AN ITERATIVE MAXIMUM EIGENVALUE METHOD FOR SQUARED DISTANCE SCALING

JAN DE LEEUW

ABSTRACT. Meet the abstract. This is the abstract.

1. INTRODUCTION

The problem we consider in this paper is a *metric multidimensional scaling* or *MDS* problem. In MDS the data are *dissimilarities* between n objects, and we want to map these objects into points of low-dimensional Euclidean space, in such a way that the *distances* between the points are approximately equal to the dissimilarities.

In this paper we translate the MDS objective into a squared error loss function on the squared dissimilarities. Thus we want to minimize

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

Here $W = \{w_{ij}\}$ and $\Delta = \{\delta_{ij}\}$, and $D(X) = \{d_{ij}(X)\}$ are the non-negative symmetric and hollow matrices of order n with, respectively, *weights* and *dissimilarities*. $D(X) = \{d_{ij}(X)\}$ is a symmetric and hollow matrix-valued function *(Euclidean) distances* between the rows of the $n \times p$ *configuration* X.

1.1. **Reindexing.**

Some of the weights may be zero and we want to get rid of the symmetries in the loss function. First some additional notation.

Date: January 20, 2008 — 12h 51min — Typeset in TIMES ROMAN.

2000 *Mathematics Subject Classification.* 00A00.

Key words and phrases. Multidimensional Scaling, Majorization.

Using unit vectors e_i and e_j we can write

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

where $A_{ij} = (e_i - e_j)(e_i - e_j)'$. We get a more compact definition by renumbering the index-pairs (i, j) with non-zero weights as $k = 1, \cdots, m$, where $m \leq \binom{n}{2}$. Then

$$\mathbf{sstress}(X) = \sum_{k=1}^{m} w_k(\delta_k^2 - \mathbf{tr}\, X'A_kX)^2.$$

1.2. **Using a Basis.** It is convenient to choose a basis for the linear space of configurations X that interest us. This could be all of $\mathbb{R}^{n\times p}$, or it could be some subspace. Using such a basis can eliminate indeterminacy due to translation and rotation, and can be used to impose additional restrictions on the configuration [**?**].

Thus we write

$$X = \sum_{s=1}^{p} \theta_s Y_s.$$

In this new parametrization

$$\mathbf{sstress}(\theta) = \sum_{k=1}^{m} w_k(\delta_k^2 - \theta'C_k\theta)^2,$$

where C_k is symmetric and positive semi-definite, of order p, with elements

$$\{C_k\}_{st} = \mathbf{tr}\, Y_s'A_kY_t.$$

1.3. **Linear Transformation.** We can suppose, without loss of generality, that the dissimilarities are scaled in such a way that

$$\sum_{k=1}^{m} w_k\delta_k^2 = 1.$$

If we define

$$B = \sum_{k=1}^{m} w_k\delta_k^2 C_k$$

we find

$$\mathbf{sstress}(\theta) = 1 - 2\theta'B\theta + \sum_{k=1}^{m} w_k(\theta'C_k\theta)^2.$$

Use the spectral decomposition $B = K\Lambda^2K'$. Define $E_k = \Lambda^{-\frac{1}{2}}K'C_kK\Lambda^{-\frac{1}{2}}$ and change variables with $\xi = \Lambda^{\frac{1}{2}}K\theta$. Using these new variables

$$\textbf{sstress}(\xi) = 1 - 2\xi'\xi + \sum_{k=1}^{m} w_k(\xi'E_k\xi)^2.$$

Note that we still have

$$\xi'E_k\xi = \theta'C_k\theta = \textbf{tr}\, X'A_kX = d_k^2(X).$$

1.4. **Use of Homogeneity.** Minimizing loss over ξ is the same as minimizing

$$\textbf{sstress}(\gamma,\zeta) = 1 - 2\gamma + \gamma^2 \sum_{k=1}^{m} w_k(\zeta'E_k\zeta)^2$$

over γ and $\zeta'\zeta = 1$, and this is equivalent to minimizing

$$\rho(\zeta) = \sum_{k=1}^{m} w_k(\zeta'E_k\zeta)^2$$

over $\zeta'\zeta = 1$. Note that ρ is convex in ζ, and homogeneous of order four, because $\rho(\gamma\zeta) = \gamma^4\rho(\zeta)$. In fact, ρ is a non-negative homogeneous polynomial of order four in ζ. Of course ρ attains is global minimum, equal to zero, at zero. For non-zero ζ we have $\rho(\zeta) = 0$ if and only ζ is in the null-space of all E_k, which can only happen if the sum of all E_k is singular,

This is the "canonical form" for our MDS problem. We want to *minimize a weighted sum of squares of positive semi-definite quadratic forms over the unit sphere*. This problem may be of some interest in itself.

2. Basis of Length Two

If there are only two E_k, then we know they can both be diagonalized by a non-singular S. Say $E_1 = S\Omega S'$ and $E_2 = S\Psi S'$. Let $\kappa = S\zeta$. Then

$$\rho(\kappa) = w_1(\kappa'\Omega\kappa)^2 + w_2(\kappa'\Psi\kappa)^2$$

3. MAJORIZATION

3.1. **Algorithm.** By the AM-GM inequality

$$(\zeta'E_k\zeta)(\eta'E_k\eta)=\sqrt{(\zeta'E_k\zeta)^2(\eta'E_k\eta)^2}\leq\frac{1}{2}\{(\zeta'E_k\zeta)^2+(\eta'E_k\eta)^2\},$$

or

$$\frac{1}{2}(\zeta'E_k\zeta)^2\geq(\zeta'E_k\zeta)(\eta'E_k\eta)-\frac{1}{2}(\eta'E_k\eta)^2.$$

It follows that

$$\frac{1}{2}\rho(\zeta)\geq\zeta'H(\eta)\zeta-\frac{1}{2}\rho(\eta).$$

where

$$H(\eta)=\sum_{k=1}^m w_k(\eta'E_k\eta)E_k.$$

Thus an iterative algorithm which updates $\zeta^{(\nu)}$ by choosing for $\zeta^{(\nu+1)}$ the normalized eigenvector, or one of the normalized eigenvectors, corresponding with the largest eigenvalue of $H(\zeta^{(\nu)})$ produces an increasing sequence $\rho^{(\nu)}=\rho(\zeta^{(\nu)})$.

