Multidimensional Scaling with Upper Bounds

Jan de Leeuw

Version 22, January 13, 2017

Abstract

We give an algorithm, with R code, to minimize the multidimensional scaling stress loss function under the condition that some or all of the fitted distances are smaller than given upper bounds.

Contents

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

1 Introduction

In this paper we study the minimization of *stress*, defined as

$$\textbf{stress}(X) := \frac{1}{4}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(\delta_{ij} - d_{ij}(X))^2$$

over $n \times p$ *configuration matrices* X. Here $W = \{w_{ij}\}$ and $\Delta = \{\delta_{ij}\}$ are given symmetric, hollow, and non-negative matricesof, respectively, *weights* and *dissimilarities*. The $d_{ij}(X)$ are Euclidean distances between the points with coordinates in the rows of X. Thus

$$d_{ij}^2(X) := (x_i - x_j)'(x_i - x_j) = (e_i - e_j)'XX'(e_i - e_j) = \textbf{tr}\ X'A_{ij}X.$$

Here e_i and e_j are unit vectors, with all elements equal to zero except for the single element i or j, which is equal to one. Also $A_{ij} := (e_i - e_j)(e_i - e_j)'$.

The problem of minimizing stress is well-studied (see, for example, Borg and Groenen (2005)) . In this paper we analyze the problem of minimizing stress over all configurations X for which $d_{ij}(X) \leq \alpha_{ij}$ for some or all pairs i, j, where the α_{ij} are positive bounds. As a special case we have *MDS from below*, where we require $d_{ij}(X) \leq \delta_{ij}$ for all i, j.

Note that constraints of this type are not exactly independent. If we require $d_{ij}(X) \leq \alpha_{ij}$ and $d_{ik}(X) \leq \alpha_{ik}$ then, by the triangle inequality, we automatically require $d_{jk}(X) \leq \alpha_{ij} + \alpha_{ik}$. On the other hand the constraints are always consistent, because they will be satisfied if we take X small enough.

The constraints $d_{ij}(X) \leq \alpha_{ij}$, or equivalently $d_{ij}^2(X) \leq \alpha_{ij}^2$, define a convex set in configuration space, but stress is not convex. We can, however, use basic `smacof` theory (De Leeuw (1977)) to majorize stress by a convex quadratic. This can be used to define a simple iterative algorithm. In each iteration we majorize stress, and then minimize the convex quadratic majorizer over configurations satisfying the convex quadratic constraints $d_{ij}^2(X) \leq \alpha_{ij}^2$. Compute a new majorization at the minimizer of the majorized problem, and so on. These are the two steps in the majorization or MM approach to optimization (De Leeuw (1994), Lange (2016)). To minimize a convex quadratic under convex quadratic constraints we use the dual Newton method from De Leeuw (2017).

1.1 Simplifying stress

A simplified expression for stress, first used in De Leeuw (1993), is

$$\textbf{stress}(x) = \frac{1}{2}\sum_{k=1}^{K} w_k(\delta_k - \sqrt{x'A_kx})^2.$$

We have put the elements below the diagonal of W and Δ in a vector of length $\frac{1}{2}n(n-1)$. Also $x := \textbf{vec}(X)$ and the A_k are the direct sums of p copies of the corresponding A_{ij}. Now expand the square, and assume without loss of generality that $\frac{1}{2}\sum_{k=1}^{K} w_k\delta_k^2 = 1$.

Then

$$\textbf{stress}(x) = 1 - \sum_{k=1}^K w_k \delta_k \sqrt{x' A_k x} + \frac{1}{2} x' V x,$$

with $V := \sum_{k=1}^K w_k A_k$.

Suppose $V = K\Lambda^2 K'$ is a complete eigen-decomposition of V. Some care is needed to handle the fact that V is singular, but under the assumption of irreducibility (De Leeuw (1977)) this is easily taken care of. Change variables with $z = \Lambda^{\frac{1}{2}} K' x$. Then

$$\textbf{stress}(z) = 1 - \sum_{k=1}^K w_k \delta_k \sqrt{z' \tilde{A}_k z} + \frac{1}{2} z' z,$$

with $\tilde{A}_k = \Lambda^{-\frac{1}{2}} K' A_k K \Lambda^{-\frac{1}{2}}$, so that $\sum_{k=1}^K w_k \tilde{A}_k = I$.

1.2 Smacof Notation and Theory

We have see in the previous section that we can write

$$\textbf{stress}(x) = 1 - \rho(x) + \frac{1}{2} x' x,$$

where

$$\rho(x) := \sum_{k=1}^K w_k \delta_k d_k(x),$$

with distances $d_k(x) := \sqrt{x' A_k x}$, and the A_k positive semi-definite matrices that add up to the identity. By Cauchy-Schwarz,

$$\rho(x) = x' B(x) x \geq x' B(y) y,$$

where

$$B(x) := \sum_{k=1}^K w_k \frac{\delta_k}{d_k(x)} A_k.$$

If we define the *Guttman transform* if x as $\Gamma(x) := B(x)x$, then for all x

$$(stress)(x) = 1 - x'\Gamma(x) + \frac{1}{2} x' x = (1 - \frac{1}{2}\Gamma(x)'\Gamma(x)) + \frac{1}{2}(x - \Gamma(x))'(x - \Gamma(x)),$$

and for all x, y

$$(stress) \leq 1 - x'\Gamma(y) + \frac{1}{2} x' x = (1 - \frac{1}{2}\Gamma(y)'\Gamma(y)) + \frac{1}{2}(x - \Gamma(y)'(x - \Gamma(y)),$$

In this notation a `smacof` iteration is $x^{(k+1)} = \Gamma(x^{(k)})$.

1.3 Necessary Conditions

We look at the necessary conditions for an optimum of the bounded MDS problem. The Lagrangian is

$$\mathcal{L}(x,\lambda) := \mathbf{stress}(x) + \frac{1}{2}\sum_{k=1}^K \lambda_k(x'A_kx - \delta_k^2)$$

If x is feasible and optimal for the problem of minimizing stress from below then there is a $\lambda \geq 0$, not all zero, such that

$$\mathcal{D}_1\mathcal{L}(x,\lambda) = \left(I + \sum_{k=1}^K \lambda_k A_k\right)x - \Gamma(x) = 0$$

and the multipliers λ and constraint functions are complementary, i.e. $\lambda_k(x'A_kx - \delta_k^2) = 0$ for all k. Compare this with the necessary condition for a local minimum in regular `smacof`, which is simply $x =$
$Gamma(x)$.

