge to the same stationary value as **smacofa**, which is not surprising because they all use the reduced basis. **strategy2** and **strategy3** seem to converge a saddlepoint.

8.4 Morse

Next, similarities between 36 morse signals, taken from Rothkopf (1957).

```
data(morse, package = "smacof")
morse <- 1 - morse
```

```
hmorse <- exampleRun(morse)
```

8.4.1 Results

8.4.1.1 Smacof Meets Newton

```
##                itel       stress         emprate          minhess         time
## smacof          831 0.1760679948 9.676536873e-01 -2.714543931e-15    72.710917
## smacofrelax     450 0.1760679948 9.390097747e-01 -3.265064144e-15    41.688882
## smacofa       94359 0.1765537739 9.998984186e-01  1.013903007e-04  9099.202734
## smacofarelax  50617 0.1765537739 9.997530920e-01  1.013801725e-04  5066.727602
## newton          250 2.3348813599 5.618110024e+00 -1.118156235e-01   543.965245
## newtona         250 0.2041685333 8.073197549e-01 -7.612760274e-01   541.815246
## strategy2       133 0.1769586660 4.703540688e-07 -2.096935642e-15   266.017840
## strategy2a     5169 0.1777018041 8.583657112e-06  6.714577385e-04 11216.130970
## strategy3       280 0.1760679948 9.905927116e-08 -2.857947696e-15   564.867824
## strategy3a     3529 0.1765537739 6.576444715e-05  1.013700413e-04  7678.822596
## strategy1-2      43 0.1765810284 2.023350347e-06 -3.653247388e-15    31.019780
## strategy1-4     371 0.1760679948 9.566240633e-07 -2.800006049e-15    47.325111
## strategy1-6     553 0.1760679948 4.282207201e-06 -2.843573644e-15    55.178087
```

```
hsmorse <- exampleRun2(morse)
print(hsmorse$results, digits = 10)
```

8.4.1.2 Smacof Quickstep

```
##                            itel       stress      time
## otmax = 1,    nterms = 1    73 0.1765127194 15.605461
## otmax = 5,    nterms = 1    62 0.1765127255 21.458539
## otmax = 10,   nterms = 1    62 0.1765127422 21.747835
## otmax = 100,  nterms = 1    62 0.1765127422 22.414700
## otmax = 1,    nterms = 2    53 0.1765123936 19.630349
## otmax = 5,    nterms = 2    37 0.1765123845 19.652858
## otmax = 10,   nterms = 2    35 0.1765123568 20.928040
## otmax = 100,  nterms = 2    34 0.1765123566 21.459277
## otmax = 1,    nterms = 3    43 0.1765122705 20.505617
## otmax = 5,    nterms = 3    33 0.1765122544 21.663457
## otmax = 10,   nterms = 3    30 0.1765122552 22.958688
## otmax = 100,  nterms = 3    28 0.1765122482 23.793120
## otmax = 1,    nterms = 4    37 0.1765122041 22.340285
## otmax = 5,    nterms = 4    30 0.1765122059 24.102875
## otmax = 10,   nterms = 4    28 0.1765121983 26.258491
## otmax = 100,  nterms = 4    28 0.1765122101 28.152732
```

8.4.2 Discussion

8.5 De Gruijter

Similarities between nine Dutch political parties in 1966, taken from De Gruijter (1967). They were computed by using the complete method of triads, followed by averaging over a politically heterogeneous group of 100 students.

```
source("gruijter.R")
hgruijter <- exampleRun(gruijter)
```

8.5.1 Results

8.5.1.1 Smacof Meets Newton

```
##                itel        stress         emprate          minhess       time
## smacof         1117 0.02230169129 9.861521851e-01  3.878130944e-16  60.368933
## smacofrelax     600 0.02230169129 9.730252717e-01 -4.341758314e-16  34.404863
## smacofa        7893 0.02449551136 9.977555156e-01  2.244435229e-03 408.947202
## smacofarelax   4116 0.02449551136 9.955337984e-01  2.244435377e-03 218.617043
## newton           15 0.02857105093 9.134578614e-07 -2.342201358e-01   6.721212
## newtona         125 0.05825528603 3.574317609e-06 -1.614072759e+00  44.457694
## strategy2        43 0.02449551136 1.554811517e-06 -2.389367779e-16  16.356704
## strategy2a     1163 0.02323104236 8.262462508e-07  9.749051769e-04 427.466328
## strategy3        57 0.02230169129 6.740550916e-07  2.557172303e-16  21.331029
## strategy3a     1496 0.02449551136 5.726315696e-08  2.244435525e-03 551.363572
## strategy1-2      79 0.02230169129 3.193405904e-07  6.047477807e-16  14.405596
## strategy1-4     154 0.02230169129 8.886248189e-07  2.702524186e-16  10.699893
## strategy1-6     459 0.02230169129 5.818046455e-06  2.958439331e-16  25.483591
```

```
hsgruijter <- exampleRun2(gruijter)
print(hsgruijter$results, digits = 10)
```

8.5.1.2 Smacof Quickstep

```
##                            itel        stress      time
## otmax = 1,    nterms = 1    57 0.02230297714  6.334746
## otmax = 5,    nterms = 1    47 0.02230297055  9.311920
## otmax = 10,   nterms = 1    46 0.02230297289  9.834055
## otmax = 100,  nterms = 1    46 0.02230293136 10.584560
## otmax = 1,    nterms = 2    45 0.02230221250  7.305626
## otmax = 5,    nterms = 2    32 0.02230224942  7.098658
## otmax = 10,   nterms = 2    31 0.02230222041  8.098033
## otmax = 100,  nterms = 2    30 0.02230223405  8.979287
## otmax = 1,    nterms = 3    38 0.02230197809  7.866055
## otmax = 5,    nterms = 3    30 0.02230199254  8.157237
```

```
## otmax = 10,   nterms = 3   28 0.02230197328  9.197776
## otmax = 100,  nterms = 3   27 0.02230197703 10.177512
## otmax = 1,    nterms = 4   33 0.02230187639  8.919222
## otmax = 5,    nterms = 4   27 0.02230188084  8.997081
## otmax = 10,   nterms = 4   25 0.02230188266  9.834506
## otmax = 100,  nterms = 4   24 0.02230188104 11.839529
```

8.5.2 Discussion

8.6 Trading

Trading between countries, taken from Cox and Cox (2001)

```
data(trading, package = "smacof")
htrading <- exampleRun(trading)
```

