Accelerated Least Squares Multidimensional Scaling

Jan de Leeuw - University of California Los Angeles

Started July 23 2024, Version of August 11, 2024

Abstract

We discuss a simple accelerations of MDS smacof iterations, and compare them with recent boosted difference-of-convex algortithms.

Contents

1 Introduction

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

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

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 (1) 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

$$\frac{1}{2}\sum\sum_{1\leq i<j\leq n} w_{ij}\delta_{ij}^2 = 1. \tag{2}$$

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

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

Following De Leeuw (1977) we define

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

where

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

with

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

Also define

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

where

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

Thus

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

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 homogeneous convex functions, with η being a norm on the space of column-centered configurations. The equations $d_{ij}(X) = 0$ for all (i,j) for which $w_{ij}\delta_{ij} > 0$ define a linear subspace of configuration space. If that subspace only contains the zero matrix then ρ is a norm as well.

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

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

we see that ρ, although not differentiable, is globally Lipschitz.

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

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

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 $\text{tr } Z'X = \rho(X)$ 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

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

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

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

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

$$\partial\rho(X) = B(X)X + Z \tag{14}$$

with

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

It also follows that

$$\partial\sigma(X) = VX - \partial\rho(X) \tag{16}$$

Since σ is not convex the subdifferential in (16) is the Clarke subdifferential (Clarke (1975)).

Now X is a 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 fixed points of the Guttman transform. A necessary condition for σ to have a local minimum at X is that X is a 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 convex analysis is rarely needed in practice. It is shown by De Leeuw (1984) that a necessary condition for a local minimum at X is that $d_{ij}(X) > 0$ for all (i, j) for which $w_{ij}\delta_{ij} > 0$. At those points σ is differentiable, and thus the subdifferential (16) is a singleton, containing only the gradient. We have $\nabla\rho(X) = B(X)X$ and $\nabla\sigma(X) = VX - B(X)X$. Stationary points satisfy $X = V^+B(X)X$.

If $w_{ij}\delta_{ij} = 0$ for some (i, j) then there can be local minima where σ is not differentiable. This typically happens 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 $dij(X) > 0$ this follows directly from the Cauchy-Schwartz inequality

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

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

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

for all Y, with equality if $Y = X$. Not "if and only if", but just "if". We also have equality if $Y = \alpha X$ for some $\alpha \geq 0$.

Using the Guttman transform we can use (**??**) as the basic smacof equality

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

for all X and the basic smacof inequality

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

for all X and Y.

Taken together (19) and (20) imply the *sandwich inequality*

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

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 (21) that $\eta^2(\Phi(Y)) \leq 1$. Thus the Guttman transform of any configuration is in a convex and compact set, in fact an ellipsoid, containing the origin.

2 One-point Iteration

2.1 Basic Iteration

The basic smacof algorithm generates the iterative sequence

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

where it is understood that we stop if $X^{(k)}$ is a fixed point. If $X^{(k)}$ is not a fixed point it follows from (21) that $\sigma(X^{(k+1)}) < \sigma(X^{(k)})$.

De Leeuw (1988) derives some additional results. Using up-arrows and down-arrows to indicate monotone convergence we have

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

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

$$\eta^2(X^{(k+1)} - X^{(k)}) \rightarrow 0. \tag{23}$$

Since the subdifferential is a upper semi-continuous (closed) map, and 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 from theorem 3.1 of Meyer (1976) that all accumulation points are fixed points and have the same function value σ_∞. Moreover, from theorem 26.1 of Ostrowski (1973), either the sequence converges or the accumulation points form a continuum.

In order to prove actual 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 our case, because of rotational indeterminacy. If X_∞ is a fixed point, then the continuum of rotations of X_∞ are all fixed points.

De Leeuw (1988) argues that the results so far are sufficient from a practical point of view. If we define an ϵ-fixed-point as any X with $\eta(X - \Phi(X)) < \eta$ then smacof produces such an ϵ-fixed-point in a finite number of steps.

In two very recent papers Ram and Sabach (2024 (in press)) and Robini, Wang, and Zhu (2024) use the powerful Kurdyka-Łojasiewicz (KL) framework (ref) to prove actual global convergence of smacof to a fixed point. We shall use a more classical approach, based on the differentiability of the Guttman transform.