It should be noted that $x = \Gamma(x)$ implies $\rho(x) = x'x$, and consequently at a stationary point $\sigma(x) = 1 - \frac{1}{2}x'x$, which implies $x'x \leq 2$. Thus $\sum_{k=1}^K w_k d_k^2(x) \leq 2$, which implies $d_k(x) \leq \sqrt{2/w_k}$. Even in regular `smacof`, without bound constraints, the distances at a stationary point are bounded, and imposing the constraints $x'A_kx \leq 2/w_k$ still allows the unconstrained `smacof` solution. If the minimum stress is non-zero, then none of the constraints will be active, and all the distances will be strictly smaller than the a priori bounds.

2 MDS from Below

Classical MDS (Torgerson (1958)) can be interpreted as minimizing the loss function *strain*, defined as

$$\mathbf{strain}(X) := \frac{1}{4}\mathbf{tr}\ J_n(\Delta^2 - D^2(X))J_n(\Delta^2 - D^2(X)),$$

over all $n \times p$ centered configuration matrices X, where J_n is the centering operator $J_n = I_n - \frac{1}{n}ee'$, Δ^2 is a matrix with squared dissimilarities, and $D^2(X)$ is a matrix of squared Euclidean distances.

The loss function interpretation is based on the identity

$$-\frac{1}{2}J_nD^2(X)J_n = XX',$$

which implies

$$\mathbf{strain}(X) = \mathbf{tr}\ (C - XX')^2,$$

with $C := -\frac{1}{2}J_n\Delta^2J_n$.

If C has signature (n_+, n_0, n_-) then we can write $C = \overline{C} - \underline{C}$ with both $\overline{C}$ and $\underline{C}$ positive semi-definite, of ranks n_+ and n_-, and with $\mathbf{tr}\ \overline{C}\underline{C} = 0$. $\overline{C}$ is the projection of C on the

convex cone of positive semi-definite matrices, and $\underline{C} = -(C - \overline{C})$ is the negative of the projection on its polar cone. In terms of the eigenvalues and eigenvectors of C we can write $C = \overline{K\Lambda K}' + \underline{K\Lambda K}$, with overlining used for the positive eigenvalues and their eigenvectors, and underlining for the negative ones. Thus

$$\mathbf{strain}(X) = \mathbf{tr}\ (\overline{C} - \underline{C} - XX')^2 = \mathbf{tr}\ (\overline{C} - XX')^2 + \mathbf{tr}\ \underline{C}^2 + 2\mathbf{tr}\ X'\underline{C}X,$$

and

$$\min_X \mathbf{strain}(X) \geq \min_Z \mathbf{tr}\ (\overline{C} - ZZ')^2 + \mathbf{tr}\ \underline{C}^2 + 2\min_Y \mathbf{tr}\ Y'\underline{C}Y = \min_Z \mathbf{tr}\ (\overline{C} - ZZ')^2 + \mathbf{tr}\ \underline{C}^2.$$

Thus if $p \geq n_+$ we choose Z so that $ZZ' = \overline{C}$, if $p < n_+$ we use the p largest eigenvalues and corresponding eigenvectors of $\overline{C}$ to find ZZ'.

Now consider the squared distances $d_{ij}^2(\overline{C}) = \overline{c}_{ii} + \overline{c}_{jj} - 2\overline{c}_{ij}$ and the squared distances $d_{ij}^2(X_r) = \sum_{s=1}^r (x_{is} - x_{js})^2$. Then for all i, j

$$d_{ij}^2(X_1) \leq d_{ij}^2(X_2) \leq \cdots \leq d_{ij}^2(X_{n+}) = d_{ij}^2(\overline{C}).$$

Thus solutions are *nested* and we use *quadratic approximation from below* of the positive semi-definite part of C (De Leeuw and Meulman (1986))

The idea of approximation from below, which is natural in the case of classical MDS, can also be applied to the minimization of stress, using the coinstraints $d_{ij}(X) \leq \delta_{ij}$ for all i and j.

This may seem somewhat perverse, because why would we ever impose such constraints ? In this context, however, it makes sense to interpret the δ_{ij} as *bounds* instead of *dissimilarities*. The problem is to locate objects in space in such a way that their distances do not exceed their upper bounds, and are as close to these bounds as possible. Thus the upper bound problem is perhaps more like a location problem than a multidimensional scaling problem.

3 Examples

3.1 Equidistances

Our first example has $n = 10$ with all δ_{ij} equal. The optimal `smacof` solution in two dimensions needs 131 iterations to arrive at stress 0.1098799783. Approximation from below uses 30 iterations and finds stress 0.1520077612. The two configurations are in figure 1. Note they produce two different local minima, one with nine points equally spaced on the circle and a tenth point in the center, and the other with ten points equally spaced on the circle. At the solution of the bounded problem there are 10 active constraints, which have $d_{ij}(X) = \delta_{ij}(X) = 1$.

