An Alternating Least Squares Approach to Squared Distance Scaling

Jan de Leeuw, Patrick Groenen, and Raoul Pietersz

Version 011, November 10, 2016

Abstract

We reproduce the 1975 derivation of the alternating least squares algorithm for squared distance scaling, from an internal report that got lost in the folds of time. In addition, we present a derivation and a substantial speed improvement based on majorization.

Contents

Note: This is a working paper which will be expanded/updated frequently. All suggestions for improvement are welcome. The directory deleeuwpdx.net/pubfolders/lost has a pdf version, the complete Rmd file with all code chunks, the bib file, and the R source code.

1 Introduction

Volume 73 of the *Journal of Statistical Software* is a festschrift for Jan de Leeuw (me). It contains a very nice article by Yoshio Takane, discussing our early interactions and some of my "lost papers" (Takane (2016)). One of these lost papers is De Leeuw (1975), which discusses the ELEGANT algorithm, an augmentation method for squared distance scaling. The algorithm has been discussed in the past by Takane (1977), Takane (1980), Browne (1987), De Leeuw, Groenen, and Pietersz (2004), and now again by Takane (2016), but with different derivations than those in the original paper. In this paper we reconstruct the original derivation, slighty generalized to include weights. Note that De Leeuw, Groenen, and Pietersz (2004) also use weights, and also describe a close approximation of the original algorithm.

Both Takane (1977) and Browne (1987) find the ELEGANT algorithm to be prohibitively slow to converge. After discussing the original algorithm and analyzing its convergence rate we use quadratic majorization to improve the convergence speed. The fact that the original ELEGANT algorithm can also be derived by using quadratic majorization was first observed by De Leeuw, Groenen, and Pietersz (2004), using a different derivation.

2 The ELEGANT Algorithm

The paper De Leeuw (1975) deals with the problem of minimizing the *sstress* loss function, defined on the space of $n \times p$ *configurations* as

$$\sigma(X) := \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n w_{ij}(\epsilon_{ij} - e_{ij}(X))^2. \tag{1}$$

Here the w_{ij} are given non-negative *weights* and the ϵ_{ij} are given non-negative *dissimilarities*. Both weights and dissimilarties are *symmetric* and *hollow*, i.e. have a zero diagonal. The $e_{ij}(X)$ are *squared Euclidean distances*, defined by

$$e_{ij}(X) := (x_i - x_j)'(x_i - x_j) = (e_i - e_j)'XX'(e_i - e_j) = \mathbf{tr}\ X'A_{ij}X,$$

where $A_{ij} := (e_i - e_j)(e_i - e_j)'$ and the e_i are unit vectors with a 1 in position i and 0 elsewhere. I am pretty sure the original formulation of ELEGANT did not include weights, but we include them here for the reasons simlar to those reviewed by Groenen and Van de Velden (2016), section 6. The alternative derivations by Takane (1977) and Browne (1987) did not consider weights either, but De Leeuw, Groenen, and Pietersz (2004) did include weights.

At that time we were trying to generalize the *alternating least squares* algorithms used for fitting non-metric linear and bilinear models to multidimensional scaling, i.e. to the least squares fitting of distances and squared distances. One way to attack this problem is the approach chosen in ALSCAL (Takane, Young, and De Leeuw (1977)), where the problem of fitting the optimal configuration is attacked by block coordinate descent. The approach in De Leeuw (1975) is quite different, because it does not partition the variables into blocks, but instead uses *augmentation* (De Leeuw (1994)). Note that augmentation was applied to multidimensional scaling shortly before *majorization* became the preferred approach. In this context majorization methods are another way to extend alternating least squares. They were first announced in the proceedings of the 1975 *US-Japan Seminar on Multidimensional Scaling and Related Techniques*.

To explain the original formulation we introduce, for each quadruple (i, j, k, l),

$$e_{ijkl} := (x_i - x_j)'(x_k - x_\ell) = (e_i - e_j)'XX'(e_k - e_\ell) = \mathbf{tr}\ XX'(e_i - e_j)(e_k - e_\ell)'.$$

Also define

$$\epsilon_{ijk\ell} := \begin{cases}\epsilon_{ij} & \text{if } (i,j)=(k,\ell),\\ \text{arbitrary} & \text{otherwise,}\end{cases} \tag{2}$$

and $w_{ijk\ell} := \sqrt{w_{ij}w_{kl}}$.