8.6.1 Results

8.6.1.1 Smacof Meets Newton

```
##                itel        stress         emprate          minhess        time
## smacof          757 0.03556267022 9.794532017e-01  6.758230563e-17   50.213561
## smacofrelax     410 0.03556267022 9.604728432e-01 -9.527605636e-16   26.472429
## smacofa       26278 0.03542731412 9.994585067e-01  5.415968766e-04 1740.421136
## smacofarelax  13799 0.03542731412 9.989217979e-01  5.415967289e-04  944.827739
## newton          250 0.07866088391 6.237785912e-01 -4.801794233e-01  167.399228
## newtona         250 0.10315153940 7.577402692e-01 -3.102375743e-01  162.699603
## strategy2        84 0.03556267022 1.621081302e-06 -2.398603457e-18   59.081164
## strategy2a     3357 0.03542731412 2.842514318e-08  5.415965811e-04 2429.942613
## strategy3        84 0.03556267022 1.621081302e-06 -2.398603457e-18   59.925354
## strategy3a     3272 0.03542731412 3.258043371e-07  5.415965811e-04 2339.677259
## strategy1-2     112 0.03555063190 1.296712418e-07 -1.217175791e-01   35.706818
## strategy1-4     118 0.03556267022 9.352528012e-07  5.431747396e-17   11.985899
## strategy1-6     316 0.03556267022 1.063773980e-06  4.546585086e-17   22.157958
```

```
hstrading <- exampleRun2(trading)
print(hstrading$results, digits = 10)
```

8.6.1.2 Smacof Quickstep

8.6.2 Discussion

8.7 Wish

Eighteen students rated similarities between 12 nations (Kruskal and Wish (1978), p 31) on a scale from zero to nine. The data are the average similarity ratings, converted to dissimilarities by

subtracting them from nine.

```
source("wish.R")
hwish <- exampleRun(wish)
```

8.7.1 Results

8.7.1.1 Smacof Meets Newton

```
##                itel        stress         emprate          minhess      time
## smacof          432 0.02906376247 9.602005756e-01  2.979562443e-17 25.198190
## smacofrelax     240 0.02906376247 9.259203221e-01 -7.018597772e-16 14.323432
## smacofa         748 0.02895458552 9.719895705e-01  2.801063234e-02 41.363547
## smacofarelax    403 0.02895458552 9.468082086e-01  2.801063226e-02 23.325638
## newton          174 0.08109079232 3.096752359e-05 -3.132519623e-01 74.705731
## newtona         114 0.05876421547 3.635698391e-06 -1.649508203e+00 49.873015
## strategy2        27 0.02906376247 1.536567911e-07 -2.402137809e-16 12.341246
## strategy2a       86 0.02895458552 1.834585579e-06  2.801063217e-02 36.020017
## strategy3        27 0.02906376247 8.235199983e-01 -1.930590976e-16 12.471257
## strategy3a       87 0.02895458552 1.283111535e-06  2.801063217e-02 36.487827
## strategy1-2      39 0.02906376247 1.047673705e-06 -3.352533786e-16  8.530337
## strategy1-4      96 0.02906376247 3.430500455e-07 -2.428458810e-16  8.231570
## strategy1-6     208 0.02906376247 3.792319279e-07 -8.551137900e-17 13.426270
```

```
hswish <- exampleRun2(wish)
print(hswish$results, digits = 10)
```

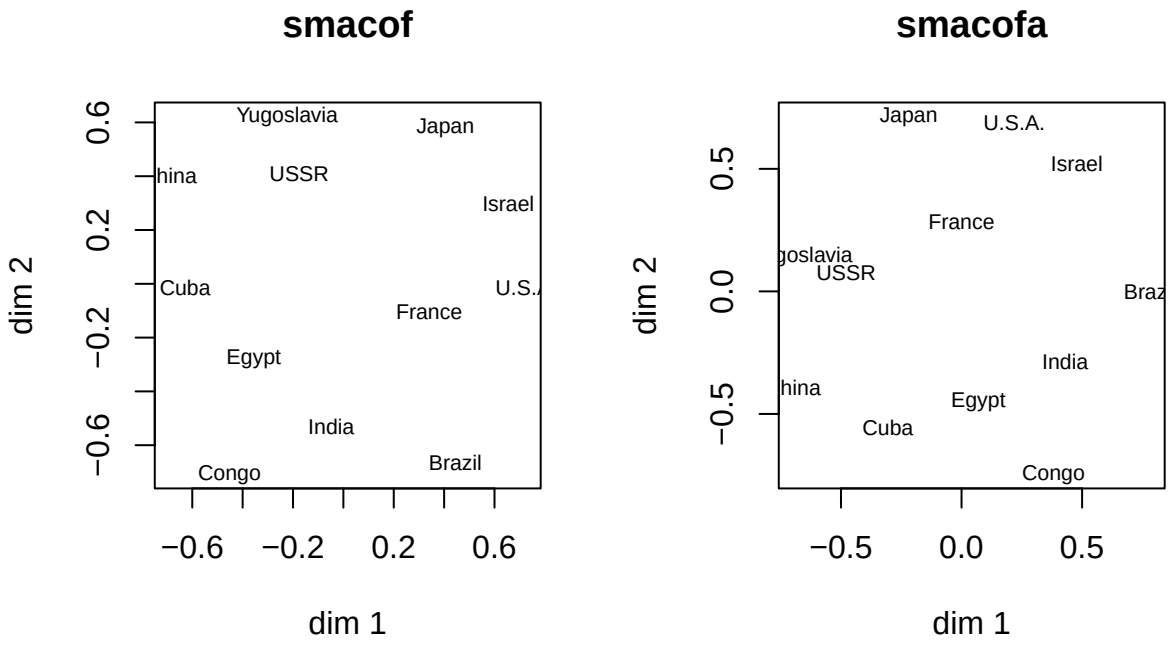
8.7.1.2 Smacof Quickstep

```
##                            itel        stress     time
## otmax = 1,    nterms = 1    39 0.02906434590 4.573263
## otmax = 5,    nterms = 1    34 0.02906437748 5.873291
## otmax = 10,   nterms = 1    34 0.02906438805 6.316501
## otmax = 100,  nterms = 1    34 0.02906437227 6.372876
## otmax = 1,    nterms = 2    29 0.02906400103 4.918319
## otmax = 5,    nterms = 2    24 0.02906397903 5.444390
## otmax = 10,   nterms = 2    24 0.02906396002 6.292475
## otmax = 100,  nterms = 2    24 0.02906395941 6.612685
## otmax = 1,    nterms = 3    24 0.02906387777 5.390393
## otmax = 5,    nterms = 3    21 0.02906385823 5.826756
## otmax = 10,   nterms = 3    19 0.02906389217 6.489685
## otmax = 100,  nterms = 3    19 0.02906389041 6.697842
## otmax = 1,    nterms = 4    20 0.02906384578 5.653736
## otmax = 5,    nterms = 4    18 0.02906382646 6.266932
## otmax = 10,   nterms = 4    17 0.02906382668 7.063275
```

```
## otmax = 100,  nterms = 4   17 0.02906382864 7.982905
```