2.2 Majorization and DCA

The original derivation of the smacof algorithm (De Leeuw (1977), De Leeuw and Heiser (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 (20) defines a majorization scheme for stress. Convergence then follows from the general *majorization principle* (these days mostly known as the *MM principle*). 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. The convex subproblem in each step is quadratic, and has the closed form solution provided by the Guttman transform. DCA is applied to MDS in Le Thi and Tao (2001), and extensive recent surveys of the DC/DCA approach are Le Thi and Tao (2018) and Le Thi and Tao (2024).

2.3 Rate of Convergence

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

$$\mathcal{D}\Phi_X(H) = V^+ \sum w_{ij} \frac{\delta_{ij}}{d_{ij}(X)} \left\{ A_{ij}H - \frac{\text{tr } X'A_{ij}H}{\text{tr } X'A_{ij}X} A_{ij}X \right\}. \tag{24}$$

It follows that $\mathcal{D}\Phi_X(X) = 0$ for all X and the Jacobian has at least one zero eigenvalue. If we think of (24) as a map on the space of all $n \times p$ matrices, then there are an additional p zero eigenvalues corresponding with translational invariance. If we define (24) on the column-centered matrices, then these 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 $\mathcal{D}\Phi_X$ 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.

Since $\Phi(X) = V^+\mathcal{D}\rho(X)$ the Jacobian of the Guttman transform has a simple relationship with the second derivatives of ρ. Since ρ is convex, all eigenvalues of the Jacobian are real and nonnegative.

We have the bilinear form

$$\mathcal{D}^2\rho_X(G, H) = \sum w_{ij} \frac{\delta_{ij}}{d_{ij}(X)} \left\{ \text{tr } G'A_{ij}H - \frac{\text{tr } G'A_{ij}X \text{tr } H'A_{ij}X}{d_{ij}^2(X)} \right\} = \text{tr } G'V\mathcal{D}\Phi_X(H). \tag{25}$$

It follows that $0 \lesssim \mathcal{D}^2\rho_X \lesssim B(X)$ in the Loewner sense. Of course $\mathcal{D}^2\sigma_X = V - \mathcal{D}^2\rho_X$ At a local minimum $\mathcal{D}^2\sigma_X \gtrsim 0$, and consequently $\mathcal{D}\Phi_X \lesssim I$. Thus all eigenvalues of the Jacobian at a local minimum are between zero and one.

We compute and store the Jacobian as a partitioned matrix with p rows and columns of the $n \times n$ matrices $\mathcal{D}_t\Phi_s(X)$, with $\Phi_s(X)$ column s of X.

$$\mathcal{D}^2\rho_X(e_ke_s', e_le_t') = \delta^{st}\{B(X)\}_{kl} - \{U_{st}(X)\}_{kl}$$

$$U_{st}(X) = \sum w_{ij}\delta_{ij}\frac{(x_{is}-x_{js})(x_{it}-x_{jt})}{d_{ij}^3(X)}A_{ij}$$

We apply basic iterations to the two-dimensional MDS analysis of the color-circle example from Ekman (1954), which has $n = 14$ points. We always start with the classical Torgerson-Gower solution and we stop if $\sigma(X^{(k)}) - \sigma(X^{(k+1)}) < 1e-15$. The fit in this example is very good and convergence is rapid. The results for the final iteration are

```
## itel  56 sold 2.1114112739 snew 2.1114112739 chng 0.0000000000 labd 0.7669784529
```

In this output "chng" is $\eta(X^{(k)} - X^{(k+1)})$, and "labd" is an estimate of the asymptotic convergence ratio, the "chng" divided by the "chng" of the previous iteration. To fifteen decimals stress is 2.1114112739076

We compute the Jacobian using the numDeriv package (Gilbert and Varadhan (2019)). Its eigenvalues are

```
##  [1]   +1.0000000000   +0.7669964993   +0.7480939418   +0.7185926294
##  [5]   +0.7007452309   +0.6920114813   +0.6859492534   +0.6593334529
##  [9]   +0.6541779410   +0.6477573343   +0.6237683213   +0.6178713316
## [13]   +0.5735285951   +0.5483330653   +0.5260355535   +0.5112510730
## [17]   +0.5064703617   +0.5059294792   +0.4919752630   +0.4827646555
## [21]   +0.4782034995   +0.4757907675   +0.4682965894   +0.4619226490
## [25]   +0.4559704884   -0.0000000000   +0.0000000000   +0.0000000000
```

Note that the second largest and first non-trivial eigenvalue is equal to "labd" from the final iteration.

