Smacof at 50: A Manual
Part x: Acceleration

Jan de Leeuw

November 3, 2024

TBD

Table of contents

Note: This is a working manuscript which will be expanded/updated frequently. All suggestions for improvement are welcome. All Rmd, tex, html, pdf, R, and C files are in the public domain. Attribution will be appreciated, but is not required. The files can be found at https://github.com/deleeuw in the repositories smacofCode, smacofManual, and smacofExamples.

1 Introduction

In this paper we study minimization of the multidimensional scaling (MDS) loss function

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

over all $n \times p$ *configuration* matrices X. Following Kruskal (1964a), Kruskal (1964b) we call $\sigma(X)$ the *stress* of X. The symbol $:=$ is used for definitions.

In definition @ref(eq:sdef) matrices $W = \{w_{ij}\}$ and $\Delta = \{\delta_{ij}\}$ are known non-negative, symmetric, and hollow. They contain, respectively, *weights* and *dissimilarities*. The matrix-valued function D, with $D(X) = \{d_{ij}(X)\}$, contains *Euclidean distances* between the rows of X.

Throughout we assume, without loss of generality, that W is irreducible, that X is column-centered, and that Δ is normalized by

$$\frac{1}{2}\sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}^2 = 1. (\#eq:delnorm) \tag{2}$$

1.1 Notation

It is convenient to have some matrix notation for the MDS problem. We use the symmetric matrices A_{ij}, of order n, which have $+1$ for elements (i,i) and (j,j), -1 for elements (i,j) and (j,i), and zeroes everywhere else. Using unit vectors e_i and e_j we can write

$$A_{ij} := (e_i - e_j)(e_i - e_j)'. (\#eq:adef) \tag{3}$$

Following De Leeuw (1977) we define

$$\rho(X) := \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}d_{ij}(X) = \text{tr } X'B(X)X, (\#eq:rhodef) \tag{4}$$

where

$$B(X) := \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}r_{ij}(X)A_{ij}, (\#eq:bdef) \tag{5}$$

with

$$r_{ij}(X) = \begin{cases} d_{ij}^{-1}(X) & \text{if } d_{ij}(X) > 0, \\ 0 & \text{if } d_{ij}(X) = 0. \end{cases} (\#eq:rdef) \tag{6}$$

Also define

$$\eta^2(X) := \sum_{1\leq i<j\leq n}\sum w_{ij}d_{ij}^2(X) = \text{tr } X'VX, (\#eq : etadef) \tag{7}$$

where

$$V := \sum_{1\leq i<j\leq n}\sum w_{ij}A_{ij}.(\#eq : vdef) \tag{8}$$

Thus

$$\sigma(X) = 1 - \rho(X) + \frac{1}{2}\eta^2(X) = 1 - \text{tr } X'B(X)X + \frac{1}{2}\text{tr } X'VX.(\#eq : sform) \tag{9}$$

Both $B(X)$ and V are positive semi-definite and doubly-centered. Because of the irreducibility of W the matrix V has rank $n-1$, with only the constant vectors in its null space. Both ρ and η are positively homogeneous convex functions, with η being a norm on the space of column-centered configurations.

Note that ρ is continuous, but it is not differentiable at X if $d_{ij}(X) = 0$ for some (i, j) for which $w_{ij}\delta_{ij} > 0$. Because

$$|d_{ij}(X) - d_{ij}(Y)|^2 \leq \text{tr } (X-Y)'A_{ij}(X-Y) \leq 2\|X-Y\|^2(\#eq : lipschitz) \tag{10}$$

we see that ρ, although not differentiable at some points, is globally Lipschitz. η^2 is locally Lipschitz, and consequently so is σ.

1.2 The Guttman Transform

The *Guttman transform* of a configuration X, so named by De Leeuw and Heiser (1980) to honor the contribution of Guttman (1968), is defined as the set-valued map

$$\Phi(X) = V^+\partial\rho(X), (\#eq : phidef) \tag{11}$$

with V^+ the Moore-Penrose inverse of V and $\partial\rho(X)$ the subdifferential of ρ at X, i.e. the set of all Z such that $\rho(Y) \geq \rho(X) + \text{tr } Z'(Y-X)$ for all Y. Because ρ is homogeneous of degree one we have that $Z \in \partial\rho(X)$ if and only if tr $Z'X = \rho(X)$ and $\rho(Y) \geq \text{tr } Z'Y$ for all Y. For each X the subdifferential $\partial\rho(X)$, and consequently the Guttman transform, is compact and convex. The map $\partial\rho$ is also positively homogeneous of degree zero, i.e. $\partial\rho(\alpha X) = \partial\rho(X)$ for all X and all $\alpha \geq 0$. And consequently so is the Guttman transform.

We start with the subdifferential of the distance function between rows i and j of an $n \times p$ matrix. Straightforward calculation gives

$$\partial d_{ij}(X) = \begin{cases} \{d_{ij}^{-1}(e_i - e_j)(x_i - x_j)'\} & \text{if } d_{ij}(X) > 0, \\ \{Z \mid Z = (e_i - e_j)z' \text{ with } z'z \leq 1\} & \text{if } d_{ij}(X) = 0. \end{cases} (\#eq : dsubsef) \tag{12}$$

Thus if $d_{ij}(X) > 0$, i.e. if d_{ij} is differentiable at X, then $\partial d_{ij}(X)$ is a singleton, containing only the gradient at X.

From subdifferential calculus (Rockafellar (1970), theorem 23.8 and 23.9) the subdifferential of ρ is the linear combination

$$\partial\rho(X) = \sum_{1\leq i<j\leq n}\sum w_{ij}\delta_{ij}\partial d_{ij}(X)(\#eq : subdif) \tag{13}$$

Summation here is in the Minkovski sense, i.e. $\partial\rho(X)$ is the compact convex set of all linear combinations $\sum\sum_{1\leq i<j\leq n} w_{ij}\delta_{ij}z_{ij}$, with $z_{ij} \in \partial d_{ij}(X)$.

It follows that

$$\partial\rho(X) = B(X)X + Z(\#eq : rhosubdef) \tag{14}$$

with

$$Z \in \sum\sum\{w_{ij}\delta_{ij}\partial d_{ij}(X) \mid d_{ij}(X) = 0\}.(\#eq : zsubdef) \tag{15}$$

It also follows that

$$\partial\sigma(X) = VX - \partial\rho(X)(\#eq : sigsubdef) \tag{16}$$

Since σ is not convex the subdifferential $\partial\sigma(X)$ is the Clarke subdifferential (Clarke (1975)).

Now X is a Clarke stationary point of σ if $0 \in \partial\sigma(X)$, i.e. if and only if $X \in V^+\partial\rho(X)$. This means that stationary points are generalized fixed points of the Guttman transform. A necessary condition for σ to have a local minimum at X is that X is a Clarke stationary point. The condition is far from sufficient, however, since stationary points can also be saddle points or local maxima. De Leeuw (1993) shows that stress only has a single local maximum at the origin $X = 0$, but generally there are many saddle points.

This little excursion into nonsmooth and convex analysis is rarely needed in practice. We call a configuration X *friendly* if $d_{ij}(X) > 0$ for all (i, j) for which $w_{ij}\delta_{ij} > 0$. In De Leeuw (1988) such configurations were called *usable*, but that seems somewhat misleading, because configurations which are not "usable" in this sense can sometimes even be optimal. Unfortunately the set of friendly configurations is far from convex. If X is friendly, then so is $-X$, and halfway between the two is the zero configuration, which is very unfriendly.

The equation $d_{ij}(X) = 0$, or equivalently $x_i = x_j$, defines a subspace of configuration space, and a configuration is friendly if it is not in the union of the subspaces for all (i, j) for which $w_{ij}\delta_{ij} = 0$ (i.e. if it is in the intersection of their complements).

Suppose $d_{ij}(X) > 0$ and $d_{ij}(Y) > 0$. There is an α sich that $d_{ij}(\alpha X + (1-\alpha)Y) = 0$ iff $x_{is} - x_{js} = \lambda_{ij}(y_{is} - y_{js})$ with $\lambda_{ij} < 0$.

If X is friendly then $X + \epsilon Y$ is friendly for ϵ small enough.

If $X_1, \cdots, X_m$ are friendly then all Y in their convex hull are a.s. friendly.

If X is friendly the rank of $B(X)$ is $n-1$. More importantly, it was shown by De Leeuw (1984) that if σ has local minimum at X then X is friendly. At friendly configurations (and thus at local minima) σ is differentiable, and the subdifferential @ref(eq:sigsubdef) is a singleton, containing only the gradient. Stationary points then satisfy $X = V^+B(X)X$. In the case in which $w_{ij}\delta_{ij} = 0$ for some (i, j), however, then there can be local minima where σ is not differentiable. This happens, for example, in multidimensional unfolding (Mair, De Leeuw, and Wurzer (2015)).

By the definition of the subdifferential $Z \in \partial\rho(X)$ implies $\rho(X) \geq \text{tr } Z'X$ and $\rho(Y) \geq \text{tr } Z'Y$ for all Y. If $d_{ij}(X) > 0$ this follows directly from the Cauchy-Schwartz inequality

$$d_{ij}(Y) \geq d_{ij}^{-1}(X)\text{tr } X'A_{ij}Y.(\#eq : csineq) \tag{17}$$

Multiplying both sides by $w_{ij}\delta_{ij}$ and summing gives

$$\rho(Y) \geq \text{tr } Y'B(X)X(\#eq : rhoineq) \tag{18}$$

for all Y, with equality if $Y = X$. Note that we also have equality if $Y = \alpha X$ for some $\alpha \geq 0$, and more generally if for all $i < j$ with $w_{ij}\delta_{ij}(X) > 0$ we have equality in @ref(eq:csineq).

Using the Guttman transform we can use @ref(eq:rhoineq) to derive the basic smacof equality

$$\sigma(X) = 1 + \eta^2(X - \Phi(X)) - \eta^2(\Phi(X))(\#eq : smacofequality) \tag{19}$$

for all X and the basic smacof inequality

$$\sigma(X) \leq 1 + \eta^2(X - \Phi(Y)) - \eta^2(\Phi(Y))(\#eq : smacofinequality) \tag{20}$$

for all X and Y.

Taken together @ref(eq:smacofequality) and @ref(eq:smacofinequality) imply the *sandwich inequality*

$$\sigma(\Phi(Y)) \leq 1-\eta^2(\Phi(Y)) \leq 1+\eta^2(Y-\Phi(Y))-\eta^2(\Phi(Y)) = \sigma(Y).(\#eq : sandwich) \tag{21}$$

If Y is not a fixed point of Φ then the second inequality in the chain is strict and thus $\sigma(\Phi(Y)) < \sigma(Y)$. As we mentioned, the first inequality may not be strict.