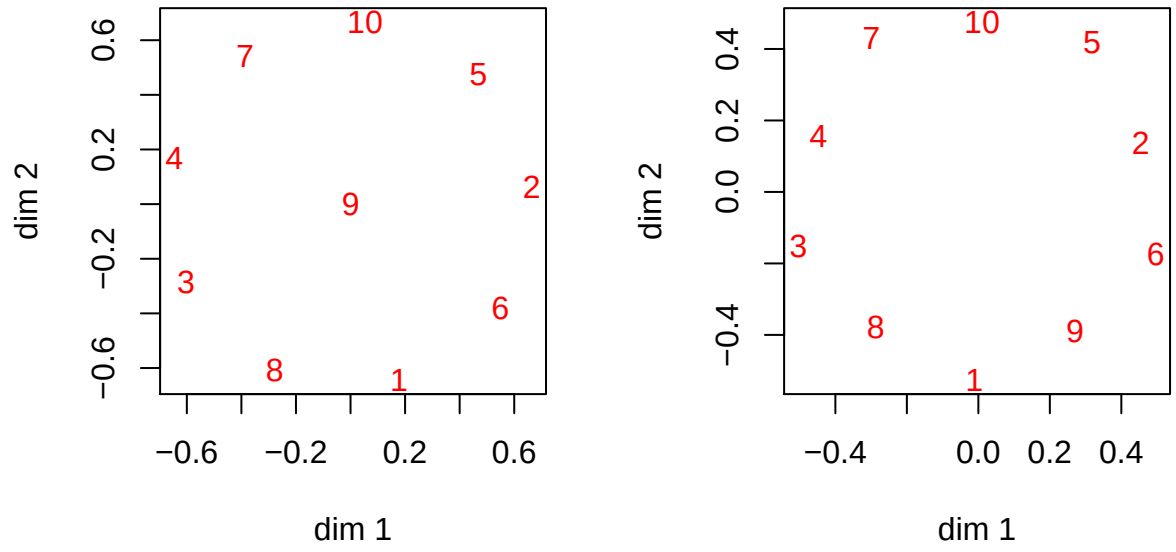

Figure 1: Equidistance Data, Regular Left, From Below Right

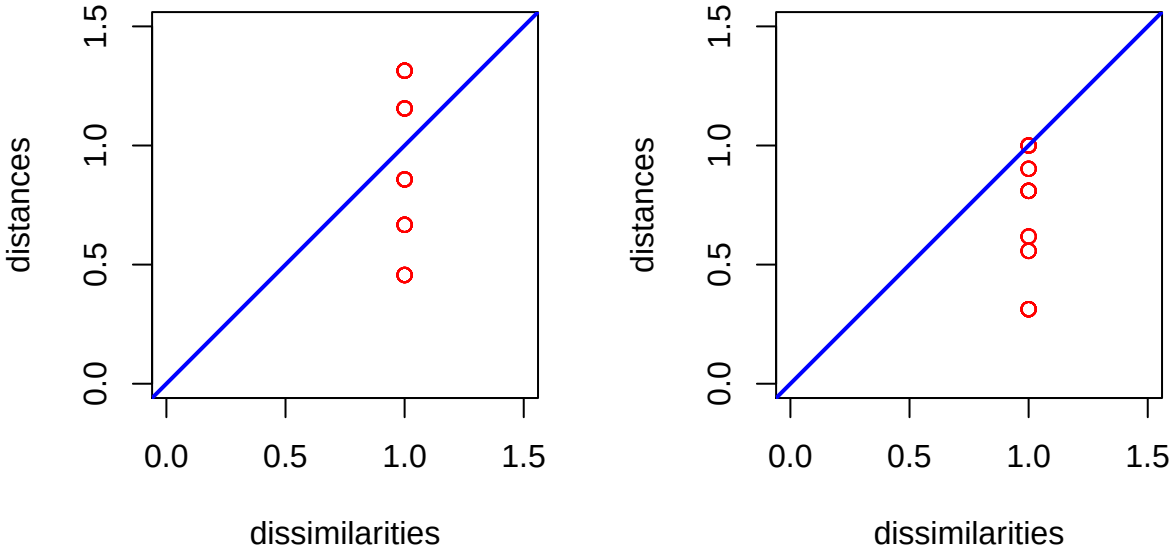

Figure 2: Equidistance Data, Regular Left, From Below Right

To illustrate complementary slackness we give the values of the constraint functions $d_{ij}^2(X) - \delta_{ij}^2$, which are all non-positive, and the values of the Lagrange multipliers, which are all non-negative. We see that a constraint function that is equal to zero means a positive Lagrange multiplier, and a constraint function that is negative means a zero Lagrange multiplier.

```
mprint (h2b$constraints, d = 6, f = "+")
```

```
##  [1] -0.172287 -0.309015 -0.172287 +0.000000 -0.309015 +0.000000 -0.451059
##  [8] -0.451058 +0.000000 +0.000000 -0.093093 -0.451058 -0.451058 -0.172283
## [15] -0.093098 -0.344574 -0.344576 -0.451059 +0.000000 +0.000000 -0.309023
## [22] -0.451053 -0.172291 -0.172292 -0.172283 +0.000000 -0.451058 -0.344574
## [29] -0.093098 -0.344576 -0.309023 -0.309009 +0.000000 -0.172288 -0.451054
## [36] +0.000000 -0.172292 -0.451053 -0.172292 -0.172287 +0.000000 -0.451055
```

```
## [43] -0.344584 -0.093096 -0.093096
```

```
mprint (h2b$multipliers, d = 6, f = "+")
```

```
##  [1] +0.000000 +0.000000 +0.000000 +0.088502 +0.000000 +0.088503 +0.000000
##  [8] +0.000000 +1.220853 +1.220844 +0.000000 +0.000000 +0.000000 +0.000000
## [15] +0.000000 +0.000000 +0.000000 +0.000000 +0.088508 +0.088477 +0.000000
## [22] +0.000000 +0.000000 +0.000000 +0.000000 +1.220844 +0.000000 +0.000000
## [29] +0.000000 +0.000000 +0.000000 +0.000000 +1.220840 +0.000000 +0.000000
## [36] +0.088508 +0.000000 +0.000000 +0.000000 +0.000000 +1.220840 +0.000000
## [43] +0.000000 +0.000000 +0.000000
```

3.2 Dutch Political Parties 1967

As the next illustration we use data from De Gruijter (1967), with average dissimilarity judgments between Dutch political parties in 1967. The data are

```
##       KVP PvdA  VVD  ARP  CHU  CPN  PSP   BP
## PvdA 5.63
## VVD  5.27 6.72
## ARP  4.60 5.64 5.46
## CHU  4.80 6.22 4.97 3.20
## CPN  7.54 5.12 8.13 7.84 7.80
## PSP  6.73 4.59 7.55 6.73 7.08 4.08
## BP   7.18 7.22 6.90 7.28 6.96 6.34 6.88
## D66  6.17 5.47 4.67 6.13 6.04 7.42 6.36 7.36
```

The optimal `smacof` solution in two dimensions needs 319 iterations to arrive at stress 0.044603386. Approximation from below uses 32 iterations and finds stress 0.0752702106. At the solution there are 10 active constraints, but as the Shepard plots in figure 4 show, the active constraints now do not correspond with the largest dissimilarities, and the two configurations in figure 3 are quite different. Although that could, of course, also be because they correspond with two different local minima, which mostly differ in the location of the D66 party (a neoliberal party, full of civilized protest).

