ACCELERATED LEAST SQUARES MULTIDIMENSIONAL SCALING

JAN DE LEEUW

ABSTRACT. We discuss some simple acceleration methods for the SMACOF iterations used in Multidimensional Scaling (MDS). The relaxation method proposed by De Leeuw and Heiser in 1980 tends to speed up the basic algorithm by a factor two. The method proposed in this paper leads to an algorithm that is about four times as fast as the original one. It can easily be implemented within the usual SMACOF programs.

1. CANONICAL MDS PROBLEMS

The least squares metric multidimensional scaling or MDS problem is the minimization of

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

over all $m\times p$ *configurations* X. Here w_{ij} are given non-negative *weights* and δ_{ij} are given non-negative *dissimilarities*. The $d_{ij}(X)$ are the Euclidean distances between rows i and j of X. Thus

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

The MDS problem is the minimization of the function (1), which is often called *stress*, over the $n\times p$ variables X. Parametrization the problem in terms of the configurations is mathematically not very convenient. Distances are invariant under translations and rotations, which means there are really only $np-\frac{1}{2}p(p+1)$ variables. The fact that some coordinates are in the same row and others are in different rows leads to unpleasant expressions, for example for the higher partial derivatives.

Date: May 26, 2008 — 21h 18min — Typeset in TIMES ROMAN.

2000 *Mathematics Subject Classification.* 62H25.

Key words and phrases. Multivariate Analysis, Multidimensional Scaling, Convergence Acceleration.

By reparametrizing we can put the MDS problems in a more simple canonical form [De Leeuw, 1993]. As a first step, use the fact that

$$d_{ij}^2(X) = (e_i - e_j)'XX'(e_i - e_j) = \mathbf{tr}\, X'A_{ij}X,$$

where the e_i and e_j are unit vectors (columns of the identity matrix), and where $A_{ij} = (e_i - e_j)(e_i - e_j)'$.

As a second step, use a basis of $n \times p$ matrices $Y_1, \cdots, Y_K$ to express the configuration as $X = \sum_{k=1}^K \theta_k Y_k$. Using a basis allows one to eliminate the indeterminacy due to translation and rotation, and also makes it possible to easily include MDS problems in which X is constrained to lie in a subspace of $\mathbb{R}^{n\times p}$.

Using our new parameters

$$d_{ij}^2(\theta) = \theta' G_{ij}\theta,$$

where G_{ij} is a matrix of order K with element (k,ℓ) equal to $\mathbf{tr}\, Y_k'A_{ij}Y_\ell$. Thus

$$\sigma(\theta) = 1 + \frac{1}{2}\mathbf{tr}\ \theta'V\theta - \sum_{i=1}^n\sum_{j=1}^n w_{ij}\delta_{ij}\sqrt{\theta' G_{ij}\theta}.$$

where $V = \sum_{i=1}^n\sum_{j=1}^n w_{ij}G_{ij}$.

The next simplification is to use a decomposition $V = SS'$ to define new parameters $\xi = S'\theta$. Define

$$\sigma(\xi) = 1 + \frac{1}{2}\|\xi\|^2 - \sum_{i=1}^n\sum_{j=1}^n w_{ij}\delta_{ij}\sqrt{\xi' C_{ij}\xi},$$

where $C_{ij} = S^{-1}G_{ij}(S^{-1})'$, and thus $\sum_{i=1}^n\sum_{j=1}^n w_{ij}C_{ij} = I$.

The final step is to forget about the double indexing altogether. Just assume we have M positive semi-definite matrices C_m of order K, and two M-vectors w and δ, such that $\sum_{m=1}^M w_m C_m = I$ and $\frac{1}{2}\sum_{m=1}^M \delta_m^2 = 1$. Find ξ that minimizes

$$\sigma(\xi) = 1 + \frac{1}{2}\|\xi\|^2 - \sum_{m=1}^M w_m\delta_m d_m(\xi), \tag{2}$$

where $d_m(\xi) = \sqrt{\xi' C_m\xi}$. This is our final form of the MDS problem. Computationally this usually is not the best parametrization, because the matrices C_m can be large and sparse, but conceptually (2) is simpler to deal with than (1). Formula's become shorter and more elegant, expressions are easier to manipulate, and programs are simpler to write.

It is shown in De Leeuw [1984] that in the neighborhood of a local minimum of stress we have $d_m(\xi) > 0$ for all m. For completeness we give a proof here. Define $M_+(\xi)$ as the set of all m with $d_m(\xi) > 0$ and $M_0(\xi)$ as the set of all m for which $d_m(\xi) = 0$. Also define

$$B(\xi) = \sum_{m \in M_+(\xi)} w_m \frac{\delta_m}{d_m(\xi)} C_m.$$

Then the directional derivatives of stress at ξ in the direction τ exist and are equal to

$$\sigma'(\xi, \tau) = \lim_{\varepsilon \downarrow 0} \frac{\sigma(\xi + \varepsilon\tau) - \sigma(\xi)}{\varepsilon} = (\xi - B(\xi)\xi)'\tau - \sum_{m \in M_0(\xi)} w_m \delta_m d_m(\tau).$$

At a local minimum $\hat{\xi}$ we must have $\sigma'(\hat{\xi}, \tau) \geq 0$ for all τ. If $\hat{\xi} - B(\hat{\xi})\hat{\xi} \neq 0$ then clearly there is a τ such that $\sigma'(\hat{\xi}, \tau) < 0$. Thus $\hat{\xi} - B(\hat{\xi})\hat{\xi} = 0$ and $\sigma'(\hat{\xi}, \tau) = -\sum_{m \in M_0(\hat{\xi})} w_m \delta_m d_m(\tau)$. If $M_0(\hat{\xi})$ is non-empty then again there is a τ such that $\sigma'(\hat{\xi}, \tau) < 0$. Thus $M_0(\hat{\xi})$ is empty, and $d_m(\hat{\xi}) > 0$ for all m. In fact there is an open sphere $\mathscr{S}(\hat{\xi})$, centered at $\hat{\xi}$, such that $d_m(\xi) > 0$ for all $\xi \in \mathscr{S}(\hat{\xi})$.

At a point where $d_m(\xi) > 0$ for all m the partials are

$$\mathscr{D}\sigma(\xi) = \xi - B(\xi)\xi.$$

The second partials are

$$\mathscr{D}^2\sigma(\xi) = I - H(\xi),$$

with

$$H(\xi) = \sum_{m=1}^{M} w_m \frac{\delta_m}{d_m(\xi)} \left\{ C_m - \frac{C_m \xi \xi' C_m}{\xi' C_m \xi} \right\}.$$

Observe that $H(\xi)$ is positive semi-definite, and that its smallest eigenvalue, corresponding to the eigenvector ξ, is equal to zero. At a strict local minimum $\mathscr{D}^2\sigma(\xi)$ is positive definite, which means that $H(\xi)$ has all its eigenvalues strictly less than one.

2. THE SMACOF ALGORITHM

The SMACOF algorithm for MDS [De Leeuw, 1977] is derived from the Cauchy-Schwarz inequality. For all ξ and for all θ such that $d_m(\theta) > 0$ we have

$$d_m(\xi) \geq \frac{1}{d_m(\theta)} \xi' C_m \theta.$$

This implies that

$$\sum_{m=1}^{m} w_m \delta_m d_m(\xi) \geq \xi' B(\theta)\theta,$$

and thus

$$\sigma(\xi) \leq 1 + \frac{1}{2}\|\xi\|^2 - \xi' B(\theta)\theta. \tag{3}$$

Define the *Guttman transform* $\Phi(\xi) = B(\xi)\xi$ and the *majorization function*

$$\eta(\xi, \theta) = 1 + \frac{1}{2}\|\xi - \Phi(\theta)\|^2 - \frac{1}{2}\|\Phi(\theta)\|^2. \tag{4}$$

Clearly $\sigma(\xi) \leq \eta(\xi, \theta)$ for all ξ, θ and $\sigma(\xi) = \eta(\xi, \xi)$ for all ξ.