It also follows from @ref(eq:sandwich) that $\eta^2(\Phi(Y)) \leq 1$. Thus the Guttman transforms are all in a convex and compact set, in fact an ellipsoid, containing the origin.

2 Basic Iteration

2.1 Function Values

The basic smacof algorithm generates the iterative sequence

$$X^{(k+1)} = \Phi(X^{(k)}), (\#eq : basic) \tag{22}$$

where it is understood that we stop iterating if $X^{(k)}$ is a fixed point. If $X^{(k)}$ is not a fixed point it follows from @ref(eq:sandwich) that $\sigma(X^{(k+1)}) < \sigma(X^{(k)})$. Thus, without any additional assumptions, and using basically only the Cauchy-Schwartz inequality, the algorithm either stops at a fixed point or produces a strictly decreasing sequence of loss function values. Since stress is bounded below by zero the sequence $\sigma(X^k)$ converges to, say, σ_∞.

It was clear from the beginning (De Leeuw and Heiser (1977)) that the case $p = 1$ is special. The smacof algorithm always stops at a fixed point after a finite, and usually small, number of iterations. This is not as good as it sounds, because there are many (up to $n!$) local minima. The unidimensional problem (Mair and De Leeuw (2015)) is essentially combinatorial, and requires a specialized treatment. In this paper we assume throughout that $p \geq 2$.

The original derivation of the smacof algorithm (De Leeuw (1977)) used the theory of maximization a ratio of norms discussed by Robert (1967). Later derivations (De Leeuw and Heiser (1980), De Leeuw (1988)) used the fact that @ref(eq:smacofinequality) defines a majorization scheme for stress. Convergence then follows from the general *majorization principle* (these days mostly known as the *MM principle*), introduced in De Leeuw (1994). A recent overview of the MM approach is Lange (2016).

It was also realized early on that the smacof algorithm was a special case of the the difference-of-convex functions algorithm (DCA), introduced by Pham Dinh Tao around 1980. Pham Dinh also started his work in the context of ratio's of norms, using Robert's fundamental ideas. Around 1985 he generalized his approach to minimizing DC functions of the form $h = f - g$, with both f and g convex. The basic idea is to use the subgradient inequality $g(x) \geq g(y) + z'(x - y)$, with $z \in \partial g(x)$, to construct the majorization $h(x) := f(x) - g(y) - z'(x - y)$. Now h is obviously convex in x. The DC algorithm then chooses the successor of y as the minimizer of this convex majorizer over x. In smacof the role of f is played by η^2 and the role of g by ρ. DCA is applied to MDS in Le Thi and Tao (2001). Extensive recent surveys of the DC/DCA approach are Le Thi and Tao (2018) and Le Thi and Tao (2024).

Thus the smacof algorithm is both MM and DCA, which means that it inherits all results that have been established for these more general classes of algorithms. But additional results can be obtained by using the special properties of the stress loss function and the smacof iterations. In the DCA context, for example, the convex subproblem that must be solved by smacof in each step is quadratic, and has the closed form solution provided by the Guttman transform.

The loss function values are a bounded decreasing, and thus converging, sequence. De Leeuw (1988) derives some additional smacof-specific results. Using up-arrows and down-arrows for monotone convergence

- $\rho(X^{(k)}) \uparrow \rho_\infty$,
- $\eta^2(X^{(k)}) \uparrow \eta^2_\infty = \rho_\infty$,
- $\sigma(X^{(k)}) \downarrow \sigma_\infty = 1 - \rho_\infty$,

and, last but not least, the sequence $\{X^{(k)}\}$ is *asymptotically regular*, i.e.

$$\omega^2(X) := \eta^2(X^{(k+1)} - X^{(k)}) \rightarrow 0.(\#eq : etaconv) \tag{23}$$

This last, very important, result follows because

$$\eta^2(X^{(k+1)} - X^{(k)}) = \eta^2(X^{(k+1)}) + \eta^2(X^{(k)}) - 2\rho(X^{(k)}), (\#eq : etanull) \tag{24}$$

which converges to zero because $\eta^2_\infty = \rho_\infty$. Note that these results are based completely on the Cauchy-Schwartz inequality and are consequently true for the general iteration $X^{(k+1)} \in V^+\partial\rho(X^{(k)})$, without assuming differentiability.

If

$$\partial\sigma(X^{(k)}) = VX^{(k)} - \partial\rho(X^{(k)}) = V(X^{(k)} - X^{(k+1)})$$

Consequently @ref(eq:etaconv) can equivalently be written as

$$\|\partial\sigma(X^{(k)})\| \rightarrow 0$$

Strictly spoken, the results so far prove convergence of the scalar sequences $\{\rho(X^{(k)})\}$, $\{\eta^2(X^{(k)})\}$ and $\{\eta^2(X^{(k+1)} - X^{(k)})\}$ associated with the iterations, and they do not prove convergence of the sequence $X^{(k)}$. But in De Leeuw (1988) I argue that these scalar convergence results are sufficient from a practical point of view. If we define an ϵ-fixed-point as any configuration X with $\eta(X - \Phi(X)) < \epsilon$ then smacof produces such an ϵ-fixed-point in a finite number of steps.

Also, we can use the general convergence result in theorem 3.1 of Meyer (1976) to get results about $\{X^{(k)}\}$. Because

- the subdifferential is a upper semi-continuous (closed) map,
- all iterates are in the compact set $\eta^2(X) \leq 1$, and
- Φ is strictly monotonic (decreases stress at non-fixed points),

it follows that the sequence $\{X^{(k)}\}$ has accumulation points (converging subsequences) and that

- all accumulation points are fixed points, and

- all accumulation points have the same function value σ_∞.

Moreover, from asymptotic regularity and theorem 26.1 of Ostrowski (1973),

- either the sequence $\{X^{(k)}\}$ converges or its accumulation points form a continuum (a connected and compact set).

In order to prove actual Cauchy convergence, additional conditions are needed. Meyer (1976) proves convergence if the number of fixed points with function value σ_∞ is finite, or if the sequence has an accumulation point that is an isolated fixed point. Both these conditions are not met in MDS, because of rotational indeterminacy. If X_∞ is a fixed point, then all elements of the continuum of rotations of X_∞ are fixed points.

It should also be mentioned that smacof can converge to stationary points that are not local minima (and thus saddle points). Suppose all weights are equal to one, $\delta_{12} > 0$, and $\delta_{1j} = \delta_{2j}$ for all $j > 2$. If $d_{12}(X) = 0$ then also $d_{12}(\Phi(X)) = 0$, and $d_{12}(X)$ will be zero for all iterates, and thus for all subsequential limits, which consequently cannot be local minima. Another example uses the result that yet another necessary consition for a local minimum is that X has full column rank. Suppose we start iterations at $(X \mid 0)$, i.e. X with some columns of zeroes added. All updates will also have this form, and convergence again cannot be to a local minimum. This last example can be generalized to any X with rank less than p, because all updates will then also have rank less than p. As a consequence if $q > p$ all local minima for $p-$dimensional MDS are saddle points for $q-$dimensional MDS.

In two very recent impressive papers Ram and Sabach (2024 (in press)) and Robini, Wang, and Zhu (2024) use the powerful Kurdyka-Łojasiewicz (KL) framework (Bolte, Daniilidis, and Lewis (2007), Bolte, Sabach, and Teboulle (2014)) to prove actual global convergence of the smacof iterates to a fixed point. We shall use the more classical local convergence analysis, based on the differentiability of the Guttman transform.

We apply basic iterations to the two-dimensional MDS analysis of the classical color-circle example from Ekman (1954), which has $n = 14$ points. In our numerical examples we always use weights equal to one. We always start with the classical Torgerson-Gower solution and we stop if $\sigma(X^{(k)}) - \sigma(X^{(k+1)}) < 1e-15$. We distinguish $f-convergence$, which happens if the stress value from one iteration to the next changes less than a small ϵ, and $x-convergence$, which happens if $\eta(\sigma(X^{(k)}) - \sigma(X^{(k+1)})$ is less than another small ϵ.

The fit for the Ekman example is very good and convergence is rapid. In iteration 56, the final iteration, stress is 2.1114112739076. The change CHNG $\eta(X^{(k)} - X^{(k-1)})$ in the final iteraton is 2.26054683805646e-16. The estimated asymptotic rate of convergence or *EARC* is the CHNG divided by the CHNG of the previous iteration. In the final iteration of this analysis it is 0.766978439824377.

2.2 Asymptotic Rate of Convergence

In order to study the asymptotic rate of convergence (ARC) of smacof, we have to compute the derivative of the Guttman transform and its eigenvalues (Ortega and Rheinboldt (1970), chapter 10). Thus we assume we are in the neighborhood of a configuration where the Guttman transform is (infinitely many times) differentiable, for example near a local minimizer.

The derivative of Φ at X, first given in De Leeuw (1988), is the linear transformation $\mathcal{D}\Phi_X$, mapping the space of column-centered $n \times p$ matrices into itself. Its value at matrix H is equal to

$$\mathcal{D}\Phi_X(H) = V^+ \sum w_{ij} \frac{\delta_{ij}}{d_{ij}(X)} \left\{ A_{ij}H - \frac{\text{tr } X'A_{ij}H}{\text{tr } X'A_{ij}X} A_{ij}X \right\} .(\#eq:jacobian) \quad (25)$$

It follows that $\mathcal{D}\Phi_X(X) = 0$ for all X and the derivative has at least one zero eigenvalue. If we think of equation @ref(eq:jacobian) as a linear transformation on the space of all $n \times p$ matrices, then there are an additional p zero eigenvalues corresponding with translational invariance. If we define @ref(eq:jacobian) on the space of column-centered matrices, then those zero eigenvalues disappear.

If S is anti-symmetric and $H = XS$ then tr $X'A_{ij}H = 0$ and thus $\mathcal{D}\Phi_X(XS) = \Phi(X)S$. If in addition X is a fixed point then $\mathcal{D}\Phi_X(XS) = XS$, which means that at a fixed point $\mathcal{D}\Phi_X$ has $\frac{1}{2}p(p-1)$ eigenvalues equal to one. These correspond to the rotational indeterminacy of the MDS problem and the smacof iterations. It also follows from $\mathcal{D}\Phi_X(XS) = \Phi(X)S$ that for all X and all anti-symmetric S the inner product tr $\Phi(X)'V\mathcal{D}\Phi_X(XS)$ is zero, i.e. $\Phi(X)$ is orthogonal to the $\frac{1}{2}p(p-1)$ dimensional subspace of all $\mathcal{D}\Phi_X(XS)$.

