# Second Derivatives of rStress, with Applications

Jan de Leeuw, Patrick Groenen, Patrick Mair

Version 009, January 20, 2016

## Contents

Note: This is a working paper which will be expanded/updated frequently. The directory deleeuwpdx.net/pubfolders/secstress has a pdf copy of this article, the complete Rmd file that includes all code chunks, and R files with the code. Suggestions are welcome 24/7.

## 1 Problem

Define the multidimensional scaling (MDS) loss function

$$\sigma_r(x) = \sum_{i=1}^n w_i(\delta_i - (x'A_ix)^r)^2, \tag{1}$$

with $r > 0$ and the $A_i$ positive semi-definite. The $w_i$ are positive *weights*, the $\delta_i$ are non-negative *dissimilarities*. We call this *rStress* (De Leeuw, Groenen, and Mair (2016)). Special cases are *stress* (Kruskal 1964) for $r = \frac{1}{2}$, *sstress* (Takane, Young, and De Leeuw 1977) for $r = 1$, and the loss function used in MULTISCALE (Ramsay 1977) for $r \rightarrow 0$.

In this paper we are interested in the first and second derivatives of rStress, and in the various applications of these derivatives to the problem of minimizing rStress.

## 2 Derivatives

Compact expression for the first and second derivatives of $\sigma_r$ can be given by defining the matrices

$$B_r(x) := \sum_{i=1}^n w_i\delta_i(x'A_ix)^{r-1}A_i, \tag{2}$$

$$C_r(x) := \sum_{i=1}^n w_i(x'A_ix)^{2r-1}A_i, \tag{3}$$

$$S_r(x) := \sum_{i=1}^n w_i\delta_i(x'A_ix)^{r-1}\left[A_i + 2(r-1)\frac{A_ixx'A_i}{x'A_ix}\right], \tag{4}$$

$$T_r(x) := \sum_{i=1}^n w_i(x'A_ix)^{2r-1}\left[A_i + 2(2r-1)\frac{A_ixx'A_i}{x'A_ix}\right]. \tag{5}$$

We then have

$$\mathcal{D}\sigma_r(x) = -4r\{B_r(x) - C_r(x)\}x, \tag{6}$$

and

$$\mathcal{D}^2\sigma_r(x) = -4r\{S_r(x) - T_r(x)\}. \tag{7}$$

Note that $S_r(x)$ is positive semi-definite for $r \geq \frac{1}{2}$ and $T_r(x)$ is positive-semi-definite for $r \geq \frac{1}{4}$.

We have written the `R` function `mdsDerivatives()` to evaluate the gradient and Hessian. Just to make sure our formulas are correct, the code can optionally compute numerical derivatives using the `numDeriv` package of Gilbert and Varadhan (2014).

## 3 Newton's Method

Newton's method to minimize rStress is

$$x^{(k+1)} = x^{(k)} - [S_r(x^{(k)}) - T_r(x^{(k)})]^{-1}(B_r(x^{(k)}) - C_r(x^{(k)}))x^{(k)}. \tag{8}$$

As we can expect in highly nonlinear situations like MDS, Newton's method without safeguards sometimes works and sometimes doesn't. If it works, it is generally fast, which is of some interest at least because the majorization method developed in De Leeuw, Groenen, and Mair (2016) for minimizing rStress can be very slow, especially for $r \geq \frac{1}{2}$.

## 4 Dutch Political Parties

Our main example in the paper is are the dissimilarity measures for nine Dutch political parties, collected by De Gruijter (1967).

```
##       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
```

Newton's method converges in all cases, although it often behaves very erratically in the early iterations. Table 1 shows the number of iterations, the rStress value, the maximum norm of the gradient, and the smallest eigenvalue of the Hessian at the solution.

```
## r:  0.40  iters:     36  rStress:  0.05153249  maxGrad:  0.00000000  minHess:  -14.80
## r:  0.45  iters:     13  rStress:  0.06911461  maxGrad:  0.00000000  minHess:  -7.682
## r:  0.50  iters:    123  rStress:  0.10559640  maxGrad:  0.00000002  minHess:  -9.331
## r:  0.55  iters:     78  rStress:  0.05524495  maxGrad:  0.00000000  minHess:  -0.000
## r:  0.65  iters:     31  rStress:  0.09931966  maxGrad:  0.00000000  minHess:  -0.980
## r:  0.75  iters:      9  rStress:  0.14507211  maxGrad:  0.00000000  minHess:  -3.125
## r:  0.90  iters:     56  rStress:  0.13421690  maxGrad:  0.00000000  minHess:  -0.000
## r:  1.00  iters:     26  rStress:  0.14925820  maxGrad:  0.00000000  minHess:  -0.000
## r:  2.00  iters:     47  rStress:  0.35796584  maxGrad:  0.00000000  minHess:  -0.000
```

Table 1: Newton solutions with various r

Clearly for the majority of solutions Newton stops at a saddle point, or at least a flat spot fairly close to a local minimum. Only for large values of r do we find a proper local minimum.

For values of r less than .40 we cannot get Newton to work. It rapidly diverges into regions with very large values of both $x$ and rStress. The configurations in figure 1 also seem to differ quite a bit for smaller values of r. Note the increased clustering for increasing r, until finally for $r = 2$ parties are put in the edges of an equilateral triangle.