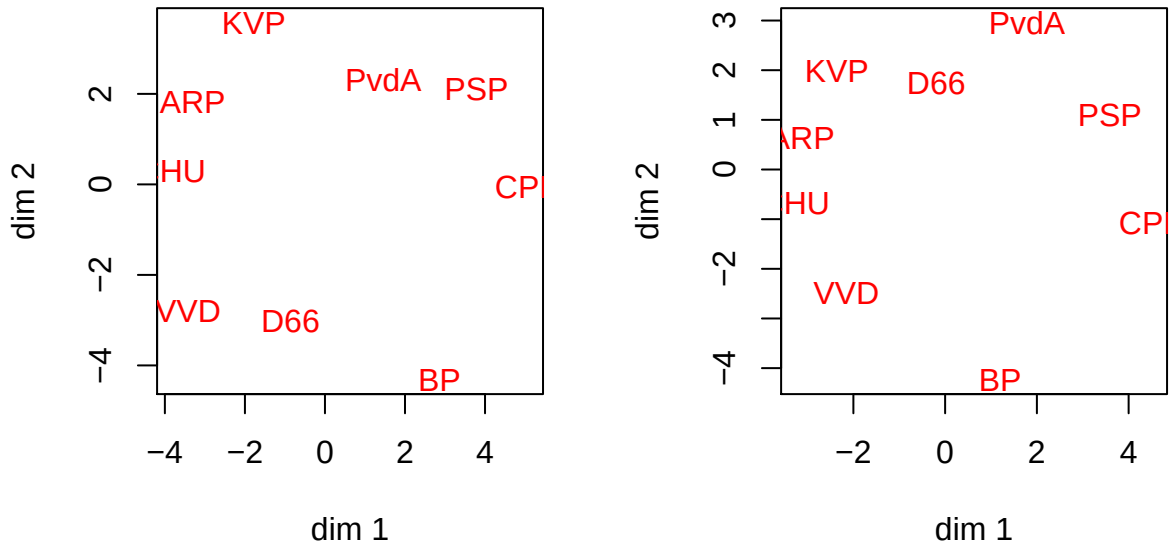

Figure 3: De Gruijter Data, Regular Left, From Below Right

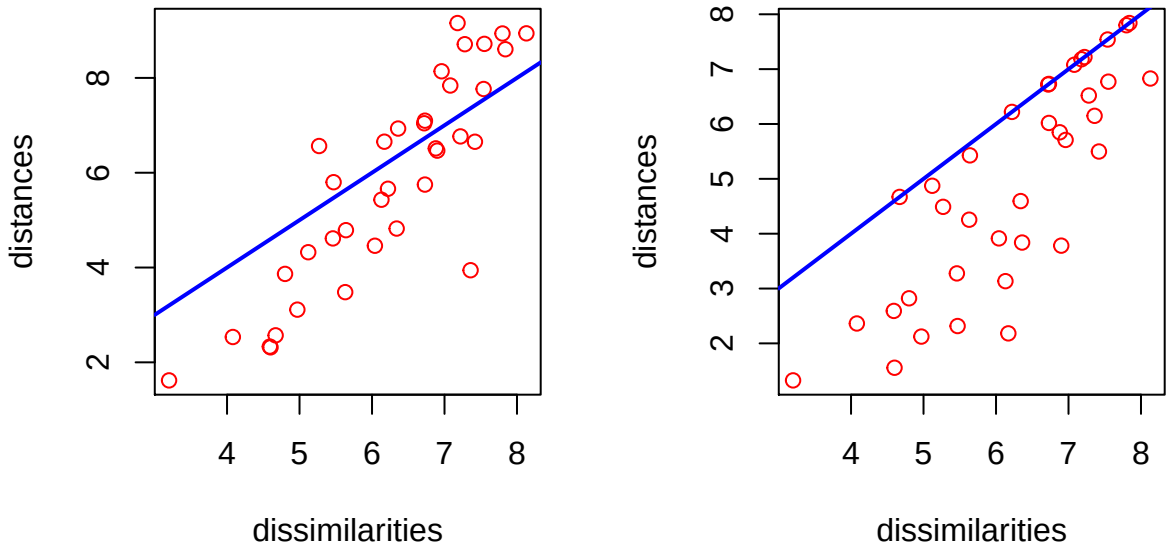

Figure 4: De Gruijter Data, Regular Left, From Below Right

In the next analysis we require that all distances are less than or equal to 8.13, the maximum dissimilarity. In order words, the maximum distances must be less than or equal to the maximum dissimilarity.

The optimal solution under these constraints in two dimensions needs 70 iterations to arrive at stress 0.0537645693. At the solution there are 6 active constraints, i.e. six distances equal to 8.13.