Since $\Phi(X) = V^+\mathcal{D}\rho(X)$ the derivative of the Guttman transform has a simple relationship with the second derivatives of ρ. The second derivative, again from De Leeuw (1988), is giuven by the quadratic form

$$\mathcal{D}^2\rho_X(H,H) = \sum w_{ij} \frac{\delta_{ij}}{d_{ij}(X)} \left\{ \text{tr } H'A_{ij}H - \frac{(\text{tr } H'A_{ij}X)^2}{d_{ij}^2(X)} \right\} .(\#eq:hessian) \quad (26)$$

Since ρ is convex, all eigenvalues of $\mathcal{D}^2\rho_X$, and thus of $\mathcal{D}\Phi_X$, are real and nonnegative. It also follows that if G and H are eigenvectors of $\mathcal{D}\Phi_X$ with different eigenvalues then tr $G'VH = 0$. In addition we see from equation @ref(eq:hessian) that $0 \lesssim \mathcal{D}^2\rho_X \lesssim B(X)$ in the Loewner sense. Since $\mathcal{D}^2\sigma_X = V - \mathcal{D}^2\rho_X$, and we have $\mathcal{D}^2\sigma_X \gtrsim 0$ at a local minimum, it follows that $\mathcal{D}\Phi_X \lesssim I$. Thus all eigenvalues of the derivative $\mathcal{D}\Phi_X$ at a local minimum X are between zero and one.

We compute the Jacobian corresponding to the derivative $\mathcal{D}\Phi_X$ in two ways. First with a loop over $i = 1, \cdots, n$ and $s = 1, \cdots, p$ by

setting H equal to each $e_i e_s'$ in turn in formula @ref(eq:jacobian). Second, just to be sure, by using the jacobian function from the numDeriv package (Gilbert and Varadhan (2019)). If the two results agree, we use the one based on @ref(eq:jacobian).

The eigenvalues of the derivative $\mathcal{D}\Phi_X$ at the solution are

```
 [1]   +1.0000000000   +0.7669964993   +0.7480939418   +0.7185926294
 [5]   +0.7007452309   +0.6920114813   +0.6859492533   +0.6593334529
 [9]   +0.6541779410   +0.6477573343   +0.6237683213   +0.6178713316
[13]   +0.5735285951   +0.5483330653   +0.5260355535   +0.5112510731
[17]   +0.5064703617   +0.5059294792   +0.4919752630   +0.4827646555
[21]   +0.4782034995   +0.4757907675   +0.4682965893   +0.4619226491
[25]   +0.4559704884   +0.0000000000   +0.0000000000   -0.0000000000
```

Note that the largest non-trivial eigenvalue, which is another and usually better estimate of the ARC, is equal to the EARC in the final iteration.

2.3 Modifications

As De Leeuw (1988) mentions, we cannot directly apply the basic point-of-attraction theorem 10.1.3 and the equally basic linear convergence theorem 10.1.4 from Ortega and Rheinboldt (1970), because at a fixed point of smacof there are $\frac{1}{2}p(p-1$ eigenvalues equal to one.

$$\Xi(X) = \Pi(\Phi(X))$$
$$\mathcal{D}\Xi_X(H) = \mathcal{D}\Pi_{\Phi(X)}(D\Phi_X(H))$$
$$\Xi(X) = \Pi(X, \Phi(X))$$
$$\mathcal{D}\Xi_X(H) = \Pi(X+H, \Phi(X)+\mathcal{D}\Phi_X(H)) = \mathcal{D}_1\Pi_{X,\Phi(X)}(H)+\mathcal{D}_2\Pi_{X,\Phi(X)}(\mathcal{D}\Phi_X(H)).$$

3 Orthogonalization

3.1 Modification

One way around this problem (De Leeuw (2019)) is to rotate each update to orthogonality, i.e. to principal components. Thus the update formula becomes $\Xi(X) = \Pi(\Phi(X))$, with $\Pi(X) = XL$, where L are the right singular vectors of X. The reasoning is simple. If Ω is any differentiable mapping with $\Omega(XK) = \Omega(X)K$ for all orthonormal K then

$\mathcal{D}\Omega_X(XA) = XA$ for all anti-symmetric A. But if $\Omega(XK) = \Omega(X)$ for all orthonormal K then $\mathcal{D}\Omega_X(XA) = 0$ for all anti-symmetric A.

With orthogonality restrictions we can expect isolated local minima, where the largest eigenvalue of the algorithmic map is strictly less than one. Such local minima are points of attraction, which means convergence to that point if the iterations get close enough. It also means that if we assume that there is only a finite number of these orthogonal stationary points, then the smacof algorithm converges globally to one of them.

3.2 Function Values

This modified algorithm generates the same sequence of ρ, η, and σ values as basic smacof.

But $\mathfrak{e}$ is a different sequence In fact $\mathfrak{e}(XL) = \eta^2(X) + \eta^2(\Phi(X)) - 2 \text{ tr } X'V\Phi(X)$

Moreover $\Xi^n(X) = \Pi(\Phi^n(X))$, which means that we can find any term of the orthogonalized sequence by orthogonalizing the corresponding term in the basic sequence. Thus, in actual computation, there is no need to orthogonalize, we may as well compute the basic sequence and orthogonalize after convergence.

In iteration 55, the final iteration, stress is 2.1114112739076. The CHNG is 3.864164805803e-16 and the EARC is 0.766992047059491.

3.3 Asymptotic Rate of Convergence

Reference: De Leeuw (2021), section 5.4.5.2, De Leeuw (2019)

Suppose the singular value decomposition of X is $X = K\Lambda L'$. We transform X to $\Pi(X) = XL = K\Lambda$.

Thus ref

$$\mathcal{D}\Pi_X(H) = HL + K\Lambda S.(\#eq : finalpca) \tag{27}$$

where S is anti-symmetric, with off-diagonal elements

$$s_{ij} = -\frac{\lambda_i u_{ij} + \lambda_j u_{ji}}{\lambda_i^2 - \lambda_j^2}, (\#eq : spca) \tag{28}$$

where $U := K'HL$.

If X is orthogonal, then $L = I$ and $X'X = \Lambda^2$.

- If $H = XA$ then $U = \Lambda L'AL$, $\Lambda U = \Lambda^2 L'AL$ and $U'\Lambda = L'A'L\Lambda^2$. If A is anti-symmetric then $\Lambda U + U'\Lambda = \Lambda^2 L'AL - L'AL\Lambda^2$ and $\{\Lambda U + U'\Lambda\}_{ij} = (\lambda_i^2 - \lambda_j^2)\{L'AL\}_{ij}$. Thus $S = -L'AL$ and $\mathcal{D}\Pi_X(XA) = 0$.
- If $H = K\Lambda^{-1}A$ with A anti-symmetric, then $U = \Lambda^{-1}AL$ $\Lambda U + U'\Lambda = AL + L'A'$ If $L = I$ then $S = 0$.
- If $H = K\Lambda^{-1}D$ with D diagonal, then $U = \Lambda^{-1}DL$ and $\Lambda U + U'\Lambda = DL + L'D$. If $L = I$ this is diagonal, and thus $S = 0$.

Also $XS = 0$ if and only if $S = 0$ if and only if nondiag$(X'H)$ is anti-symmetric. Thus true for $H = X_\perp B$ as well as for H with $X'H$ us diagonal. If $H = XU$ then $l_i'(H'X + X'H)l_j$−{U'□+□U}*{ij}*=*u*{ji}□_j+□*iu*{ij}$ Eigenvalues of $\mathcal{D}\Pi_X(H)$. $XL = K\Lambda^{\frac{1}{2}}$ $H = K\Lambda^{\frac{1}{2}}A + K_\perp B$.

$$L'(H'X + X'H)L$$

$X = K\Lambda L'$ $H = KAL' + K_\perp BL'$ $L'X'HL = \Lambda A$ $\Lambda A + A\Lambda = 0$

We next compute the derivative of Ξ. By the chain rule

$$\mathcal{D}\Xi_X(H) = \mathcal{D}\Pi_{\Phi(X)}(\mathcal{D}\Phi_X(H)).(\#eq : chain) \tag{29}$$

Thus, from equations @ref(eq:chain) and @ref(eq:finalpca)

$$\mathcal{D}\Xi_X(H) = \mathcal{D}\Phi_X(H)L + \Phi(X)LS(\#eq : xideriv) \tag{30}$$

with L and S computed from the singular value decomposition of $\Phi(X)$.

At a fixed point of Ξ we have both $\Phi(X) = X$ and $\Pi(X) = X$, and consequently $L = I$ and $X'X = \Lambda$. (Is this true ? We could have $\Phi(X) = XK$, because then still $\Xi(X) = X$)

Equation @ref(eq:xideriv) becomes

$$\mathcal{D}\Xi_X(H) = \mathcal{D}\Phi_X(H) + XS, (\#eq : xiderivfixed) \tag{31}$$

where now

$$s_{ij} = -\frac{(H'X + X'H)_{ij}}{\lambda_i - \lambda_j}.(\#eq : sdeffixed) \tag{32}$$

At a fixed point X the eigenvectors H of $\mathcal{D}\Phi_X$ with eigenvalue one are of the form $H = XA$ with A any anti-symmetric matrix. From @ref(eq:xiderivfixed)

$$\mathcal{D}\Xi_X(XA) = XA + XS, (\#eq : xiderivasym) \tag{33}$$

where

$$s_{ij} = -\frac{(A'\Lambda + \Lambda A)_{ij}}{\lambda_i - \lambda_j} = -a_{ij}.(\#eq : sdefasym) \tag{34}$$

Thus $\mathcal{D}\Xi_X(XA) = 0$ and the unit eigenvalue has been replaced by a zero eigenvalue.

Moreover, at a fixed point X of Φ, if H is an eigenvector of $\mathcal{D}\Phi_X$ with eigenvalue $\lambda < 1$, then $H + \lambda^{-1}XS$ is an eigenvector of $\mathcal{D}\Xi_X$ with eigenvalue λ. This follows from

$$\mathcal{D}\Xi_X(H + \lambda^{-1}XS) = \mathcal{D}\Xi_X(H) = \lambda H + XS = \lambda(H + \lambda^{-1}XS).(\#eq : evaltrans) \tag{35}$$

Thus, except for the trivial unit eigenvalue which becomes zero, both sets of eigenvalues are the same, and so is the ARC.

$$\mathcal{D}\Xi_X(H) = \mathcal{D}\Phi_X(H) + XS$$
$$\mathcal{D}\Xi_X(X) = \mathcal{D}\Phi_X(X) + XS = XS$$

If $\mathcal{D}\Phi_X(H) = \lambda H$ then

$$\text{tr } X'V\mathcal{D}\Xi_X(H) = \lambda \text{ tr } X'VH + \text{tr } X'VXS = 0$$

which implies $\lambda = 1$. The eigenvalues of the Jacobian are

```
 [1]   +0.7669964950   +0.7480939419   +0.7185926296   +0.7007452320
 [5]   +0.6920114814   +0.6859492533   +0.6593334535   +0.6541779411
 [9]   +0.6477573344   +0.6237683215   +0.6178713316   +0.5735285953
[13]   +0.5483330652   +0.5260355534   +0.5112510730   +0.5064703617
[17]   +0.5059294792   +0.4919752629   +0.4827646555   +0.4782035015
[21]   +0.4757907660   +0.4682965891   +0.4619226497   +0.4559704891
[25]   -0.0000000000   -0.0000000000   +0.0000000000   +0.0000000000
```

Orthogonalization gives the same EARC as the basic sequence, but the Jacobian of Ξ at a local minimum does not have the unit eigenvalues any more. They are replaced by zeroes, reflecting the fact that we are iterating on the nonlinear manifold or orthogonal column-centered matrices.