If a superscript (ν) is used for the iteration counter, then one iteration step of the SMACOF algorithm computes $\xi^{(\nu+1)}$ by minimizing $\eta(\xi, \xi^{(\nu)})$, i.e. by setting $\xi^{(\nu+1)} = \Phi(\xi^{(\nu)})$. If $\xi^{(\nu+1)} = \xi^{(\nu)} = \Phi(\xi^{(\nu)})$ we stop. If not, the *sandwich inequality*

$$\sigma(\xi^{(\nu+1)}) \leq \eta(\xi^{(\nu+1)}, \xi^{(\nu)}) < \eta(\xi^{(\nu)}, \xi^{(\nu)}) = \sigma(\xi^{(\nu)})$$

tell us the value of stress is always decreased, i.e. the algorithm is *stable* [Varadhan and Roland, 2008]. For ease of reference we call SMACOF algorithm $\mathscr{A}_1$.

Algorithm $\mathscr{A}_1$ (SMACOF)
$\xi^{(\nu+1)} = \Phi(\xi^{(\nu)})$

The *iteration function* is $\Phi(\xi) = B(\xi)\xi$, and the *iiteration sequence*, which depends on the starting point $\xi^{(0)}$, is the Picard sequence $\xi^{(\nu+1)} = \Phi(\xi^{(\nu)})$. It is shown in De Leeuw [1977] what to do if the algorithm happens to stumble on a point ξ where one or more of the $d_m(\xi)$ are zero.

Suppose the SMACOF algorithm converges to $\xi_\star$, a fixed point of Φ and a point where the derivative of σ is zero. The linear convergence rate of this algorithm is $\rho(\mathscr{D}\Phi(\xi_\star))$, the spectral radius (the largest eigenvalue in modulus, or the square of the largest singular value) of the derivative of the iteration function. See Ostrowski [1966] or Ortega and Rheinboldt [1970, Chapter 10]. Note that, by De Leeuw [1984], this derivative exists at a local minimum. Since $\mathscr{D}\Phi(\xi) = H(\xi)$, the derivative of the iteration function at a strict local minimum $\xi_\star$ has all its eigenvalues between zero and one, with the largest one strictly less than one and the smallest one precisely equal to zero [De Leeuw, 1988].

More precisely, suppose $\lambda_i(\xi_\star)$ are the K eigenvalues of $H(\xi_\star)$ ordered as

$$0 = \lambda_K(\xi_\star) \leq \lambda_{n-1}(\xi_\star) \leq \cdots \leq \lambda_2(\xi_\star) \leq \lambda_1(\xi_\star) < 1.$$

The rate of convergence of the iteration $\xi^{(\nu+1)} = \Phi(\xi^{(\nu)})$ is

$$\rho_1(\xi_\star) = \rho(\mathscr{D}\Phi(\xi_\star)) = \max_{k=1}^{K} |\lambda_i(\xi_\star)| = \lambda_1(\xi_\star).$$

Thus if $\xi^{(\nu)}$ converges to a local minimum at $\xi_\star$, then

$$\limsup_{k\to\infty} \frac{\|\xi^{(\nu+1)} - \xi_\star\|}{\|\xi^{(\nu)} - \xi_\star\|} = \lambda_1(\xi_\star),$$

and

$$\limsup_{k\to\infty} \frac{\sigma(\xi^{(\nu+1)}) - \sigma(\xi_\star)}{\sigma(\xi^{(\nu)}) - \sigma(\xi_\star)} = \lambda_1^2(\xi_\star).$$

3. Linear Multipoint Acceleration

Now define the more general multipoint iteration function

$$\Psi_\alpha(\xi) = \sum_{r=0}^{s} \alpha_r \Phi_r(\xi), \tag{5}$$

where $\Phi_0(\xi) = \xi$ and $\Phi_r(\xi) = \Phi(\Phi_{r-1}(\xi))$ for $r > 0$. To make sure fixed points of Φ are also fixed points of Ψ we require $\sum_{r=0}^{r} \alpha_r = 1$. Note that we do not require the α_r to be non-negative, but we do require them to be the same in all iterations. In a later section we shall drop this stationarity requirement.

Now

$$\mathscr{D}\Psi_\alpha(\xi) = \sum_{r=0}^{s} \alpha_r \mathscr{D}\Phi_r(\xi) = \sum_{r=0}^{s} \alpha_r [\mathscr{D}\Phi(\xi)]^r.$$

The eigenvalues of the iteration function Ψ_α at ξ are $\sum_{r=0}^{s} \alpha_r \lambda_k^r(\xi)$, and the rate of convergence at $\xi_\star$ is

$$\rho(\mathscr{D}\Psi_\alpha(\xi_\star)) = \max_{k=1}^{K} |\sum_{r=0}^{s} \alpha_r \lambda_k^r(\xi_\star)|. \tag{6}$$

This implies that for linear multipoint methods the best attainable rate is

$$\hat{\rho}_s(\xi_\star) = \min_{\alpha_0+\cdots+\alpha_s=1} \rho(\mathscr{D}\Psi_\alpha(\xi_\star)). \tag{7}$$

Define the Vandermonde matrix $\Lambda(\xi_\star)$ by

$$\begin{bmatrix} 1 & \lambda_1(\xi_\star) & \lambda_1^2(\xi_\star) & \cdots & \lambda_1^s(\xi_\star) \\ 1 & \lambda_2(\xi_\star) & \lambda_2^2(\xi_\star) & \cdots & \lambda_2^s(\xi_\star) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_K(\xi_\star) & \lambda_K^2(\xi_\star) & \cdots & \lambda_K^s(\xi_\star) \end{bmatrix}.$$

Then $\Lambda(\xi_\star)$ is of dimension $K\times(s+1)$. Suppose, for the moment, that all eigenvalues are different. Then if $s=K$ there will be a $\hat\alpha(\xi_\star)$ such that $\Lambda(\xi_\star)\hat\alpha(\xi_\star)=0$. The $\hat\alpha(\xi_\star)$ are the coefficients of a polynomial of degree s in λ that has $\lambda_1(\xi_\star),\cdots,\lambda_K(\xi_\star)$ as its roots. The polynomial is unique up to multiplication by a constant, and we can use this constant to normalize the coefficients so they add up to one. This gives

$$\pi_\star(\lambda)=\prod_{k=1}^K\frac{\lambda-\lambda_k(\xi_\star)}{1-\lambda_k(\xi_\star)}=\sum_{r=0}^s\hat\alpha_r(\xi_\star)\lambda^r.$$

Note that $\pi_\star(1)=\sum_{r=0}^s\hat\alpha_r(\xi_\star)=1$. Also note that if $\lambda_K(\xi_\star)=0$, as in MDS, then also $\hat\alpha_0(\xi_\star)=0$. If some eigenvalues are equal, then some rows of $\Lambda(\xi_\star)$ will be equal, and we must eliminate these multiples. It then suffices to choose the degree of the polynomial equal to the number of distinct eigenvalues of $\mathscr{D}\Phi(\xi_\star)$. We can easily compute the coefficients $\hat\alpha(\xi_\star)$ by using the `charPoly()` function in the Appendix. The iteration function using $\hat\alpha(\xi_\star)$ has quadratic convergence, but of course cases in which the $\lambda_k(\xi_\star)$ are known beforehand will be rare indeed.