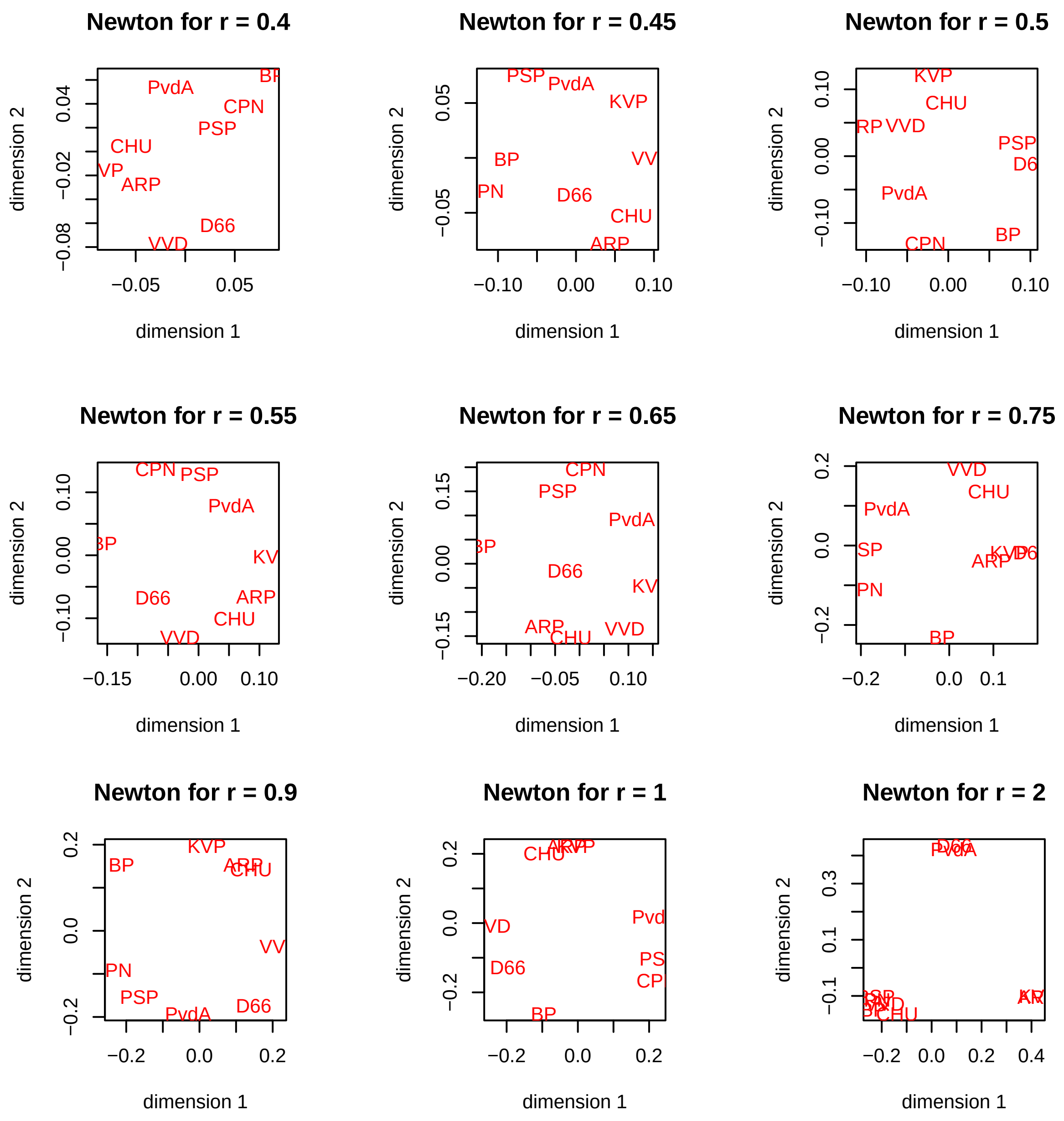

Figure 1: Newton configurations with various r

## 5 Majorizing Newton

In this section we limit ourselves to the case $r \geq \frac{1}{2}$. Without loss of generality we assume the dissimilarities are scaled by

$$\sum_{i=1}^{n} w_i\delta_i^2 = 1. \tag{9}$$

Next, it is convenient to define

$$\rho_r(x) := \sum_{i=1}^{n} w_i\delta_i(x'A_ix)^r, \tag{10}$$

and

$$\eta_r(x) := \sum_{i=1}^{n} w_i(x'A_ix)^{2r}, \tag{11}$$

so that

$$\sigma_r(x) = 1 - 2\rho_r(x) + \eta_r(x). \tag{12}$$

If $r \geq \frac{1}{2}$ then both $\rho_r$ and $\eta_r$ are convex (De Leeuw, Groenen, and Mair (2016)). Thus

$$\rho_r(x) \geq \rho_r(y) + (x-y)'\mathcal{D}\rho(y), \tag{13}$$

for all $x$ and $y$, which translates to the majorization

$$\sigma_r(x) = \min_y \zeta(x,y), \tag{14}$$

where

$$\zeta(x,y) := 1 - 2\rho_r(y) - 4r(x-y)'B_r(y)y + \eta_r(x). \tag{15}$$

Now consider the algorithm where we use block relaxation to alternate over minimization over $\zeta$ over $x$ and $y$. By definition

$$\arg\min_y \zeta(x,y) = x, \tag{16}$$

so minimization over $y$ for given $x$ is trivial. We minimize $\zeta$ over $x$ for given $y$ by using one or more steps of Newton's method, relying on the fact that $\zeta$ is convex in $x$ for given $y$. Thus there will be no local minima problem with Newton, although we may observe non-convergence. Note that it will not be neceesary for convergence to iterate Newton to convergence between updates of $y$. In fact we propose an algorithm in which only a single Newton step is done.

The derivatives needed for the Newton steps are

$$\mathcal{D}_1\zeta(x,y) = -4r(B_r(y)y - C_r(x)x), \tag{17}$$

and

$$\mathcal{D}_{11}\zeta(x,y) = 4rT_r(x). \tag{18}$$

Thus the two-block algorithm with a single Newton step becomes

$$y^{(k)} = x^{(k)}, \tag{19}$$

$$x^{(k+1)} = x^{(k)} - [T_r(x^{(k)})]^{-1}(B_r(y^{(k)})y^{(k)} - C_r(x^{(k)})x^{(k)}), \tag{20}$$

but this is of course equivalent to the algorithm

$$x^{(k+1)} = x^{(k)} - [T_r(x^{(k)})]^{-1}(B_r(x^{(k)}) - C_r(x^{(k)}))x^{(k)}. \tag{21}$$

This is what we have implemented in our `R` program, using the parameter `linearize=TRUE`. By default `linearize=FALSE`, which is the standard uncorrected Newton method.