It is now sufficient for local linear convergence to assume that the largest eigenvalue of the Jacobian at the solution is strictly less than one, or alternatively assume that one of the accumulation points is an isolated local minimum.

4 Subspace Rotation

4.1 Modification

Instead of orthogonality we can also require X to be in the subspace of all lower triangular column-centered $n \times p$ matrices (which means $x_{ij} = 0$ for all $i < j$). This also identifies X in the manifold of rotated solutions.

In subspace rotation we use a rotation of X to lower triangular form. We use the same notation as in the previous section, overloading some symbols. The transformation of the update is again Π and the transformed update is Ξ. Thus $\Xi(X) = \Pi(\Phi(X))$.

Suppose X_1 are the first p rows of X, and $X_1' = QR$ is the QR-decomposition of the transpose. Thus Q is square orthonormal and R is upper triangular. Then $X_1 Q = R'Q'Q = R'$, which is lower triangular, as desired. Note that the transformation to lower triangular form only uses the first p rows of X, and does not depend on the other $(n-p) \times p$ elements.

4.2 Function Values

The results are pretty much the same as for the rotation to principal components in the previous section.

In iteration 55, the final iteration, stress is 2.1114112739076. CHNG $\eta(X^{(k)} - X^{(k+1)})$ is 3.84673519544424e-16 and EARC is 0.766987804354728.

4.3 Asymptotic Rate of Convergence

To compute the derivative of Π we first compute the derivative of the QR decomposition of a square non-singular matrix X, using the results of De Leeuw (2023). Perturb $X = QR$, with Q square orthonormal and R upper triangular, to $X + H = (Q + P)(R + S)$. Collecting the first order terms gives

$$H = QS + PR. (\#eq : qrfirst) \tag{36}$$

Because $(Q + P)'(Q + P) = I$ we see that $Q'P + P'Q = 0$, and thus $P = QA$ with A anti-symmetric.

$$H = QS + QAR. (\#eq : qrsecond) \tag{37}$$

Pre-multiplying by Q' and post-multiplying by R^{-1} gives

$$A = Q'HR^{-1} - SR^{-1} (\#eq : qrthird) \tag{38}$$

Both S and R^{-1} are upper triangular, and so is their product. Suppose lt replaces the upper triangular part (including the diagonal) of a matrix by zeroes. Then, from @ref(eq:qrthird),

$$\text{lt}(A) = \text{lt}(Q'HR^{-1})(\#eq : qrfourth) \tag{39}$$

and by anti-symmetry the upper-triangular part of A is minus the transpose of $\text{lt}(A)$. This gives the derivative

$$\mathcal{D}Q_X(H) = QA.(\#eq : qrfifth) \tag{40}$$

In our rotation procedure we apply QR to X_1, which is the transpose of the leading $p \times p$ submatrix of the $n \times p$ matrix X (assumed to be non-singular). Let H_1 be the transpose of the corresponding submatrix of H. Then **??**eq:qrfifth) applies with Q and A computed at X_1 and H_1. Thus

$$\mathcal{D}\Pi_X(H) = HQ + XQA$$

(Eigenvalues Jacobian one zero, one negative (equal to trace), 13 minus one, 13 plus one)

and thus

$$\mathcal{D}\Xi_X(H) = \mathcal{D}\Pi_{\Phi(X)}(D\Phi_X(H)) = HQ + \Phi(X)QA$$

with Q and A now computed at the submatrices $\{\Phi(X)\}_1$ and $\{D\Phi_X(H)\}_1$.

At a fixed point $\Phi(X) = X$ and both $Q = I$ and $R = I$. Thus

$$\mathcal{D}\Xi_X(H) = H + XA.$$

5 Subspace Restriction

5.1 Modification

Method two uses the theory of constrained smacof of De Leeuw and Heiser (1980). In this case this means computing the Guttman update and then projecting it on the subspace of lower triangular matrices. We create p column-centered matrices Y_s, $s = 1, \cdots, p$, of dimension $n \times (n-s)$, that satisfy $Y_s'VY_s = I$ and have their first $s-1$ rows equal to zero. Now column s of X is restricted to be of the form $x_s = Y_s\theta_s$. The transformation Π, for dimension s, is

$$\Pi(X)_s = Y_sY_s'Vx_s(\#eq : subspace) \tag{41}$$

Alt: Minimize

$$\text{tr}\,(X_1 - Y_1)'V_{11}(X_1 - Y_1) + 2\text{tr}\,(X_1 - Y_1)'V_{12}(X_2 - Y_2) + \text{tr}\,(X_2 - Y_2)'V_{22}(X_2 - Y_2)$$

requiring that Y_1 is upper-triangular. Now

$$X_2 - Y_2 = V_{22}^{-1}V_{21}(X_1 - Y_1)$$

and thus it suffices to minimize

$$\text{tr } (X_1 - Y_1)'V_{1|2}(X_1 - Y_1)$$

with $V_{1|2}$ the Schur complement $V_{11} - V_{12}V_{22}^{-1}V_{21}$ over upper-triangular Y.

Alt: direct

$$d_{ij}(\theta) = \sqrt{\sum_{s=1}^{p} \theta_s' Y_s' A_{ij} Y_s \theta_s}.$$

$$\rho(\theta) = \theta_s' Y_s' B(\theta) Y_s \theta_s \geq \theta_s' Y_s' B(\tilde{\theta}) Y_s \tilde{\theta}_s.$$

$$\theta_s^{(k+1)} = Y_s' B(\theta^{(k)}) Y_s \theta_s^{(k)}$$

5.2 Function Values

The subspace restrictions have a devastating effect on the rate of convergence of the smacof iterations.

Although the final stress is the correct 2.11141127390763, and the final CHNG is 5.95185745918126e-16, it takes 443 iterations and the EARC is 0.962237154391956.

5.3 Asymptotic Rate of Convergence

In this case computing the derivatives of @ref(eq:subspace) is very simple indeed. We

```
 [1]   +0.9622371565   +0.7669637116   +0.7480360811   +0.7105397440
 [5]   +0.7007452013   +0.6920054108   +0.6859241126   +0.6593334167
 [9]   +0.6541666413   +0.6476391845   +0.6234727079   +0.6172949237
[13]   +0.5586355758   +0.5478680315   +0.5260205816   +0.5111279921
[17]   +0.5064290866   +0.4929119272   +0.4843668597   +0.4783221142
[21]   +0.4758041101   +0.4686440280   +0.4665512407   +0.4575763680
[25]   -0.0000000000   -0.0000000000   -0.0000000000   +0.0000000000
```

6 Relaxed

6.1 Modification

De Leeuw and Heiser (1980) first suggested the "relaxed" update

$$\Psi(X) := 2\Phi(X) - X.(\#eq : relax) \tag{42}$$

The reason for recommending @ref(eq:relax) is two-fold. First, the smacof inequality @ref(eq:smacofinequality) says

$$\sigma(X) \leq 1 + \eta^2(X - \Phi(Y)) - \eta^2(\Phi(Y)).(\#eq : smaineq) \tag{43}$$

If $X = \alpha\Phi(Y) + (1-\alpha)Y$ then this becomes

$$\sigma(\alpha\Phi(Y) + (1-\alpha)Y) \leq 1 + (1-\alpha)^2\eta^2(Y - \Phi(Y)) - \eta^2(\Phi(Y)) \tag{44}$$

If $(1-\alpha)^2 \leq 1$ then

$$1 + (1-\alpha)^2\eta^2(Y - \Phi(Y)) - \eta^2(\Phi(Y)) \leq 1 + \eta^2(Y - \Phi(Y)) - \eta^2(\Phi(Y)) = \sigma(Y) \tag{45}$$

Thus updating with $X^{(k+1)} = \alpha\Phi(X^{(k)}) + (1-\alpha)X^{(k)}$ is a stricly monotone algorithm as long as $0 \leq \alpha \leq 2$.

But if $\alpha = 2$ an $Y = \lambda X$!!

The second reason for choosing the relaxed update @ref(eq:relax) given by De Leeuw and Heiser (1980) is that its asymptotic convergence rate is

$$\max_s |2\lambda_s - 1| = \max(2\lambda_{\text{max}} - 1, 1 - 2\lambda_{\text{min}}). \tag{46}$$

De Leeuw and Heiser (1980) then somewhat carelessly assume that this is equal to $2\lambda_{\text{max}} - 1$ and argue that if $\lambda_{\text{max}} = 1 - \epsilon$ with ϵ small, as it usually is in MDS, then

$$2\lambda_{\text{max}} - 1 = 1 - 2\epsilon \approx (1-\epsilon)^2 = \lambda_{\text{max}}^2, \tag{47}$$

so that the relaxed update requires approximately half the number of iterations of the basic update. Despite the somewhat sloppy reasoning, the approximate halving of the number of iterations is often observed in practice.

6.2 Function Values

It turns out (Groenen, Glunt, and Hayden (1996), De Leeuw (2006)), however, that applying the relaxed update has some unintended consequences, which basically imply that it should never be used without additional precautions. Let's take a look at the Ekman results.

```
itel   25 sold 3.994627066568261 snew 3.994627066568263 chng 3.766431585321326 labd 1.
```

In iteration 25, the final iteration, stress is 3.99462706656826. The "change" $\eta(X^{(k)}-X^{(k+1)})$ is 3.76643158532133 and the estimate of the asymptotic convergence ratio, the "change" divided by the "change" of the previous iteration, is 1.

The loss function values converge and the number of iterations is reduced from 57 to 23. But we see that $\eta(X^{(k+1)}-X^{(k)})$ does not converge to zero, and that σ_k converges to a value which does not correspond to a local minimum of σ.

