# R code -- class example -- Torgerson's classical metric MDS

# Enter original proximity (dissimilarity) ratings as a vector.

# Note they may not satisfy the metric axioms.

prox <-

c(2,

4, 3,

5, 4, 0)

D = matrix(numeric(16),4,4) # define empty 4 x 4 matrix

# the following code puts elements of the lowerhalf vector P into the full matrix D

k<-1

for (i in 2:4)

{ for (j in 1:(i-1))

{ D[i,j] <- prox[k] # plug elements of P into lowerhalf of D

k<-k+1 }}

for (i in 1:4) D[i,i]=0 # put 0s on diagonal

D = D + t(D) # add D to its transpose to make symmetric

# so now symmetry and minimality are satisfied

# Now write R code to check the triangle inequality (TI) for all triples

# of points. Then fix any violations of TI by adding a constant c to all entries.

# m = size of matrix (# of stimuli).

m<-4; minc<-0 # note you can put two R statements on one line - separate them with semicolon

for (k in 3:m)

{ for (j in 2:(k-1))

{ for (i in 1:(j-1))

{ i;j;k

c1<-D[i,j]+D[j,k]-D[i,k]

c2<-D[j,k]+D[i,k]-D[i,j]

c3<-D[i,j]+D[i,k]-D[j,k]

c <- min(c1,c2,c3)

if (c<0) minc<-min(c,minc) }}}

# if minc<0, then the TI is violated, by abs(c)

C<-matrix(numeric(16),4,4) # creates a matrix of size 4 x 4 (with entries = 0)

C<-C+abs(minc) # (matrix + scalar) adds the scalar quantity elementwise to the matrix

D

C

Delta<-D+C

for (i in 1:m) Delta[i,i]=0 # put 0s on diagonal of matrix "Delta"

# Delta now contains "pseudo-distances", i.e. proximities transformed to satisfy the metric axioms

Delta

DeltaSq<-Delta^2

DeltaSq

# now compute the row / col means and the grand mean

aveDsq<-c(1:4)

for (i in 1:m) aveDsq[i]<-mean(DeltaSq[i,])

m<-4

aveDsq

grmean<-mean(aveDsq[])

grmean

# now we can define matrix B\*, the quasi-scalar products matrix

B<-matrix(numeric(16),4,4)

for (i in 1:m)

{ for (j in 1:m)

{ B[i,j] <- -0.5\*(DeltaSq[i,j]-aveDsq[i]-aveDsq[j]+grmean)

}}

B

# now factor matrix B\*; start with eigendecomposition

# Function "eigen" puts eigenvalues into object "values", eigenvectors into "vectors"

# NOTE: we must factor matrix B\* as a \*covariance\* matrix, to use the diagonal values

Bcomp<-eigen(B)

Bcomp

#define principal components (use first two only)

wts<-matrix(numeric(4),2,2)

for (i in 1:2) wts[i,i]<-sqrt(Bcomp$values[i])

wts

evec<-Bcomp$vectors[,1:2]

evec

P<-evec%\*%wts

P

# plot the final 2-dim configuration

plot(P,pch="")

points<-c("a","b","c","d") # prepare to label points

text(P,points)

save1<-Bcomp$values # save the estimated eigenvalues for investigation below

save1

# note that the obtained configuration matrix (X) is close to the plot we used

# to generate the proximity "data".

# Let's check the (linear) fit of the solution

# First, calculate Euclidean distances between points in the 2D solution space.

nprox<-m\*(m-1)/2

dist<-rep(0,nprox)

ij=0

for (i in 1:(m-1))

{ for (j in (i+1):m)

{ ij<-ij+1

for (r in 1:2) { dist[ij]<-dist[ij]+(P[i,r]-P[j,r])^2 }

dist[ij]<-sqrt(dist[ij])

}}

# now calculate the correlation of the model distances with the original proximities

fitsol<-cor(prox,dist)

fitsol

#========= On the Additive Constant problem ================

# however, the large negative eigenvalue (#4) is perhaps problematic. Using a larger additive

# constant will generally increase all the eigenvalue, so it might help. But figuring out the

# best constant to use is the "additive constant problem".

# Below is an demonstration of how increasing the additive constant can "fix" the problem of

# negative eigenvalues.

# However, this "fix" leads to spuriously increasing the apparent dimensionality.

# To understand why, imagine using an additive constant of 100. Then, all the pseudodistances

# (transformed dissimilarities) would be roughly equal. Thus, the derived solution would be

# a "simplex" in (m-1)dimensional space. That is, the solution would be uninformative about

# meaningful structure in the original proximity data.

save\_eig<-matrix(28,4,7)

colnames(save\_eig)<-c("0","0.5","1","1.5","2","2.5","3")

#save\_eig[,1]<-save0

#save\_eig[,2]<-save0.5

#save\_eig[,3]<-save1

#save\_eig[,4]<-save1.5

#save\_eig[,5]<-save2

#save\_eig[,6]<-save2.5

#save\_eig[,7]<-save3

#save\_eig

# 0 0.5 1 1.5 2 2.5 3

#[1,] 16.5664 20.2496 24.1899 28.3848 32.8326 37.5325 42.4840

#[2,] 1.4336 2.5193 3.8530 5.4354 7.2670 9.3480 11.6785

#[3,] 0.0000 0.0000 0.0000 0.5548 1.4004 2.4945 3.8376

#[4,] -0.5000 -0.3938 -0.0429 0.0000 0.0000 0.0000 0.0000