2.4 Orthogonalized Iteration

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

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 $\Xi(X) = \Pi(\Phi(X))$, $\Pi(X) = XL$, where L are the right singular vectors of X.

We compute the Jacobian of Ξ. By the chain rule

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

If $X'XL = L\Lambda$ with $L'L = LL' = I$, assuming the eigenvalues in Λ are all different,

$$\mathcal{D}\Pi_X(H) = HL + XLS$$

where S is the anti-symmetric matrix with elements

$$s_{ij} = -\frac{l_i'(H'X + X'H)l_j}{\lambda_i - \lambda_j}$$

Thus, from … and …

$$\mathcal{D}\Xi_X(H) = \mathcal{D}\Phi_X(H)L + \Phi(X)LS$$

with L and S computed at $\Phi(X)$.

At a fixed point of Ξ we have $\Phi(X) = X$ and $\Pi(X) = X$ and consequently $L = I$ and $X'X = \Lambda$. Thus … becomes

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

where now

$$s_{ij} = -\frac{(H'X + X'H)_{ij}}{\lambda_i - \lambda_j}$$

If $H = XA$ with A anti-symmetric then

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

$$s_{ij} = -\frac{(A'\Lambda + \Lambda A)_{ij}}{\lambda_i - \lambda_j} = -a_{ij}$$

Thus $\mathcal{D}\Xi_X(XA) = 0$.

2.4.1 end intermezzo

Now clearly this modified algorithm generates the same sequence of function values as basic smacof. Moreover $\Phi^n(X) = \Pi(\Phi^n(X))$, which means that we can find any term of the orthogonal 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.

Theoretically, however, orthogonalization gives the same convergence rate as the basic sequence, but the Jacobian of Φ_o 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 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.

```
## itel  56 sold 2.1114112739 snew 2.1114112739 chng 0.0000000000 labd 0.7669940352
```

```
##  [1]   +0.7669964994   +0.7480939418   +0.7185926294   +0.7007452309
##  [5]   +0.6920114813   +0.6859492533   +0.6593334528   +0.6541779411
##  [9]   +0.6477573343   +0.6237683211   +0.6178713315   +0.5735285949
## [13]   +0.5483330653   +0.5260355534   +0.5112510731   +0.5064703617
## [17]   +0.5059294792   +0.4919752628   +0.4827646550   +0.4782034998
## [21]   +0.4757907672   +0.4682965894   +0.4619226491   +0.4559704888
## [25]   +0.0000000002   -0.0000000001   -0.0000000001   +0.0000000001
```

$$\Phi^o(X) = \Phi(X)K(\Phi(X))$$

2.5 Subspace Restrictions

Instead of orthogonalizing we can also restrict X to be in the subspace of all lower triangular column-centered $n \times p$ matrices (which means $x_{ij} = 0$ for all $i < j$). There are two different ways to accomplish this.

Method one uses a rotation of X to lower triangular form. The theory is pretty much the same as for the rotation to principal components in the previous section.

```
## itel  57 sold 2.1114112739 snew 2.1114112739 chng 108.2873334994 labd 1.0000000000
```

The results are the same as for the basis sequence, as predicted. The eigenvalues of the Jacobian are

```
##  [1]  -0.7669965027  -0.7480939418  -0.7185926293  -0.7007452300
##  [5]  -0.6920114811  -0.6859492533  -0.6593334524  -0.6541779410
##  [9]  -0.6477573342  -0.6237683212  -0.6178713316  -0.5735285947
## [13]  -0.5483330654  -0.5260355535  -0.5112510731  -0.5064703617
## [17]  -0.5059294793  -0.4919752630  -0.4827646550  -0.4782034983
## [21]  -0.4757907684  -0.4682965896  -0.4619226489  -0.4559704883
## [25]  +0.0000000001  +0.0000000001  -0.0000000000  -0.0000000000
```

The unit eigenvalues from the unrotated solution have been replaced by zeroes.