If we check the conditions of theorem 3.1 in Meyer (1976) we see that, although the algorithmic map is closed and the iterates are in a compact set, Ψ is not strictly monotone at some non-fixed points. The problem was first discussed in Groenen, Glunt, and Hayden (1996). Suppose X is a fixed point and $\tau \neq 1$. Then $\tau\overline{X}$ is not a fixed point of Ψ, because $\Psi(\tau\overline{X}) = (2-\tau)\overline{X}$. And

$$\sigma(\tau\overline{X}) = 1-\tau\rho(\overline{X})+\frac{1}{2}\tau^2\eta^2(\overline{X}) = 1-\frac{1}{2}\tau(2-\tau)\rho(\overline{X}) = \sigma((2-\tau)\overline{X})(\#eq:sigmatau) \tag{48}$$

Thus the algorithm has convergent subsequences which may not converge to a fixed point of Ψ (and thus of Φ). And indeed, the computational results show that the method produces a sequence $X^{(k)}$ with two subseqences. If $\overline{X}$ is a fixed point of Φ then there is a $\tau > 0$ such that the subsequence with k even converges to $\tau\overline{X}$ while the subsequence with k odd converges to $(2-\tau)\overline{X}$.

This suggests a simple fix. After convergence of the funcion values we make a final update using Φ instead of Ψ. Computationally this is simple to do. If the final iteration updates $X^{(k)}$ to $X^{(k+1)} = \Psi(X^{(k)})$ then set the final solution to the average $\frac{1}{2}(X^{(k)} + X^{(k+1)})$. Making this adjustment at the end of the Ekman sequence gives us a final stress equal to 2.11141127390763.

6.3 Asymptotic Rate of Convergence

The eigenvalues of the Jacobian are

7 Doubling

7.1 Modification

The analysis in the previous section suggest the update function Ψ^2, i.e.

$$\Xi(X) = \Psi(\Psi(X)).$$

7.2 Function Values

The algorithm generates the same sequence of function values and configurations as the relaxed algoroithm Ψ. The only difference is that we test for convergence, and compute CHNG and EARC, every other iteration.

With Ψ^2 the algorithm is everywhere strictly monotonic and does converge to a fixed point. But not all problems associated with Ψ have disappeared. If X is a stationary point of σ, and thus a fixed point of Φ, Ψ, and Ψ^2, then τX is a fixed point of Ψ^2 for all $\tau > 0$. Thus we cannot exclude the possibility that the sequence converges to a fixed point proportional to X, but not equal to X.

Here are the results for the Ekman data if we use Ψ^2.

In iteration 13, the final iteration, stress is 3.99462706656826. The CHNG is 3.76583425614514e-16 and the EARC is 0.273802752120992.

Again we need some adjustment. A final update using Φ will do the trick. After this adjustment stress is 3.99462706656826

7.3 Asymptotic Rate of Convergence

$$\mathcal{D}\Psi^2_X(H) = \mathcal{D}\Psi_{\Psi(X)}(\mathcal{D}\Psi_X(H))$$

The asymptotic convergence rate is

$$\max_s (2\lambda_s - 1)^2 = \max\{(2\lambda_{\max} - 1)^2, (2\lambda_{\min} - 1)^2\}$$

.

8 Dilation

8.1 Modification

De Leeuw (2006) discusses some other ways to fix the relaxed update problem. The first one, borrowed from Groenen, Glunt, and Hayden (1996), defines Defines

$$\Pi(X) := \frac{\rho(X)}{\eta^2(X)}X$$

and

$$\Xi(X) := \Pi(\Psi(X))$$

Function Values

Here are the results for the Ekman data if we use dilation.

In iteration 26, the final iteration, stress is 2.1114112739076. The CHNG is 1.77621337696992e-16 and the EARC is 0.533991473995601.

8.2 Asymptotic Rate of Convergence

First, differentiate Π of ... Using the product and quotient rules for differentiation we find

$$\mathcal{D}\Pi_X(H) = \frac{\rho(X)}{\eta^2(X)}H + \frac{\eta^2(X)\mathcal{D}\rho_X(H) - \rho(X)\mathcal{D}\eta_X^2(H)}{\eta^4(X)}X$$

Using

$$\mathcal{D}\rho_X(H) = \text{tr } H'B(X)X$$
$$\mathcal{D}\eta_X^2(H) = 2\text{tr } H'VX$$

this becomes

$$\mathcal{D}\Pi_X(H) = \frac{\rho(X)}{\eta^2(X)}H + \text{tr } H'\left\{\frac{\eta^2(X)B(X)X - 2\rho(X)VX}{\eta^4(X)}\right\}X$$

The chain rule says

$$\mathcal{D}\Xi_X(H) = \mathcal{D}\Psi_X(H) - \frac{\text{tr } X'V\mathcal{D}\Psi_X(H)}{\text{tr } X'VX}X$$

Since $\mathcal{D}\Psi_X(X) = -X$ we have

$$\mathcal{D}\Xi_X(X) = -X + \frac{\text{tr } X'VX}{\text{tr } X'VX}X = 0$$

Thus the offending eigenvector X of $\mathcal{D}\Psi$ is eliminated.

More generally, if $\mathcal{D}\Psi_X(H) = \lambda H$ with $H \neq X$ then $\mathcal{D}\Xi_X(H - X) = \lambda(H - X)$, and thus $\mathcal{D}\Xi$ has the same eigenvalues as $\mathcal{D}\Psi$.

9 Stabilizing

9.1 Modification

Another strategy

$$\Xi(X) := \Phi(\Psi(X)) \tag{49}$$

$$\mathcal{D}\Xi_X(H) = \mathcal{D}\Phi_{\Psi(X)}(\mathcal{D}\Psi_X(H)) = 2\mathcal{D}\Phi_{\Psi(X)}(\mathcal{D}\Phi_X(H)) - \mathcal{D}\Phi_{\Psi(X)}(H)$$

$$\max_s |\lambda_s(2\lambda_s - 1)| \tag{50}$$

9.2 Function Values

Here are the results for the Ekman data if we use stabilization.

In iteration 19, the final iteration, stress is 2.1114112739076. The CHNG is 1.29239746041767e-16 and the EARC is 0.40956832382978.

9.3 Asymptotic Rate of Convergence

10 Benchmarking

We compare the eight different upgrades using the microbenchmark package (Mersmann (2023)).

```
Warning in microbenchmark(smacofAccelerate(delta, xold = xold, ndim = 2, : less
accurate nanosecond times to avoid potential integer overflows
```

```
Unit: milliseconds
                                                                                                ex
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 1, halt = 2,      verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 2, halt = 2,      verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 3, halt = 2,      verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 4, halt = 2,      verbose = FALS
                  smacofAccelerate(delta, ndim = 2, opt = 5, halt = 2, verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 6, halt = 2,      verbose = FALS
 smacofAccelerate(delta, ndim = 2, xold = xold, opt = 7, halt = 2,      verbose = FALS
 smacofAccelerate(delta, ndim = 2, xold = xold, opt = 8, halt = 2,      verbose = FALS
       min        lq      mean    median        uq       max neval
  3.044455  3.146053  3.448287  3.186110  3.314153  5.294576   100
  4.148954  4.305738  4.739418  4.377754  4.688248  6.815430   100
  3.829769  3.957504  4.165095  4.007606  4.072468  6.222365   100
 26.974228 28.709122 29.319847 29.039111 29.516822 50.105854   100
  1.614129  1.663493  1.838397  1.699819  1.748937  6.934945   100
  1.079079  1.112556  1.199962  1.124487  1.143921  5.839507   100
  1.515032  1.581841  1.704306  1.607754  1.641599  3.645228   100
  1.493630  1.544860  1.655274  1.563576  1.597032  3.725219   100
```

De Gruijter (1967)

```
Unit: milliseconds
                                                                                                ex
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 1, halt = 2,      verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 2, halt = 2,      verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 3, halt = 2,      verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 4, halt = 2,      verbose = FALS
```

```
                  smacofAccelerate(delta, ndim = 2, opt = 5, halt = 2, verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 6, halt = 2,      verbose = FALS
 smacofAccelerate(delta, ndim = 2, xold = xold, opt = 7, halt = 2,      verbose = FALS
 smacofAccelerate(delta, ndim = 2, xold = xold, opt = 8, halt = 2,      verbose = FALS
     min       lq     mean   median       uq      max neval
 44.56356 45.42333 47.14609 46.44152 46.87905 68.94703   100
 61.57872 62.66573 64.18375 63.27837 64.45190 83.48600   100
 57.70192 58.66200 60.11207 59.05974 60.60173 78.60196   100
 56.66696 57.81754 58.96594 58.21838 59.09945 77.35921   100
 20.29418 21.77850 21.97197 22.11845 22.37103 24.10640   100
 14.69190 14.94430 15.75168 15.58898 16.44651 18.72437   100
 23.09633 24.79844 25.09315 25.00746 25.33203 27.71325   100
 21.50270 23.05369 23.37570 23.34841 23.64210 26.69973   100

Unit: milliseconds
                                                                                ex
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 1, halt = 2,      verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 2, halt = 2,      verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 3, halt = 2,      verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 4, halt = 2,      verbose = FALS
                  smacofAccelerate(delta, ndim = 2, opt = 5, halt = 2, verbose = FALS
 smacofAccelerate(delta, xold = xold, ndim = 2, opt = 6, halt = 2,      verbose = FALS
 smacofAccelerate(delta, ndim = 2, xold = xold, opt = 7, halt = 2,      verbose = FALS
 smacofAccelerate(delta, ndim = 2, xold = xold, opt = 8, halt = 2,      verbose = FALS
      min        lq      mean    median        uq       max neval
  42.83639  46.24685  47.50839  46.54912  47.28733  66.76551   100
  55.42913  56.56801  58.14801  57.06767  58.80995  80.01474   100
  52.75601  53.75082  55.24188  54.26299  55.70772  76.80842   100
 107.38769 108.96640 115.01316 110.18508 116.25702 152.92635   100
  18.94048  19.66083  21.17640  20.10187  22.41228  45.11419   100
  14.31864  14.77636  16.61831  14.99965  17.51516  42.79695   100
  24.47405  26.37624  29.26654  27.93303  28.79045  50.26055   100
  23.35577  24.20747  27.54380  26.71755  27.48287  49.37405   100
```