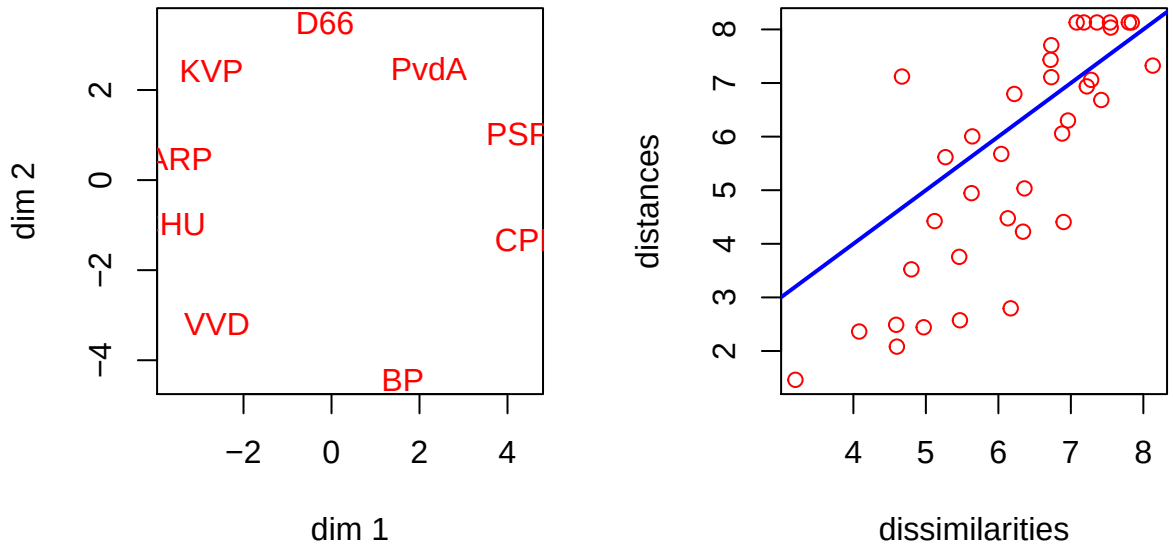

Figure 5: De Gruijter Data, Distances Bounded by Maximum Dissimilarity

And finally an analysis in which we require that distances between ARP, CHU, KVP (the Christian Democrat parties) are less than 0.1, and the distances between PvdA, PSP, CPN (the leftist parties) are also less than 0.1. This will effectively collapse them in the configuration plot.

The optimal solution under these constraints in two dimensions needs 85 iterations to arrive at stress 0.1092519844. At the solution all 6 constraints are active.

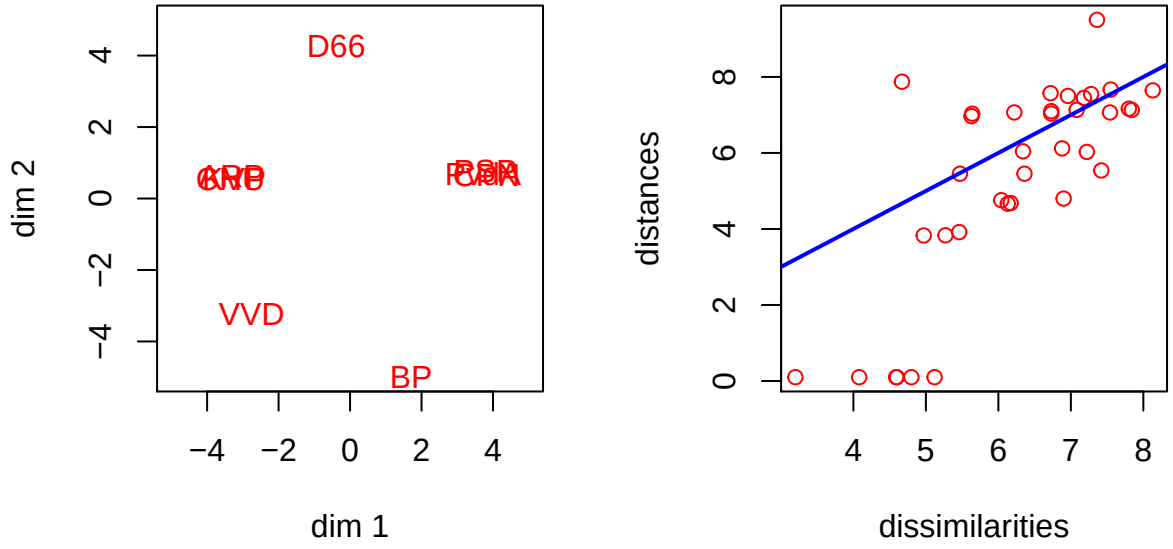

Figure 6: De Gruijter Data, Imposing Small Distances

3.3 Ekman Color Data

The final example are the color data from Ekman (1954). The optimal `smacof` solution in two dimensions needs 26 iterations to arrive at stress 0.0172132469. Approximation from

below uses 16 iterations and finds stress 0.0274106473. At the solution there are 15 active constraints. Again the bounded solution is a shrunken version of the regular `smacof` solution, and figure 8 shows that the active constraints correspond with the largest dissimilarities and distances.