We now define an augmented version of sstress, which is a function of both the configuration X and the *disparities* $\epsilon_{ijk\ell}$.

$$\sigma_\star(X,E) := \frac12\sum_{i=1}^n\sum_{j=1}^n\sum_{k=1}^n\sum_{\ell=1}^n w_{ijk\ell}(\epsilon_{ijk\ell}-e_{ijk\ell}(X))^2, \tag{3}$$

The relation with sstress from (1) is

$$\sigma(X) = \min_{E\in\mathcal{E}}\sigma_\star(X,E),$$

where $\mathcal{E}$ is the set of disparities satisfying (2). The ELEGANT algorithm in De Leeuw (1975) uses alternating least squares to minimize $\sigma_\star$. Thus we alternate minimization over X keeping E fixed with minimization over $E\in\mathcal{E}$ keeping X fixed.

Minimizing $\sigma_\star$ over $E\in\mathcal{E}$ for fixed X is trivial. We simply set

$$\epsilon_{ijk\ell} := \begin{cases}\epsilon_{ij} & \text{if } (i,j)=(k,\ell),\\ e_{ijk\ell}(X) & \text{otherwise.}\end{cases} \tag{4}$$

The more complicated and interesting problem is to minimize $\sigma_\star$ over X for fixed E. Observe first

$$\sum_{i=1}^n\sum_{j=1}^n\sum_{k=1}^n\sum_{\ell=1}^n w_{ijk\ell}\epsilon_{ijk\ell}e_{ijk\ell} = \sum_{i=1}^n\sum_{j=1}^n\sum_{k=1}^n\sum_{\ell=1}^n w_{ijk\ell}\epsilon_{ijk\ell}(e_i-e_j)'XX'(e_k-e_\ell) = \mathbf{tr}\ X'B(X)X,$$

with

$$B(X) := \sum_{i=1}^n\sum_{j=1}^n\sum_{k=1}^n\sum_{\ell=1}^n w_{ijk\ell}\epsilon_{ijk\ell}(e_i-e_j)(e_k-e_\ell)'. \tag{5}$$

Next

$$\sum_{i=1}^n\sum_{j=1}^n\sum_{k=1}^n\sum_{\ell=1}^n w_{ijk\ell}e_{ijk\ell}^2 = \sum_{i=1}^n\sum_{j=1}^n\sum_{k=1}^n\sum_{\ell=1}^n w_{ij}^{\frac12}w_{k\ell}^{\frac12}(e_i-e_j)'XX'(e_k-e_\ell)(e_k-e_\ell)'XX'(e_i-e_j) = \mathbf{tr}\ (X'VX)^2,$$

with

$$V := \sum_{i=1}^n\sum_{j=1}^n w_{ij}^{\frac12}A_{ij}. \tag{6}$$

It follows that the X minimizing $\sigma_\star$ for fixed E must satisfy the stationary equations $B(X)X = VX(X'VX)$. If Z are the normalized eigenvectors corresponding with the p largest eigenvalues of the generalized eigen problem $B(X)Z = VZ\Lambda$, with $Z'VZ = I$, then $X = Z\Lambda^{\frac{1}{2}}$.

Working with four-dimensional arrays, and computing $e_{ijkl}(X)$ for all quadruples, is unattractive. It can be avoided, however. Write (4) in the form

$$\epsilon_{ijk\ell} = \delta^{ik}\delta^{j\ell}\epsilon_{ij} + (1-\delta^{ik}\delta^{j\ell})e_{ijk\ell}(X) = \delta^{ik}\delta^{j\ell}(\epsilon_{ij} - e_{ij}) + e_{ijkl}(X)$$

and substitute this into (5). After some simplification we find the value of $B(X)$ at this ϵ_{ijkl} to be

$$B(X) = R(X) + VXX'V, \tag{7}$$

with

$$R(X) := \sum_{i=1}^n\sum_{j=1}^n w_{ij}(\epsilon_{ij} - e_{ij}(X))A_{ij}. \tag{8}$$

Note that (6) says that $v_{ij} = -2\sqrt{w_{ij}}$ for $i \neq j$, and the diagonal is filled in such that rows and columns add up to zero. In the same way (8) says $r_{ij}(X) = -w_{ij}(\epsilon_{ij} - e_{ij}(X))$ for $i \neq j$, and the diagonal elements again make rows and columns sum to zero.