(APPENDIX) Appendices

11 Code

11.1 smacofAccelerate.R

```
library(MASS)
library(microbenchmark)
library(numDeriv)

source("smacofUtils.R")
source("smacofDerivatives.R")

smacofAccelerate <- function(delta,
                             wgth = 1 - diag(nrow(delta)),
                             ndim = 2,
                             xold = smacofTorgerson(delta, ndim),
                             opt = 1,
                             halt = 0,
                             wd = 4,
                             dg = 15,
                             itmax = 1000,
                             epsx = 1e-10,
                             epsf = 1e-15,
                             verbose = 2) {
  nobj <- nrow(delta)
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  if (opt == 4) {
    bs <- smacofMakeBasis(nobj, ndim, vmat)
  }
  xold <- smacofCenter(xold)
  if ((opt == 2) || (opt == 4)) {
    xrot <- qr.Q(qr(xold[1:ndim, ]))
    xold <- xold %*% xrot
  }
  if (opt == 3) {
```

```
  xrot <- svd(xold)$v
  xold <- xold %*% xrot
}
dold <- as.matrix(dist(xold))
sold <- sum(wgth * (delta - dold) ^ 2)
cold <- Inf
itel <- 1
repeat {
  if (opt == 1) {
    h <- smacofOptionOne(xold, delta, wgth, vmat, vinv)
  }
  if (opt == 2) {
    h <- smacofOptionTwo(xold, delta, wgth, vmat, vinv)
  }
  if (opt == 3) {
    h <- smacofOptionThree(xold, delta, wgth, vmat, vinv)
  }
  if (opt == 4) {
    h <- smacofOptionFour(xold, delta, wgth, vmat, vinv, bs)
  }
  if (opt == 5) {
    h <- smacofOptionFive(xold, delta, wgth, vmat, vinv)
  }
  if (opt == 6) {
    h <- smacofOptionSix(xold, delta, wgth, vmat, vinv)
  }
  if (opt == 7) {
    h <- smacofOptionSeven(xold, delta, wgth, vmat, vinv)
  }
  if (opt == 8) {
    h <- smacofOptionEight(xold, delta, wgth, vmat, vinv)
  }
  labd <- sqrt((h$cnew) / cold)
  if (verbose == 2) {
    smacofLinePrint(itel, sold, h$snew, h$cnew, labd, wd = wd, dg = dg)
  }
  if (halt == 1) {
    converge <- h$cnew < epsx
  } else {
    converge <- (sold - h$snew) < epsf
```

```
    }
    if ((itel == itmax) || converge) {
      break
    }
    itel <- itel + 1
    sold <- h$snew
    xold <- h$xnew
    cold <- h$cnew
    dold <- h$dnew
  } # end of repeat loop
  if (verbose == 1) {
    smacofLinePrint(itel, sold, h$snew, h$cnew, labd, wd = wd, dg = dg)
  }
  adjust <- list(xnew = NULL, dnew = NULL, snew = NULL)
  if (opt == 5) {
    adjust$xnew <- (h$xnew + xold) / 2
    adjust$dnew <- as.matrix(dist(adjust$xnew))
    adjust$snew <- sum(wgth * (delta - adjust$dnew) ^ 2)
  }
  if (opt == 6) {
    bold <- -wgth * delta / (h$dnew + diag(nobj))
    diag(bold) <- -rowSums(bold)
    adjust$xnew <- vinv %*% bold %*% h$xnew
    adjust$dnew <- as.matrix(dist(adjust$xnew))
    adjust$snew <- sum(wgth * (delta - adjust$dnew) ^ 2)
  }
  return(
    list(
      x = h$xnew,
      s = h$snew,
      d = h$dnew,
      itel = itel,
      chng = h$cnew,
      labd = labd,
      wgth = wgth,
      delta = delta,
      adjust = adjust
    )
  )
}
```

```r
smacofOptionOne <- function(xold, delta, wgth, vmat, vinv) {
  xnew <- smacofCenter(smacofGuttman(xold, delta, wgth, vinv))
  dnew <- as.matrix(dist(xnew))
  snew <- sum(wgth * (delta - dnew) ^ 2)
  cnew <- sum((xold - xnew) * (vmat %*% (xold - xnew)))
  return(list(
    xnew = xnew,
    dnew = dnew,
    snew = snew,
    cnew = cnew
  ))
}

smacofOptionTwo <- function(xold, delta, wgth, vmat, vinv) {
  ndim <- ncol(xold)
  xbar <- smacofCenter(smacofGuttman(xold, delta, wgth, vinv))
  xrot <- smacofSignEigenVectors(qr.Q(qr(t(xbar[1:ndim, ]))))
  #xrot <- qr.Q(qr(t(xbar[1:ndim, ])))
  xnew <- xbar %*% xrot
  dnew <- as.matrix(dist(xnew))
  snew <- sum(wgth * (delta - dnew) ^ 2)
  cnew <- sum((xold - xnew) * (vmat %*% (xold - xnew)))
  return(list(
    xnew = xnew,
    dnew = dnew,
    snew = snew,
    cnew = cnew
  ))
}

smacofOptionThree <- function(xold, delta, wgth, vmat, vinv) {
  xbar <- smacofCenter(smacofGuttman(xold, delta, wgth, vinv))
  xrot <- smacofSignEigenVectors(svd(xbar)$v)
  xnew <- xbar %*% xrot
  dnew <- as.matrix(dist(xnew))
  snew <- sum(wgth * (delta - dnew) ^ 2)
  cnew <- sum((xold - xnew) * (vmat %*% (xold - xnew)))
  return(list(
    xnew = xnew,
    dnew = dnew,
```

```
    snew = snew,
    cnew = cnew
  ))
}

smacofOptionFour <- function(xold, delta, wgth, vmat, vinv, bs) {
  ndim <- ncol(xold)
  nobj <- nrow(xold)
  xnew <- matrix(0, nobj, ndim)
  xbar <- smacofCenter(smacofGuttman(xold, delta, wgth, vinv))
  for (s in 1:ndim) {
    aux <- crossprod(bs[[s]], vmat %*% xbar[, s])
    xnew[, s] <- bs[[s]] %*% aux
  }
  dnew <- as.matrix(dist(xnew))
  snew <- sum(wgth * (delta - dnew) ^ 2)
  cnew <- sum((xold - xnew) * (vmat %*% (xold - xnew)))
  return(list(
    xnew = xnew,
    dnew = dnew,
    snew = snew,
    cnew = cnew
  ))
}


smacofOptionFive <- function(xold, delta, wgth, vmat, vinv) {
  xbar <- smacofCenter(smacofGuttman(xold, delta, wgth, vinv))
  xnew <- 2 * xbar - xold
  dnew <- as.matrix(dist(xnew))
  snew <- sum(wgth * (delta - dnew) ^ 2)
  cnew <- sum((xold - xnew) * (vmat %*% (xold - xnew)))
  return(list(
    xnew = xnew,
    dnew = dnew,
    snew = snew,
    cnew = cnew
  ))
}
```

```r
smacofOptionSix <- function(xold, delta, wgth, vmat, vinv) {
  nobj <- nrow(xold)
  xbar <- smacofCenter(smacofGuttman(xold, delta, wgth, vinv))
  xaux <- 2 * xbar - xold
  daux <- as.matrix(dist(xaux))
  baux <- -wgth * delta / (daux + diag(nobj))
  diag(baux) <- -rowSums(baux)
  xbaz <- vinv %*% baux %*% xaux
  xnew <- 2 * xbaz - xaux
  dnew <- as.matrix(dist(xnew))
  snew <- sum(wgth * (delta - dnew) ^ 2)
  cnew <- sum((xold - xnew) * (vmat %*% (xold - xnew)))
  return(list(
    xnew = xnew,
    dnew = dnew,
    snew = snew,
    cnew = cnew
  ))
}

smacofOptionSeven <- function(xold, delta, wgth, vmat, vinv) {
  xbar <- smacofCenter(smacofGuttman(xold, delta, wgth, vinv))
  xaux <- 2 * xbar - xold
  daux <- as.matrix(dist(xaux))
  alpa <- sum(wgth * daux * delta) / sum(wgth * daux ^ 2)
  xnew <- alpa * xaux
  dnew <- alpa * daux
  snew <- sum(wgth * (delta - dnew) ^ 2)
  cnew <- sum((xold - xnew) * (vmat %*% (xold - xnew)))
  return(list(
    xnew = xnew,
    dnew = dnew,
    snew = snew,
    cnew = cnew
  ))
}

smacofOptionEight <- function(xold, delta, wgth, vmat, vinv) {
  nobj <- nrow(xold)
  xbar <- smacofCenter(smacofGuttman(xold, delta, wgth, vinv))
```

```
  xaux <- 2 * xbar - xold
  daux <- as.matrix(dist(xaux))
  baux <- -wgth * delta / (daux + diag(nobj))
  diag(baux) <- -rowSums(baux)
  xnew <- vinv %*% baux %*% xaux
  dnew <- as.matrix(dist(xnew))
  snew <- sum(wgth * (delta - dnew) ^ 2)
  cnew <- sum((xold - xnew) * (vmat %*% (xold - xnew)))
  return(list(
    xnew = xnew,
    dnew = dnew,
    snew = snew,
    cnew = cnew
  ))
}
```