If we want to spend less time in each iteration, and are prepared to use more iterations, then we can update by a single power method iteration, using

$$\zeta^{(\nu+1)}=\frac{H(\zeta^{(\nu)})\zeta^{(\nu)}}{\|H(\zeta^{(\nu)})\zeta^{(\nu)}\|}.$$

In any case, the increasing sequence $\rho^{(\nu)}$ is bounded above on the unit sphere, and thus it is convergent to, say, ρ_∞. Because the eigenvector map is continuous, or at least closed, all accumulation points will be fixed points of the algorithmic map. At an accumulation point ζ_∞ we'll have $H(\zeta_\infty)=\rho_\infty\zeta_\infty$, and in fact ζ_∞ will be the eigenvector associated with the largest eigenvalue ρ_∞ of $H(\zeta_\infty)$.

3.2. **Rate of Convergence.**

APPENDIX A. CODE

```
1  iterSSQQ<-function(e,xi=rnorm(nrow(e)),itmax=1000,
       eps=1e-6,verbose=TRUE) {
2  xi<-xi/sqrt(sum(xi^2)); itel<-1; m<-nrow(e); fold
       <-1e6
3  repeat {
4           u<-apply(e,3,function(x) sum(outer(xi,xi)*x
               ))
5           au<-apply(u*aperm(e),c(2,3),sum)
6           ue<-eigen(au)
7           nu<-ue$vectors[,m]
8           v<-apply(e,3,function(x) sum(outer(nu,nu)*x
               ))
9           av<-apply(v*aperm(e),c(2,3),sum)
10          ve<-eigen(av)
11          xi<-ve$vectors[,m]
12          fnew<-sum(u*v)
13          if (verbose)
14                  cat(" Iter: ",formatC(itel,digits
                       =6,width=6),
15                    " Eigen1: ",formatC(ue$values[m
                         ],digits=6,width=12,format="
                         f"),
16                    " Eigen2: ",formatC(ve$values[m
                         ],digits=6,width=12,format="
                         f"),
17                      " Sumuu: ",formatC(sum(u^2)
                           ,digits=6,width=12,
                           format="f"),
18                      " Sumvv: ",formatC(sum(v^2)
                           ,digits=6,width=12,
                           format="f"),
```

```
                              " Sumuv: ",formatC(fnew,
                                 digits=6,width=12,
                                 format="f"),"\n")
          if ((itel == itmax) || ((fold - fnew) < eps
             )) break()
          itel<-itel+1
          fold<-fnew
          }
          print(ue$values)
          print(ve$values)
return(list(x=xi,y=nu,f=fnew,u=u,iter=itel))
}

make.ee<-function(w,d,y) {
d<-d/sqrt(sum(w*d^2)); m<-length(w)
n<-dim(y)[1]; p<-dim(y)[3]; aa<-makeaa(n)
cc<-array(0,c(p,p,m))
for (j in 1:m) {
          for (s in 1:p) {
                    for (t in 1:p) {
                              cc[s,t,j]<-sum(y[,,s]*(aa
                                 [,,j]%*%y[,,t]))
                              }
                    }
          }
bb<-matrix(0,p,p)
for (i in 1:m) bb<-bb+w[i]*d[i]*cc[,,i]
be<-eigen(bb)
bv<-be$values; bw<-sqrt(ifelse(bv<1e-10,0,1/bv))
by<-be$vectors; bp<-outer(bw,bw)
for (i in 1:m) cc[,,i]<-crossprod(by,cc[,,i]%*%by)
   *bp
bb<-matrix(0,p,p)
for (i in 1:m) bb<-bb+w[i]*d[i]*cc[,,i]
```

```
#print(bb)
return(cc)
}

makeaa<-function (n) {
nn<-n*(n-1)/2; aa<-array(0,c(n,n,nn)); k<-1
for (i in 1:(n-1)) for (j in (i+1):n){
         aa[i,i,k]<-aa[j,j,k]<-1
         aa[i,j,k]<-aa[j,i,k]<--1
         k<-k+1
         }
return(aa)
}

basemat<-function(n) {
a<-matrix(rnorm(n^2),n,n)
a[,1]<-1
return(qr.Q(qr(a))[,-1])
}

base1<-function(n) {
x<-basemat(n)
y<-array(0,c(n,1,n-1))
for (i in 1:(n-1)) y[,,i]<-x[,i]
return(y)
}

base2<-function(n) {
x<-basemat(n)
z<-rbind(0,basemat(n-1))
y<-array(0,c(n,2,(n-1)+(n-2))); k<-1
for (i in 1:(n-1)) {
         y[,1,k]<-x[,i]
         y[,2,k]<-0
```

```
82          k<-k+1
83          }
84  for (j in 1:(n-2)) {
85          y[,1,k]<-0
86          y[,2,k]<-z[,j]
87          k<-k+1
88          }
89  return(y)
90  }
```

REFERENCES

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.

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