8.7.2 Discussion

This is small example, with a good fit. There seem to be three different stationary points. **newtona** converges to a saddle point. One of the local minima is found by **smacofa**, **newton**, **strategy 2a**, and **strategy 3a**. Its stress (for **newton**) is . The other local minimum is found by **smacof**, **strategy2** and **strategy 3**. It has a slightly higher stress (for **smacof**) of . In the two plots the first and "best" local minimum is on the left, the second is on the right. Although the grouping of the nations in the two plots is similar, their actual locations are quite different.



8.8 Airline Distances

Airline distances between 18 cities are taken from Izenman (2008), page 466, who collected them from the National Geographic Society's *Atlas of the World*, sixth edition, 1995. Because of the spherical earth the larger distances cannot properly be projected on the plane, and the MDS will be like a spherical projection.

```
source("airline.R")
hairline <- exampleRun(airline)
```

The cities are Beijing.

8.8.1 Results

8.8.1.1 Smacof Meets Newton

```
##              itel        stress         emprate         minhess       time
## smacof        297 0.00988993672 9.428533733e-01 1.110223025e-15  18.426999
```

```
## smacofrelax   168 0.00988993672 8.964618992e-01 1.332267630e-15  11.814519
## smacofa      1935 0.00988993672 9.916922247e-01 8.307369205e-03 117.761102
## smacofarelax 1024 0.00988993672 9.836464616e-01 8.307369194e-03  65.544486
## newton          9 0.00988993672 1.395947034e-07 2.442490654e-15   7.169752
## newtona         8 0.00988993672 1.953624139e-07 8.307369182e-03   6.679310
## strategy2      10 0.00988993672 5.115172131e-08 1.554312234e-15   7.468560
## strategy2a      9 0.00988993672 3.988699891e-08 8.307369182e-03   6.925515
## strategy3      10 0.00988993672 3.694707637e-07 1.332267630e-15   7.473234
## strategy3a      7 0.00988993672 2.152088807e-06 8.307369182e-03   5.979768
## strategy1-2    13 0.00988993672 1.392805054e-05 2.220446049e-15   6.363938
## strategy1-4    66 0.00988993672 3.756790000e-06 1.332267630e-15   7.639530
## strategy1-6   143 0.00988993672 1.069919901e-06 1.332267630e-15  11.470324
```

```
hsairline <- exampleRun2(airline)
print(hsairline$results, digits = 10)
```

8.8.1.2 Smacof Quickstep

```
##                            itel         stress     time
## otmax = 1,    nterms = 1   25 0.009890391116 3.358802
## otmax = 5,    nterms = 1   24 0.009890366162 3.907177
## otmax = 10,   nterms = 1   24 0.009890366162 4.117466
## otmax = 100,  nterms = 1   24 0.009890366162 3.833049
## otmax = 1,    nterms = 2   20 0.009890072856 3.884791
## otmax = 5,    nterms = 2   17 0.009890074833 4.163591
## otmax = 10,   nterms = 2   17 0.009890073609 4.271462
## otmax = 100,  nterms = 2   17 0.009890073609 4.123247
## otmax = 1,    nterms = 3   16 0.009890022949 4.122427
## otmax = 5,    nterms = 3   14 0.009890013048 4.428902
## otmax = 10,   nterms = 3   14 0.009890002767 4.639232
## otmax = 100,  nterms = 3   14 0.009890004884 4.644562
## otmax = 1,    nterms = 4   14 0.009889981214 4.682733
## otmax = 5,    nterms = 4   13 0.009889972369 4.953169
## otmax = 10,   nterms = 4   12 0.009889984652 5.021926
## otmax = 100,  nterms = 4   12 0.009889984652 5.010815
```

8.8.2 Discussion

9 General Discussion

1. MDS may have local minima which are quite different but have very close stress values.
2. If the dissimilarities have little variation there tend to be many local minima and smacof converges very slowly.
3. It is imperative to iterate smacof to (very) high precision.

4. The reduced basis may have some theoretical appeal but greatly slows down computation.
5. strategy 1 iterates smacof to low precision and then switches to newton. This seems to work well, especially for larger examples.
6. strategy 2 and 3 require computation of both the smacof and newton update in each iteration. This is (maybe too) expensive.
7. It is imperative to inspect the eigenvalues of the Hessian at convergence.
8. The adaptive version of the relaxed update works well and does approximately half the number of iterations.
9. The relaxed update $2\Gamma(\theta) - \theta$ should not be used.
10. It is informative to inspect the eigenvalues of $B(\theta)$ at convergence.
11. Since using the full basis for unrestricted MDS is equivalent to working with x in $\mathbb{R}^{np}$, we may as well not use Y and θ if there are no constraints.
12. It is informative to compute the full-dimensional MDS solution.
13. It is informative to use the Hessian to draw pseudo-confidence ellipses.
14. It is a good idea to use more than one of the computational strategies outlined in this report on a single data set.
15. Using unsafeguarded newton from the start is a bad idea.
16. For really large examples iterating smacof to high precision may take an eternity.
17. smacof always converges to a local minimum, except when started in a saddle point. Strategies that mix smacof and newton could end up in saddle points.