11.2 smacofDerivatives.R

```
library(numDeriv)

source("smacofUtils.R")
source("smacofPCADerivative.R")
source("smacofQRDerivative.R")

smacofRhoHessian <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  ntot <- nobj * ndim
  dmat <- as.matrix(dist(x))
  fac1 <- wgth * delta / (dmat + diag(nobj))
  fac2 <- wgth * delta / ((dmat + diag(nobj)) ^ 3)
  bmat <- -fac1
  diag(bmat) <- -rowSums(bmat)
  hess <- matrix(0, ntot, ntot)
  for (s in 1:ndim) {
    ns <- (s - 1) * nobj + 1:nobj
    hess[ns, ns] <- bmat
    for (t in 1:ndim) {
```

```
      nt <- (t - 1) * nobj + 1:nobj
      ds <- outer(x[, s], x[, s], "-")
      dt <- outer(x[, t], x[, t], "-")
      aux <- -fac2 * ds * dt
      diag(aux) <- -rowSums(aux)
      hess[ns, nt] <- hess[ns, nt] - aux
    }
  }
  return(hess)
}

smacofBasicDerivative <- function(x, h, delta, wgth, vinv, dmat) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  bmat <- wgth * delta / (dmat + diag(nobj))
  bmat <- -bmat
  diag(bmat) <- -rowSums(bmat)
  hmat <- wgth * delta / ((dmat + diag(nobj)) ^ 3)
  for (i in 1:nobj) {
    for (j in 1:nobj) {
      xhij <- sum((x[i, ] - x[j, ]) * (h[i, ] - h[j, ]))
      hmat[i, j] <- hmat[i, j] * xhij
    }
  }
  hmat <- -hmat
  diag(hmat) <- -rowSums(hmat)
  deri <- vinv %*% (bmat %*% h - hmat %*% x)
  return(deri)
}

smacofBasicJacobianFormula <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  ntot <- nobj * ndim
  dmat <- as.matrix(dist(x))
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  jacob <- matrix(0, ntot, ntot)
  e <- function(i, n) {
```

```
    ifelse(i == 1:n, 1, 0)
  }
  k <- 1
  for (j in 1:ndim) {
    for (i in 1:nobj) {
      h <- outer(e(i, nobj), e(j, ndim))
      r <- smacofBasicDerivative(x, h, delta, wgth, vinv, dmat)
      jacob[, k] <- as.vector(r)
      k <- k + 1
    }
  }
  return(jacob)
}

smacofBasicJacobianNumerical <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  ntot <- nobj * ndim
  dmat <- as.matrix(dist(x))
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  func <- function(x, nobj, ndim, delta, wgth) {
    x <- matrix(x, nobj, ndim)
    xbar <- smacofGuttman(x, delta, wgth, vinv)
    return(as.vector(xbar))
  }
  jacob <- jacobian(
    func,
    as.vector(x),
    nobj = nobj,
    ndim = ndim,
    delta = delta,
    wgth = wgth
  )
  return(jacob)
}

smacofPCADerivative <- function(x, h, delta, wgth, vinv, dmat) {
  xbar <- smacofGuttman(x, delta, wgth, vinv)
```

```
  dexh <- smacofBasicDerivative(x, h, delta, wgth, vinv, dmat)
  deri <- PCADerivative(xbar, dexh)
  return(deri)
}

smacofPCAJacobianFormula <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  ntot <- nobj * ndim
  dmat <- as.matrix(dist(x))
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  jacob <- matrix(0, ntot, ntot)
  e <- function(i, n) {
    ifelse(i == 1:n, 1, 0)
  }
  k <- 1
  for (j in 1:ndim) {
    for (i in 1:nobj) {
      h <- outer(e(i, nobj), e(j, ndim))
      r <- smacofPCADerivative(x, h, delta, wgth, vinv, dmat)
      jacob[, k] <- as.vector(r)
      k <- k + 1
    }
  }
  return(jacob)
}

smacofPCAJacobianNumerical <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  ntot <- nobj * ndim
  dmat <- as.matrix(dist(x))
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  func <- function(x, nobj, ndim, delta, wgth) {
    x <- matrix(x, nobj, ndim)
    xbar <- smacofGuttman(x, delta, wgth, vinv)
```

```
    l <- smacofSignEigenVectors(svd(xbar)$v)
    return(xbar %*% l)
  }
  jacob <- jacobian(
    func,
    as.vector(x),
    nobj = nobj,
    ndim = ndim,
    delta = delta,
    wgth = wgth
  )
  return(jacob)
}


smacofQRDerivative <- function(x, h, delta, wgth, vinv, dmat) {
  xbar <- smacofGuttman(x, delta, wgth, vinv)
  dexh <- smacofBasicDerivative(x, h, delta, wgth, vinv, dmat)
  deri <- QRDerivative(xbar, dexh)
  return(deri)
}

smacofQRJacobianFormula <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  ntot <- nobj * ndim
  dmat <- as.matrix(dist(x))
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  jacob <- matrix(0, ntot, ntot)
  e <- function(i, n) {
    ifelse(i == 1:n, 1, 0)
  }
  k <- 1
  for (j in 1:ndim) {
    for (i in 1:nobj) {
      h <- outer(e(i, nobj), e(j, ndim))
      r <- smacofQRDerivative(x, h, delta, wgth, vinv, dmat)
      jacob[, k] <- as.vector(r)
```

```
      k <- k + 1
    }
  }
  return(jacob)
}

smacofQRJacobianNumerical <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  ntot <- nobj * ndim
  dmat <- as.matrix(dist(x))
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  func <- function(x, nobj, ndim, delta, wgth) {
    x <- matrix(x, nobj, ndim)
    xbar <- smacofGuttman(x, delta, wgth, vinv)
    l <- qr.Q(qr(t(xbar[1:ndim, ])))
    return(xbar %*% l)
  }
  jacob <- jacobian(
    func,
    as.vector(x),
    nobj = nobj,
    ndim = ndim,
    delta = delta,
    wgth = wgth
  )
  return(jacob)
}

smacofYbasDerivative <- function(x, h, delta, wgth, vmat, vinv, dmat, bs) {
  ndim <- ncol(x)
  nobj <- nrow(x)
  ntot <- nobj * ndim
  dexh <- smacofBasicDerivative(x, h, delta, wgth, vinv, dmat)
  deri <- matrix(0, nobj, ndim)
  for (i in 1:ndim) {
    deri[, i] <- bs[[i]] %*% crossprod(bs[[i]], vmat %*% dexh[, i])
  }
```

```
  return(deri)
}

smacofYbasJacobianFormula <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  ntot <- nobj * ndim
  dmat <- as.matrix(dist(x))
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  bs <- smacofMakeBasis(nobj, ndim, vmat)
  jacob <- matrix(0, ntot, ntot)
  e <- function(i, n) {
    ifelse(i == 1:n, 1, 0)
  }
  k <- 1
  for (j in 1:ndim) {
    for (i in 1:nobj) {
      h <- outer(e(i, nobj), e(j, ndim))
      r <- smacofYbasDerivative(x, h, delta, wgth, vmat, vinv, dmat, bs)
      jacob[, k] <- as.vector(r)
      k <- k + 1
    }
  }
  return(jacob)
}

smacofYbasJacobianNumerical <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  bs <- smacofMakeBasis(nobj, ndim, vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  func <- function(x, nobj, ndim, delta, wgth, vmat, vinv, bs) {
    x <- matrix(x, nobj, ndim)
    xbar <- smacofGuttman(x, delta, wgth, vinv)
    for (i in 1:ndim) {
      xbar[, i] <- bs[[i]] %*% crossprod(bs[[i]], vmat %*% xbar[, i])
```

```
    }
    return(xbar)
  }
  jacob <- jacobian(
    func,
    as.vector(x),
    nobj = nobj,
    ndim = ndim,
    delta = delta,
    wgth = wgth,
    vmat = vmat,
    vinv = vinv,
    bs = bs
  )
  return(jacob)
}


smacofRelaxDerivative <- function(x, h, delta, wgth, vinv, dmat) {
  ndim <- ncol(x)
  nobj <- nrow(x)
  ntot <- nobj * ndim
  dexh <- smacofBasicDerivative(x, h, delta, wgth, vinv, dmat)
  deri <- 2 * dexh - h
  return(deri)
}

smacofRelaxJacobianFormula <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  ntot <- nobj * ndim
  dmat <- as.matrix(dist(x))
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  jacob <- matrix(0, ntot, ntot)
  e <- function(i, n) {
    ifelse(i == 1:n, 1, 0)
  }
  k <- 1
```

```
  for (j in 1:ndim) {
    for (i in 1:nobj) {
      h <- outer(e(i, nobj), e(j, ndim))
      r <- smacofRelaxDerivative(x, h, delta, wgth, vinv, dmat)
      jacob[, k] <- as.vector(r)
      k <- k + 1
    }
  }
  return(jacob)
}

smacofRelaxJacobianNumerical <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  bs <- smacofMakeBasis(nobj, ndim, vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  func <- function(x, nobj, ndim, delta, wgth, vinv) {
    x <- matrix(x, nobj, ndim)
    xbar <- smacofGuttman(x, delta, wgth, vinv)
    xbaz <- 2 * xbar - x
    return(xbaz)
  }
  jacob <- jacobian(
    func,
    as.vector(x),
    nobj = nobj,
    ndim = ndim,
    delta = delta,
    wgth = wgth,
    vinv = vinv
  )
  return(jacob)
}

smacofDoubleDerivative <- function(x, h, delta, wgth, vinv, dmat) {
  ndim <- ncol(x)
  nobj <- nrow(x)
  ntot <- nobj * ndim
```

```
  dexh <- smacofBasicDerivative(x, h, delta, wgth, vinv, dmat)
  deri <- 2 * dexh - h
  return(deri)
}

smacofDoubleJacobianFormula <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  ntot <- nobj * ndim
  dmat <- as.matrix(dist(x))
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  jacob <- matrix(0, ntot, ntot)
  e <- function(i, n) {
    ifelse(i == 1:n, 1, 0)
  }
  k <- 1
  for (j in 1:ndim) {
    for (i in 1:nobj) {
      h <- outer(e(i, nobj), e(j, ndim))
      r <- smacofDoubleDerivative(x, h, delta, wgth, vinv, dmat)
      jacob[, k] <- as.vector(r)
      k <- k + 1
    }
  }
  return(jacob)
}

smacofDoubleJacobianNumerical <- function(x, delta, wgth) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  bs <- smacofMakeBasis(nobj, ndim, vmat)
  vinv <- solve(vmat + (1 / nobj)) - (1 / nobj)
  func <- function(x, nobj, ndim, delta, wgth, vinv) {
    x <- matrix(x, nobj, ndim)
    xbar <- 2 * smacofGuttman(x, delta, wgth, vinv) - x
    xbaz <- 2 * smacofGuttman(xbar, delta, wgth, vinv) - xbar
```

```
    return(xbaz)
  }
  jacob <- jacobian(
    func,
    as.vector(x),
    nobj = nobj,
    ndim = ndim,
    delta = delta,
    wgth = wgth,
    vinv = vinv
  )
  return(jacob)
}

smacofDilateJacobianFormula <- function() {}

smacofDilateJacobianNumerical <- function() {}


smacofStabilizeJacobianFormula <- function() {}

smacofStabilizeJacobianNumerical <- function() {}
```

11.3 smacofPCADerivative.R

```
PCADerivative <- function(x, h) {
  ndim <- ncol(x)
  eixx <- eigen(crossprod(x))
  evec <- eixx$vectors
  evec <- evec %*% diag(sign(diag(evec)))
  eval <- eixx$values
  xh <- crossprod(x, h)
  xh <- xh + t(xh)
  s <- matrix(0, ndim, ndim)
  for (i in 1:ndim) {
    for (j in 1:ndim) {
      if (i == j) {
        next
```

```
      }
      s[i, j] <- -sum(evec[, i] * (xh %*% evec[, j])) / (eval[i] - eval[j])
    }
  }
  return(h %*% evec + x %*% evec %*% s)
}

PCAJacobianFormula <- function(x) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  np <- nobj * ndim
  e <- function(i, n) {
    ifelse(i == 1:n, 1, 0)
  }
  d <- matrix(0, np, np)
  k <- 1
  for (j in 1:ndim) {
    for (i in 1:nobj) {
      h <- outer(e(i, nobj), e(j, ndim))
      r <- PCADerivative(x, h)
      d[, k] <- as.vector(r)
      k <- k + 1
    }
  }
  return(d)
}

PCAJacobianNumerical <- function(x) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  func <- function(x, nobj, ndim) {
    x <- matrix(x, nobj, ndim)
    l <- svd(x)$v
    l <- l %*% diag(sign(diag(l)))
    return(x %*% l)
  }
  jacob <- jacobian(func, as.vector(x), nobj = nobj, ndim = ndim)
  return(jacob)
}
```

```
PCATester <- function(x, h) {
  func <- function(x) {
    l <- svd(x)$v
    l <- l %*% diag(sign(diag(l)))
    return(x %*% l)
  }
  x0 <- func(x)
  xh <- func(x + h)
  xd <- x0 + PCADerivative(x, h)
  print(cbind(x0, xh, xd))
}
```