We have shown that if s is equal to the number of distinct eigenvalues, then at least theoretically we can attain quadratic convergence by an appropriate choice of coefficients. For smaller s computing the best rate $\rho_s(\xi_\star)$ for given $\lambda_1(\xi_\star),\cdots,\lambda_K(\xi_\star)$ is a discrete linear Chebyshev problem. By eliminating α_0 we get

$$\rho_s(\xi_\star)=\min_\alpha\max_{k=1}^K|\sum_{r=1}^s\alpha_r(1-\lambda_k^r(\xi_\star))-1|. \tag{8}$$

The Appendix gives an `R` interface `chebR()` to the `FORTRAN` routine `CHEB`, to compute discrete Chebyshev approximations [Barrodale and Philips, 1975].

Now consider a more general situation in which the coefficients of the linear combination also depend on the current solution. Thus

$$\Psi_\alpha(\xi)=\sum_{r=0}^s\alpha_r(\xi)\Phi_r(\xi), \tag{9}$$

where $\Phi_0(\xi)=\xi$ and $\Phi_r(\xi)=\Phi(\Phi_{r-1}(\xi))$ for $r>0$, and where now we require $\sum_{r=0}^r\alpha_r(\xi)=1$ for all ξ. If the $\alpha_r(\xi)$ are differentiable functions of ξ, then

$$\mathscr{D}\Psi_\alpha(\xi)=\sum_{r=0}^s\mathscr{D}\alpha_r(\xi)\Phi_r(\xi)+\sum_{r=0}^s\alpha_r(\xi)[\mathscr{D}\Phi(\xi)]^r,$$

and at a stationary point $\xi_\star$ where $\Phi_r(\xi_\star)=\xi_\star$ for all r we have

$$\mathscr{D}\Psi_\alpha(\xi_\star)=\sum_{r=0}^s\alpha_r(\xi_\star)[\mathscr{D}\Phi(\xi_\star]^r.$$

Thus the convergence rate is still

$$\rho(\mathscr{D}\Psi_\alpha(\xi_\star)) = \max_{k=1}^{K} |\sum_{r=0}^{s} \alpha_r(\xi_\star)\lambda_k^r(\xi_\star)|. \tag{10}$$

Note that the derivatives of the coefficients $\alpha_r(\xi)$ at $\xi_\star$ do not play a role in (10). For differentiable coefficients the rate is computed as if the coefficients are the constants $\alpha_r(\xi_\star)$. Thus the Chebyshev optimal bounds still apply to the case with non-constant, but differentiable, coefficients in the linear combinations.

4. THREE-POINT ITERATIONS

We do not have a practical procedure, however, to implement general optimal linear multipoint iterations in the MDS case, and thus we look for simplifications. The first simplification will be to limit ourselves, for the rest of the paper, to three-point iterations of the form

$$\Psi(\xi) = \alpha_0\xi + \alpha_1\Phi(\xi) + \alpha_2\Phi(\Phi(\xi)),$$

with, of course, $\alpha_0 + \alpha_1 + \alpha_2 = 1$. The convergence rate is

$$\rho(\mathscr{D}\Psi_{\alpha_0,\alpha_1,\alpha_2}(\xi_\star)) = \max_{k=1}^{K} |\alpha_0\xi_\star + \alpha_1\lambda_k(\xi_\star) + \alpha_2\lambda_k^2(\xi_\star)|,$$

and the optimal convergence rate is

$$\hat{\rho}_2(\xi_\star) = \min_{\alpha_0+\alpha_1+\alpha_2=1} \rho(\mathscr{D}\Psi_{\alpha_0,\alpha_1,\alpha_2}(\xi_\star)).$$

Instead of tackling the general three-point problem directly, we simplify it some more, first by setting $\alpha_2 = 0$. The optimal rate becomes

$$\hat{\rho}_1(\xi_\star) = \min_\alpha \max_{k=K}^{n} |\alpha\lambda_k(\xi_\star) + (1-\alpha)|.$$

The iteration function is $\Psi_\alpha(\xi) = \alpha\Phi(\xi) + (1-\alpha)x = x - \alpha\mathscr{D}\sigma(\xi)$, i.e. to a gradient method with constant step-size α.

The computation of $\hat{\rho}_1$ is illustrated in Figure 2, which shows an example with $\lambda = 0.00(0.01)0.99$. Each of the 100 functions graphed in the plot is a V-shaped function of the form $|\alpha\lambda_k + (1-\alpha)|$, and the boundary of the wedge-shaped area in the middle is the function defined as the maximum of those 100 V's. The minimum is attained at the intersection of $\alpha\lambda_1 + (1-\alpha)$ (in green) and $-\alpha\lambda_K - (1-\alpha)$ (in blue), i.e. at

$$\hat{\alpha} = \frac{2}{2 - (\lambda_1 + \lambda_K)}.$$

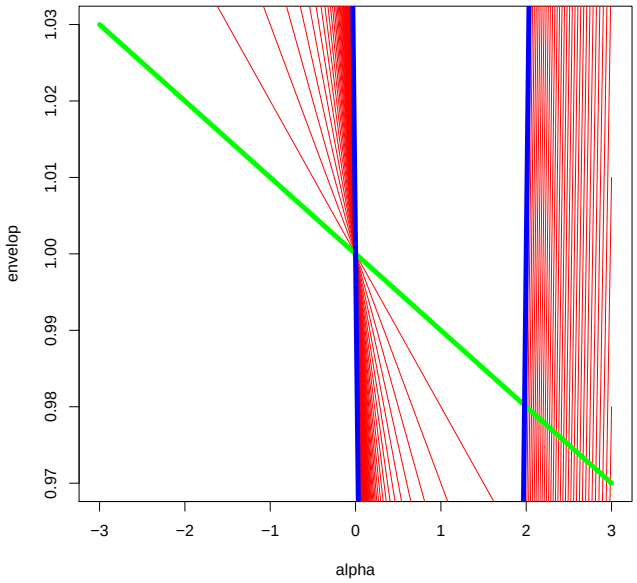

FIGURE 1. Computing $\hat{\rho}_1$

At this point the rate is

$$\hat{\rho}_1 = \frac{\lambda_1 - \lambda_K}{2 - (\lambda_1 + \lambda_K)}. \tag{11a}$$

If $\lambda_K = 0$, as in SMACOF, the best rate is

$$\hat{\rho}_1 = \frac{\lambda_1}{2 - \lambda_1}, \tag{11b}$$

which is attained at

$$\hat{\alpha} = \frac{2}{2 - \lambda_1}.$$

If, for example, λ_1 is 0.99 and thus the optimal two step rate is $\hat{\rho}_1 = 0.980198$, attained for $\hat{\alpha} = 1.980198$.

Of course in actual examples we do not know λ_1, so we cannot implement

$$\Psi_\alpha(\xi) = \frac{2}{2 - \lambda_1}\Phi(\xi) - \frac{\lambda_1}{2 - \lambda_1}x$$

directly. In an implementation we can replace λ_1 by a simple estimate, which does not influence the asymptotic convergence rate. Thus the algorithm, which is generally not stable, becomes

Algorithm $\mathscr{A}_2$ (STEP)

$$\begin{aligned}
\zeta^{(\nu)} &= \Phi(\xi^{(\nu)}),\\
\hat\lambda_1^{(\nu)} &= \frac{\|\zeta^{(\nu)}-\xi^{(\nu)}\|}{\|\zeta^{(\nu-1)}-\xi^{(\nu-1)}\|},\\
\hat\alpha^{(\nu)} &= \frac{2}{2-\hat\lambda_1^{(\nu)}},\\
\xi^{(\nu+1)} &= \hat\alpha^{(\nu)}\zeta^{(\nu)}+(1-\hat\alpha^{(\nu)})\eta^{(\nu)}.
\end{aligned}$$

This algorithm can be just as easily implemented in the original configuration parametrization, there is no need computationally to switch to the canonical form involving the C_m.

4.1. **The Relaxed Update.** Equation (4) implies that *stability* $\sigma(\xi^{(\nu+1)})\leq\sigma(\xi^{(\nu)})$ is guaranteed if we update by

$$\xi^{(\nu+1)}=\alpha\Phi(\xi^{(\nu)})+(1-\alpha)\xi^{(\nu)}=\xi^{(\nu)}-\alpha\mathscr{D}\sigma(\xi^{(\nu)})$$

for any $0\leq\alpha\leq 2$.