The algorithm is now simply to compute the update $X^{(k+1)}$ by solving the generalized eigen problem defined by $B(X^k)$ and V. Note that the fixed point equations for this algorithm $(R(X) + VXX'V)X = VX(X'VX)$ are equivalent to $R(X)X = 0$, which are the stationary equations found by setting the derivatives of σ equal to zero. It is easy to understand why, in my youthful enthusiasm, I baptized the algorithm ELEGANT. Note that if all off-diagonal weights are equal to one then $VX = 2nX$, and thus $B(X) = R(X) + 4n^2XX'$ and we must compute eigenvalues and eigenvectors of $XX' + \frac{1}{4n^2}R(X)$.

3 Ekman example

```r
data(ekman, package="smacof")
lbs<-attr(ekman, "Labels")
ekman <- (1 - as.matrix (ekman)) ^ 2
ha <- elegant(ekman, verbose = FALSE, itmax = 5000)
```

The algorithm takes 3498 iterations to come up with sstress 3.3187849896 and the following solution.

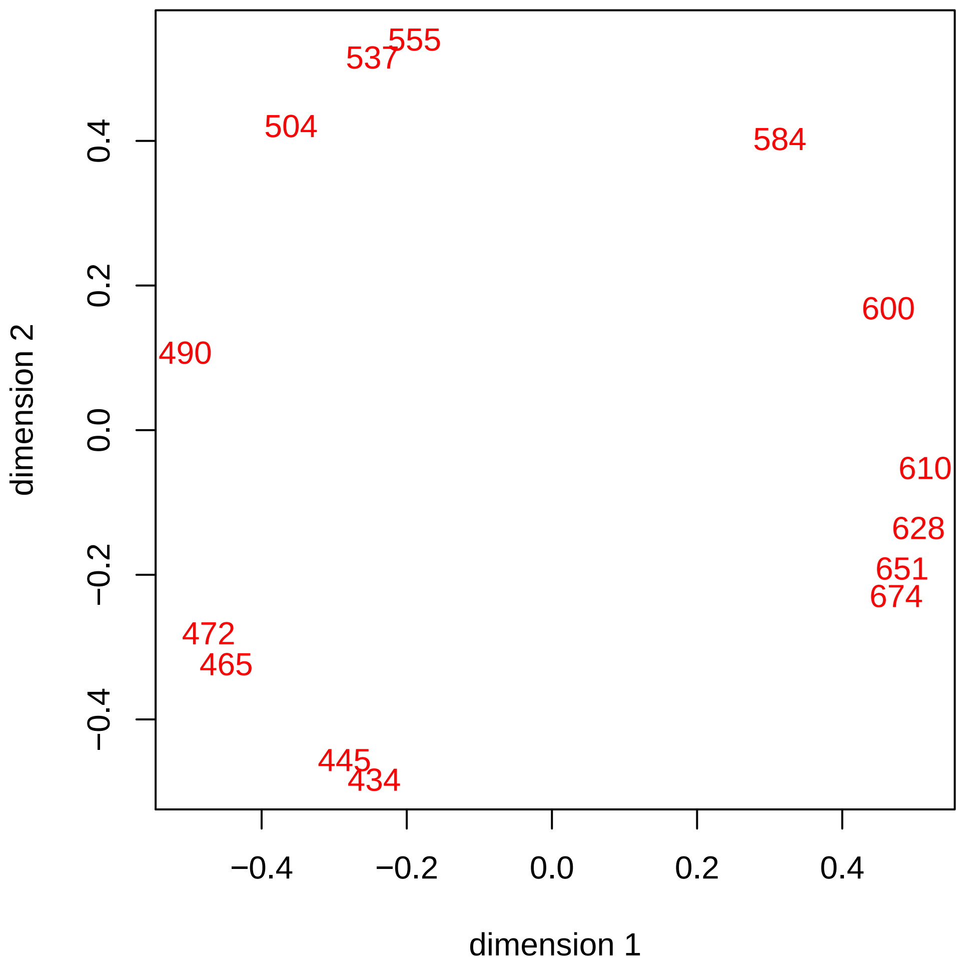

Figure 1: Ekman Unweighted

Now consider using the weights $w_{ij}=(2\epsilon_{ij}^{\frac12})^{-1}$. We have the approximation