10 Appendix: R Code

10.1 utils.R

```
# returns unit vector i of length n

ei <- function(i, n) {
  return(ifelse(i == 1:n, 1, 0))
}

# return distancing matrix (i,j) of order n

aij <- function(i, j, n) {
  df <- ei(i, n) - ei(j, n)
  return(outer(df, df))
}

# y is a list of matrices
# returns a matrix with the direct sum of the y[[i]]

directSum <- function(y) {
  mr <- sum(sapply(y, nrow))
  mc <- sum(sapply(y, ncol))
```

```
  nr <- c(0, cumsum(sapply(y, nrow)))
  nc <- c(0, cumsum(sapply(y, ncol)))
  p <- length(y)
  z <- matrix(0, mr, mc)
  for (i in 1:p) {
    ind <- (nr[i] + 1):nr[i + 1]
    jnd <- (nc[i] + 1):nc[i + 1]
    z[ind, jnd] <- y[[i]]
  }
  return(z)
}

# y is a matrix
# returns a matrix with the direct sum of p copies of y

directExpand <- function(y, p) {
  return(directSum(rep(list(y), p)))
}

# double centers a symmetric matrix

doubleCenter <- function(x) {
  rs <- apply(x, 1, mean)
  ss <- mean(x)
  return(x - outer(rs, rs, "+") + ss)
}

# mPrint() formats a matrix (or vector, or scalar) of numbers
# for printing

mPrint <- function(x,
                   digits = 6,
                   width = 8,
                   format = "f",
                   flag = "+") {
  print(noquote(
    formatC(
      x,
      digits = digits,
      width = width,
      format = format,
      flag = flag
    )
  ))
```

```
}

# classical MDS

torgerson <- function(delta, p = 2) {
  e <- eigen(-.5 * doubleCenter(as.matrix(delta) ^ 2))
  l <- sqrt(pmax(0, e$values[1:p]))
  if (p == 1) {
    return(as.matrix(e$vectors[, 1] * l))
  } else {
    return(e$vectors[, 1:p] %*% diag(l))
  }
}

# converts dist object w to M-matrix

mmatrix <- function(w) {
  v <- -as.matrix(w)
  diag(v) <- -rowSums(v)
  return(v)
}

# create dist with all elements equal to one

oneDist <- function(n) {
  return(as.dist(matrix(1, n, n)))
}

## partitioned matrix to single matrix

par2mat <- function(x) {
  nc <- length(x[[1]])
  nr <- length(x)
  h <- NULL
  for (i in 1:nr) {
    hi <- NULL
    for (j in 1:nc) {
      hi <- cbind(hi, x[[i]][[j]])
    }
    h <- rbind(h, hi)
  }
  return(h)
}
```

```
# vector to partitioned list, inverse of unlist()

vec2List <- function(x, y) {
  z <- list()
  h <- sapply(y, ncol)
  p <- length(h)
  ind <- cumsum(c(1, h))[-(p + 1)]
  jnd <- ind + (h - 1)
  for (k in 1:p) {
    z <- c(z, list(x[ind[k]:jnd[k]]))
  }
  return(z)
}

# tmForm of a matrix X has x_{ij}=0 if j geq i
# while the distances satisfy D(tlForm(X)) = D(X)

tmForm <- function(x) {
  x <- x - matrix(x[1,], nrow(x), ncol(x), byrow = TRUE)
  x[-1,] <- x[-1,] %*% qr.Q(qr(t(x[-1,])))
  return(x)
}

# dlForm of a matrix X has e'X = 0 and x_{ij}=0 if j > i
# while the distances satisfy D(dlForm(X)) = D(X)

dlForm <- function(x) {
  x <- apply(x, 2, function(x)
    x - mean(x))
  return(x %*% qr.Q(qr(t(x))))
}

# returns a configuration scaled to minimize stress

scalConf <- function(delta, x, w = oneDist(attr(delta, "Size"))) {
  d <- dist(x)
  lbd <- sum(w * delta * d) / sum(w * (d ^ 2))
  return(lbd * x)
}

# just returns stress

stressComp <- function(delta, x, w = oneDist(attr(delta, "Size"))) {
  return(sum(w * (delta - dist(x)) ^ 2) / 2)
```

```
}

# returns a weighted sum of matrices in a list x with weights in a

listSum <- function(x, a = rep(1, length(x))) {
  n <- length(x)
  y <- array(0, dim(x[[1]]))
  for (i in 1:n) {
    y <- y + a[i] * x[[i]]
  }
  return(y)
}

guttman <- function(delta, x, w = oneDist(nrow(x))) {
  v <- mmatrix(w)
  b <- mmatrix(w * delta / dist(x))
  return(solve(v + 1, b %*% x))
}
```

10.2 basis.R

```
aTriBas <- function(n, p) {
  np <- n - (p - 1)
  x <- diag(np)
  x[, 1] <- 1
  x <- rbind(matrix(0, p - 1, np), x)
  return(qr.Q(qr(x))[, -1, drop = FALSE])
}

aBasis <- function(n, p) {
  return(lapply(1:p, function(s)
    aTriBas(n, s)))
}

bFulBas <- function(n) {
  x <- diag(n)
  x[, 1] <- 1
  return(qr.Q(qr(x))[, -1])
}

bBasis <- function(n, p) {
  x <- bFulBas(n)
  return(lapply(1:p, function(s)
```

```
    x))
}

uBasis <- function(n, p) {
  x <- bBasis(n, p)
  for (i in 1:n) {
    ei <- ifelse(i == 1:n, 1, 0)
    ei <- as.matrix(ei - mean(ei))
    x <- c(x, list(ei))
  }
  return(x)
}

makeBasis <- function(p, basis, w) {
  v <- mmatrix(w)
  n <- nrow(v)
  y <-
    switch(basis,
           A = aBasis(n, p),
           B = bBasis(n, p),
           C = cBasis(n, p))
  ev <-
    lapply(1:p, function(s)
      eigen(crossprod(y[[s]], v %*% y[[s]])))
  y <-
    lapply(1:p, function(s)
      y[[s]] %*% ev[[s]]$vectors %*% diag(1 / sqrt(ev[[s]]$values)))
  return(y)
}
```