De Leeuw and Heiser [1980] have proposed to use the update with $\alpha=2$. The reasoning was that if $\lambda_1=1-\varepsilon$, with ε small, then $2\lambda_1-1=\lambda_1^2-\varepsilon^2<\lambda_1^2$. Thus the convergence rate goes from λ_1 to λ_1^2 and the algorithm uses only approximately half the number of iterations. This, however, does not tell the whole story. Because $\lambda_K=0$ is an eigenvalue of the SMACOF iteration, the convergence rate of the relaxed update $\Psi(\xi)=2\Phi(\xi)-\xi$ is $\max_{k=1}^K|2\lambda_k-1|=1$, which indicates sublinear convergence. The sandwich inequality for the relaxed update is

$$\sigma(\xi^{(\nu+1)})\leq\eta(\xi^{(\nu+1)},\xi^{(\nu)})=\eta(\xi^{(\nu)},\xi^{(\nu)})=\sigma(\xi^{(\nu)}),$$

which means we cannot show strict improvement in each iteration.

In fact, suppose we start iteration ν with $\xi^{(\nu)}=\tau\hat\xi$, where $\hat\xi$ is a fixed point of the Guttman transform and $\tau\neq 1$. Then for the relaxed update $\xi^{(\nu+1)}=\Psi(\xi^{(\nu)})=(2-\tau)\hat\xi$ and $\xi^{(\nu+2)}=\Psi(\Psi(\xi^{(\nu)}))=\tau\hat\xi=\xi^{(\nu)}$. Thus there is no convergence and the sequence cycles between the two different points $\tau\hat\xi$ and $(2-\tau)\hat\xi$. Both points have stress value $1+\frac{1}{2}\tau(\tau-2)\|\hat\xi\|^2>\sigma(\hat\xi)$. Moreover any point of the form $\tau\hat\xi$ is a fixed point of $\Psi_2(\xi)=\Psi(\Psi(\xi))$.

4.2. **Normalization.** To effectively deal with the shortcomings of the relaxed update we can define a normalized relaxed update, following Groenen et al. [1999].

Suppose Ψ is any MDS iteration function. Instead of updating ξ by $\Psi(\xi)$, we update by $\tau\Psi(\xi)$, where the scalar τ is chosen to minimize stress. This defines the normalized algorithm

Algorithm $\mathscr{A}_3(NORM)$
$\eta^{(\nu)} = \Psi(\xi^{(\nu)}),$
$\tau^{(\nu)} = \dfrac{(\eta^{(\nu)})'\Psi(\eta^{(\nu)})}{(\eta^{(\nu)})'\eta^{(\nu)}},$
$\xi^{(\nu+1)} = \tau^{(\nu)}\eta^{(\nu)}.$

The normalizing scalar τ is equal to one at a local minimum $\hat{\xi}$. If the original algorithm is stable, then the normalized algorithm is stable, because $\sigma(\xi^{(\nu+1)}) \leq \sigma(\eta^{(\nu)}) \leq \sigma(\xi^{(\nu)})$. Also

$$\mathscr{D}\tau(\hat{\xi}) = -\frac{\hat{\xi}}{\hat{\xi}'\hat{\xi}},$$

and thus for the derivative of the normalized iteration function $\tilde{\Psi}$ at the local minimum $\hat{\xi}$ we find

$$\mathscr{D}\tilde{\Psi}(\hat{\xi}) = (I - \frac{\hat{\xi}\hat{\xi}'}{\hat{\xi}'\hat{\xi}})\mathscr{D}\Psi(\hat{\xi}).$$

Thus derivatives of iteration functions which have $\mathscr{D}\Psi(\xi)\xi = \xi$, such as the relaxed update, are deflated and the trivial unit eigenvalue is removed. All other eigenvalues remain the same.

The rate of convergence of the normalized relaxed update becomes

$$\lambda_3 = \max\{|2\lambda_1 - 1|, |2\lambda_{K-1} - 1|\}. \tag{12}$$

We have a rate $\lambda_3 = 2\lambda_1 - 1$ if $\lambda_1 > \frac{1}{2}$ and $\lambda_{K-1} > \frac{1}{2}$. We also have $\lambda_3 = 2\lambda_1 - 1$ if $\lambda_1 > \frac{1}{2}$ and $\lambda_{K-1} < \frac{1}{2}$ and $\lambda_1 + \lambda_{K-1} > 1$. In all other cases $\lambda_3 = 1 - 2\lambda_{K-1}$. For the example with $\lambda = 0.00(0.01)0.99$ the rate is 0.98, which is actually slightly better than the optimal rate for two-point convex combinations of the form $\alpha\Phi(\xi) + (1-\alpha)\xi$, which was 0.980198. Since the $d_m(\xi^{(\nu)})$ can be re-used in the next iteration to quickly compute the $d_m(\xi^{(\nu+1)})$ normalizing the relaxed update does not significantly increase the amount of work in an iteration.

4.3. **Stabilization.** There is an alternative method to normalize, which we call *stabilization*, following Varadhan and Roland [2008]. We set

Algorithm $\mathscr{A}_4(STABLE)$
$\eta^{(\nu)} = \Psi(\xi^{(\nu)}),$
$\xi^{(\nu+1)} = \Phi(\eta^{(\nu)}).$

If we apply stabilization to the relaxed update, we find $\sigma(\xi^{(\nu+1)}) \leq \sigma(\eta^{(\nu)}) \leq \sigma(\xi^{(\nu)})$. The convergence rate is

$$\rho(\mathscr{D}\Phi(\Psi(\xi)\mathscr{D}\Psi(\xi)) = \max_{k=1}^{m} |\lambda_k(2\lambda_k - 1)|.$$

On the unit interval the convex quadratic $\lambda(2\lambda - 1)$ has roots at zero and $\frac{1}{2}$, and a minimum at $\frac{1}{4}$, which minimum value equal to $-\frac{1}{8}$. Thus the maximum of $|\lambda_k(2\lambda_k - 1)|$ is attained at either the eigenvalue between zero and $\frac{1}{2}$ for which $\lambda(1-2\lambda)$ is maximized (which is the eigenvalue closest to $\frac{1}{4}$) or at λ_1. Assuming that $\lambda_1 > \frac{1}{4}(1+\sqrt{2}) \approx 0.6035534$ we have

$$\rho(\mathscr{D}\Phi(\Psi(\xi)\mathscr{D}\Psi(\xi)) = \lambda_1(2\lambda_1 - 1).$$

But for $\lambda_1 = 1-\varepsilon$ with ε small $\lambda_1(2\lambda_1 - 1) = \lambda_1^3 - \lambda_1\varepsilon^2 < \lambda_1^3$, and thus stabilized relaxed updates converge at least three times as fast as raw SMACOF. If $\lambda_1 = .99$ then $\lambda_1^3 = 0.970299$ and $\lambda_1(2\lambda_1 - 1) = 0.9702$.

4.4. **Three-point Relaxation.** The original idea behind relaxation can be generalized to three-point acceleration. Define the relaxed update

$$\Psi(\xi) = 3\Phi(\Phi(\xi)) - 3\Phi(\xi) + \xi,$$

and normalize this.