$$\sigma(X)=\frac12\sum_{i=1}^n\sum_{j=1}^n\frac{1}{2\epsilon_{ij}^{\frac12}}(\epsilon_{ij}-e_{ij})^2\approx\frac12\sum_{i=1}^n\sum_{j=1}^n\frac{1}{\epsilon_{ij}^{\frac12}+e_{ij}^{\frac12}}(\epsilon_{ij}-e_{ij})^2=\frac12\sum_{i=1}^n\sum_{j=1}^n(\epsilon_{ij}^{\frac12}-e_{ij}^{\frac12})^2$$

Thus using these weights approximates an unweighted least squares loss function defined on the distances, not the squared distances.

```
n <- nrow (ekman)
w <- ifelse (ekman == 0, 0, 1 / (2 * sqrt (ekman)))
hb <- elegant (ekman, w = w, verbose = FALSE, itmax = 5000)
```

After 306 iterations we find the following solution.

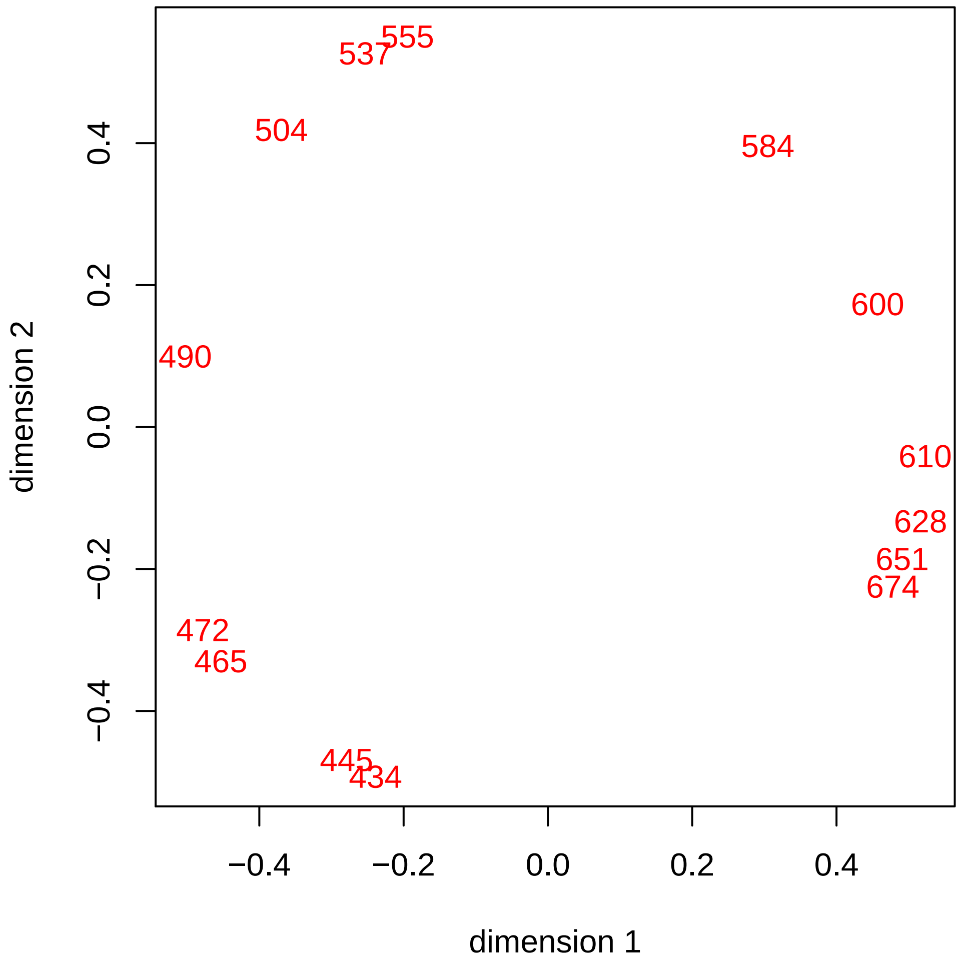

Figure 2: Ekman Weighted

4 ELEGANT Majorization

The ELEGANT algorithm uses *augmentation*. Suppose we want to minimize a function $f : X \rightarrow \mathbb{R}$, and the problem is too difficult or cumbersome to be attacked directly. It helps if we can find an *augmented function* $g : X \otimes Y \rightarrow \mathbb{R}$ such that $f(x) = \min_{y\in Y} g(x, y)$ for all $x \in X$ and the subproblems of minimizing g over $x \in X$ for fixed $y \in Y$ and minimizing over $y \in Y$ for fixed $x \in X$ are both easy. We can then alternate the two subproblems to update iterate $(x^{(k)}, y^{(k)})$ by

$$
\begin{aligned}
y^{(k+1)} &= \mathop{\textbf{argmin}}_{y\in Y} g(x^{(k)}, y),\\
x^{(k+1)} &= \mathop{\textbf{argmin}}_{x\in X} g(x, y^{(k+1)}),
\end{aligned}
$$

and under weak conditions accumulation points of the sequence $x^{(k)}$ will be stationary points of f (see De Leeuw (1994)). In ELEGANT the *augmentation variables* y are the off-diagonal elements of the four-dimensional array $\mathcal{E} = \{\epsilon_{ijk\ell}\}$.