Method two uses the theory of constrained smacof from De Leeuw and Heiser (1980). 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, 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 $x_s = Y_s\theta_s$.

$$x_s^{(k+1)} = Y_sY_s'V\{\Phi(X^{(k)})\}_s$$

3 Two Point Iteration

3.1 Basic

De Leeuw and Heiser (1980) suggested the “relaxed” update

$$\Psi(X) := 2\Phi(X) - X \tag{26}$$

The reasoning here is two-fold. First, the smacof inequality (20) says

$$\sigma(X) \leq 1 + \eta^2(X - \Phi(Y)) - \eta^2(\Phi(Y)) \tag{27}$$

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

$$\sigma(\alpha\Phi(Y) + (1-\alpha)Y) \leq 1 + (1-\alpha)^2\eta^2(Y - \Phi(Y)) - \eta^2(\Phi(Y)) \tag{28}$$

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

$$1 + (1-\alpha)^2\eta^2(Y - \Phi(Y)) - \eta^2(\Phi(Y)) \leq 1 + \eta^2(Y - \Phi(Y)) - \eta^2(\Phi(Y)) = \sigma(Y) \tag{29}$$

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

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

$$\max_s |2\lambda_s - 1| = \max(2\lambda_{\text{max}} - 1, 1 - 2\lambda_{\text{min}}). \tag{30}$$

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 then

$$2\lambda_{\text{max}} - 1 = 1 - 2\epsilon \approx (1-\epsilon)^2 = \lambda_{\text{max}}^2, \tag{31}$$

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.

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

```
## itel 418 sold 2.1114112739 snew 2.1114112739 chng 0.0000000000 labd 0.9622371695
```

The loss function value decreases. The number of iterations is reduced from 57 to 23. But we see that $\eta^2(X^{(k+1)} - X^{(k)})$ does not converge to zero, and that σ_k converges to a value which does not even correspond to a local minimum of σ.

The eigenvalues of the Jacobian are

```
## [1]  -1.0000000000  -1.0000000000  -1.0000000000  +0.9999999999
## [5]  +0.5339930116  +0.4961879098  +0.4371856275  +0.4014904423
## [9]  +0.3840229515  +0.3718985108  +0.3186668913  +0.3083558925
## [13] +0.2955146860  +0.2475366678  +0.2357426702  +0.1470571705
## [17] +0.0966661311  -0.0880590227  -0.0761546972  -0.0634068160
## [21] +0.0520711061  -0.0484184543  -0.0435930088  -0.0344706947
## [25] +0.0225021452  -0.0160494761  +0.0129407209  +0.0118589425
```

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. Suppose X is a fixed point and $\tau\neq 1$. Then $\tau\overline{X}$ is not a fixed point, because $\Psi(\tau\overline{X})=(2-\tau)\overline{X}$. And

$$\sigma(\tau\overline{X})=1-\tau\rho(\overline{X})+\frac{1}{2}\tau^2\eta^2(\overline{X})=1-\frac{1}{2}\tau(2-\tau)\rho(\overline{X})=\sigma((2-\tau)\overline{X}) \tag{32}$$

Thus the algorithm has convergent subsequences which may not converge to a fixed point of Ψ (and thus of Φ). And indeed, an analysis of the 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 $\frac{1}{2}(X^{(k)}+X^{(k+1)})$. Making thus final adjustment in the Ekman sequence gives us a final stress equal to 2.11141127390762.

3.2 Doubling

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

$$X^{(k+1)}=\Psi(\Psi(X^{(k)})).$$

The asymptotic convergence rate is

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

.