The idea is to give up some speed (and quadratic convergence) by gaining stability. In table 2 we do see larger numbers of iterations (but iterations are marginally faster because they do not need $S_r(x)$). We also have observed monotone convergence of loss function values in all cases, and we see that convergence is always to a local minimum. In most cases, except for $r = 1$, the solution found has a lower loss function value than the one found by the Newton method. Remember, however, that our majorization method is only guaranteed to work for $r \geq \frac{1}{2}$.

```
## r:  0.40  iters:    288  rStress:  0.02854517  maxGrad:  0.00000011  minHess:  -0.000
## r:  0.45  iters:    268  rStress:  0.03823655  maxGrad:  0.00000009  minHess:  -0.000
## r:  0.50  iters:    729  rStress:  0.04460338  maxGrad:  0.00000011  minHess:  -0.000
## r:  0.55  iters:    186  rStress:  0.05524495  maxGrad:  0.00000009  minHess:  -0.000
## r:  0.65  iters:    104  rStress:  0.07731578  maxGrad:  0.00000006  minHess:  -0.000
## r:  0.75  iters:     96  rStress:  0.10711307  maxGrad:  0.00000006  minHess:  -0.000
## r:  0.90  iters:    150  rStress:  0.13989729  maxGrad:  0.00000005  minHess:  -0.000
## r:  1.00  iters:   1000  rStress:  0.15444014  maxGrad:  0.00000008  minHess:  -0.000
## r:  2.00  iters:     53  rStress:  0.23176557  maxGrad:  0.00000005  minHess:  -0.000
```

Table 2: Majorization solutions with various r

The configurations found by the majorization method are more stable over different values of r, and show the familar effect of becoming more and more clustered if r increases. Note that for $r = 2$ the majorization method finds a better location of the parties to the edges, although finding the optimum allocation is of course a combinatorial problem.

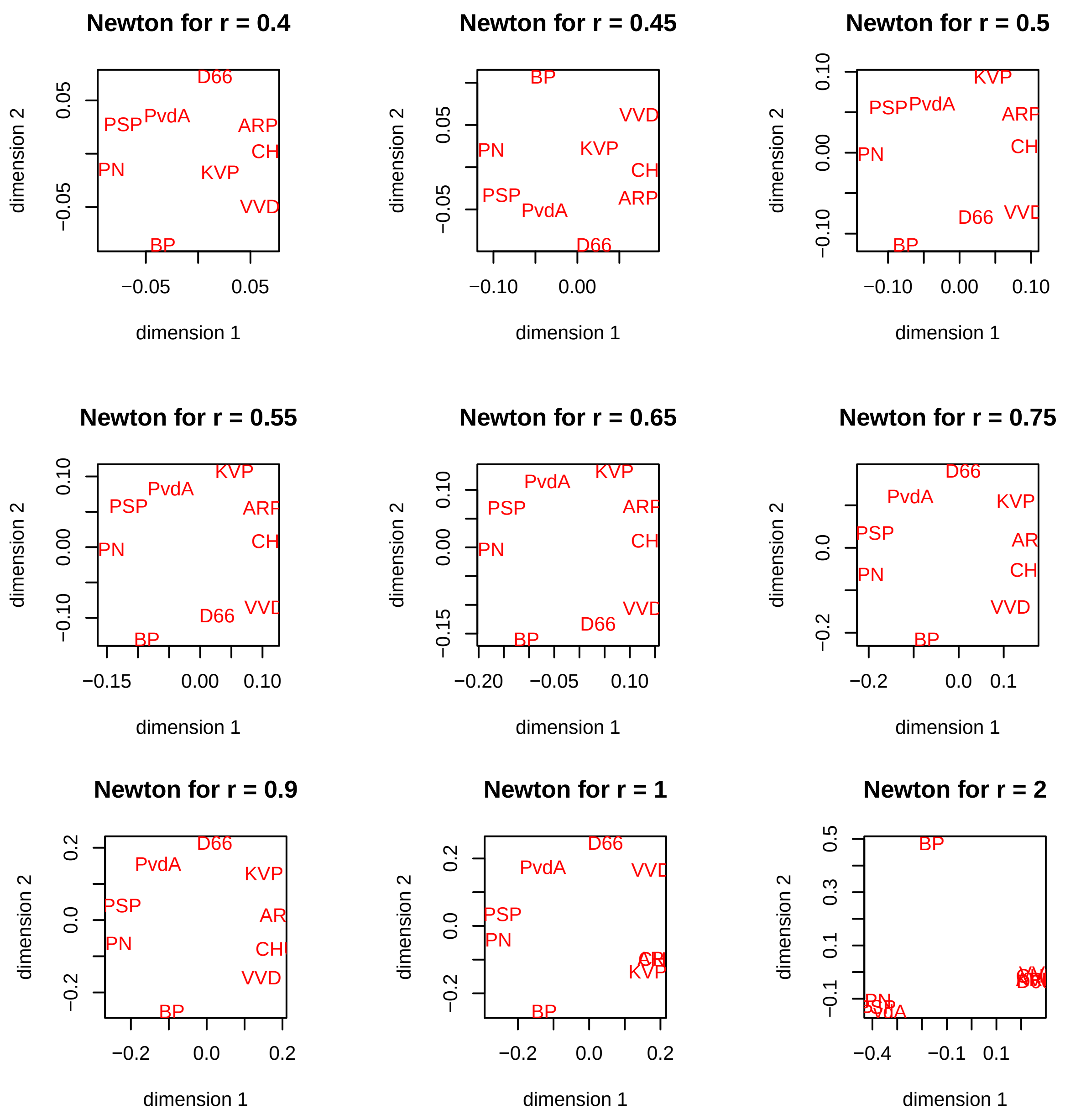
Figure 2: Majorization with various r

In table 3 we give the rStress values and iteration numbers for the *scalar* majorization algorithm of De Leeuw, Groenen, and Mair (2016). We see that the rStress values for $r \geq \frac{1}{2}$ are basically the same as the ones from the majorization algorithm in this paper, but the number of iterations is much larger. In fact, it is larger than 100,000 for $r = 1$ and $r = 2$.

```
## r:  0.10  iters:  29103  rStress:  0.00546400
## r:  0.25  iters:   3605  rStress:  0.00631000
## r:  0.50  iters:   3566  rStress:  0.04460300
## r:  0.75  iters:   3440  rStress:  0.10711300
## r:  1.00  iters:     NA  rStress:  0.15539200
## r:  2.00  iters:     NA  rStress:  0.23487700
```

Table 3: Scalar majorization solutions with various r

## 6 Sensitivity Analysis

The second derivatives can also be used to draw sensitivity regions around points in an MDS solution. At a point $x$ where the first derivatives vanish and the Hessian is positive semi-definite, we have

$$\sigma_r(y) \approx \sigma_r(x) + \frac{1}{2}(x-y)'\mathcal{D}^2\sigma_r(x)(x-y), \tag{22}$$

and thus $\{y \mid \sigma_r(y) \leq \alpha\}$ is approximately the ellipsoid

$$\{y \mid (x-y)'\mathcal{D}^2\sigma_r(x)(x-y) \leq 2(\alpha - \sigma_r(x))\}. \tag{23}$$

For graphics in the plane we take $2\times 2$ principal submatrices of the Hessian and draw ellipses, for example by using the `R` package `car` (Fox and Weisberg (2011)). We have to remember that in `car` the shape matrix is the inverse of our second derivative matrix, while their radius parameter corresponds with our $\sqrt{2(\alpha - \sigma_r(x))}$.