Majorization algorithms (nowadays often called the *MM algorithms*, see Lange (2016)) are augmentation algorithms that use a *majorization function*. Again $f : X \rightarrow \mathbb{R}$, but now $g : X \otimes X \rightarrow \mathbb{R}$ satisfies $g(x, y) \geq f(x)$ for all $x, y \in X$ and $g(x, x) = f(x)$ for all $x \in X$. This implies $f(x) = \min_{y \in X} g(x, y)$ and $x^{(k)} = \mathbf{argmin}_{y \in Y}\, g(x^{(k)}, y)$. Thus the majorization algorithm is simply

$$x^{(k+1)} = \mathop{\mathbf{argmin}}_{x \in X} g(x, x^{(k)}).$$

One particular form of majorization we will use in this paper is the quadratic approximation method of Vosz and Eckhardt (1980) or Böhning and Lindsay (1988), also known as the quadratic upper bound method (Lange (2016), section 4.6). If f is twice differentiable and $\mathcal{D}^2 f(x) \lesssim B$ with B positive definite for all $x \in X$ then

$$g(x, y) = f(y) + (x - y)'\mathcal{D}f(y) + \frac{1}{2}(x - y)'B(x - y)$$

is a majorization function. The majorization algorithm is

$$x^{(k+1)} = x^{(k)} - B^{-1}\mathcal{D}f(x^{(k)}).$$

In order to get a quadratic upper bound for sstress we prove a useful inequality.

Lemma 1: [Inequality]

$$\sum_{i=1}^n \sum_{j=1}^n w_{ij}(A_{ij} \otimes A_{ij}) \lesssim \left(\sum_{i=1}^n \sum_{j=1}^n w_{ij}^{\frac{1}{2}} A_{ij}\right) \otimes \left(\sum_{i=1}^n \sum_{j=1}^n w_{ij}^{\frac{1}{2}} A_{ij}\right) = V \otimes V.$$

Proof: Suppose X and Y are an arbitrary $n \times p$ matrices, and define $C = XY'$ and $c = \mathbf{vec}(C)$. Also let $e_{ijk\ell}(X, Y) = (x_i - x_j)'(y_k - y_\ell)$ and $e_{ij}(X, Y) = (x_i - x_j)'(y_i - y_j)$. Then

$$\sum_{i=1}^n \sum_{j=1}^n w_{ij} e_{ij}^2(X, Y) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}\ \mathbf{tr}\ A_{ij}C'A_{ij}C = \sum_{i=1}^n \sum_{j=1}^n w_{ij}\ c'(A_{ij} \otimes A_{ij})c, \tag{9}$$

and

$$\sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \sum_{\ell=1}^n w_{ijkl} e_{ijkl}^2(X, Y) = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n \sum_{\ell=1}^n w_{ij}^{\frac{1}{2}} w_{kl}^{\frac{1}{2}} \mathbf{tr}\ A_{k\ell}C'A_{ij}C = \mathbf{tr}\ VC'VC = c'(V \otimes V)c. \tag{10}$$

