Multidimensional Scaling with Anarchic Distances

Jan de Leeuw

Version 23, December 28, 2016

Abstract

Using anarchic distances means using a different configuration for each dissimilarity. We give the anarchic version of the smacof majorization algorithm, and apply it to additive constants, individual differences, and scaling of asymmetric dissimilarities.

Contents

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

1 Introduction

We start with a ridiculously general MDS problem: each distance d_{ij} is computed in its own configuration X_{ij}. It is possible that $X_{ij} \neq X_{ji}$, although we can suppose without loss of generality that $X_{ii} = 0$ for all i. It is also possible that the X_{ij} have different numbers of columns. Use $\mathcal{X}$ for the set of all X_{ij}.

Using standard notation we write

$$d_{ij}^2(X_{ij}) := \mathbf{tr}\ X_{ij}'A_{ij}X_{ij}, \tag{1}$$

with $A_{ij} = (e_i - e_j)(e_i - e_j)'$, where e_i and e_j are unit vectors (one element equal to one, the rest is zero). Not that we could also compute $d_{k\ell}(X_{ij})$ and $d_{ij}(X_{kl})$ but these cross-distances play no role in our techniques.

The *stress* loss function is

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

where the w_{ij} and δ_{ij} are non-negative weights and dissimilarities. We do not assume that $W = \{w_{ij}\}$ and $\Delta = \{\delta_{ij}\}$ are symmetric, but we assume without loss of generality that they are *hollow*, i.e. $w_{ii} = \delta_{ii} = 0$ for all i. Minimizing stress over all possible $\mathcal{X}$ does not make sense, because we can always attain zero stress by choosing the X_{ij} such that $d_{ij}(X_{ij}) = \delta_{ij}$. Thus the problem only becomes interesting if there are restrictive constraints on the X_{ij} that prohibit these trivial solutions.

1.1 smacof

Overviews of the `smacof` theory and algorithms for MDS are in De Leeuw (1977), De Leeuw and Heiser (1977), De Leeuw and Heiser (1980), Borg and Groenen (2005), De Leeuw and Mair (2009), Groenen and Van de Velden (2016). `smacof` stands for *scaling by majorizing a convex function*. We adapt the notation and results to our ridiculously general situation. For convenience assume that the dissimilarities are normalized so that the weighted sum of squares is equal to one. Expand stress as

$$\sigma(\mathcal{X}) = 1 - 2\sum_{i=1}^n\sum_{j=1}^n w_{ij}\delta_{ij}\ \sqrt{\mathbf{tr}\ X_{ij}'A_{ij}X_{ij}} + \sum_{i=1}^n\sum_{j=1}^n w_{ij}\ \mathbf{tr}\ X_{ij}'A_{ij}X_{ij}.$$

We now apply majorization (a.k.a. MM) theory (De Leeuw (1994), Lange (2016)). Suppose we have two sets of configuration matrices $\mathcal{X}$ and $\mathcal{Y}$. Define the auxilary quantities

$$\xi_{ij}(X) := \begin{cases} \frac{\delta_{ij}}{d_{ij}(X)} & \text{if } d_{ij}(X) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Note that $d_{ij}(\xi_{ij}(X)X) = \delta_{ij}$ for every X. Thus $\xi_{ij}(X)$ scales a configuration X in such a way that the distance between points i and j is δ_{ij}.

An application of Cauchy-Schwarz gives the majorization inequality

$$\sigma(\mathcal{X}) \leq 1 - 2\sum_{i=1}^n\sum_{j=1}^n w_{ij}\xi_{ij}(Y_{ij})\mathbf{tr}\ X_{ij}'A_{ij}Y_{ij} + \sum_{i=1}^n\sum_{j=1}^n w_{ij}\mathbf{tr}\ X_{ij}'A_{ij}X_{ij}. \tag{3}$$

The application of Cauchy-Schwarz to define the majorization function is what gives `smacof` its name. If we define

$$\rho(\mathcal{X}, \mathcal{Y}) := \sum_{i=1}^n\sum_{j=1}^n w_{ij}\ \mathbf{tr}\ (X_{ij} - \xi_{ij}(Y_{ij})Y_{ij})'A_{ij}(X_{ij} - \xi_{ij}(Y_{ij})Y_{ij}),$$

the majorization inequality (3) can also be written as $\sigma(\mathcal{X}) \leq \rho(\mathcal{X}, \mathcal{Y})$ for all $\mathcal{X}$ and $\mathcal{Y}$, while $\sigma(\mathcal{X}) = \rho(\mathcal{X}, \mathcal{X})$ for all $\mathcal{X}$.

A `smacof` iteration in this general context is

$$\mathcal{X}^{(k+1)} = \underset{\mathcal{X}}{\mathbf{argmin}}\, \rho(\mathcal{X}, \mathcal{X}^{(k)}) \tag{4}$$

Of course there will be constraints on $\mathcal{X}$, otherwise the solutions is completely trivial and arbitrary.