Algorithm $\mathscr{A}_5(RELAX3-N)$
$\chi^{(\nu)} = \Phi(\xi^{(\nu)}),$
$\zeta^{(\nu)} = \Phi(\chi^{(\nu)}),$
$\eta^{(\nu)} = 3\zeta^{(\nu)} - 3\chi^{(\nu)} + \xi^{(\nu)},$
$\tau^{(\nu)} = \dfrac{(\eta^{(\nu)})'\Psi(\eta^{(\nu)})}{(\eta^{(\nu)})'\eta^{(\nu)}},$
$\xi^{(\nu+1)} = \tau^{(\nu)}\eta^{(\nu)}.$

The convex quadratic $3\lambda^2-3\lambda+1$ is non-negative on the unit interval and is symmetric around its minimum at $\frac{1}{2}$. Thus the convergence rate is

$$\rho_5=\max_{k=1}^{K-1}|3\lambda_k^2-3\lambda_k+1|=\max(3\lambda_1^2-3\lambda_1+1,3\lambda_{K-1}^2-3\lambda_{K-1}+1).$$

Thus $\rho_5=3\lambda_1^2-3\lambda_1+1$ if $\lambda_1+\lambda_{K-1}>1$. If $\lambda_1=1-\varepsilon$, then $\rho_5=\lambda_1^3+\varepsilon^3$ and thus we expect the number of iterations to be divided by a factor of three. Note that the analysis depends critically on using the normalization, and on the fact that the eigenvalues of the SMACOF iteration function are between zero and one. Thus the method will not necessarily perform well with other fixed point iterations.

The stabilized three-point relaxation is

Algorithm $\mathscr{A}_6(RELAX3-S)$
$\chi^{(\nu)}=\Phi(\xi^{(\nu)}),$
$\zeta^{(\nu)}=\Phi(\chi^{(\nu)}),$
$\eta^{(\nu)}=3\zeta^{(\nu)}-3\chi^{(\nu)}+\xi^{(\nu)},$
$\xi^{(\nu+1)}=\Phi(\eta^{(\nu)}).$

The third degree polynomial $\lambda(3\lambda^2-3\lambda+1)$ is non-negative and increasing on the unit interval. Thus

$$\rho_6(\xi)=\lambda_1(\xi)(3\lambda_1^2(\xi)-3\lambda_1(\xi)+1),$$

and if $\lambda_1(\xi)=1-\varepsilon$, then $\rho_6(\xi)=\lambda_1^4(\xi)+\lambda_1(\xi)\varepsilon^2$. For $\lambda=.99$ we have $\lambda^4=0.960596$ and $\lambda(3\lambda^2-3\lambda+1)=0.960597$.

4.5. **Three-point Self-Scaling Iterations.** An iteration function Ψ is called *self-scaling* if $\Psi(\theta\xi)=\Psi(\xi)$ for all ξ and for all real θ, i.e. if the iteration function is homogenous of degree zero. This implies, by Euler's Theorem, that $\mathscr{D}\Psi(\xi)\xi=0$ for all ξ. The SMACOF iteration function Φ is self-scaling. Note that three-point algorithms are self-scaling if and only if $\alpha_2=0$. This is even true after normalization and stabilization, although both normalization and stabilization guarantee that $\mathscr{D}\Psi(\xi)\xi=0$,

So let us look at three-point self-scaling algorithms with $\alpha_2=0$. The optimal rate is

$$\lambda_5=\min_\alpha\max_{k=1}^K|\alpha\lambda_k^2+(1-\alpha)\lambda_k|.$$

Rate λ_5 corresponds with the iteration $\Psi(\xi) = \alpha\Phi(\Phi(\xi)) + (1-\alpha)\Phi(\xi)$, which is self-scaling because Φ is.

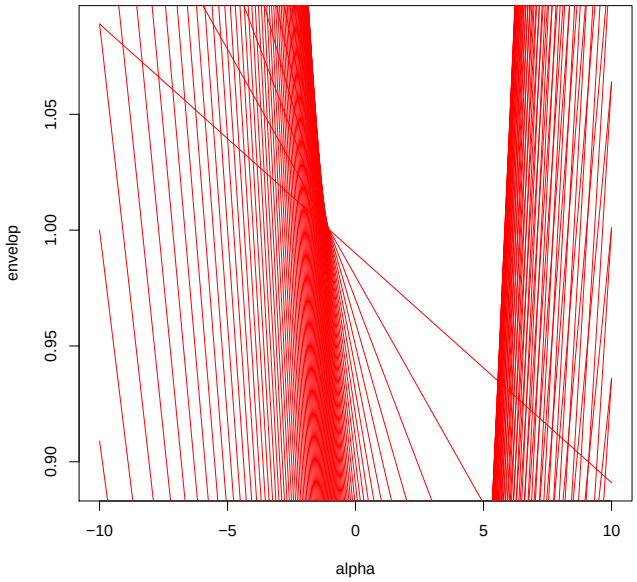

FIGURE 2. Computing λ_5

The 100 V-shaped functions $|\alpha\lambda_k^2 + (1-\alpha)\lambda_k|$ and their envelop are given in Figure 2. We find the minimum by intersecting the line with the steepest increase, which is $\alpha\lambda_1^2 + (1-\alpha)\lambda_1$, with one of the $n-1$ reflected parts $-(\alpha\lambda_k^2 + (1-\alpha)\lambda_k)$. We conclude that the minimum occurs for some k at

$$\hat{\alpha} = \min_{k>1} \frac{\lambda_1 + \lambda_k}{(\lambda_1 + \lambda_k) - (\lambda_1^2 + \lambda_k^2)}.$$

Using the $\hat{k}$ that yields the minimum shows that

$$\lambda_5 = \hat{\alpha}\lambda_1^2 + (1-\hat{\alpha})\lambda_1 = \frac{\lambda_1\lambda_{\hat{k}}(\lambda_1 - \lambda_{\hat{k}})}{(\lambda_1 + \lambda_{\hat{k}}) - (\lambda_1^2 + \lambda_{\hat{k}}^2)} \tag{13}$$

In our example the optimum $\hat{\alpha}$ is 5.559968, which gives a rate of $\lambda_4 = 0.9349563$. The accelerated algorithm is 6.7 times as fast as the basic SMACOF algorithm (but it uses twice as many computations for the update, so we expect it to be about three to four times as fast).

During the iterations we usually do not have information about the intermediate eigenvalues, but we can estimate what λ_1 is. Thus it is advantageous to have an expression that only involves λ_1, similar to what we had in the two-step case. Now

$$\hat{\alpha} = \min_{k>1} \frac{1}{1 - \frac{\lambda_1^2 + \lambda_k^2}{\lambda_1 + \lambda_k}} \geq \min_{0 \leq x \leq 1} \frac{1}{1 - \lambda_1 \frac{1+x^2}{1+x}} = \frac{1}{1 - \lambda_1 (2\sqrt{2} - 2)}.$$

Letting $\overline{\lambda} = \frac{1}{2}(1 + \sqrt{2}) \approx 1.207107$ we find

$$\hat{\alpha} \geq \frac{\overline{\lambda}}{\overline{\lambda} - \lambda_1}.$$

If we us this lower bound for $\hat{\alpha}$ we find the convergence rate

$$\lambda_6 = \lambda_1^2 \frac{\overline{\lambda} - 1}{\overline{\lambda} - \lambda_1} \geq \lambda_5. \tag{14}$$

As our examples show, λ_5 and λ_6 are generally close. In our example with 100 equally spaced eigenvalues the bound for the rate is $\lambda_6 = 0.9349563$, the same as the optimal rate λ_5.

In our implementation we replace λ_1 by the same estimate we used before. Thus the algorithm, which is generally non-monotone, becomes

Algorithm $\mathscr{A}_7(SCALING)$
$\eta^{(\nu)} = \Phi(\xi^{(\nu)}),$
$\zeta^{(\nu)} = \Phi(\eta^{(\nu)}),$
$\hat{\rho}^{(\nu)} = \frac{\|\zeta^{(\nu)} - \eta^{(\nu)}\|}{\|\eta^{(\nu)} - \xi^{(\nu)}\|},$
$\hat{\alpha}^{(\nu)} = \frac{\overline{\lambda}}{\overline{\lambda} - \hat{\rho}^{(\nu)}},$
$\xi^{(\nu+1)} = \hat{\alpha}^{(\nu)} \zeta^{(\nu)} + (1 - \hat{\alpha}^{(\nu)}) \eta^{(\nu)}.$

5. EXAMPLES

5.1. **Equal Dissimilarities.** If all dissimilarities are equal then optimal MDS solutions in two dimensions will be on one or more concentric circles [De Leeuw and Stoop, 1984]. In A.4.1 we give a small `R` program `eqdiss()` that computes normalized dissimilarities and a basis for order n.