Thus

$$c'\left\{\sum_{i=1}^n \sum_{j=1}^n w_{ij}\ (A_{ij} \otimes A_{ij})\right\} c \leq c'(V \otimes V)c$$

for all C of the form XY', which means for all C. **QED**

The inequality in lemma 1 is now used to obtain a quadratic majorization of sstress. Think of sstress as a function of $c = \mathbf{vec}(C)$, with $C = XX'$. Then

$$\mathcal{D}^2\sigma(c) = \sum_{i=1}^n \sum_{j=1}^n w_{ij}(A_{ij} \otimes A_{ij}), \tag{11}$$

and thus for all pairs of square symmetric C and $\tilde{C}$

$$\sigma(C) \leq \sigma(\tilde{C}) - \mathbf{tr}\ R(\tilde{C})(C - \tilde{C}) + \frac{1}{2}\mathbf{tr}\ V(C - \tilde{C})V(C - \tilde{C}).$$

Differentiating the majorization function with repect to X and setting the derivative equal to zero gives

$$(R(\tilde{C}) + V\tilde{C}V)X = VX(X'VX),$$

which is the update formula for the ELEGANT algorithm.

5 Beyond ELEGANT

In the unweighted case we replace the current configuration X with the configuration giving the best rank p approximation of $XX' + \frac{1}{4n^2}R(X)$. This leads to slow convergence, because the inertia part XX' is weighted much more heavily than the change part $R(X)$. Both De Leeuw (1975) and Browne (1987) suggest using a step size procedure than gives a larger weight to the change part. If the larger step size decreases the loss we go to the next iteration, if it does not decrease the loss we take a smaller step size and try again. This is somewhat ad hoc, but it works.

The majorization derivation of ELEGANT, however, gives us a more rigorous way to speed up the algorithm. The expression (11) for the second derivatives of stress makes it possible to easily derive bounds of the form $\mathcal{D}^2\sigma(C) \lesssim \beta I$ for some scalar β, where $\lesssim$ is used for the Loewner order of square symmetric matrices, with $A \lesssim B$ if $B - A$ is positive semidefinite. The algorithm then computes best rank p approximations to $XX' + \frac{1}{\beta}R(X)$. For ELEGANT with unit weights $\beta = 4n^2$.

For β we can make two choices. The first is the largest eigenvalue of the Hessian, the matrix on the right hand side of (11).

Lemma 2: [**Hessian**] If $w_{ij} = 1$ for all $i \neq j$ then the largest eigenvalue of the Hessian is $4n$.

Proof: Define the matrix S with elements $s_{ij,k\ell} = \mathbf{tr}\ A_{ij}A_{k\ell}$. S has the same eigenvalues as the Hessian from (11). All elements of S are non-negative, each row has $4n - 8$ elements equal to $+1$ and two elements equal to $+4$. The other elements are zero. Thus the largest (Frobenius) eigenvalue, corresponding to an eigenvector with all elements equal to $+1$, is $4n$. **QED**

Thus in the case of equal unit weights we can use $\beta = 4n$ instead of $4n^2$, which suggests the majorization algorithm based on the largest eigenvalue will be about n times as fast. It does require initially solving the eigenvalue problem for a large matrix of order n^2, however. If we choose β as the trace of the Hessian we find $\beta = 4\sum_{i=1}^n\sum_{j=1}^n w_{ij}$, which becomes $4n(n-1)$ in the case of unit weights. This (almost) coincides with the original ELEGANT algorithm for the unweighted case. We have programmed the majorization methods with a scalar β, and the resulting simplications, in a separate R function `beyond`.