We illustrate this with the majorization solution for $r = \frac{1}{2}$, which has rStress 0.0446034. In figure 3 we choose $\alpha - \sigma_r = .001$, which means we look for the solutions which have rStress larger than 0.0446034 by 0.001 or less.

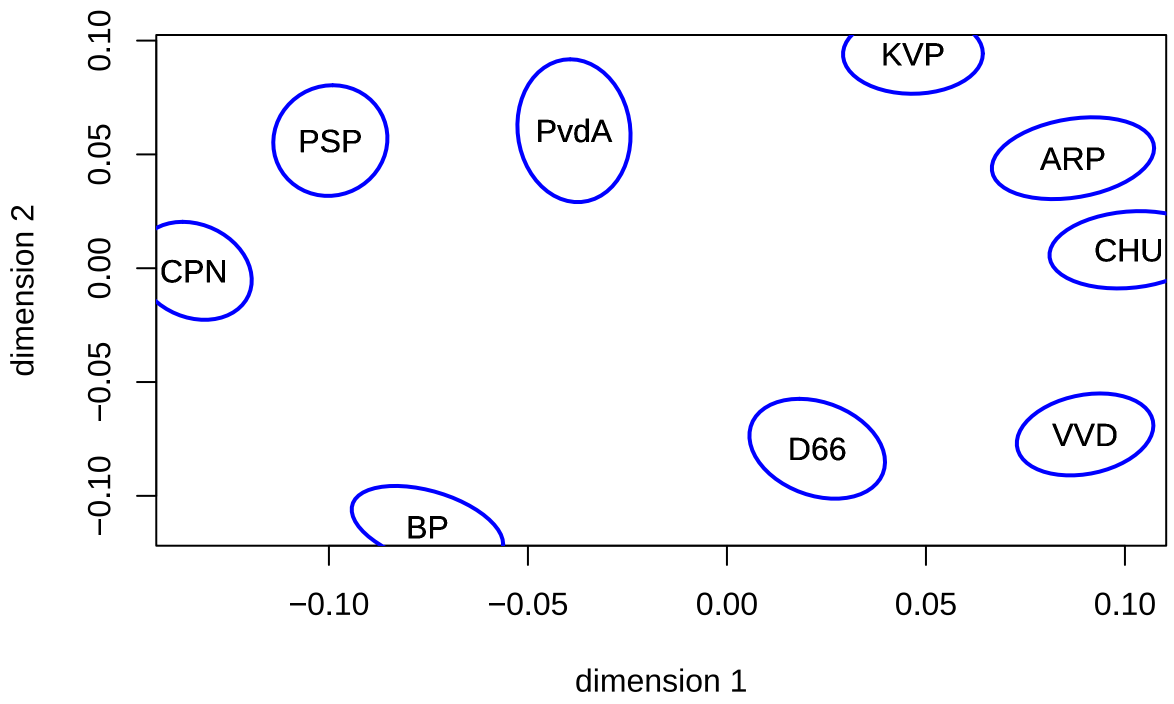

Figure 3: Sensitivity regions for r = 0.5

# 7 Nonmetric MDS

Our main function `newtonMe()` has parameter `nonmetric`, by default `FALSE`, and `ties`, by default `"primary"`. It uses the algorithm from De Leeuw (2016) to perform a monotonic regression after updating the configuration. We start with three runs for $r = \frac{1}{2}$. The first is Newton, the second Newton with majorization, and the third non-metric majorized Newton. Since the data do not have many ties (in fact just one) there is no opportunity to compare primary, secondary, and tertiary.

For the number of iterations in the three runs we find

```
## [1] 123 729 489
```

and for rStress

```
## [1] 0.105596403 0.044603383 0.008436025
```

If we compare the configurations in figure 4 we see how the non-metric solution is less fine-grained than the metric one, although of course the fit is vastly improved.

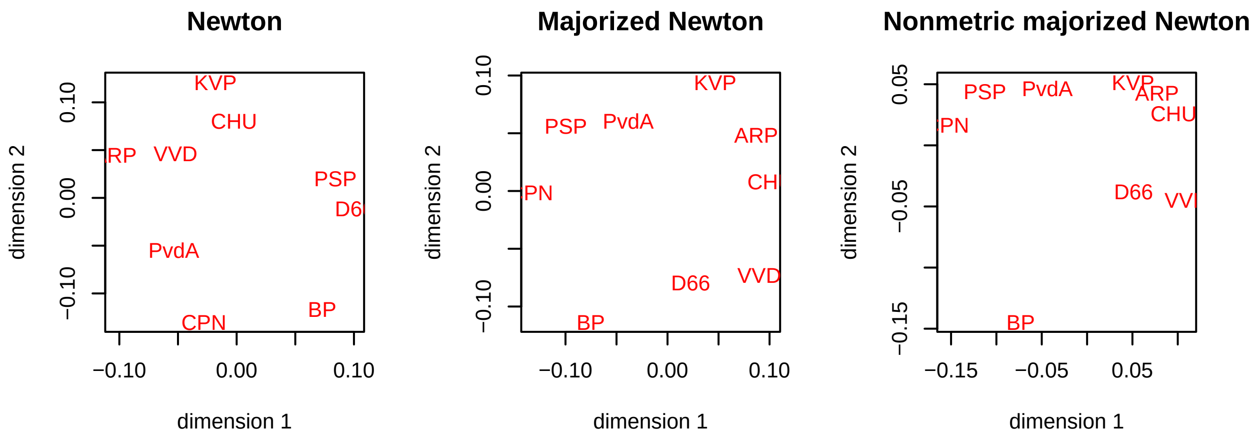

Figure 4: Three solutions for r = 1/2

The Shepard diagram in figure 5 shows the optimal non-metric transformation of the data.