11.4 smacofQRDerivative.R

```
QRDerivative <- function(x, h) {
  ndim <- ncol(x)
  z <- t(x[1:ndim, ])
  g <- t(h[1:ndim, ])
  qq <- qr(z)
  q <- qr.Q(qq)
  r <- qr.R(qq)
  b <- crossprod(q, g %*% solve(r))
  a <- matrix(0, ndim, ndim)
  i <- outer(1:ndim, 1:ndim, ">")
  a <- ifelse(i, b, 0)
  a <- a - t(a)
  deri <- h %*% q + x %*% q %*% a
  return(deri)
}

QRJacobianFormula <- function(x) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  np <- nobj * ndim
  e <- function(i, n) {
    ifelse(i == 1:n, 1, 0)
  }
  d <- matrix(0, np, np)
```

```
  k <- 1
  for (j in 1:ndim) {
    for (i in 1:nobj) {
      h <- outer(e(i, nobj), e(j, ndim))
      r <- QRDerivative(x, h)
      d[, k] <- as.vector(r)
      k <- k + 1
    }
  }
  return(d)
}

QRJacobianNumerical <- function(x) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  func <- function(x, nobj, ndim) {
    x <- matrix(x, nobj, ndim)
    l <- qr.Q(qr(t(x[1:ndim, ])))
    return(x %*% l)
  }
  jacob <- jacobian(func, as.vector(x), nobj = nobj, ndim = ndim)
  return(jacob)
}

QRTester <- function(x, h) {
  func <- function(x) {
    ndim <- ncol(x)
    l <- qr.Q(qr(t(x[1:ndim, ])))
    l <- l %*% diag(sign(diag(l)))
    return(x %*% l)
  }
  x0 <- func(x)
  xh <- func(x + h)
  xd <- x0 + QRDerivative(x, h)
  print(cbind(x0, xh, xd))
}
```

11.5 smacofCompare.R

```
smacofCompare <- function(delta, ndim = 2) {
  nobj <- nrow(delta)
  wgth <- 1 - diag(nobj)
  xold <- smacofTorgerson(delta, ndim)
  return(
    microbenchmark(
      smacofAccelerate(
        delta,
        xold = xold,
        ndim = 2,
        opt = 1,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        xold = xold,
        ndim = 2,
        opt = 2,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        xold = xold,
        ndim = 2,
        opt = 3,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        xold = xold,
        ndim = 2,
        opt = 4,
        halt = 2,
        verbose = FALSE
```

```
      ),
      smacofAccelerate(
        delta,
        ndim = 2,
        opt = 5,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        xold = xold,
        ndim = 2,
        opt = 6,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        ndim = 2,
        xold = xold,
        opt = 7,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        ndim = 2,
        xold = xold,
        opt = 8,
        halt = 2,
        verbose = FALSE
      )
    )
  )
}
```

11.6 smacofUtils.R

```
smacofMatrixPrint <- function(x,
                    digits = 10,
                    width = 15,
                    format = "f",
                    flag = "+") {
  print(noquote(
    formatC(
      x,
      digits = digits,
      width = width,
      format = format,
      flag = flag
    )
  ))
}

smacofLinePrint <- function(itel, sold, snew, cnew, labd, wd, dg) {
  cat(
    "itel",
    formatC(itel, width = wd, format = "d"),
    "sold",
    formatC(sold, digits = dg, format = "f"),
    "snew",
    formatC(snew, digits = dg, format = "f"),
    "chng",
    formatC(cnew, digits =  dg, format = "f"),
    "labd",
    formatC(labd, digits =  dg, format = "f"),
    "\n"
  )
}

smacofMakeBasis <- function(n, ndim, vmat) {
  y <- lapply(1:ndim, function(k)
    matrix(0, n, n - k))
  for (s in 0:(ndim - 1)) {
    ns <- n - s
```

```
    aux <- qr.Q(qr(ns * diag(ns) - 1))[, -ns]
    aux <- rbind(matrix(0, s, ns - 1), aux)
    sux <- crossprod(aux, vmat %*% aux)
    y[[s + 1]] <- aux %*% smacofMatrixPower(sux, -0.5)
  }
  return(y)
}

smacofMakeBisas <- function(n, ndim, vmat) {
  y <- rep(list(matrix(0, n, n - 1)), ndim)
  for (s in 0:(ndim - 1)) {
    ns <- n
    aux <- qr.Q(qr(n * diag(n) - 1))[, -n]
    sux <- crossprod(aux, vmat %*% aux)
    y[[s + 1]] <- aux %*% smacofMatrixPower(sux, -0.5)
  }
  return(y)
}

smacofTorgerson <- function(delta, ndim) {
  n <- nrow(delta)
  dd <- delta ^ 2
  rd <- rowSums(dd) / n
  sd <- mean(dd)
  cc <- -.5 * (dd - outer(rd, rd, "+") + sd)
  ee <- eigen(cc)
  x <- ee$vectors[, 1:ndim] %*% diag(sqrt(ee$values[1:ndim]))
  return(x)
}

smacofGuttman <- function(x, delta, wgth, vinv) {
  nobj <- nrow(x)
  dmat <- as.matrix(dist(x))
  bmat <- -wgth * delta / (dmat + diag(nobj))
  diag(bmat) <- -rowSums(bmat)
  return(vinv %*% bmat %*% x)
}

smacofCenter <- function(x) {
  return(apply(x, 2, function(x)
```

```
    x - mean(x)))
}

smacofSignEigenVectors <- function(x) {
  return(x %*% diag(sign(diag(x))))
}

smacofMatrixPower <- function(s, power) {
  e <- eigen(s)
  eval <- e$values
  evec <- e$vectors
  dval <- ifelse(abs(eval) < 1e-10, 0, abs(eval) ^ power)
  return(tcrossprod(evec %*% diag(dval), evec))
}
```

References

Bolte, J., A Daniilidis, and A. Lewis. 2007. "The Łojasiewicz Inequality for Nonsmooth Subanalytic Functions with Applications to Subgradient Dynamical Systems." *SIAM Journal of Optimization* 17 (4): 1205–23.

Bolte, J., S. Sabach, and M. Teboulle. 2014. "Proximal Alternating Linearized Minimization for Nonconvex and Nonsmooth Problems." *Mathematical Programming* 146: 459–94.

Clarke, F. H. 1975. "Generalized Gradients and Applications." *Transactions of the American Mathematical Society* 205: 247–62.

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.

———. 1993. "Fitting Distances by Least Squares." Preprint Series 130. Los Angeles, CA: UCLA Department of Statistics. https://jansweb.netlify.app/publication/deleeuw-r-93-c/deleeuw-r-93-c.pdf.

———. 1994. "Block Relaxation Algorithms in Statistics." In *Information Systems and Data Analysis*, edited by H. H. Bock, W. Lenski, and M. M. Richter, 308–24. Berlin: Springer

Verlag. https://jansweb.netlify.app/publication/deleeuw-c-94-c/deleeuw-c-94-c.pdf.
———. 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.
———. 2019. "Convergence of SMACOF." 2019. https://jansweb.netlify.app/publication/deleeuw-e-19-h/deleeuw-e-19-h.pdf.
———. 2021. *Least Squares Euclidean Multidimensional Scaling*. https://jansweb.netlify.app/publication/deleeuw-b-21-a/deleeuw-b-21-a.pdf.
———. 2023. "Differentiating the QR Decomposition." 2023. https://jansweb.netlify.app/publication/deleeuw-e-23-a/deleeuw-e-23-a.pdf.
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.
Ekman, G. 1954. "Dimensions of Color Vision." *Journal of Psychology* 38: 467–74.
Gilbert, P., and R. Varadhan. 2019. *numDeriv: Accurate Numerical Derivatives*. https://CRAN.R-project.org/package=numDeriv.
Groenen, P. J. F., W. Glunt, and T. L. Hayden. 1996. "Fast Algorithms for Multidimensional Scaling: A Comparison of Majorization and Spectral Gradient Methods." Department of Data Theory, University of Leiden.
Guttman, L. 1968. "A General Nonmetric Technique for Fitting the Smallest Coordinate Space for a Configuration of Points." *Psychometrika* 33: 469–506.
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.
Lange, K. 2016. *MM Optimization Algorithms*. SIAM.
Le Thi, H. A., and P. D. Tao. 2001. "D.c. Programming Approach to the Multidimensional Scaling Problem." In *From Local to Global Optimization*, edited by A. Migdalas, P. M. Pardalos, and P. Värbrand, 231–76. Springer Verlag.
———. 2018. "DC Programming and DCA: Thirty Years of Developments." *Mathematical Programming, Series B*.
———. 2024. "Open Issues and Recent Advances in DC Programming and DCA." *Journal of Global Optimization* 88: 533–90.
Mair, P., and J. De Leeuw. 2015. "Unidimensional Scaling." In *Wiley StatsRef: Statistics Reference Online*, 1–3. Wiley.
Mair, P., J. De Leeuw, and M. Wurzer. 2015. "Multidimensional Unfolding." In *Wiley StatsRef: Statistics Reference Online*, 1–4. Wiley.
Mersmann, O. 2023. *microbenchmark: Accurate Timing Functions*. https://CRAN.R-project.

org/package=microbenchmark.
Meyer, R. R. 1976. “Sufficient Conditions for the Convergence of Monotonic Mathematical Programming Algorithms.” *Journal of Computer and System Sciences* 12: 108–21.
Ortega, J. M., and W. C. Rheinboldt. 1970. *Iterative Solution of Nonlinear Equations in Several Variables*. New York, N.Y.: Academic Press.
Ostrowski, A. M. 1973. *Solution of Equations in Euclidean and Banach Spaces*. Third Edition of Solution of Equations and Systems of Equations. Academic Press.
Ram, N., and S. Sabach. 2024 (in press). “A Globally Convergent Inertial First-Order Optimization Method for Multidimensional Scaling.” *Journal of Optimization Theory and Applications*, 2024 (in press).
Robert, F. 1967. “Calcul du Rapport Maximal de Deux Normes sur $\mathbb{R}^n$.” *Revue Francaise d’Automatique, d’Informatique Et De Recherche Operationelle* 1: 97–118.
Robini, M., L. Wang, and Y. Zhu. 2024. “The Appeals of Quadratic Majorization-Minimization.” *Journal of Global Optimization* 89: 509–58.
Rockafellar, R. T. 1970. *Convex Analysis*. Princeton University Press.