We'll try these alternative majorization methods on the Ekman data. First with the eigenvalue bound.

```
data(ekman, package="smacof")
ekman <- (1 - as.matrix (ekman)) ^ 2
hc <- beyond(ekman, bound = "eval", verbose = FALSE, itmax = 5000)
```

The beyond algorithm now takes 298 iterations (instead of 3498) to come up with sstress 3.3187849627, the same solution as ELEGANT. About 12 times faster in iteration count. Individual iterations are also faster, but there is some overhead because of the large initial eigenvalue problem.

```
data(ekman, package="smacof")
ekman <- (1 - as.matrix (ekman)) ^ 2
hd <- beyond(ekman, bound = "trace", verbose = FALSE, itmax = 5000)
```

For the trace bound we need 3268 iterations to arrive at sstress sstress 3.3187849875, again the same solution as ELEGANT. There is no overhead to compute the eigenvalue bound, but the iteration gain is minimal (probably reflecting the difference between n^2 and $n(n-1)$).

Next we use `microbenchmark` (Mersmann (2015)) to compare computing times of ELEGANT and the two quadratic majorization bounds.

```
## Unit: milliseconds
##                                                            expr        min
##                  elegant(ekman, verbose = FALSE, itmax = 5000) 442.435321
##   beyond(ekman, verbose = FALSE, bound = "eval", itmax = 5000) 191.455628
##  beyond(ekman, verbose = FALSE, bound = "trace", itmax = 5000) 397.601953
##          lq        mean     median          uq        max neval cld
##  465.593165 475.5283196 471.488237 481.8184840 582.078795   100   c
##  208.286437 243.4682953 211.918767 308.3162115 326.199085   100 a
##  419.952887 437.7884735 426.484254 438.1270405 546.422315   100  b
```

We see a speedup factor of about 2.2 if we use the eigenvalue bound, which is 56, and about 10 percent gain over ELEGANT if we use the trace bound, which is $4 \times 13 \times 14 = 728$. The speed gain is considerably less than the gain in iteration count. For larger problems we expect a larger speed gain. Note our algorithms use the `slanczos` function from the `mgcv` package (Wood (2016)) to compute one or a few dominant eigenvalues.

6 Appendix: Code

```r
suppressPackageStartupMessages(library(mgcv, quietly = TRUE))

e <- function (i, n)
  ifelse (i == 1:n, 1, 0)

a <- function (i, j, n)
  outer (e(i, n) - e(j, n), e(i, n) - e(j, n))

torgerson <- function (delta, p = 2) {
  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)
  }
  z <- slanczos(-doubleCenter(delta / 2), p)
  v <- pmax(z$values, 0)
  return(z$vectors %*% diag(sqrt(v)))
}

mpower <- function (x, p, eps = 1e-16) {
  z <- eigen (x, symmetric = TRUE)
  zval <- z$values[1:(nrow(x) - 1)] ^ p
  zvec <- z$vectors[, 1:(nrow(x) - 1)]
  return (zvec %*% diag (zval) %*% t (zvec))
}

elegant <-
  function (delta,
            w = 1 - diag (nrow (delta)),
            p = 2,
            xold = torgerson (delta, p),
            itmax = 1000,
            eps = 1e-10,
            debug = FALSE,
            verbose = TRUE) {
    itel <- 1
    v <- -2 * sqrt (w)
    diag (v) <- -rowSums (v)
    vinv <- mpower (v,-1 / 2)
    dold <- as.matrix (dist (xold)) ^ 2
    sold <- sum (w * (delta - dold) ^ 2)
    repeat {
      b <- -w * (delta - dold)
      diag (b) <- -rowSums (b)
      h <- tcrossprod (v %*% xold)
      s <- b + h
      z <- slanczos (vinv %*% s %*% vinv, p)
      u <- vinv %*% z$vectors
      evz <- pmax(z$values, 0)
      xnew <- u %*% diag(sqrt(evz))
      dnew <- as.matrix (dist (xnew)) ^ 2
      snew <- sum (w * (delta - dnew) ^ 2)
      if (debug)
        browser ()
      if (verbose) {
        cat (
          formatC (itel, width = 4, format = "d"),
          formatC (
            sold,
            digits = 10,
            width = 15,
            format = "f"
          ),
          formatC (
            snew,
            digits = 10,
            width = 15,
            format = "f"
          ),
          "\n"
        )
      }
      if ((itel == itmax) || (sold - snew) < eps)
        break
      itel <- itel + 1
      xold <- xnew
      dold <- dnew
      sold <- snew
    }
    return (list (
      x = xnew,
      d = dnew,
      itel = itel,
        s = snew
      ))
    }

beyond <-
  function (delta,
            w = 1 - diag (nrow (delta)),
            p = 2,
            xold = torgerson (delta, p),
            bound = "eval",
            itmax = 1000,
            eps = 1e-10,
            debug = FALSE,
            verbose = TRUE) {
    n <- nrow (delta)
    itel <- 1
    vv <- matrix (0, n ^ 2, n ^ 2)
    if (bound == "eval") {
      for (i in 1:n)
        for (j in 1:n)
          vv <- vv + w[i, j] * kronecker (a (i, j, n), a(i, j, n))
        lbd <- slanczos(vv, 1)$values
    }
    if (bound == "trace")
      lbd <- 4 * sum (w)
    dold <- as.matrix (dist (xold)) ^ 2
    sold <- sum (w * (delta - dold) ^ 2)
    repeat {
      b <- -w * (delta - dold)
      diag (b) <- -rowSums (b)
      h <- tcrossprod (xold)
      s <- b / lbd + h
      z <- slanczos (s, p)
      u <- z$vectors
      evz <- pmax(z$values, 0)
      xnew <- u %*% diag(sqrt(evz))
      dnew <- as.matrix (dist (xnew)) ^ 2
      snew <- sum (w * (delta - dnew) ^ 2)
      if (debug)
        browser ()
      if (verbose) {
        cat (
          formatC (itel, width = 4, format = "d"),
          formatC (
            sold,
            digits = 10,
            width = 15,
            format = "f"
          ),
          formatC (
            snew,
            digits = 10,
            width = 15,
            format = "f"
          ),
          "\n"
        )
      }
      if ((itel == itmax) || (sold - snew) < eps)
        break
      itel <- itel + 1
      xold <- xnew
      dold <- dnew
      sold <- snew
    }
    return (list (
      x = xnew,
      d = dnew,
      itel = itel,
      bound = lbd,
      s = snew
    ))
  }
```

