SQUARED EUCLIDEAN DISTANCE SCALING BY POLYNOMIAL MAJORIZATION

JAN DE LEEUW

ABSTRACT. Meet the abstract. This is the abstract.

1. PROBLEM

In squared distance Euclidean multidimensional scaling we minimize a loss function of the form

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

over the $n \times r$ matrices X. The matrix X is the *configuration* or the *map*, r is the *dimensionality* of the map. The w_{ij} are known positive *weights*, the δ_{ij} are known non-negative *dissimilarities*. The quantity $d_{ij}^2(X)$ is the *squared Euclidean distance* between rows i and j of X.

Our first example are the railway distances in kilometers between four cities in The Netherlands, taken from the EUrail website. They are given below the diagonal in table 1.

Date: Thursday 30th June, 2011 — 16h 19min — Typeset in TIMES ROMAN.

Key words and phrases. Template, LaTeX.

Amsterdam	–	0.103	0.022	0.556
's-Hertogenbosch	59	–	0.023	0.721
Utrecht	27	28	–	0.398
Groningen	137	156	116	–

TABLE 1. Rail Distances

For convenience we rescale the data such that the weighted sum of squares of the squared railway distances is 1. With units weights this gives the elements above the diagonal of table 1.

2. USE OF BASES

We can get rid of rotational and translational indeterminacy parametrizing the configuration as

$$X = \begin{array}{c|cc} \text{A} & 0 & 0 \\ \text{H} & x_1 & x_2 \\ \text{U} & x_3 & 0 \\ \text{G} & x_4 & x_5 \end{array}$$

Amsterdam is in the origin and the line Amsterdam-Utrecht is along the horizontal axis.

We can write this new parametrization in the form

$$X = \sum_{s=1}^{p} x_s U_s. \tag{2}$$

In our railway example $p = 5$ and the U_s are 4×2 unit matrices with all elements equal to zero and only a single element equal to one. In other examples we can have other bases of $n \times p$ matrices for the space of configurations, not necessarily binary or orthogonal. In fact using bases is an effective way to impose linear constraints on the configuration.

Now define matrices E_{ij} of order n which have elements (i,i) and (j,j) equal to $+1$, and elements (i,j) and (j,i) equal to -1. All other elements are zero. Thus $d_{ij}^2(X) = \mathbf{tr}\, XE_{ij}X'$. By using (2) we see that

$$d_{ij}^2(X) = \sum_{s=1}^{p}\sum_{t=1}^{p} x_s x_t \mathbf{tr}\, U_s E_{ij} U_t',$$

and by defining A_{ij} as the $p \times p$ matrix with element (s,t) equal to

$$\mathbf{tr}\, U_s E_{ij} U_t' = \mathbf{tr}\, E_{ij} U_t' U_s$$

we can write $d_{ij}^2(X) = x'A_{ij}x$.

Using our normalizations see that we can fit a subset of K of all possible distances by minimizing

$$\sigma(x) = 1 - 2x'\{\sum_{k=1}^{K} w_k \delta_k A_k\}x + \sum_{k=1}^{K} w_k (x'A_k x)^2. \tag{3}$$

3. Change of Variables

Suppose K are the eigenvectors and Λ^2 is the corresponding diagonal matrix of eigenvalues of $B = \sum_{k=1}^{K} w_k \delta_k A_k$.

Change the variables to $y = \Lambda K' x$. Then

$$\sigma(y) = 1 - 2y'y + \sum_{k=1}^{K} w_k (y'H_k y)^2,$$

where $H_k = \Lambda^{-1}K'A_k K\Lambda^{-1}$, with Λ^{-1} the Moore-Penrose inverse. Note that $\sum_{k=1}^{K} w_k \delta_k H_k = I$.

Now write y in the form $y = \beta z$, with $z'z = 1$. This additional change of variables gives

$$\sigma(z,\beta) = 1 - 2\beta^2 + \beta^4 \sum_{k=1}^{K} w_k (z'H_k z)^2,$$

and

$$\min_{\beta} \sigma(z,\beta) = 1 - \frac{1}{\sum_{k=1}^{K} w_k (z'H_k z)^2},$$

where the minimum is attained for

$$\hat{\beta}^2 = \frac{1}{\sum_{k=1}^K w_k (z'H_k z)^2}.$$

This shows we can also solve our original problem by computing the minimum of

$$\tau(z) = \sum_{k=1}^K w_k (z'H_k z)^2$$

over $z'z = 1$. Then scale the solution to find the optimal y, transform back to find the optimal x, and use the basis to find the optimal X.