For $n = 4$ the global minimum in two dimensions is four points at the corners of a square. Since it does not matter which point we put in which corner, the

minimum is not unique. The basis consists of 5 matrices C_m of order 6. Minimum stress is $\sigma(\xi_\star) = 0.0285955$. The four non-zero eigenvalues of $H(\xi_\star)$ are 0.840537, 0.585786, 0.585786, and 0.492796. The convergence rate of raw SMACOF is $\lambda_1(\xi_\star) = 0.840537$. Note that two of the eigenvalues are equal. The optimal Chebyshev rate for three-step algorithms is 0.349517, and for four-step algorithms it is 0.0773212. Five-step algorithms can attain quadratic convergence.

step	α_0	α_1	α_2	α_3	α_4	rate
2	-0.724937	1.724937	–	–	–	0.724937
3	0.349517	-3.428725	4.079207	–	–	0.349517
4	-0.077322	5.377768	-16.081755	11.781308	–	0.077322
5	0.000000	-7.242595	35.677494	-57.284050	29.849152	0.000000

TABLE 1. Chebyshev Algorithms

5.2. **Example with Perfect Fit.** For the dissimilarities we use the normalized distances of the two-dimensional configuration in Table 2.

	1	2
1	0.09	0.53
2	−1.17	−0.67
3	−0.57	−0.02
4	0.59	−1.58
5	−1.66	−0.88
6	−0.73	−0.47
7	0.84	−1.33
8	1.09	−0.24
9	0.12	0.28
10	−0.91	−0.73

TABLE 2. Random Configuration

Thus the global minimum of stress should be zero. We start with ξ equal to $1,2,\cdots,17$. All the relevant code is in the Appendix. Basic SMACOF uses 772 iterations and time 4.871 seconds. Some of this is overhead, of course. If we define a version of SMACOF which makes two SMACOF steps in each iteration before

recomputing stress and deciding to stop or not, then we need 415 of these double iterations and only 3.619 seconds of time. SMACOF with relaxation and normalization uses 410 iterations and time 3.429. Especially in this last case some time could be gained by more efficient programming. Three-point relaxed uses 288 iterations and time 3.062. The λ-algorithm uses 145 iterations with time 1.271, which is promising.

For this example we have computed eigenvalues of the algorithmic map at the solution, and thus the theoretical convergence rates. They are in Table 3. The three-point Chebyshev solution has $\hat{\alpha} = 7.780288$ and $\hat{\alpha} = -7.684987$ and $\hat{\gamma} = 0.904699$.

algorithm	formula	value
basic	λ_1	0.987749
two-step	λ_1^2	0.975648
three-step	λ_1^3	0.963695
three-point	λ_1	0.904684
optimal	λ_2	0.975795
relax/norm	λ_3	0.975498
relax(2)/norm	λ_4	0.963697
$\hat{\alpha}$	λ_5	0.921032
λ	λ_6	0.921159

TABLE 3. Theoretical Convergence Rates

The code in the Appendix also implements Newton's method. The iteration formula is $\Psi(\xi) = (I - H(\xi))^{-1}B(\xi)\xi$. With the same start for ξ we need only 26 iterations, and use only 0.284 seconds. Unfortunately convergence is to a saddle point, and not to the global minimum. If we try ten random permutations of ξ as starting points, then Newton's method uses anywhere from 13 to 128 iterations, with a running time of 0.142 to 1.363 seconds. It only finds the global minimum in three of the ten runs.

5.3. **Ekman Color Circle.** This is an example with 14 points in two dimensions [Ekman, 1954]. We apply the one-step, one-step-relaxed, two-step, and the λ algorithms. They all start at the same point and converge to the same solution, which

has largest eigenvalue $\lambda_1 = 0.964896$. Table 5 shows the computed rates (which are very close to the theoretical ones) and the time to convergence.

algorithm	empirical rate	iterations	time
basic	0.964898	519	6.876
two-step	0.931027	270	5.174
relaxed	0.929765	217	4.937
relaxed(two)	0.898387	187	4.428
λ	0.796165	90	1.730

TABLE 4. Ekman Example

As in the previous example, we see convergence rates indicating that the λ-algorithm is needs about one sixth of the iterations of basic SMACOF and, if we take the amount of work within iterations into account, is about four times faster. We can compute the Chebyshev weights at the optimum. This gives $\hat{\alpha} = 6.601501$ and $\hat{\alpha} = -6.369762$ and $\hat{\gamma} = 0.768261$, with a rate of $\lambda_1 = 0.768264$.

5.4. **Rothkopf Morse Code Data.** For the next example [Rothkopf, 1957] there are 36 morse-signals being compared, and thus 630 dissimilarities. A basis for the two-dimensional configurations has 67 elements. The confusion data are first transformed to dissimilarities using the Shepard-Luce formula

$$\delta_{ij} = -\log\sqrt{\frac{n_{ij}n_{ji}}{n_{ii}n_{jj}}}.$$

Programming efficiency really begins to matter in this case, because the array contaning the C_m has $67 \times 67 \times 630 = 2,828,070$ elements. We use basic SMACOF, the normalized relaxed update, and the λ-algorithm. We show the user time, the system time, and the elapsed time.

Again we see that basic SMACOF takes about six times as many iterations and about four times as much time as the λ-algorithm. The optimal Chebyshev rate in this example, by the way, is $\lambda_1 = 0.947381$. The Newton-Raphson method, without any safeguards, wildly fluctuates all over the place, and is pretty useless in this case.

algorithm	empirical rate	iterations	u-time	s-time	e-time
basic	0.993152	1214	347.662	238.964	588.013
relaxed	0.986304	655	268.575	172.530	441.900
relaxed(two)	0.979597	457	258.531	165.137	423.767
λ	0.954831	187	83.372	61.496	145.105