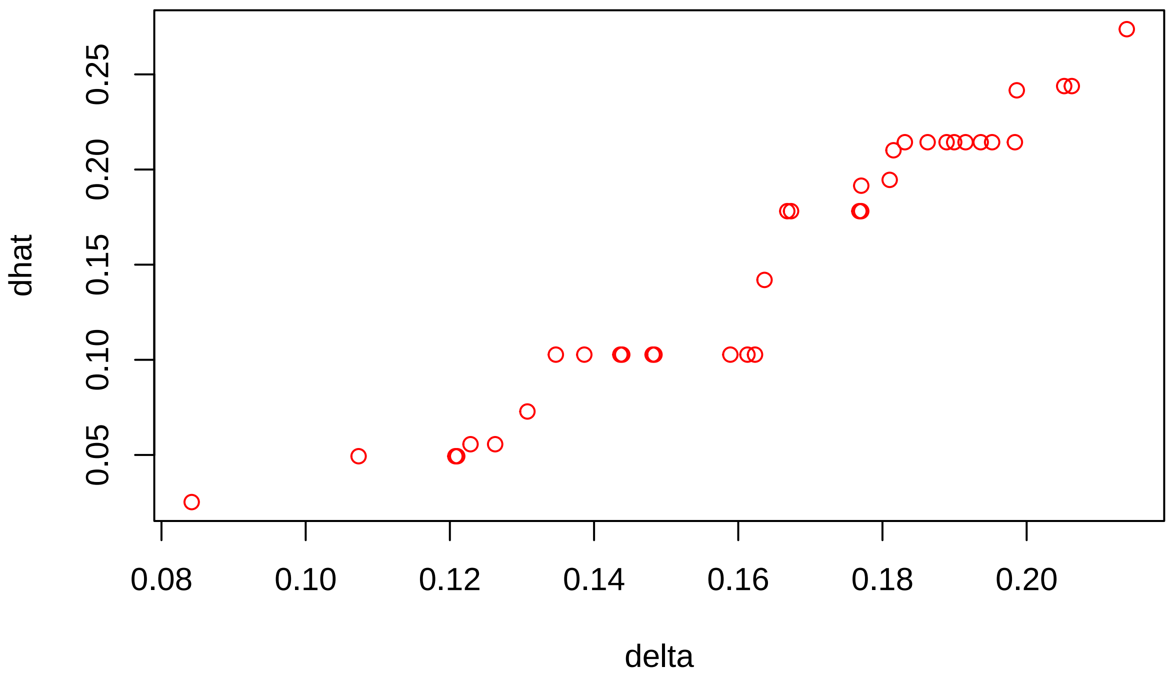

Figure 5: Shepard diagram for non-metric solution

A somewhat more elaborate example uses the Ekman (1954) color data. These have also been analyzed with various values of r in De Leeuw, Groenen, and Mair (2016). The data are well known for their excellent fit and for the very regular circular pattern in the recovered configurations. The data have quite a few ties, the 91 dissimilarities have only 47 unique values.

```
##      434  445  465  472  490  504  537  555  584  600  610  628  651
## 445 0.14
## 465 0.58 0.50
## 472 0.58 0.56 0.19
## 490 0.82 0.78 0.53 0.46
## 504 0.94 0.91 0.83 0.75 0.39
## 537 0.93 0.93 0.90 0.90 0.69 0.38
## 555 0.96 0.93 0.92 0.91 0.74 0.55 0.27
## 584 0.98 0.98 0.98 0.98 0.93 0.86 0.78 0.67
## 600 0.93 0.96 0.99 0.99 0.98 0.92 0.86 0.81 0.42
## 610 0.91 0.93 0.98 1.00 0.98 0.98 0.95 0.96 0.63 0.26
## 628 0.88 0.89 0.99 0.99 0.99 0.98 0.98 0.97 0.73 0.50 0.24
## 651 0.87 0.87 0.95 0.98 0.98 0.98 0.98 0.98 0.80 0.59 0.38 0.15
## 674 0.84 0.86 0.97 0.96 1.00 0.99 1.00 0.98 0.77 0.72 0.45 0.32 0.24
```

We analyze the data for both $r = \frac{1}{2}$ and $r = 1$, with Newton, majorized Newton, and non-metric majorized Newton with both primary and secondary approach to ties. This gives a total of 8 analyses.

Let's look at the results for $r = \frac{1}{2}$ first. Both Newton and Majorized Newton converge to the same solution. The two non-metric solutions have much lower rStress, and take more iterations to converge. But the configurations in figure 6 show all four configurations are basically the same.

```
## Newton:                                iterations:     7  rStress:  0.01721325
## Majorized Newton:                      iterations:    47  rStress:  0.01721325
## Non-metric Majorized Newton (Primary):   iterations:   191  rStress:  0.00053373
## Non-metric Majorized Newton (Secondary): iterations:   115  rStress:  0.00099767
```