10.3 derivatives.R

```
# returns the gradient of stress at X as n x p matrix

gradientx <- function(x, b, v) {
  return((v - b) %*% x)
}

# returns the gradient of stress at theta as list of vectors

gradientt <- function(x, b, v, y) {
  gradx <- gradientx(x, b, v)
  return(lapply(1:length(y), function(s) drop(crossprod(y[[s]], gradx[, s]))))
```

```
}

# returns the Guttman transform at x as matrix

guttmanx <- function (x, b, vinv) {
  return(vinv %*% b %*% x)
}

# returns the Guttman transform at theta as list of vectors

guttmant <- function(x, b, vinv, y) {
  guttx <- guttmanx(x, b, vinv)
  return(lapply(1:length(y), function(s) drop(crossprod(y[[s]], guttx[, s]))))
}

# returns the hessian of stress at x as partioned matrix

hessianx <- function(x, delta, w) {
  p <- ncol(x)
  n <- nrow(x)
  d <- dist(x)
  aux1 <- w * delta / d
  aux2 <- w * delta / (d ^ 3)
  hesx <- list()
  for (s in 1:p) {
    hessrows <- list()
    di <- as.dist(outer(x[, s], x[, s], "-"))
    for (t in 1:p) {
      dj <- as.dist(outer(x[, t], x[, t], "-"))
      hesspart <- mmatrix(aux2 * di * dj)
      if (s == t) {
        hesspart <- mmatrix(w) - (mmatrix(aux1) - hesspart)
      }
      hessrows <- c(hessrows, list(hesspart))
    }
    hesx <- c(hesx, list(hessrows))
  }
  return(hesx)
}

# returns the hessian of stress at theta as partioned matrix

hessiant <- function(x, delta, w, y) {
  hesx <- hessianx(x, delta, w)
```

```
  hest <- hesx
  p <- length(y)
  for (s in 1:p) {
    for (t in 1:p) {
      hest[[s]][[t]] <- crossprod(y[[s]], (hesx[[s]][[t]] %*% y[[t]]))
    }
  }
  return(hest)
}


# computes first, second, and third derivatives (in x space for now).

thirdx <- function(delta, x, w = 1 - diag(nrow(x))) {
  n <- nrow(x)
  p <- ncol(x)
  np <- n * p
  w <- 2 * w / sum(w)
  delta <- delta / sqrt(sum(w * delta ^ 2) / 2)
  d <- as.matrix(dist(x))
  y <- as.vector(x)
  func <- 0.0
  fij <- array(0, np)
  ftot <- array(0, np)
  sij <- array(0, c(np, np))
  stot <- array(0, c(np, np))
  tij <- array(0, c(np, np, np))
  ttot <- array(0, c(np, np, np))
  for (j in 1:(n - 1)) {
    ej <- ifelse(j == 1:n, 1, 0)
    for (i in (j + 1):n) {
      dij <- d[i, j]
      wij <- w[i, j]
      gij <- delta[i, j]
      func <- func + wij * (dij - gij) ^ 2
      ei <- ifelse(i == 1:n, 1, 0)
      aij <- outer(ei - ej, ei - ej)
      a <- directSum(rep(list(aij), p))
      u <- drop(a %*% y)
      for (s in 1:np) {
        fij[s] <- u[s] / dij
        for (t in 1:np) {
          sij[s, t] <- a[s, t] / dij - (u[s] * u[t]) / (dij ^ 3)
          for (r in 1:np) {
```

```
            tij[s, t, r] <- (u[s] * u[t] * u[r]) / (dij ^ 5)
            tij[s, t, r] <-
              tij[s, t, r] - ((u[s] * a[t, r]) + (u[t] + a[s, r]) + (u[r] * a[s, t])) /
          }
        }
      }
      ttot <- ttot - w[i, j] * delta[i, j] * tij
      stot <- stot - w[i, j] * delta[i, j] * sij
      ftot <- ftot - w[i, j] * delta[i, j] * fij
      for (s in 1:np) {
        ftot[s] <- ftot[s] + w[i, j] * u[s]
        for (t in 1:np) {
          stot[s, t] <- stot[s, t] + w[i, j] * a[s, t]
        }
      }
    }
  }
  return(list(
    func = func / 2.0,
    ftot = ftot,
    stot = stot,
    ttot = ttot
  ))
}
```