References

Böhning, D., and B.G. Lindsay. 1988. "Monotonicity of Quadratic-approximation Algorithms." *Annals of the Institute of Statistical Mathematics* 40 (4): 641–63.

Browne, M.W. 1987. "The Young-Householder Algorithm and the Least Squares Multdimensional Scaling of Squared Distances." *Journal of Classification* 4: 175–90.

De Leeuw, J. 1975. "An Alternating Least Squares Approach to Squared Distance Scaling." Department of Data Theory FSW/RUL.

———. 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.

http://deleeuwpdx.net/janspubs/1994/chapters/deleeuw_C_94c.pdf.

De Leeuw, J., P.J.F. Groenen, and R. Pietersz. 2004. "Augmentation and Majorization Algorithms for Squared Distance Scaling." http://deleeuwpdx.net/janspubs/2004/notes/deleeuw_groenen_pietersz_U_04.pdf.

Groenen, P.J.F., and M. Van de Velden. 2016. "Multidimensional Scaling by Majorization: A Review." *Journal of Statistical Software* 73 (8): 1–26. doi:10.18637/jss.v073.i08.

Lange, K. 2016. *MM Optimization Algorithms*. SIAM.

Mersmann, O. 2015. *Microbenchmark: Accureate Timing Functions*. https://CRAN.R-project.org/package=microbenchmark.

Takane, Y. 1977. "On the Relations among Four Methods of Multidimensional Scaling." *Behaviormetrika* 4: 29–42.

———. 1980. *Multidimensional Scaling (in Japanese)*. University of Tokyo Press.

———. 2016. "My Early Interactions with Jan and Some of His Lost Papers." *Journal of Statistical Software* 73 (7): 1–14. doi:10.18637/jss.v073.i07.

Takane, Y., F.W. Young, and J. De Leeuw. 1977. "Nonmetric Individual Differences in Multidimensional Scaling: An Alternating Least Squares Method with Optimal Scaling Features." *Psychometrika* 42: 7–67. http://deleeuwpdx.net/janspubs/1977/articles/takane_young_deleeuw_A_77.pdf.

Vosz, H., and U. Eckhardt. 1980. "Linear Convergence of Generalized Weiszfeld's Method." *Computing* 25: 243–51.

Wood, S.N. 2016. *Mixed GAM Computation Vehicle with GCV/AIC/REML Smoothness Estimation*. https://CRAN.R-project.org/package=mgcv.