Table 4: Ekman Solutions with r = 1/2

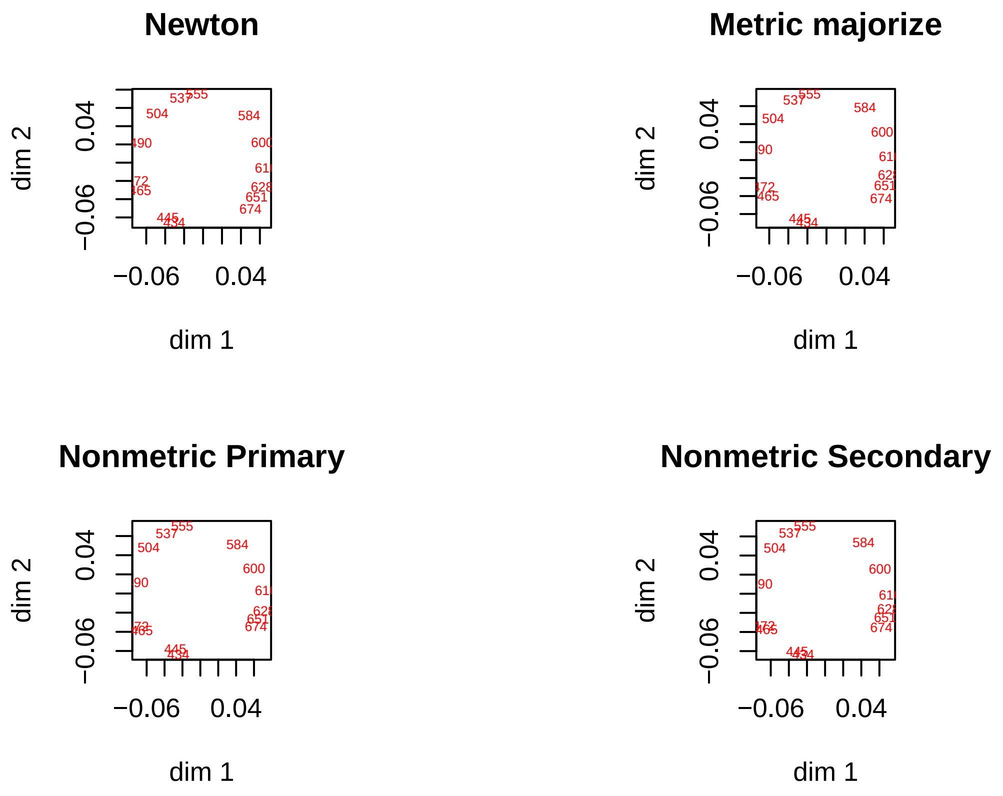

Figure 6: Four Ekman solutions for r = 1/2

Shepard plots for the primary and secondary approach to ties are tight and slightly convex. Again, they differ only in detail.

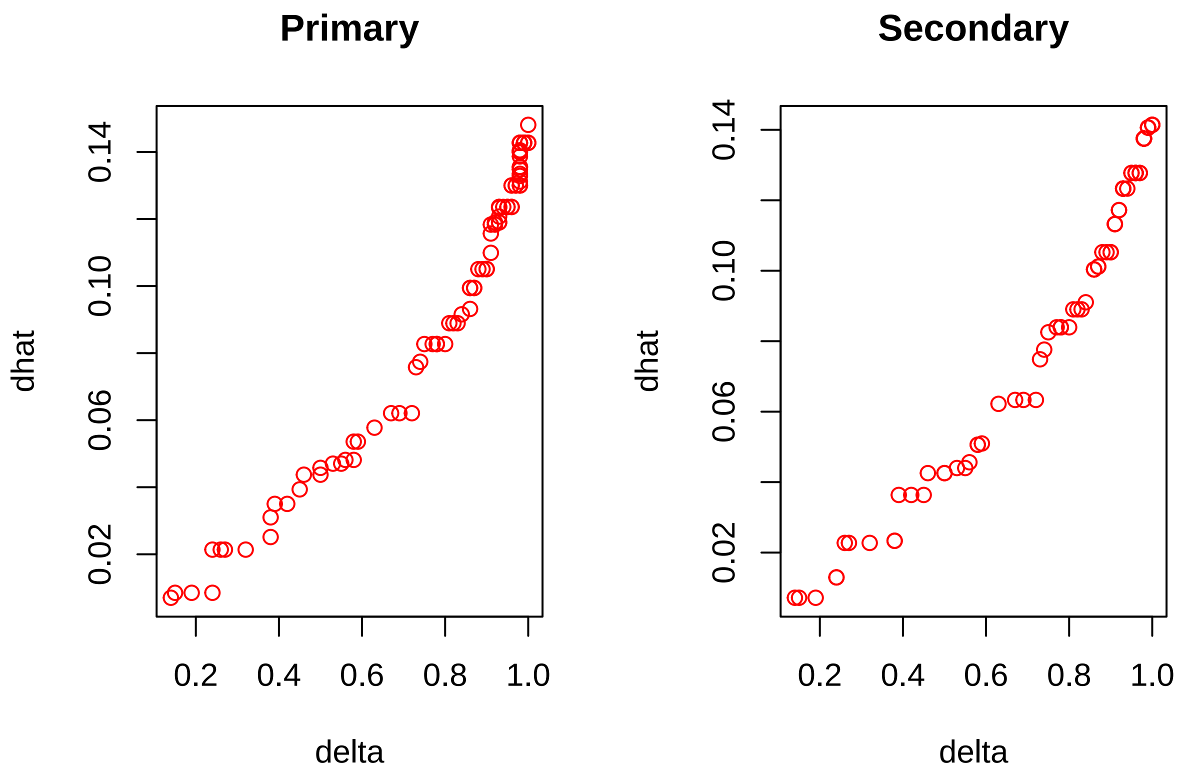

Figure 7: Ekman Shepard Plots r = 1/2

For $r = 1$ Newton converges to a local maximum, with all points in the origin. The other three solutions in figure 8 are basically the same. They do not differ much from the plots for $r = \frac{1}{2}$, maybe exhibit a bit more clustering. The Shepard plots in figure 9 are cnsiderably more convex and non-linear than the ones for $r = \frac{1}{2}$, again indicating clustering (small dissimilarities become smaller after transformation, larger ones become larger).

```
## NA  iterations:      4  rStress:  1.00000000
## NA  iterations:     65  rStress:  0.09306315
## NA  iterations:    281  rStress:  0.00090145
## NA  iterations:    139  rStress:  0.00238525
```