With Ψ^2 the algorithm is everywhere strictly monotonic and converges to a fixed point. But not all problems 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.

```
## itel  23 sold 3.9946270666 snew 3.9946270666 chng 3.7664315853 labd 1.0000000000
```

```
## [1]  +1.0000000001  -1.0000000001  -1.0000000000  -1.0000000000
## [5]  +0.5339929984  +0.4961878837  +0.4371852589  +0.4014904620
## [9]  +0.3840229626  +0.3718985067  +0.3186669058  +0.3083558821
## [13] +0.2955146686  +0.2475366426  +0.2357426630  +0.1470571901
## [17]   +0.0966661308   -0.0880590232   -0.0761547018   -0.0634068213
## [21]   +0.0520711070   -0.0484184653   -0.0435930008   -0.0344706891
## [25]   +0.0225021462   -0.0160494740   +0.0129407234   +0.0118589585
```

stress is 2.1114112739076

3.3 Scaling

De Leeuw (2006) discusses some other ways to fix of the relaxed update problem.

3.4 Switching

$$\Phi(\Psi(X)) \tag{33}$$

$$\max_s |\lambda_s(2\lambda_s - 1)| \tag{34}$$

4 Benchmarking

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

```
## Warning in microbenchmark(smacofAccelerate(delta, ndim = 2, opt = 1, halt = 2,
## : less accurate nanosecond times to avoid potential integer overflows
```

```
## Unit: milliseconds
##                                                                     expr
##  smacofAccelerate(delta, ndim = 2, opt = 1, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 2, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 3, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 4, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 5, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 6, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 7, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 8, halt = 2, verbose = FALSE)
##        min        lq      mean    median        uq       max neval
##   3.302304  3.404988  3.764543  3.564909  3.681083  6.229745   100
##   4.362810  4.552537  4.920569  4.712991  4.902062  7.359951   100
##   4.031981  4.176035  4.519865  4.299977  4.459303  8.645137   100
##  24.817546 26.797497 27.845782 27.532094 28.340491 48.516120   100
##   1.570505  1.636372  1.813730  1.704247  1.802278  3.869129   100
##   1.375140  1.440514  1.597812  1.492625  1.575179  3.739774   100
##   1.805681  1.880977  2.126657  1.961932  2.060824  4.392822   100
##   1.700188  1.747133  1.901704  1.834852  1.931346  4.144936   100
```

De Gruijter (1967)

```
## Unit: milliseconds
##                                                                     expr      min
##  smacofAccelerate(delta, ndim = 2, opt = 1, halt = 2, verbose = FALSE) 43.66922
##  smacofAccelerate(delta, ndim = 2, opt = 2, halt = 2, verbose = FALSE) 60.01346
##  smacofAccelerate(delta, ndim = 2, opt = 3, halt = 2, verbose = FALSE) 54.61708
##  smacofAccelerate(delta, ndim = 2, opt = 4, halt = 2, verbose = FALSE) 53.96703
##  smacofAccelerate(delta, ndim = 2, opt = 5, halt = 2, verbose = FALSE) 20.64793
##  smacofAccelerate(delta, ndim = 2, opt = 6, halt = 2, verbose = FALSE) 14.30634
##  smacofAccelerate(delta, ndim = 2, opt = 7, halt = 2, verbose = FALSE) 23.41904
##  smacofAccelerate(delta, ndim = 2, opt = 8, halt = 2, verbose = FALSE) 20.66552
##        lq     mean   median       uq      max neval
##  44.90517 46.44803 46.22732 47.03663 59.89698   100
##  61.31038 62.45559 61.93280 63.16255 77.60300   100
##  57.03239 58.80424 57.56449 58.62414 76.46316   100
##  56.32861 57.58191 57.01751 58.66670 64.54958   100
##  22.37188 23.33055 22.77958 23.20621 45.44153   100
##  14.71978 15.90683 15.79748 16.49192 35.54975   100
##  25.37962 26.08471 25.63215 26.06641 44.67020   100
##  22.32725 22.79460 22.81800 23.31959 25.65673   100
```

```
## Unit: milliseconds
##                                                                      expr
##  smacofAccelerate(delta, ndim = 2, opt = 1, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 2, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 3, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 4, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 5, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 6, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 7, halt = 2, verbose = FALSE)
##  smacofAccelerate(delta, ndim = 2, opt = 8, halt = 2, verbose = FALSE)
##        min        lq      mean    median        uq       max neval
##   48.85945  49.81820  53.13325  50.30183  51.76998  74.76916   100
##   57.82927  58.92885  61.29368  59.59676  60.97850  88.88115   100
##   55.75360  56.94654  61.67891  57.89831  60.51864 102.74170   100
##  109.66938 111.57547 116.77156 113.64247 116.60769 138.07271   100
##   18.60367  19.39880  20.85011  19.81530  22.23578  27.08251   100
##   15.47524  15.86499  17.94653  16.14521  18.91890  38.76784   100
##   26.33061  29.11808  30.04142  29.78863  30.29945  52.95421   100
##   24.71496  25.71698  27.75145  28.27087  28.73694  48.26262   100
```