1.2 Single Configuration Case

The minimization problem gets a more familiar look if we consider the case in which all X_{ij} are constrained to be equal to a single configuration X. Then

$$X^{(k+1)} = \underset{X}{\mathbf{argmin}} \sum_{i=1}^n\sum_{j=1}^n w_{ij}\ \mathbf{tr}\ (X - \xi_{ij}(X^{(k)})X^{(k)})'A_{ij}(X - \xi_{ij}(X^{(k)})X^{(k)}),$$

which has the solution, if there are no further constraints on X,

$$X^{(k+1)} = V^+B(X^{(k)})X^{(k)}, \tag{5}$$

where

$$V := \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\ A_{ij} \tag{6}$$

and

$$B(X^{(k)}) := \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\xi_{ij}(^{(k)})A_{ij}. \tag{7}$$

Equation (5) defines what is known as the *Guttman transform* of $X^{(k)}$. The `smacof` algorithm iteratively applies Guttman transforms to the configuration matrix. De Leeuw (1988) shows linear convergence of these iterations, and De Leeuw (1984) shows that distances are non-zero at a local minimum.

2 Additive Components

We will first look at the special case in which the X_{ij} can be partitioned into a common configuration and a parametric scalar component. Thus $X_{ij} = \begin{bmatrix} X & | & \frac{1}{2}\sqrt{2}f_{ij}(\theta)I \end{bmatrix}$. The functions f_{ij} are arbitrary, but we will discuss some interesting special cases.

As a consequence $d_{ij}^2(X_{ij}) = d_{ij}^2(X) + f_{ij}^2(\theta)$. We have the partitioning

$$\rho(\mathcal{X}, \mathcal{X}^{(k)}) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\ \mathbf{tr}\ (X-\xi_{ij}(X_{ij}^{(k)})X^{(k)})'A_{ij}(X-\xi_{ij}(X_{ij}^{(k)})X^{(k)})+\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(f_{ij}(\theta)-\xi_{ij}(X_{ij}^{(k)})f_{ij}(\theta^{(k)}))^2,$$

where

$$\xi(X_{ij}^{(k)}) = \frac{\delta_{ij}}{\sqrt{d_{ij}^2(X^{(k)}) + f_{ij}^2(\theta^{(k)})}}$$

As a consequence a `smacof` step to update the configuration is simply

$$X^{(k+1)} = V^+ B(X^{(k)})X^{(k)},$$

with

$$B(X^{(k)}) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\xi(X_{ij}^{(k)})A_{ij}.$$

In the same way we update the additive part with

$$\xi^{(k+1)} = \underset{\theta}{\mathbf{argmin}} \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\ (f_{ij}(\theta) - \xi(X_{ij}^{(k)})f_{ij}(\theta^{(k)}))^2.$$

Of course the neat part is that the two additive components can be optimized separately.

3 Applications

In the simplest case the additive components f_{ij} are constants, not dependent on parameters that have to be estimated. This is analyzed in our first examples. We then analyze some more complicated cases, and even some cases in which we do not have additive components, although the general formulation still applies.

3.1 Regularized MDS

One problem with MDS is that there are configurations in which points coincide, and thus distances are zero. If $w_{ij}\delta_{ij} > 0$ then De Leeuw (1984) shows that at a local minimum $d_{ij}(X) > 0$. But this does not cover cases in which weights or dissimilarities are zero, or configurations which are not local minima. The problem can be avoided by choosing all f_{ij} equal to a small ϵ. The `smacof` iterations are Guttman transforms with

$$B(X^{(k)}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij} \ \frac{\delta_{ij}}{\sqrt{d_{ij}^2(X^{(k)}) + \epsilon^2}} A_{ij}.$$

It is interesting to look at the effect of this regularization on unidimensional scaling (De Leeuw (2005)). In unidimensional scaling we have

$$B(x^{(k)})x^{(k)} = \sum_{i=1}^n \sum_{j=1}^n w_{ij} \ \frac{\delta_{ij}}{|x_i^{(k)} - x_j^{(k)}|}(e_i - e_j)(x_i^{(k)} - x_j^{(k)}) = \sum_{i=1}^n \sum_{j=1}^n w_{ij} \ \delta_{ij} \mathbf{sign}(x_i^{(k)} - x_j^{(k)})(e_i - e_j),$$

which simplifies to $B(x^{(k)})x^{(k)} = \tau^{(k)}$, where

$$\tau_i^{(k)} = \sum_{j=1}^n (w_{ij}\delta_{ij} + w_{ji}\delta_{ji})\mathbf{sign}(x_i^{(k)} - x_j^{(k)}).$$

Thus the update only depends on the rank order of the coordinates, and the algorithm converges in a finite number of iterations. This is no longer true if we regularize with a small ϵ.