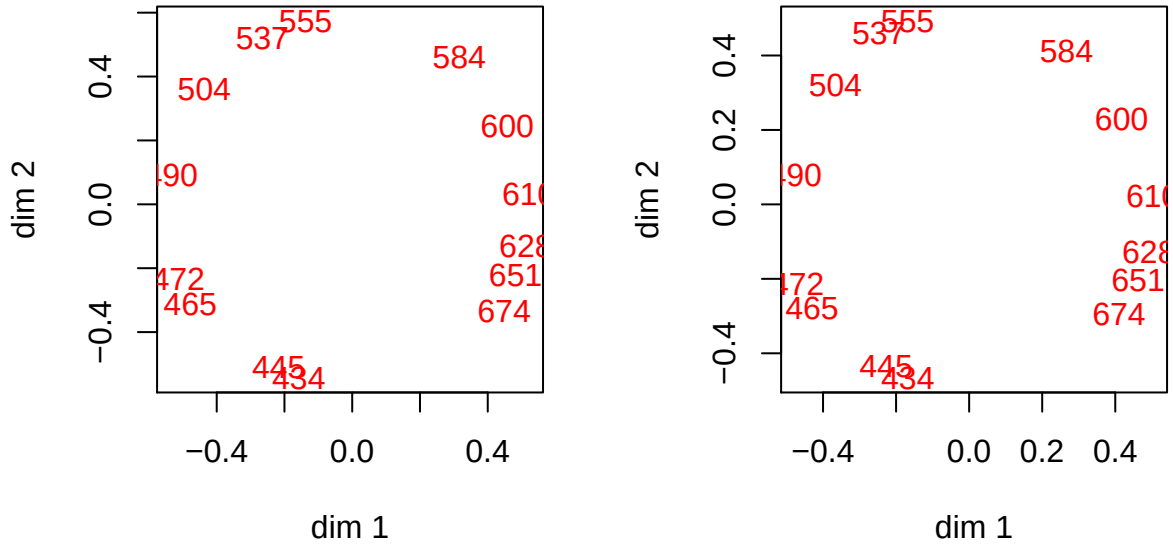

Figure 7: Ekman Data, Regular Left, From Below Right

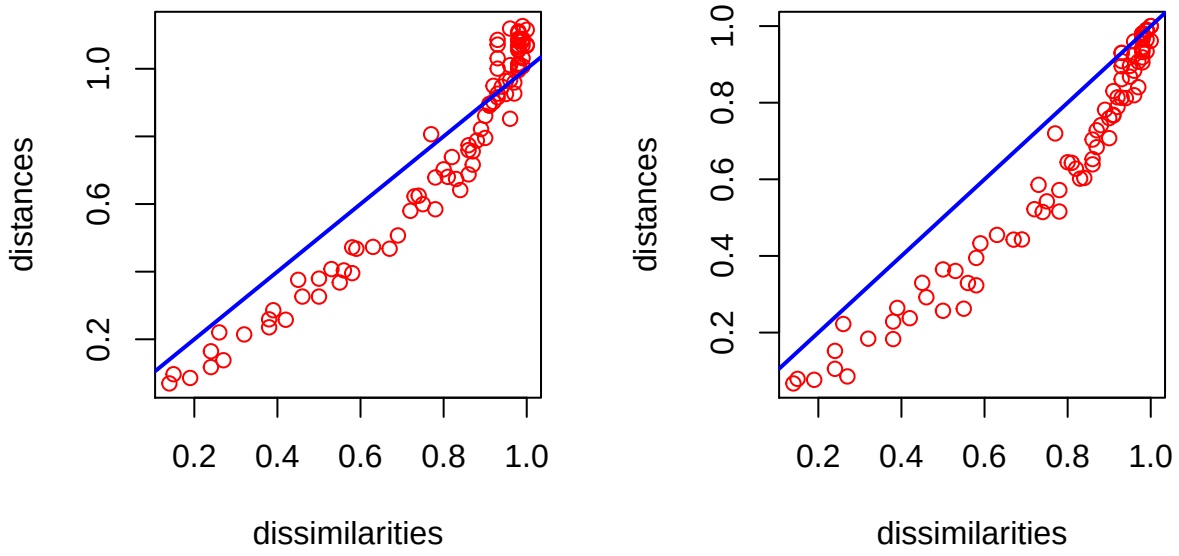

Figure 8: Ekman Data, Regular Left, From Below Right

Figure 9 shows that the two configurations in figure 7 may look to be proportional, they really are not. There are many deviations from proportionality if we compare the fitted distances in the regular and bounded `smacof` solutions.

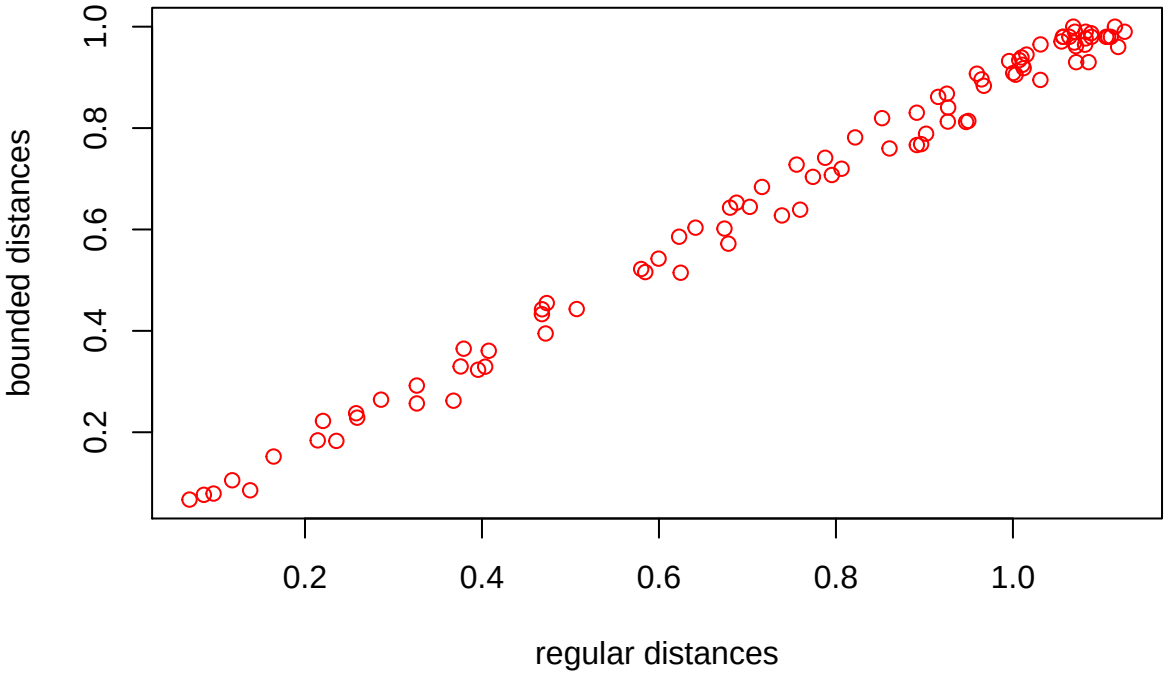

Figure 9: Ekman Data, Regular vs Bounded Distances

4 Discussion

If $p = n - 1$ our multidimensional scaling problem is full-dimensional scaling or FDS (De Leeuw, Groenen, and Mair (2016)). In that case stress can be parametrized by using the scalar products C and we must minimize

$$\textbf{stress}(C) := \sum_{i=1}^n \sum_{j=1}^n w_{ij}(\delta_{ij} - \sqrt{\textbf{tr}\ A_{ij}C})^2$$

over $C \gtrsim 0$ and $\textbf{tr}\ A_{ij}C \leq \alpha_{ij}$. This means that FDS with upper bounds means minimizing a convex function over a convex set, in fact solving a semidefinite programming problem. It also means that the concept of Gower rank starts playing a role (De Leeuw (2016b)).

MDS from below can also be used with different loss functions. Minimizing

$$\sum_{i=1}^n \sum_{j=1}^n w_{ij}|\delta_{ij} - d_{ij}(X)|$$

over $d_{ij}(X) \leq \delta_{ij}$ is the same as maximizing the weighted sum of the distances, a convex function. Thus we can use tangential majorization (De Leeuw (2016a)), which means the majorization function is linear and has no quadratic term.