4. MAJORIZATION

Optimizing $\tau(z)$ over $z'z = 1$ is done in this paper by majorization [De Leeuw, 1994; Heiser, 1995; Lange et al., 2000]. In De Leeuw and Groenen [2011] a quadratic majorization method is developed which uses bounds on the individual elements of the Hessian to arrive at an upper bound for its spectral norm.

The gradient of τ is

$$g(z) = 4 \sum_{k=1}^K w_k (z'H_k z) H_k z,$$

and the Hessian is

$$H(z) = 4 \sum_{k=1}^n w_k \{(z'H_k z)H_k + 2H_k z z' H_k\}.$$

This gives the elementwise bound

$$h_{st}(z) \leq 4 \sum_{k=1}^K w_k \{\lambda_k |(H_k)_{st}| + 2(u_k u_k')_{st}\},$$

where element s of u_k is the ℓ_2 norm of row s of H_k, and λ_k is the largest eigenvalue of H_k. Thus

$$\|H(z)\| \leq 4 \max_s \sum_t \sum_{k=1}^K w_i \{(z'H_k z)H_i + 2H_k z z' H_k\}. \tag{4}$$

The algorithm is

$$z^{(k+1)} = \frac{z^{(k)} - \frac{1}{L}g(z^{(k)})}{\|z^{(k)} - \frac{1}{L}g(z^{(k)})\|}, \tag{5}$$

with L the bound from (4).

5. EXAMPLES

An implementation in R is given in the Appendix. It is written in a straightforward way, with small examples in mind. In actual applications we can use the sparsity of the usual bases to greatly improve both storage requirements and computational efficiency.

5.1. **Four Cities.** Starting with a $z^{(0)}$ that has all elements equal to $\frac{1}{5}\sqrt{5}$ we need 282 iterations to arrive at a change in function value less than $1E-6$. The minimum loss function value is 0.003089.

6. DISCUSSION

Further research is needed to improve the bound of (4), to efficiently use the sparsity of the bases, and to apply acceleration techniques such as the ones in De Leeuw [2008a,b].

REFERENCES

J. De Leeuw. Block Relaxation Methods in Statistics. In H.H. Bock, W. Lenski, and M.M. Richter, editors, *Information Systems and Data Analysis*, Berlin, 1994. Springer Verlag. URL http://preprints.stat.ucla.edu/131/131.ps.gz.

J. De Leeuw. Polynomial Extrapolation to Accelerate Fixed Point Algorithms. Preprint Series 542, UCLA Department of Statistics, Los Angeles, CA, 2008a. URL http://preprints.stat.ucla.edu/download.php?paper=542.

J. De Leeuw. Accelerating Majorization Algorithms. Preprint Series 543, UCLA Department of Statistics, Los Angeles, CA, 2008b. URL http://preprints.stat.ucla.edu/543/acceleration.pdf.

J. De Leeuw and P. Groenen. Majorizing a Multivariate Polynomial over the Unit Sphere, with Applications. Preprint Series 630, UCLA Department of Statistics, Los Angeles, CA, 2011. URL http://preprints.stat.ucla.edu/630/polymaj.pdf.

W.J. Heiser. Convergent Computing by Iterative Majorization: Theory and Applications in Multidimensional Data Analysis. In W.J. Krzanowski, editor, *Recent Advantages in Descriptive Multivariate Analysis*, pages 157–189. Clarendon Press, Oxford, 1995.

K. Lange, D.R. Hunter, and I. Yang. Optimization Transfer Using Surrogate Objective Functions. *Journal of Computational and Graphical Statistics*, 9:1–20, 2000.