Table 5: Ekman Solutions with r = 1

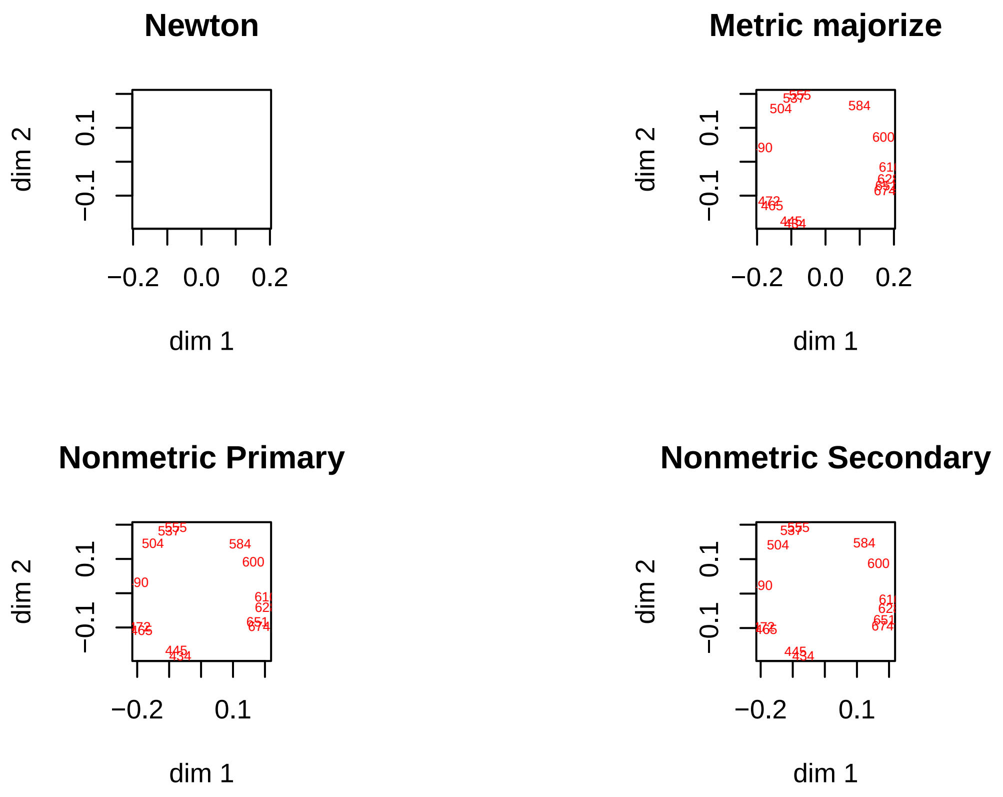

Figure 8: Four Ekman solutions for r = 1

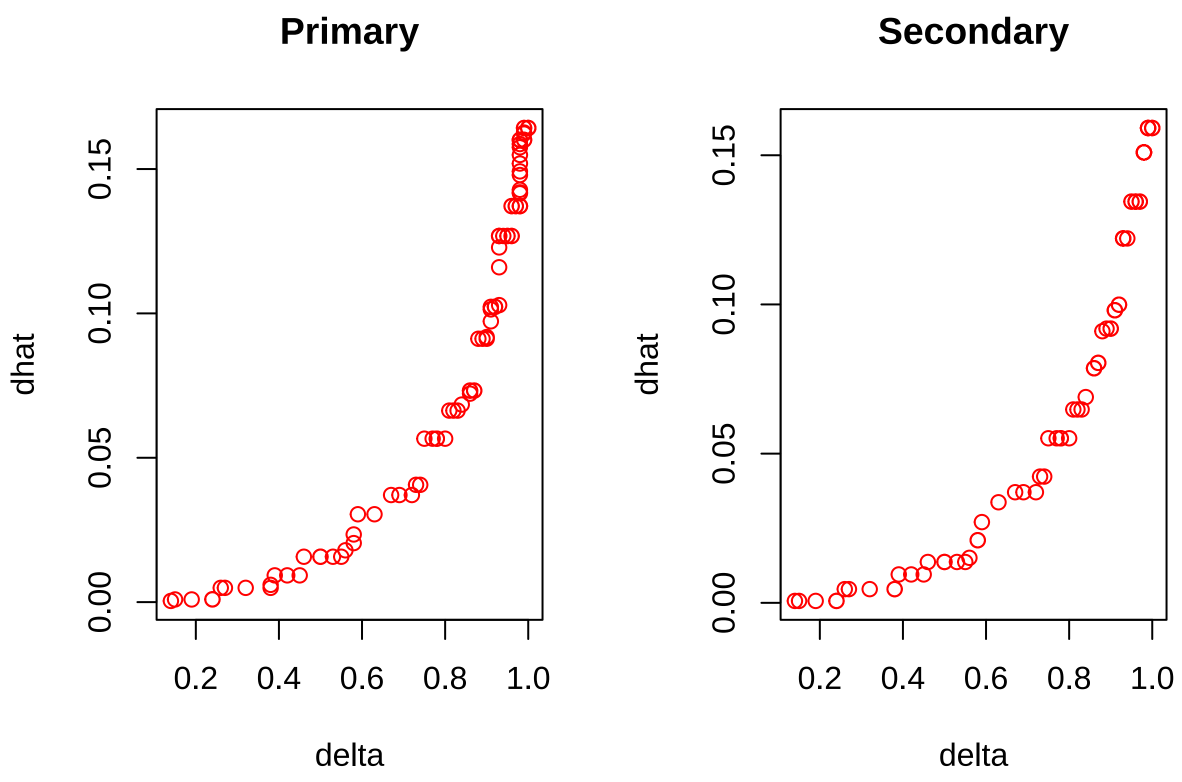

Figure 9: Ekman Shepard Plots r = 1

## 8 Code

```r
library (numDeriv)
library (MASS)

amalgm <- function (x, w = rep (1, length (x))) {
  dyn.load ("pava.so")
  n <- length (x)
  a <- rep (0, n)
  b <- rep (0, n)
  y <- rep (0, n)
  lf <-
    .Fortran (
      "AMALGM",
      n = as.integer (n),
      x = as.double (x),
      w = as.double (w),
      a = as.double (a),
      b = as.double (b),
      y = as.double (y),
      tol = as.double(1e-15),
      ifault = as.integer(0)
    )
  return (lf$y)
}

isotone <-
  function (x,
            y,
            w = rep (1, length (x)),
            ties = "secondary") {
    f <- sort(unique(x))
    g <- lapply(f, function (z)
      which(x == z))
    n <- length (x)
    k <- length (f)
    if (ties == "secondary") {
      w <- sapply (g, length)
      h <- lapply (g, function (x)
        y[x])
      m <- sapply (h, sum) / w
      r <- amalgm (m, w)
      s <- rep (0, n)
      for (i in 1:k)
        s[g[[i]]] <- r[i]
    }
    if (ties == "primary") {
      h <- lapply (g, function (x)
        y[x])
      m <- rep (0, n)
      for (i in 1:k) {
        ii <- order (h[[i]])
        g[[i]] <- g[[i]][ii]
        h[[i]] <- h[[i]][ii]
      }
      m <- unlist (h)
      r <- amalgm (m, w)
      s <- r[order (unlist (g))]
    }
    if (ties == "tertiary") {
      w <- sapply (g, length)
      h <- lapply (g, function (x)
        y[x])
      m <- sapply (h, sum) / w
      r <- amalgm (m, w)
      s <- rep (0, n)
      for (i in 1:k)
        s[g[[i]]] <- y[g[[i]]] + (r[i] - m[i])
    }
    return (s)
  }


rStress <- function (x, w, delta, a, r) {
  n <- length (a)
  s <- 0
  for (i in 1:n) {
    xax <- sum (x * (a[[i]] %*% x))
    s <- s + w[i] * (delta[i] - xax ^ r) ^ 2
  }
  return (s)
}

mdsDerivatives <- function (x, w, delta, a, r, numerical = FALSE) {
  m <- length (x)
  n <- length (a)
  b <- c <- s <- t <- matrix (0, m, m)
  for (i in 1:n) {
    xa <- drop (a[[i]] %*% x)
    xax <- sum (x * xa)
    b <- b + w[i] * delta [i] * (xax ^ (r - 1)) * a[[i]]
    c <- c + w[i] * (xax ^ (2 * r - 1)) * a[[i]]
    s <-
      s + w[i] * delta[i] * (xax ^ (r - 1)) * (a[[i]] + 2 * (r - 1) * outer(xa, xa) / xa
    t <-
      t + w[i] *  (xax ^ (2 * r - 1)) * (a[[i]] + 2 * (2 * r - 1) * outer(xa, xa) / xax)
  }
  gan <- -4 * r * drop ((b - c) %*% x)
  han <- -4 * r * (s - t)
  result <- list (
    b = b,
    c = c,
    s = s,
    t = t,
    gan = gan,
    han = han
  )
  if (numerical) {
    gnu <- grad (
      rStress,
      x,
      w = w,
      delta = delta,
      a = a,
      r = r
    )
    hnu <- hessian (
      rStress,
      x,
      w = w,
      delta = delta,
      a = a,
      r = r
    )
    result <- c (result, list (gnu = gnu, hnu = hnu))
  }
  return (result)
}

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

u <- function (i, n) {
  return (ifelse (i == 1:n, 1, 0))
}

e <- function (i, j, n) {
  d <- u (i, n) - u (j, n)
  return (outer (d, d))
}

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)
```

```
}

makeA <- function (n, p = 2) {
  m <- n * (n - 1) / 2
  a <- list()
  for (j in 1:(n - 1))