TABLE 5. Rothkopf Example

APPENDIX A. CODE

A.1. **MDS Code.**

```
dyn.load("one_up.so")

do_some_iters<-function(a,delta,x,upfunc,renorm=FALSE,stable=
    FALSE,itmax=1000,eps=1e-15,ips=1e-6,verbose=TRUE) {
        xold<-x; if (renorm) xold<-optnorm(a,delta,xold)
        fold<-stress(a,delta,xold); itel<-1; nro<-1; fro<-1
        repeat {
                xnew<-upfunc(a,delta,xold)$up
                if (renorm) xnew<-optnorm(a,delta,xnew)
                if (stable) xnew<-one_up(a,delta,xnew)$up
                fnew<-stress(a,delta,xnew)
                nrn<-norm(xold-xnew); frn<-fold-fnew
                if (verbose) cat(
                                "Iteration: ",formatC(itel,
                                    width=3, format="d"),
                                "Change: ", formatC(nrn,digits
                                    =8,width=12,format="f"),
                                "TRate: ", formatC(nrn/nro,
                                    digits=8,width=12,format="f"
                                    ),
                                "FRate: ", formatC(frn/fro,
                                    digits=8,width=12,format="f"
                                    ),
                                "O-Function: ",formatC(fold,
                                    digits=8,width=12,format="f"
                                    ),
```

```
                                    "N-Function: ",formatC(fnew,
                                        digits=8,width=12,format="f"
                                        ),
                                    "\n")
                    if ((itel == itmax) || (abs(fnew - fold) < eps)
                         || (nrn < ips)) break()
                    itel<-itel+1; fold<-fnew; xold<-xnew; nro<-nrn
                        ; fro<-frn
            }
return(the_solution(a,delta,xnew,itel))
}

make_x<-function (n,p) {
r<-(p*n)-(p*(p+1)/2); l<-1
x<-array(0,c(n,p,r))
for (i in 1:p)
            {
            qrq<-as.matrix(qr.Q(qr(outer(1:(n-i+1),0:(n-i),"^")))
                [,2:(n-i+1)])
            for (k in 1:(n-i))
                    {
                    x[1:(n-i+1),i,l]<-qrq[,k]
                    l<-l+1
                    }
            }
return(x/sqrt(n))
}

make_a_from_x<-function(x) {
n<-dim(x)[3]; m<-dim(x)[1]; mm<-m*(m-1)/2
c<-array(0,c(n,n,mm))
for (s in 1:n) for (t in 1:n)
            {
            prd<-x[,,s]%*%t(x[,,t]); k<-1
            for (i in 1:(m-1)) for (j in (i+1):m)
                    {
                    c[s,t,k]<-prd[i,i]+prd[j,j]-(prd[i,j]+prd[j,i])
                    k<-k+1
                    }
```

```
52          }
53  return(c)
54  }
55
56  make_random_a<-function(n,p) {
57  a<-array(0,c(p,p,n)); bk<-matrix(0,p,p)
58  for (k in 1:n) {
59          ak<-crossprod(matrix(rnorm(p*p),p,p))
60          a[,,k]<-ak; bk<-bk+ak
61          }
62  ev<-eigen(bk); ea<-ev$values; ek<-ev$vectors
63  ck<-matrix(0,p,p); eb<-sqrt(outer(ea,ea));  bk<-matrix(0,p,p)
64  for (k in 1:n) {
65          ak<-crossprod(ek,a[,,k]%*%ek)/eb
66          a[,,k]<-ak; bk<-bk+ak
67          }
68  return(a)
69  }
70
71  the_solution<-function(a,delta,x,itel) {
72          n<-dim(a)[3]; p<-dim(a)[1]
73          b<-matrix(0,p,p); h<-matrix(0,p,p); d<-rep(0,n)
74          for (k in 1:n) {
75                          ak<-a[,,k]; ax<-colSums(x*ak)
76                          dk<-sqrt(sum(x*colSums(x*ak))); d[k]
                                <-dk
77                          b<-b+(delta[k]/dk)*ak
78                          h<-h+(delta[k]/dk)*(ak-outer(ax,ax)/(dk
                                ^2))
79                          }
80      eb<-eigen(b,only.values=TRUE)$values
81      eh<-eigen(h,only.values=TRUE)$values
82      cp<-charPoly(c(1,eh))
83      cf<-torgerson(vecAsDist(d))
84          return(list(x=x,f=1+sum(x^2)-2*sum(delta*d),itel=itel,d
                =d,b=b,h=h,eb=eb,eh=eh,cp=cp,cf=cf))
85  }
86
87  one_up_r<-function(a,delta,x) {
```

```
88          n<-dim(a)[3]; p<-dim(a)[1]
89          b<-matrix(0,p,p)
90          for (k in 1:n) {
91                  ak<-a[,,k]
92                  dk<-sqrt(sum(x*colSums(x*ak)))
93                  b<-b+(delta[k]/dk)*ak
94          }
95  up<-colSums(x*b)
96  return(list(up=up))
97  }
98
99  one_up_c<-function(a,delta,x) {
100         n<-dim(a)[3]; p<-dim(a)[1]; b<-matrix(0,p,p)
101         res<-.C("oneUp",as.double(a),as.double(delta),as.double
                (x),as.double(b),as.integer(n),as.integer(p))
102         b<-matrix(res[[4]],p,p)
103         up<-colSums(x*b)
104         return(list(up=up))
105 }
106
107 one_up<-one_up_c
108
109 one_up_rel<-function(a,delta,x) {
110 y<-one_up(a,delta,x)$up
111 up<-2*y-x
112 return(list(up=up))
113 }
114
115 two_up_rel<-function(a,delta,x){
116 y<-one_up(a,delta,x)$up
117 z<-one_up(a,delta,y)$up
118 up<-3*z-3*y+x
119 return(list(up=up))
120 }
121
122 two_up_kp<-function(a,delta,x) {
123         y<-one_up(a,delta,x)$up
124         nold<-norm(y-x)
125         z<-one_up(a,delta,y)$up
```

```
126           nnew<-norm(z-y)
127       kappa<-nnew/nold
128           kup<-(1+sqrt(2))/2
129           alp<-kup/(kup-kappa)
130           up<-alp*z+(1-alp)*y
131           return(list(up=up))
132   }
133
134   epsilon<-function(a,delta,x){
135           y<-one_up(a,delta,x)$up
136           z<-one_up(a,delta,y)$up
137           r2<-z-y
138           r1<-y-x
139           al<-sum(r2^2)
140           bt<--2*sum(r2*r1)
141           gm<-sum(r1^2)
142           up<-(al*x+bt*y+gm*z)/(al+bt+gm)
143           return(list(up=up))
144   }
145
146   macleod<-function(a,delta,x){
147           y<-one_up(a,delta,x)$up
148           z<-one_up(a,delta,y)$up
149           r2<-z-y
150           r1<-y-x
151           kappa<-norm(r2)/norm(r1)
152           up<-z-((kappa^2)/(kappa-1))*r1
153           return(list(up=up))
154   }
155
156   nwt_up<-function(a,delta,x) {
157           n<-dim(a)[3]; p<-dim(a)[1]
158           b<-matrix(0,p,p); h<-matrix(0,p,p)
159           for (k in 1:n) {
160                   ak<-a[,,k]; ax<-colSums(x*ak)
161                   dk<-sqrt(sum(x*colSums(x*ak)))
162                   b<-b+(delta[k]/dk)*ak
163                   h<-h+(delta[k]/dk)*(ak-outer(ax,ax)/(dk^2))
164           }
```

```
y<-colSums(x*b); up<-solve(diag(p)-h,y)
return(list(up=up))
}


norm<-function (x) sqrt(sum(x^2))

stress_r<-function(a,delta,x){
        n<-dim(a)[3]; p<-dim(a)[1]
        xi<-0
        for (k in 1:n) {
                ak<-a[,,k]
                dk<-sqrt(sum(x*colSums(x*ak)))
                xi<-xi+delta[k]*dk
        }
return(1+sum(x^2)-2*xi)
}

stress_c<-function(a,delta,x){
n<-dim(a)[3]; p<-dim(a)[1]; s<-0.0
res<-.C("aLoss",as.double(a),as.double(delta),as.double(x),
    as.double(s),as.integer(n),as.integer(p))
return(res[[4]])
}

stress<-stress_c

optnorm_r<-function(a,delta,x) {
        xi<-0; n<-dim(a)[3]
        for (k in 1:n) {
                ak<-a[,,k]
                dk<-sqrt(sum(x*colSums(x*ak)))
                xi<-xi+delta[k]*dk
        }
return(xi*x/sum(x^2))
}

optnorm_c<-function(a,delta,x) {
n<-dim(a)[3]; p<-dim(a)[1]
```

```
res<-.C("aNorm",as.double(a),as.double(delta),as.double(x),
    as.integer(n),as.integer(p))
return(res[[3]])
}

optnorm<-optnorm_c

charPoly<-function(a) {
n<-length(a)
coef<-c(1,rep(0,n))
for (i in 1:n) {
        coef[2:(i+1)]<-coef[2:(i+1)]-a[i]*coef[1:i]
        }
coef<-coef/sum(coef)
return(rev(coef))
}

vecAsDist<-function(x) {
n<-(1+sqrt(1+8*length(x)))/2
e<-matrix(0,n,n); k<-0
for (i in 1:(n-1)) {
    l<-n-i; ll<-1:l
        e[i+ll,i]<-x[k+ll]
        k<-k+l
        }
return(as.dist(e))
}

torgerson<-function(diss,p=2) {
z<-eigen(-doubleCenter(as.matrix(diss)^2)/2,symmetric=TRUE)
v<-pmax(z$values,0)
return(z$vectors[,1:p]%*%diag(sqrt(v[1:p])))
}

doubleCenter<-function(x) {
n<-dim(x)[1]; m<-dim(x)[2]; s<-sum(x)/(n*m)
xr<-rowSums(x)/m; xc<-colSums(x)/n
return((x-outer(xr,xc,"+"))+s)
}
```