5 Code

5.1 smacofAccelerate.R

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

source("smacofUtils.R")

smacofAccelerate <- function(delta,
                             ndim = 2,
                             wgth = 1 - diag(nrow(delta)),
                             xold = smacofTorgerson(delta, ndim),
                             opt = 1,
                             halt = 0,
                             wd = 4,
                             dg = 15,
                             itmax = 1000,
                             epsx = 1e-10,
                             epsf = 1e-15,
                             verbose = 1) {
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- ginv(vmat)
  nobj <- nrow(xold)
  xold <- xold %*% qr.Q(qr(t(xold[1:ndim, ])))
  bs <- smacofMakeBasis(nobj, ndim, wgth)
  cold <- Inf
  itel <- 1
  repeat {
    xold <- apply(xold, 2, function(x)
      x - mean(x))
    dold <- as.matrix(dist(xold))
    sold <- sum(wgth * (delta - dold) ^ 2)
    bold <- -wgth * delta / (dold + diag(nobj))
    diag(bold) <- -rowSums(bold)
    xbar <- vinv %*% bold %*% xold
    if (opt == 1) {
      h <- smacofOptionOne(xold, xbar, delta, wgth, vmat)
    }
    if (opt == 2) {
      h <- smacofOptionTwo(xold, xbar, delta, wgth, vmat)
    }
    if (opt == 3) {
      h <- smacofOptionThree(xold, xbar, delta, wgth, vmat)
    }
    if (opt == 4) {
      h <- smacofOptionFour(xold, xbar, delta, wgth, vmat, bs)
    }
    if (opt == 5) {
      h <- smacofOptionFive(xold, xbar, delta, wgth, vmat)
    }
    if (opt == 6) {
      h <- smacofOptionSix(xold, xbar, delta, wgth, vmat, vinv)
    }
    if (opt == 7) {
      h <- smacofOptionSeven(xold, xbar, delta, wgth, vmat)
    }
    if (opt == 8) {
      h <- smacofOptionEight(xold, xbar, 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
  }
  if (verbose == 1) {
    smacofLinePrint(itel, sold, h$snew, h$cnew, labd, wd = wd, dg = dg)
  }
  if (opt == 5) {
    h$xnew <- (h$xnew + xold) / 2
    h$dnew <- as.matrix(dist(h$xnew))
    h$snew <- sum(wgth * (delta - h$dnew) ^ 2)
    smacofLinePrint(itel, sold, h$snew, h$cnew, labd, wd = wd, dg = dg)
  }
```

```r
  if (opt == 6) {
    bold <- -wgth * delta / (h$dnew + diag(nobj))
    diag(bold) <- -rowSums(bold)
    h$xnew <- vinv %*% bold %*% h$xnew
    h$dnew <- as.matrix(dist(h$xnew))
    h$snew <- sum(wgth * (delta - h$dnew) ^ 2)
    smacofLinePrint(itel, sold, h$snew, h$cnew, labd, wd = wd, dg = dg)
  }
  return(
    list(
      x = h$xnew,
      s = h$snew,
      d = h$dnew,
      itel = itel,
      chng = h$cnew,
      labd = labd,
      wgth = wgth,
      delta = delta
    )
  )
}

smacofOptionOne <- function(xold, xbar, delta, wgth, vmat) {
  xnew <- xbar
  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, xbar, delta, wgth, vmat) {
  ndim <- ncol(xold)
  xnew <- xbar %*% qr.Q(qr(t(xbar[1:ndim, ])))
  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, xbar, delta, wgth, vmat) {
  xnew <- xbar %*% svd(xbar)$v
  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, xbar, delta, wgth, vmat, bs) {
  ndim <- ncol(xold)
  nobj <- nrow(xold)
  xnew <- matrix(0, nobj, ndim)
  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, xbar, delta, wgth, vmat) {
  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,
```