Alternatively, the problem of minimizing

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

over $d_{ij}(X) \leq \delta_{ij}$ means maximizing a convex quadratic form. Although the objective function is quadratic, we still need tangential majorization to get a convex objective to minimize.

We do not address the problem in this paper of minimizing stress under constraints of the form $\beta_{ij} \leq d_{ij}(X) \leq \alpha_{ij}$. This is a fundamentally more complicated problem, because the lower bounds make the solution set non-convex. In fact, using lower bounds may make the solution set empty, because there may not be p-dimensional Euclidean distances satisfying the constraints.

It will be of some interest to apply our algorithm to unfolding examples, where we could have upper bound constraints on the row distances, the column distances, or the row-column distances. Of course in the unfolding case lower bound constraints are at least as interesting, because they can in principle be used to avoid the trivial solutions that are so common in unfolding, but we have argued that computationally lower bound constraints are far more difficult to handle.

5 Appendix: Code

5.1 below.R

```
suppressPackageStartupMessages (library (mgcv, quietly = TRUE))
suppressPackageStartupMessages (library (MASS, quietly = TRUE))
suppressPackageStartupMessages (library (lsei, quietly = TRUE))
suppressPackageStartupMessages (library (numDeriv, quietly = TRUE))
source("auxilary.R")
source("mdsUtils.R")
source("nnnewton.R")
source("qpqc.R")
source("smacof.R")
source("smacofBelow.R")
```

5.2 auxilary.R

```
mprint <- function (x,
                    d = 6,
                    w = 8,
                    f = "") {
  print (noquote (formatC (
    x,
    di = d,
    wi = w,
    fo = "f",
```

```
    flag = f
  )))
}

directSum <- function (x) {
  m <- length (x)
  nr <- sum (sapply (x, nrow))
  nc <- sum (sapply (x, ncol))
  z <- matrix (0, nr, nc)
  kr <- 0
  kc <- 0
  for (i in 1:m) {
    ir <- nrow (x[[i]])
    ic <- ncol (x[[i]])
    z[kr + (1:ir), kc + (1:ic)] <- x[[i]]
    kr <- kr + ir
    kc <- kc + ic
  }
  return (z)
}

repList <- function(x, n) {
  z <- list()
  for (i in 1:n)
    z <- c(z, list(x))
  return(z)
}

shapeMe <- function (x) {
  m <- length (x)
  n <- (1 + sqrt (1 + 8 * m)) / 2
  d <- matrix (0, n, n)
  k <- 1
  for (i in 2:n) {
    for (j in 1:(i - 1)) {
      d[i, j] <- d[j, i] <- x[k]
      k <- k + 1
    }
  }
  return (d)
}

symmetricFromTriangle <- function (x, lower = TRUE, diagonal = TRUE) {
  k <- length (x)
```

```
  if (diagonal)
    n <- (sqrt (1 + 8 * k) - 1) / 2
  else
    n <- (sqrt (1 + 8 * k) + 1) / 2
  if (n != as.integer (n))
    stop ("input error")
  nn <- 1:n
  if (diagonal && lower)
    m <- outer (nn, nn, ">=")
  if (diagonal && (!lower))
    m <- outer (nn, nn, "<=")
  if ((!diagonal) && lower)
    m <- outer (nn, nn, ">")
  if ((!diagonal) && (!lower))
    m <- outer (nn, nn, "<")
  b <- matrix (0, n, n)
  b[m] <- x
  b <- b + t(b)
  if (diagonal)
    diag (b) <- diag(b) / 2
  return (b)
}

triangleFromSymmetric <- function (x, lower = TRUE, diagonal = TRUE) {
  n <- ncol (x)
  nn <- 1:n
  if (diagonal && lower)
    m <- outer (nn, nn, ">=")
  if (diagonal && (!lower))
    m <- outer (nn, nn, "<=")
  if ((!diagonal) && lower)
    m <- outer (nn, nn, ">")
  if ((!diagonal) && (!lower))
    m <- outer (nn, nn, "<")
  return (x[m])
}
```

5.3 mdsUtils.R

```
library (mgcv)

torgerson <- function (delta, p = 2) {
```

```
  z <- slanczos(-doubleCenter((delta ^ 2) / 2), p)
  w <- matrix (0, p, p)
  v <- pmax(z$values, 0)
  diag (w) <- sqrt (v)
  return(z$vectors %*% w)
}

basisPrep <- function (n, p, w) {
  m <- n * (n - 1) / 2
  v <- -symmetricFromTriangle (w, diagonal = FALSE)
  diag (v) <- -rowSums(v)
  ev <- eigen (v)
  eval <- ev$values[1:(n - 1)]
  evec <- ev$vectors[, 1:(n - 1)]
  z <- evec %*% diag (1 / sqrt (eval))
  a <- array (0, c(n - 1, n - 1, m))
  k <- 1
  for (j in 1:(n-1)) {
    for (i in (j+1):n) {
      dif <- z[i,] - z[j,]
      a [, , k] <- outer (dif, dif)
      k <- k + 1
    }
  }
  return (list (z = z, a = a))
}

center <- function (x) {
  return (apply (x, 2, function (z) z - mean (z)))
}

doubleCenter <- function (x) {
  n <- nrow (x)
  j <- diag(n) - (1 / n)
  return (j %*% x %*% j)
}

squareDist <- function (x) {
  d <- diag (x)
  return (outer (d, d, "+") - 2 * x)
}
```