10.4 msmacof.R

```
source("utils.R")
source("basis.R")
source("exampleRun.R")
source("derivatives.R")

library(MASS)
library(microbenchmark)

mSmacof <-
  function(delta,
           # dissimilarities, class dist
           w = oneDist(attr(delta, "Size")),
           # weights, class dist
           p = 2L,
           # dimensionality
           xold = NULL,
```

```
        # initial configuration, matrix, or NULL
        basis = "B",
        # choice of basis
        itmax = 100000L,
        relax = FALSE,
        strategy = 1L,
        eps1 = 15L,
        eps2 = 10L,
        eps3 = 15L,
        verbose = FALSE) {
n <- attr(delta, "Size")
w <- w / sum(w)
v <- mmatrix(w)
vinv <- solve(w + (1 / n)) - (1 / n)
delta <- delta / sqrt(sum(w * (delta ^ 2)))
if (is.null(xold)) {
  xold <- torgerson(delta, p)
}
xold <- apply(xold, 2, function(x)
  x - mean(x))
y <- makeBasis(p, basis, w)
told <-
  lapply(1:p, function(s)
    drop(crossprod(y[[s]], xold[, s])))
xold <- sapply(1:p, function(s)
  y[[s]] %*% told[[s]])
dold <- dist(xold)
lbd  <- sum(w * delta * dold) / sum(w * (dold ^ 2))
dold <- dold * lbd
xold <- xold * lbd
bold <- mmatrix(w * (delta / dold))
sold <- sum(w * (delta - dold) ^ 2) / 2
rold <- Inf
itel <- 1L
eopt <- 0.0
repeat {
  # smacof step
  tnew <-
    lapply(1:p, function(s)
      drop(crossprod(y[[s]], bold %*% xold[, s]))) # Guttman transform at told
  grat <-
    lapply(1:p, function(s)
      told[[s]] - tnew[[s]]) # gradient at told as list
  rgrd <-
```

```
    sqrt(sum(sapply(grat, function(x)
      sum(x ^ 2)))) # norm of change in theta
  if (relax) {
    eopt <- min(1, eopt)
    tnew <- lapply(1:p, function(s)
      (1 + eopt) * tnew[[s]] - eopt * told[[s]])
  } # compute relaxed update of theta
  xnew <- sapply(1:p, function(s)
    drop(y[[s]] %*% tnew[[s]])) # compute new x after smacof step
  dnew <-
    dist(xnew) # compute new d after smacof step
  bnew <- mmatrix(w * delta / dnew)
  snew <-
    sum(w * (delta - dnew) ^ 2) / 2 # compute new stress after smacof step
  type <- "SMACOF"
  # newton step
  if ((strategy > 1) || ((strategy == 1) && (rgrd < (10 ^ -eps3)))) {
    hest <-
      hessiant(xold, delta, w, y) # hessian at told
    hist <- ginv(par2mat(hest))
    tnwt <-
      unlist(told) - drop(hist %*% unlist(grat)) # new theta with newton step
    tnwt <- vec2List(tnwt, y)
    xnwt <-
      sapply(1:p, function(s)
        y[[s]] %*% tnwt[[s]]) # compute new x after newton step
    dnwt <- dist(xnwt) # new d after newton step
    bnwt <- mmatrix(w * delta / dnwt) # bmat at tnwt
    snwt <-
      sum(w * (delta - dnwt) ^ 2) / 2 # new stress after newton step
  }
  if (((strategy == 3) && (snwt < snew)) ||
      ((strategy == 2) && (snwt < sold)) ||
      ((strategy == 1) && (rgrd < (10 ^ -eps3)))) {
    snew <- snwt
    tnew <- tnwt
    xnew <- xnwt
    dnew <- dnwt
    bnew <- bnwt
    type <- "NEWTON"
  }
  chng <-
    lapply(1:p, function(s)
      told[[s]] - tnew[[s]])  # change in theta
```

```
    rnew <-
      sqrt(sum(sapply(chng, function(x)
        sum(x ^ 2)))) # norm of change in theta
    erat <- rnew / rold # empirical rate
    eopt <- min(1, erat / (2 - erat))
    sdif <- sold - snew
    if (verbose) {
      cat(
        type,
        "itel =",
        formatC(itel, digits = 2, format = "d"),
        "snew =",
        formatC(snew, digits = min(15, eps1), format = "f"),
        "sdif =",
        formatC(sdif, digits = min(15, eps1), format = "f"),
        "chng =",
        formatC(rnew, digits = min(10, eps2), format = "f"),
        "rate =",
        formatC(erat, digits = min(10, eps2), format = "f"),
        "\n"
      )
    }
    if ((itel == itmax) ||
        ((abs(sold - snew)) < (10 ^ -eps1)) &&
        (rnew < (10 ^ -eps2))) {
      break()
    }
    itel <- itel + 1
    told <- tnew
    xold <- xnew
    dold <- dnew
    bold <- bnew
    sold <- snew
    rold <- rnew
  }
  hest <-
    hessiant(xnew, delta, w, y) # hessian after convergence
  hesx <-
    hessianx(xnew, delta, w) # hessian after convergence
  grax <- lapply(1:p, function(s)
    drop(xnew[, s] - vinv %*% bnew %*% xnew[, s])) # gradient at xnew as list
  evlt <-
    eigen(par2mat(hest))$values # eigenvalues of hessian at t
  evlx <-
```

```
    eigen(par2mat(hesx))$values # eigenvalues of hessian at x
  evlb <-
    eigen(ginv(mmatrix(w)) %*% bnew)$values # eigenvalues of V^+B matrix
  if (basis == "A") {
    trat <- rev(1 - evlt)[1] # theoretical convergence rate
  } else {
    trat <- rev(1 - evlt)[2] # theoretical convergence rate
  }
  grat <-
    lapply(1:p, function(s)
      drop(crossprod(y[[s]], grax[[s]]))) # gradient wrt t as list
  grax <- matrix(unlist(grax), n, p) # gradient wrt x as matrix
  if (itel > itmax) {
    warning("Maximum number of iterations reached")
  }
  return(
    list(
      wmat = w,
      delt = delta,
      ybas = y,
      xmat = xnew,
      bmat = bnew,
      dmat = dnew,
      itel = itel,
      loss = snew,
      thet = tnew,
      grax = grax,
      grat = grat,
      hesx = hesx,
      hest = hest,
      evlt = evlt,
      evlx = evlx,
      evlb = evlb,
      erat = erat,
      trat = trat
    )
  )
}
```

10.5 vsmacof.R

```
vSmacof <-
  function(delta,
```

```
        mbas,
        w = oneDist(attr(delta, "Size")),
        told = rep(1, length(mbas)),
        itmax = 1000,
        eps1 = 1e-15,
        eps2 = 1e-15,
        verbose = TRUE) {
w <- w / sum(w)
delta <- delta / sqrt(sum(w * (delta ^ 2)))
m <- length(mbas)
v <- mmatrix(w)
vv <- matrix(0, m, m)
for (i in 1:m) {
  for (j in 1:m) {
    vv[i, j] <- sum(mbas[[i]] * (v %*% mbas[[j]]))
  }
}
vi <- ginv(vv)
zold <- listSum(mbas, told)
dold <- dist(zold)
sold <- sum(w * (delta - dold) ^ 2) / 2.0
itel <- 1
repeat {
  bold <- mmatrix(w * (dlmat / dold))
  bb <- matrix(0, m, m)
  for (i in 1:m) {
    for (j in 1:m) {
      bb[i, j] <- sum(mbas[[i]] * bold %*% mbas[[j]])
    }
  }
  tnew <- vi %*% bb %*% told
  znew <- listSum(mbas, tnew)
  dnew <- dist(znew)
  snew <- sum(w * (delta - dnew) ^ 2) / 2.0
  chng <- sqrt(sum((told - tnew) ^ 2))
  if (verbose) {
    cat(
      "itel ",
      formatC(itel, digits = 4, format = "d"),
      "sold ",
      formatC(sold, digits = 15, format = "f"),
      "snew ",
      formatC(snew, digits = 15, format = "f"),
      "chng ",
```

```
        formatC(chng, digits = 15, format = "f"),
        "\n"
      )
    }
    if ((itel == itmax) || (((sold - snew) < eps1) && (chng < eps2))) {
      break
    }
    told <- tnew
    zold <- znew
    sold <- snew
    dold <- dnew
    itel <- itel + 1
  }
  return(list(
    c = tnew,
    z = znew,
    d = dnew,
    s = snew,
    itel = itel
  ))
}
```