    for (i in (j + 1):n) {
      d <- u (i, n) - u (j, n)
      e <- outer (d, d)
      a <- c(a, list (directSum (repList (e, p))))
    }
  return (a)
}

newtonMe <-
  function (delta,
            xini = NULL,
            w = rep (1, length (delta)),
            p = 2,
            r = .5,
            eps = 1e-15,
            itmax = 1000,
            linearize = FALSE,
            nonmetric = FALSE,
            ties = "primary",
            verbose = TRUE) {
    n <- nrow (as.matrix (delta))
    dhat <- delta / sqrt (sum (delta ^ 2))
    if (is.null (xini)) {
      xold <- as.vector (torgerson (dhat, p))
    } else {
      xold <- xini
    }
    a <- makeA (n, p)
    dold <- sapply (a, function (u)
      sum (xold * (u %*% xold)))
    eold <- dold ^ r
    sold <- sum (w * (dhat - eold) ^ 2)
    itel <- 1
    repeat {
      h <- mdsDerivatives (xold, w, dhat, a, r)
      if (linearize) {
        xnew <- drop (xold - ginv (4 * r * h$t) %*% h$gan)
      } else {
```

```
      xnew <- drop (xold - ginv (h$han) %*% h$gan)
    }
    dnew <- sapply (a, function (u)
      sum (xnew * (u %*% xnew)))
    enew <- dnew ^ r
    if (nonmetric) {
      dhat <- isotone (delta, enew, ties = ties)
      dhat <- dhat / sqrt (sum (dhat ^ 2))
    }
    snew <- sum (w * (dhat - enew) ^ 2)
    if (verbose) {
      cat (
        formatC (itel, width = 4, format = "d"),
        formatC (
          sold,
          digits = 10,
          width = 13,
          format = "f"
        ),
        formatC (
          snew,
          digits = 10,
          width = 13,
          format = "f"
        ),
        "\n"
      )
    }
    if ((itel == itmax) || (abs(sold - snew) < eps))
      break
    itel <- itel + 1
    xold <- xnew
    dold <- dnew
    sold <- snew
  }
  return (list (
    x = matrix (xnew, n, p),
    d = dnew,
    dhat = dhat,
    rstress = snew,
    g = h$gan,
    h = h$han,
    itel = itel
  ))
  }
```

## 9 NEWS

001 01/17/16 Sadly incomplete version

002 01/17/16 Added Newton with Majorization

003 01/17/16 Added addional runs, many edits

004 01/18/16 Figures redone, tables redone, discussion added

005 01/18/16 Compare with older algorithm

006 01/18/16 Add sensitivity regions

007 01/18/16 Numerous small corrections

008 01/19/16 Added nonmetric options

009 01/20/16 Added Ekman example

## References

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. 2016. "Exceedingly Simple Isotone Regression with Ties." 2016.

De Leeuw, J., P. Groenen, and P. Mair. 2016. "Minimizing rStress Using Majorization." 2016.

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

Fox, J., and S. Weisberg. 2011. *An r Companion to Applied Regression*. Second Edition. Thousand Oaks, CA: Sage. http://socserv.socsci.mcmaster.ca/jfox/Books/Companion.

Gilbert, P., and R. Varadhan. 2014. *numDeriv: Accurate Numerical Derivatives*. https://R-Forge.R-project.org/projects/optimizer/.

Kruskal, J. B. 1964. "Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis." *Psychometrika* 29: 1–27.

Ramsay, J. O. 1977. "Maximum Likelihood Estimation in Multidimensional Scaling." *Psychometrika* 42: 241–66.

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://www.stat.ucla.edu/~deleeuw/janspubs/1977/articles/takane_young_deleeuw_A_77.pdf.