5.4 smacof.R

```
smacof <-
  function (delta,
            p = 2,
            xini = torgerson (symmetricFromTriangle (delta, diagonal = FALSE), p),
            w = rep (1, length (delta)),
            itmax = 1000,
            eps = 1e-10,
            verbose = FALSE) {
    m <- length (delta)
    n <- (1 + sqrt (1 + 8 * m)) / 2
    r <- p * (n - 1)
    h <- basisPrep (n, p, w)
    xold <- rep (0, r)
    for (s in 1:p) {
      k <- (s - 1) * (n - 1) + 1:(n - 1)
      xold[k] <-
        crossprod (h$z,  xini[, s]) / diag (crossprod (h$z))
    }
    d <- rep (0, m)
    itel <- 1
    sold <- Inf
    ssq <- sum (w * delta ^ 2)
    repeat {
      bmat <- matrix (0, n - 1, n - 1)
      xx <- matrix (xold, n - 1, p)
      for (k in 1:m) {
        ak <- h$a[, , k]
        d[k] <- sqrt (sum (xx * (ak %*% xx)))
        bmat <- bmat + w[k] * (delta[k] / d[k]) * ak
      }
      xnew <- as.vector (bmat %*% xx)
      snew <- sum (w * (delta - d) ^ 2) / ssq
      if (verbose)
        cat(
          "Iteration: ",
          formatC (itel, width = 3, format = "d"),
          "sold: ",
          formatC (
            sold,
            digits = 8,
            width = 12,
            format = "f"
```

```
          ),
          "snew: ",
          formatC (
            snew,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "\n"
        )
      if ((itel == itmax) || ((sold - snew) < eps))
        break
      xold <- drop (xnew)
      sold <- snew
      itel <- itel + 1
    }
    xconf <- matrix (0, n, p)
    for (s in 1:p) {
      k <- (s - 1) * (n - 1) + 1:(n - 1)
      xconf[, s] <- h$z %*% xnew[k]
    }
    return (list (
      delta = delta,
      dist = d,
      x = xconf,
      itel = itel,
      stress = snew
    ))
  }
```

5.5 smacofBelow.R

```
smacofBelow <-
  function (delta,
            p = 2,
            xini = torgerson (symmetricFromTriangle (delta, diagonal = FALSE), p),
            w = rep (1, length (delta)),
            bnd = delta,
            itmax = 1000,
            eps = 1e-10,
            verbose = FALSE) {
    m <- length (delta)
    n <- (1 + sqrt (1 + 8 * m)) / 2
```

```
  wnd <- which (bnd < Inf)
  l <- length (wnd)
  r <- p * (n - 1)
  yold <- rep (0, l)
  h <- basisPrep (n, p, w)
  xold <- rep (0, r)
  for (s in 1:p) {
    k <- (s - 1) * (n - 1) + 1:(n - 1)
    xold[k] <-
      crossprod (h$z,  xini[, s]) / diag (crossprod (h$z))
  }
  d <- rep (0, m)
  itel <- 1
  sold <- Inf
  ssq <- sum (w * delta ^ 2)
  repeat {
    bmat <- matrix (0, n - 1, n - 1)
    xx <- matrix (xold, n - 1, p)
    for (k in 1:m) {
      ak <- h$a[, , k]
      d[k] <- sqrt (sum (xx * (ak %*% xx)))
      bmat <- bmat + w[k] * (delta[k] / d[k]) * ak
    }
    xmid <- as.vector (bmat %*% xx)
    ccomp <- c(ssq,-bnd[wnd] ^ 2) / 2
    bcomp <- matrix (0, r, l + 1)
    bcomp[, 1] <- -xmid
    acomp <- array (0, c(r, r, l + 1))
    acomp[, , 1] <- diag (r)
    t <- 1
    for (k in wnd) {
      ak <- directSum (repList (h$a[, , k], p))
      acomp[, , t + 1] <- ak
      t <- t + 1
    }
    v <- qpqc (yold, acomp, bcomp, ccomp, verbose = FALSE)
    snew <- 2 * v$f / ssq
    xnew <- v$xmin
    ynew <- v$multipliers
    cons <- v$constraints
    if (verbose)
      cat(
        "Iteration: ",
        formatC (itel, width = 3, format = "d"),
```

```
        "sold: ",
        formatC (
          sold,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "snew: ",
        formatC (
          snew,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "\n"
      )
    if ((itel == itmax) || ((sold - snew) < eps))
      break
    xold <- xnew
    sold <- snew
    yold <- ynew
    itel <- itel + 1
  }
  xconf <- matrix (0, n, p)
  for (s in 1:p) {
    k <- (s - 1) * (n - 1) + 1:(n - 1)
    xconf[, s] <- h$z %*% xnew[k]
  }
  return (
    list (
      delta = delta,
      dist = d,
      x = xconf,
      multipliers = ynew,
      constraints = cons,
      itel = itel,
      stress = sum (w * (delta - d) ^ 2) / ssq
    )
  )
}
```

References

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

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. http://www.stat.ucla.edu/~deleeuw/janspubs/1977/chapters/deleeuw_C_77.pdf.

———. 1993. "Fitting Distances by Least Squares." Preprint Series 130. Los Angeles, CA: UCLA Department of Statistics. http://www.stat.ucla.edu/~deleeuw/janspubs/1993/reports/deleeuw_R_93c.pdf.

———. 1994. "Block Relaxation Algorithms in Statistics." In *Information Systems and Data Analysis*, edited by H.H. Bock, W. Lenski, and M.M. Richter, 308–24. Berlin: Springer Verlag. http://www.stat.ucla.edu/~deleeuw/janspubs/1994/chapters/deleeuw_C_94c.pdf.

———. 2016a. *Block Relaxation Methods in Statistics*. Bookdown. https://bookdown.org/jandeleeuw6/bras/.

———. 2016b. "Gower Rank." doi:10.13140/RG.2.1.1890.5202.

———. 2017. "Quadratic Programming with Quadratic Constraints." doi:10.13140/RG.2.2.20763.87841.

De Leeuw, J., and J.J. Meulman. 1986. "Principal Component Analysis and Restricted Multidimensional Scaling." In *Classification as a Tool of Research*, edited by W. Gaul and M. Schader, 83–96. Amsterdam, London, New York, Tokyo: North-Holland. http://www.stat.ucla.edu/~deleeuw/janspubs/1986/chapters/deleeuw_meulman_C_86.pdf.

De Leeuw, J., P. Groenen, and P. Mair. 2016. "Full-Dimensional Scaling." doi:10.13140/RG.2.1.1038.4407.

Ekman, G. 1954. "Dimensions of Color Vision." *Journal of Psychology* 38: 467–74.

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

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