10.6 ssmacof.R

```
sSmacof <-
  function(delta,
           p = 2,
           xold = torgerson(delta, p),
           w = oneDist(attr(delta, "Size")),
           itmax = 10000L,
           otmax = 1L,
           nterms = 1L,
           epsouter = 1e-15,
           epsinner = 1e-6,
           verbose = FALSE) {
    m <- nterms + 1
    w <- w / sum(w)
    delta <- delta / sqrt(sum(w * delta ^ 2))
    v <- mmatrix(w)
    itel <- 1
    repeat {
      # nterms smacof iterations to build basis
      gutt <- list(xold)
```

```
    repeat {
      xnew <- guttman(delta, xold, w = w)
      gutt <- c(gutt, list(xnew))
      if (nterms == (length(gutt) - 1L)) {
        break
      }
      xold <- xnew
    }
    # at most otmax smacof iterations to find weights
    m <- length(gutt)
    told <- ei(m, m)
    xold <- gutt[[m]]
    dold <- dist(xold)
    bold <- mmatrix(w * (delta / dold))
    sold <- sum(w * (delta - dold) ^ 2) / 2
    vcon <- matrix(0, m, m)
    for (i in 1:m) {
      for (j in 1:m) {
        vcon[i, j] <- sum(gutt[[i]] * v %*% gutt[[j]])
      }
    }
    otel <- 1
    repeat {
      bcon <- matrix(0, m, m)
      for (i in 1:m) {
        for (j in 1:m) {
          bcon[i, j] <- sum(gutt[[i]] * bold %*% gutt[[j]])
        }
      }
      tnew <- drop(ginv(vcon) %*% bcon %*% told)
      xnew <- listSum(gutt, tnew)
      dnew <- dist(xnew)
      snew <- sum(w * (delta - dnew) ^ 2) / 2
      diff <- sold - snew
      bnew <- mmatrix(w * delta / dnew)
      if ((otel == otmax) || (diff < epsinner)) {
        break
      }
      otel <- otel + 1
      bold <- bnew
      told <- tnew
      sold <- snew
    }
    diff <- sold - snew
```

```
    if (verbose) {
      cat(
        "itel ",
        formatC(itel, format = "d"),
        "snew ",
        formatC(snew, digits = 15, format = "f"),
        "diff ",
        formatC(diff, digits = 15, format = "f"),
        "\n"
      )
    }
    if ((diff < epsouter) || (itel == itmax)) {
      break
    }
    itel <- itel + 1
    xold <- xnew
  }
  return(list(
    xmat = xnew,
    loss = snew,
    thet = tnew,
    itel = itel
  ))
}
```

10.7 exampleRun.R

```
exampleRun <- function(delta) {
  args <- list(
    list(delta, itmax = 1000000),
    list(delta, relax = TRUE, itmax = 1000000),
    list(delta, basis = "A", itmax = 1000000),
    list(
      delta,
      basis = "A",
      relax = TRUE,
      itmax = 1000000
    ),
    list(delta, eps3 = -1, itmax = 250),
    list(
      delta,
      basis = "A",
      eps3 = -1,
```

```
    itmax = 250
  ),
  list(delta, strategy = 2),
  list(delta, strategy = 2, basis = "A"),
  list(delta, strategy = 3),
  list(delta, strategy = 3, basis = "A"),
  list(delta, eps3 = 2),
  list(delta, eps3 = 4),
  list(delta, eps3 = 6)
)
n <- length(args)
run <- as.list(1:n)
trun <- as.list(1:n)
for (i in 1:n) {
  run[[i]] <- do.call("mSmacof", args[[i]])
  trun[[i]] <-
    suppressWarnings(fivenum(microbenchmark(do.call(
      "mSmacof", args[[i]]
    ), times = 11L)$time / (10 ^ 6)))
}
results <- matrix(0, n, 5)
row.names(results) <-
  c(
    "smacof      ",
    "smacofrelax ",
    "smacofa     ",
    "smacofarelax",
    "newton      ",
    "newtona     ",
    "strategy2   ",
    "strategy2a  ",
    "strategy3   ",
    "strategy3a  ",
    "strategy1-2 ",
    "strategy1-4 ",
    "strategy1-6 "
  )
colnames(results) <-
  c("itel", "stress", "emprate", "minhess", "time")
for (i in 1:n) {
  results[i,] <-
    c(run[[i]]$itel,
      run[[i]]$loss,
      run[[i]]$erat,
```

```
        min(run[[i]]$evlt),
        trun[[i]][[3]])
  }
  return(list(results = results, run = run))
}

exampleRun2 <- function(delta) {
  args <- list(
    list(delta, otmax = 1L, nterms = 1L),
    list(delta, otmax = 5L, nterms = 1L),
    list(delta, otmax = 10L, nterms = 1L),
    list(delta, otmax = 100L, nterms = 1L),
    list(delta, otmax = 1L, nterms = 2L),
    list(delta, otmax = 5L, nterms = 2L),
    list(delta, otmax = 10L, nterms = 2L),
    list(delta, otmax = 100L, nterms = 2L),
    list(delta, otmax = 1L, nterms = 3L),
    list(delta, otmax = 5L, nterms = 3L),
    list(delta, otmax = 10L, nterms = 3L),
    list(delta, otmax = 100L, nterms = 3L),
    list(delta, otmax = 1L, nterms = 4L),
    list(delta, otmax = 5L, nterms = 4L),
    list(delta, otmax = 10L, nterms = 4L),
    list(delta, otmax = 100L, nterms = 4L)
  )
  n <- length(args)
  run <- as.list(1:n)
  trun <- as.list(1:n)
  for (i in 1:n) {
    run[[i]] <- do.call("sSmacof", args[[i]])
    trun[[i]] <-
      suppressWarnings(fivenum(microbenchmark(do.call(
        "sSmacof", args[[i]]
      ), times = 11L)$time / (10 ^ 6)))
  }
  results <- matrix(0, n, 3)
  row.names(results) <-
    c(
      "otmax = 1,    nterms = 1",
      "otmax = 5,    nterms = 1",
      "otmax = 10,   nterms = 1",
      "otmax = 100,  nterms = 1",
      "otmax = 1,    nterms = 2",
      "otmax = 5,    nterms = 2",
```

```
      "otmax = 10,   nterms = 2",
      "otmax = 100,  nterms = 2",
      "otmax = 1,    nterms = 3",
      "otmax = 5,    nterms = 3",
      "otmax = 10,   nterms = 3",
      "otmax = 100,  nterms = 3",
      "otmax = 1,    nterms = 4",
      "otmax = 5,    nterms = 4",
      "otmax = 10,   nterms = 4",
      "otmax = 100,  nterms = 4"
    )
  colnames(results) <-
    c("itel", "stress", "time")
  for (i in 1:n) {
    results[i,] <-
      c(run[[i]]$itel,
        run[[i]]$loss,
        trun[[i]][[3]])
  }
  return(list(results = results, run = run))
}
```

References

Bentler, P. M., and D. G. Weeks. 1978. "Restricted Multidimensional Scaling Models." *Journal of Mathematical Psychology* 17: 138–51.