A.2. C Code.

```
#include <stdio.h>
#include <math.h>

void oneUp(double *a, double *delta, double *x, double *b, int
    *n, int *p)
{
int i, j, k, m, l; double dk;
for (k = 0; k < *n; k++) {
        dk = 0.0;
        for (i = 0; i < *p; i++) for (j = 0; j < *p; j++) {
                m = (k * (*p) * (*p)) + (i * (*p)) + j;
                dk += a[m] * x[i] * x[j];
                }
        dk = sqrt(dk);
        for (i = 0; i < *p; i++) for (j = 0; j < *p; j++) {
                l = (i * (*p)) + j;
                m = (k * (*p) * (*p)) + l;
                b[l] += (delta[k]/dk)*a[m];
                }
        }
}

void aLoss(double *a, double *delta, double *x, double *s, int
    *n, int *p)
{
int i, j, k, m, l; double dk, xi = 0.0, ssq = 0.0;
for (i =0; i < *p; i++) ssq += x[i]*x[i];
for (k = 0; k < *n; k++) {
        dk = 0.0;
        for (i = 0; i < *p; i++) for (j = 0; j < *p; j++) {
                m = (k * (*p) * (*p)) + (i * (*p)) + j;
                dk += a[m] * x[i] * x[j];
                }
        dk = sqrt(dk);
        xi += delta[k] * dk;
        }*s = 1 + ssq - 2 * xi;
}

```

```
void aNorm(double *a, double *delta, double *x, int *n, int *p)
{
int i, j, k, m, l; double dk, xi = 0.0, ssq = 0.0;
for (i =0; i < *p; i++) ssq += x[i]*x[i];
for (k = 0; k < *n; k++) {
        dk = 0.0;
        for (i = 0; i < *p; i++) for (j = 0; j < *p; j++) {
                m = (k * (*p) * (*p)) + (i * (*p)) + j;
                dk += a[m] * x[i] * x[j];
                }
        dk = sqrt(dk);
        xi += delta[k] * dk;
        }
for (i =0; i < *p; i++) x[i] *= xi/ssq;
}
```

A.3. **Chebyshev Code.**

```
chebR<-function(a,b,tol=1e-15,relerr=0.0) {
m<-nrow(a); n<-ncol(a); ndim<-n+3; mdim<-m+1
if (n > m) stop("number of equations exceeds number of unknowns
    ")
aa<-matrix(0,ndim,mdim); bb<-rep(0,mdim); xx<-rep(0,ndim)
aa[1:n,1:m]<-t(a); bb[1:m]<-b
rlist<-.Fortran("cheb",as.integer(m),as.integer(n),as.integer(m
    +1),as.integer(n+3),
        as.single(aa),bb=as.single(bb),as.single(tol),as.single
            (relerr),xx=as.single(xx),
        rank=as.integer(0),resmax=as.single(0.0),iter=
            as.integer(0),ocode=as.integer(0))
return(list(coefs=rlist$xx[1:n],resids=rlist$bb[1:m],rank=rlist
    $rank,iter=rlist$iter,ocode=rlist$ocode))
}

.First.lib <- function(lib, pkg) {
  library.dynam("cheb", pkg, lib)
}
```

A.4. **Examples.**

A.4.1. *Equal.*

```
source("rate.R")
eqDiss<-function(n) {
delta<-rep(1,n*(n-1)/2)
delta<<-delta/norm(delta)
x<<-1:((2*n)-3)
a<<-make_a_from_x(make_x(n,2))
}
```

A.4.2. *Perfect.*

```
source("rate.R")
`xm` <-
structure(c(0.0947966676671456, -1.17253486825229, -0
    .572723099521036,
0.589391643055378, -1.66102559283175, -0.72665553801336, 0
    .84260817782339,
1.09247852299655, 0.115975433324915, -0.906068717410187, 0
    .534867387934993,
-0.671339193612298, -0.0208153247614012, -1.58028224671476, -0
    .884705210987525,
-0.471815643394534, -1.32788885314117, -0.240439630416530, 0
    .284084940210558,
-0.728654644122993), .Dim = c(10L, 2L))
delta<-as.vector(dist(xm))
delta<-delta/norm(delta)
x<-1:17
a<-make_a_from_x(make_x(10,2))
```

A.4.3. *Ekman.*

```
source("rate.R")
library(smacof)
data(ekman)
delta<-as.vector(100-ekman)
delta<-delta/norm(delta)
x<-1:25
a<-make_a_from_x(make_x(14,2))
```

A.4.4. *Morse Code.*

```
source("rate.R")
library(smacof)
data(morse)
morse<--log((morse*t(morse))/outer(diag(morse),diag(morse)))
morse[which(morse==Inf)]<-9
delta<-as.vector(morse)
delta<-delta/norm(delta)
a<-make_a_from_x(make_x(36,2))
x<-1:67
```

REFERENCES

I. Barrodale and C. Philips. Algorithm 495 – Solutions of an Overdetermined System of Linear Equations in the Chebyshev Norm. *ACM Transactions on Mathematical Software*, 1:264–270, 1975.

J. De Leeuw. Convergence of the Majorization Method for Multidimensional Scaling. *Journal of Classification*, 5:163–180, 1988.

J. De Leeuw. Differentiability of Kruskal's Stress at a Local Minimum. *Psychometrika*, 49:111–113, 1984.

J. De Leeuw. Fitting Distances by Least Squares. Technical Report 130, Department of Statistics, UCLA, Los Angeles, California, 1993. URL `http://preprints.stat.ucla.edu/130/130.ps.gz`.

J. De Leeuw. Applications of Convex Analysis to Multidimensional Scaling. In J.R. Barra, F. Brodeau, G. Romier, and B. Van Cutsem, editors, *Recent developments in statistics*, pages 133–145, Amsterdam, The Netherlands, 1977. North Holland Publishing Company.

J. De Leeuw and W. J. Heiser. Multidimensional Scaling with Restrictions on the Configuration. In P.R. Krishnaiah, editor, *Multivariate Analysis, volume V*, pages 501–522, Amsterdam, The Netherlands, 1980. North Holland Publishing Company.

J. De Leeuw and I. Stoop. Upper Bounds for Kruskal's Stress. *Psychometrika*, 49: 391–402, 1984.

G. Ekman. Dimensions of Color Vision. *Journal of Psychology*, 38:467–474, 1954.

P.J.F. Groenen, W. Glunt, and T.L. Hayden. Fast Algorithms for Multidimensional Scaling: A Comparison of Majorization and Spectral Gradient Methods. August 1999.

J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York, N.Y., 1970.

A. M. Ostrowski. *Solution of Equations and Systems of Equations*. Academic Press, New York, N.Y., 1966.

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

R. Varadhan and C. Roland. Simple and Globally Convergent Methods for Accelerating the Convergence of any EM Algorithm. *Scandinavian Journal of Statistics*, 2008.

DEPARTMENT OF STATISTICS, UNIVERSITY OF CALIFORNIA, LOS ANGELES, CA 90095-1554

E-mail address, Jan de Leeuw: deleeuw@stat.ucla.edu

URL, Jan de Leeuw: http://gifi.stat.ucla.edu