```r
    dnew = dnew,
    snew = snew,
    cnew = cnew
  ))
}

smacofOptionSix <- function(xold, xbar, delta, wgth, vmat, vinv) {
  nobj <- nrow(xold)
  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, xbar, delta, wgth, vmat) {
  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, xbar, delta, wgth, vmat, vinv) {
  nobj <- nrow(xold)
  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
  ))
}
```

5.2 smacofHessian.R

```
numFunc <- function(x, nobj, ndim, wgth, delta, opt = 1) {
  xx <- matrix(x, nobj, ndim)
  dd <- as.matrix(dist(xx))
  vv <- -wgth
  diag(vv) <- -rowSums(vv)
  vinv <- solve(vv + (1 / nobj)) - (1 / nobj)
  bb <- -wgth * delta / (dd + diag(nobj))
  diag(bb) <- -rowSums(bb)
  xaux <- vinv %*% bb %*% xx
  if (opt == 1) {
    yy <- xaux
  }
  if (opt == 2) {
    lbd <- sqrt(sum(xaux[1, ] ^ 2))
    cs <- xaux[1, 2] / lbd
    sn <- xaux[1, 1] / lbd
    rot <- matrix(c(sn, cs, -cs, sn), 2, 2)
    yy <- xaux %*% rot
  }
  if (opt == 3) {
    yy <- xaux %*% svd(xaux)$v
  }
  if (opt == 4) {
    yy <- 2 * xaux - xx
  }
  return(as.vector(yy))
```

```
}

numHess <- function(x,
                    delta,
                    wgth = 1 - diag(nrow(x)),
                    opt = 1) {
  nobj <- nrow(x)
  ndim <- ncol(x)
  x <- as.vector(x)
  h <- jacobian(
    numFunc,
    x,
    nobj = nobj,
    ndim = ndim,
    wgth = wgth,
    delta = delta,
    opt = opt
  )
  return(h)
}

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

```r
}

smacofJacobian <- function(x, delta, wgth) {
  n <- nrow(x)
  p <- ncol(x)
  np <- n * p
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  vinv <- solve(vmat + (1 / n)) - (1 / n)
  jacob <- smacofRhoHessian(x, delta, wgth)
  for (s in 1:p) {
    ns <- (s - 1) * n + 1:n
    for (t in 1:p) {
      nt <- (t - 1) * n + 1:n
      jacob[ns, nt] <- vinv %*% jacob[ns, nt]
    }
  }
  return(jacob)
}
```

5.3 smacofCompare.R

```r
smacofCompare <- function(delta, ndim = 2) {
  nobj <- nrow(delta)
  wgth <- 1 - diag(nobj)
  xold <- smacofTorgerson(delta, ndim)
  return(
    microbenchmark(
      smacofAccelerate(
        delta,
        ndim = 2,
        opt = 1,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        ndim = 2,
        opt = 2,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        ndim = 2,
        opt = 3,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        ndim = 2,
        opt = 4,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        ndim = 2,
        opt = 5,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        ndim = 2,
        opt = 6,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        ndim = 2,
        opt = 7,
        halt = 2,
        verbose = FALSE
      ),
      smacofAccelerate(
        delta,
        ndim = 2,
        opt = 8,
        halt = 2,
        verbose = FALSE
      )
    )
  )
}
```

5.4 smacofUtils.R

```
mPrint <- 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, wgth = 1 - diag(n)) {
  vmat <- -wgth
  diag(vmat) <- -rowSums(vmat)
  y <- as.list(1:ndim)
  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]] <- tcrossprod(aux, solve(chol(sux)))
  }
  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)
}
```

References

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.

———. 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.

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.

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., 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.