Borg, I., and P. J. F. Groenen. 2005. *Modern Multidimensional Scaling*. Second Edition. Springer.

Borg, I., and D. Leutner. 1983. "Dimensional Models for the Perception of Rectangles." *Perception and Psychophysics* 34: 257–69.

Cox, T. F., and M. A. A. Cox. 2001. *Multidimensional Scaling*. Second Edition. Monographs on Statistics and Applied Probability 88. Chapman & Hall.

De Gruijter, D. N. M. 1967. "The Cognitive Structure of Dutch Political Parties in 1966." Report E019-67. Psychological Institute, University of Leiden.

De Leeuw, J. 1977. "Applications of Convex Analysis to Multidimensional Scaling." In *Recent Developments in Statistics*, edited by J. R. Barra, F. Brodeau, G. Romier, and B. Van Cutsem, 133–45. Amsterdam, The Netherlands: North Holland Publishing Company.

———. 1984. "Differentiability of Kruskal's Stress at a Local Minimum." *Psychometrika* 49: 111–13.

———. 1988. "Convergence of the Majorization Method for Multidimensional Scaling." *Journal of Classification* 5: 163–80.

———. 2006. "Accelerated Least Squares Multidimensional Scaling." Preprint Series 493. Los Angeles, CA: UCLA Department of Statistics. https://jansweb.netlify.app/publication/deleeuw-r-06-b/deleeuw-r-06-b.pdf.

———. 2014. “Bounding, and Sometimes Finding, the Global Minimum in Multidimensional Scaling.” UCLA Department of Statistics. https://jansweb.netlify.app/publication/deleeuw-u-14-b/deleeuw-u-14-b.pdf.
———. 2016. “Gower Rank.” 2016. https://jansweb.netlify.app/publication/deleeuw-e-16-k/deleeuw-e-16-k.pdf.
———. 2019. “Global Minima by Penalized Full-dimensional Scaling.” 2019. https://jansweb.netlify.app/publication/deleeuw-e-19-e/deleeuw-e-19-e.pdf.
———. 2021. *Least Squares Euclidean Multidimensional Scaling*. https://jansweb.netlify.app/publication/deleeuw-b-21-a/deleeuw-b-21-a.pdf.
———. 2023. “Powering Dissimilarities in Metric MDS.” 2023. https://doi.org/10.13140/RG.2.2.31865.44645.
De Leeuw, J., and W. J. Heiser. 1977. “Convergence of Correction Matrix Algorithms for Multidimensional Scaling.” In *Geometric Representations of Relational Data*, edited by J. C. Lingoes, 735–53. Ann Arbor, Michigan: Mathesis Press.
———. 1980. “Multidimensional Scaling with Restrictions on the Configuration.” In *Multivariate Analysis, Volume V*, edited by P. R. Krishnaiah, 501–22. Amsterdam, The Netherlands: North Holland Publishing Company.
De Leeuw, J., and P. Mair. 2009. “Multidimensional Scaling Using Majorization: SMACOF in R.” *Journal of Statistical Software* 31 (3): 1–30. https://www.jstatsoft.org/article/view/v031i03.
De Leeuw, J., and I. Stoop. 1984. “Upper Bounds for Kruskal’s Stress.” *Psychometrika* 49: 391–402.
Ekman, G. 1954. “Dimensions of Color Vision.” *Journal of Psychology* 38: 467–74.
Gower, J. C. 1966. “Some Distance Properties of Latent Root and Vector Methods Used in Multivariate Analysis.” *Biometrika* 53: 325–38.
Groenen, P. J. F., and M. Van de Velden. 2016. “Multidimensional Scaling by Majorization: A Review.” *Journal of Statistical Software* 73 (8): 1–26. https://www.jstatsoft.org/index.php/jss/article/view/v073i08.
Guttman, L. 1968. “A General Nonmetric Technique for Fitting the Smallest Coordinate Space for a Configuration of Points.” *Psychometrika* 33: 469–506.
Izenman, A. J. 2008. *Modern Multivariate Statistical Techniques*. Springer.
Kearsley, A. J., R. A. Tapia, and M. W. Trosset. 1994. “The Solution of the Metric STRESS and SSTRESS Problems in Multidimensional Scaling Using Newton’s Method.” TR94-44. Houston, TX: Department of Computational and Applied Mathematics, Rice University. https://apps.dtic.mil/sti/pdfs/ADA445621.pdf.
Kruskal, J. B. 1964a. “Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis.” *Psychometrika* 29: 1–27.
———. 1964b. “Nonmetric Multidimensional Scaling: a Numerical Method.” *Psychometrika* 29: 115–29.
Kruskal, J. B., and M. Wish. 1978. *Multidimensional Scaling*. Sage.
Lange, K. 2016. *MM Optimization Algorithms*. SIAM.
Mair, P., P. J. F. Groenen, and J. De Leeuw. 2022. “More on Multidimensional Scaling in R: smacof Version 2.” *Journal of Statistical Software* 102 (10): 1–47. https://www.jstatsoft.org/article/view/v102i10.
Malone, S. W., P. Tarazaga, and M. W. Trosset. 2002. “Better Initial Configurations for Metric Multidimensional Scaling.” *Computational Statistics and Data Analysis* 41: 143–56.

Mersmann, O. 2023. *microbenchmark: Accurate Timing Functions*. https://CRAN.R-project.org/package=microbenchmark.

Nesterov, Y., and B. T. Polyak. 2006. "Cubic Regularization of Newton Method and Its Global Performance." *Mathematical Programming* A108: 177–205.

Rothkopf, E. Z. 1957. "A Measure of Stimulus Similarity and Errors in some Paired-associate Learning." *Journal of Experimental Psychology* 53: 94–101.

Schwendinger, F., and H. W. Borchers. 1923. "CRAN Task View: Optimization and Mathematical Programming. Version 2023-09-28." https://CRAN.R-project.org/view=Optimization.

Takane, Y., H. A. L. Kiers, and J. De Leeuw. 1995. "Component Analysis with Different Sets of Constraints on Different Dimensions." *Psychometrika* 60: 259–80.

Torgerson, W. S. 1958. *Theory and Methods of Scaling*. New York: Wiley.

Trosset, M. W., and R. Mathar. 1997. "On the Existence on Nonglobal Minimizers of the STRESS Criterion for Metric Multidimensional Scaling." In *Proceedings of the Statistical Computing Section*, 158–62. Alexandria, VA: American Statistical Association.