APPENDIX A. CODE

```
data <- matrix(c
    (1,1,1,2,2,3,2,3,4,3,4,4,1,1,1,1,1,1,59,27,137,28,156,116), 6, 4)
data [,4] <- data [, 4] ^ 2
data [,4] <- data [, 4] / sqrt (sum (data [, 3] * data [, 4] ^ 2))

base <- array (0, c(4, 2, 5))

base [2, 1, 1] <- 1
base [2, 2, 2] <- 1
base [3, 1, 3] <- 1
base [4, 1, 4] <- 1
base [4, 2, 5] <- 1

library(smacof)
data(stardist)
stardata <- matrix (0, 7140, 5)
stardata[,1] <- outer(1:120,rep(0,120),"+")[outer(1:120,1:120,">")]
stardata[,2] <- outer(rep(0,120),1:120,"+")[outer(1:120,1:120,">")]
stardata[,3] <- 1
stardata[,5] <- as.vector(stardist)
stardata[,4] <- (stardata[,5]^2)/sqrt(sum(stardata[,5]^4))

starbase <- array (0, c(120, 2, 240))
k <- 1
for (i in 1:120) {
        for (j in 1:2) {
                starbase [i, j, k] <- 1
                k <- k + 1
        }
}
starbase <- starbase[,,c(3,5:240)]

mkCC <- function (base) {
        d <- dim (base)
        m <- d [3]
        n <- d [1]
        cc <- array (0, c(m, m, n * (n - 1) / 2))
        for (s in 1 : m) {
                for (t in 1 : m) {
                        w <- tcrossprod (base [, , s],base [, , t])
                        k <- 1
```

```
                        for (i in 1: (n - 1)) {
                                for (j in (i + 1) : n) {
                                        cc [s, t, k] <- w[i, i] + w[j, j
                                            ] - w[i, j] - w[j, i]
                                        k <- k + 1
                                }
                        }
                }
        }
        return (cc)
}

mkBB <- function (data, cc) {
        m <- dim (cc) [3]
        n <- dim (cc) [1]
        bb <- matrix (0, n, n)
        for (i in 1:m) {
                bb <- bb + data [i, 3] * data[i, 4] * cc[, , i]
        }
        return (bb)
}

mkSS <- function (bb) {
        eb <- eigen (bb)
        return (eb $ vectors %*% diag (1 / sqrt (eb $ values)))
}

mkHH <- function (ss, cc) {
        m <- dim (cc) [3]
        hh <- array (0, dim (cc))
        for (i in 1:m) {
                hh[, , i] <- t (ss) %*% cc[, , i] %*% ss
        }
        return (hh)
}

hessBound <- function (data, hh) {
        m <- dim (hh) [3]
        k <- dim (hh) [1]
        hb <- matrix (0, k, k)
        for (i in 1 : m) {
                hi <- hh [, , i]
                ha <- abs (hi)
```

```
83                  hs <- max (Mod (eigen (hi, only.values = TRUE) $ values)
                        )
84                  print (hs)
85          hr <- sqrt (rowSums (hi ^ 2))
86                  hb <- hb + data [i, 3] * (hs * ha + 2 * outer (hr, hr))
87          }
88          return (max (rowSums (hb)))
89  }
90
91  mds2 <- function (data, base, xold = rep(1, dim (base) [3]), itmax =
        10000, eps = 1e-6, verbose = TRUE) {
92          cc <- mkCC (base)
93          ss <- mkSS (mkBB (data, cc))
94          hh <- mkHH (ss, cc)
95          K <- hessBound (data, hh)
96          xold <- xold / sqrt (sum (xold ^2))
97          fold <- Inf
98          itel <- 1
99          m <- dim (base) [3]
100         k <- dim (data) [1]
101         w <- data [, 3]
102         repeat {
103                 dd <- rep (0,k)
104                 tt <- matrix (0, m, m)
105                 fnew <- 0
106                 for (i in 1 : k) {
107                         dd [i] <- xold %*% hh [, , i] %*% xold
108                         fnew <- fnew + w [i] * dd [i] ^2
109                         tt <- tt + w [i] * dd [i] * hh [, , i]
110                         }
111         if (verbose) {
112             cat("Iteration: ",
113                 formatC (itel, digits = 6, width = 6),
114                 "   f old :   ",
115                 formatC (fold, digits = 10, width = 15, format="f"),
116                 "   f new :   ",
117                 formatC (fnew, digits = 10, width = 15, format="f"),
118                 "    diff :   ",
119                 formatC (fold - fnew, digits = 10, width = 15, format="f
                        "),
120                 "\n")
121         }
122         xnew <- xold - 2 * drop (tt %*% xold) / K
```

```
        xnew <- xnew / sqrt (sum (xnew ^ 2))
                if (((fold - fnew) < eps) || (itel == itmax)) {
                        break
                }
                xold <- xnew
                fold <- fnew
                itel <- itel + 1
                }
                xnew <- xnew * sqrt (fnew)
                stress <- 1 - 1 / fnew
                xnew <- ss %*% xnew
                xconf <- array (0, dim (base) [1:2])
                for (i in 1 : m) {
                        xconf <- xconf + xnew [i] * base [, , i]
                }
                return (list (itel = itel, x = xconf, stress = stress))
}
```

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