We illustrate with the Guilford vegetable data from the `psych` package (Revelle (2015)). These are paired comparison proportions p_{ij}, and according to Thurstone's Case V model $\Phi^{-1}(p_{ij}) \approx x_i - x_j$, which Φ the standard normal distribution function. Thus $|\Phi^{-1}(p_{ij})|$ should give us approximate unidimensional distances.

There are five different analyses, with five different values of ϵ. The solutions, plotted in figure 1, as basically indistinguishable, until ϵ gets fairly large. The solution with $\epsilon = 0$ is plotted as a red line. Increasing ϵ flattens the scale by drawing in the extremes, which is of course what regularization is supposed to do.

```
## f: 0.000  itel =    3 stress:    1.40614364
## f: 0.001  itel =    4 stress:    1.40613401
## f: 0.010  itel =    5 stress:    1.40518700
```

```
## f: 0.100  itel =    8 stress:    1.33982251
## f: 0.250  itel =   13 stress:    1.33907623
## f: 0.500  itel =   15 stress:    3.08078523
```

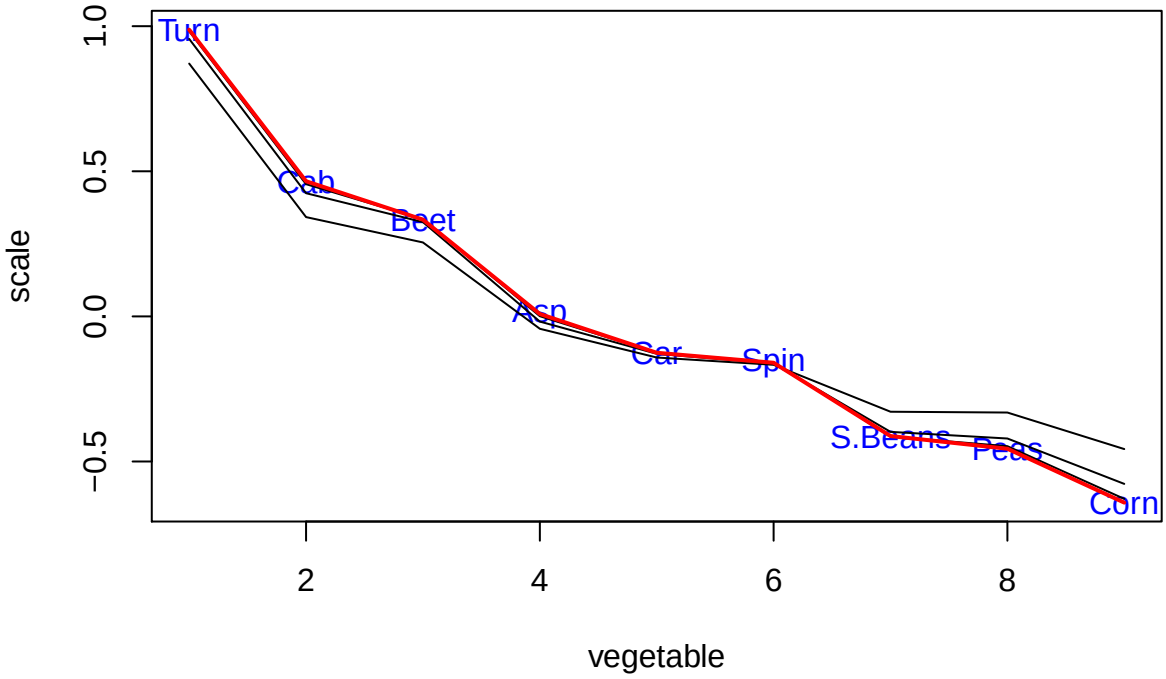

Figure 1: Guilford Vegetable Data

3.2 Nested MDS

In principal component analysis and classical MDS solutions for different dimensionalities are *nested*, in the sense that the p-dimensional solution defines the first p dimensions of the $p+1$-dimensional solution. One possible way around this problem is to define the p-dimensional solution as the first p principal components of the full-dimensional MDS solution (De Leeuw, Groenen, and Mair (2016)). But we can also use additive components to go from the p-dimensional to the $p+1$-dimensional solution. Simply set the f_{ij} equal to the distances of the p-dimensional configuration, and use

$$B(X^{(k)}) = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\ \xi_{ij}(X_{ij}^{(k)})A_{ij},$$

with

$$\xi_{ij}(X_{ij}^{(k)}) = \frac{\delta_{ij}}{\sqrt{(x_i^{(k)} - x_j^{(k)})^2 + f_{ij}^2}}.$$

to compute the next dimension.

As an illustration we use data from De Gruijter (1967), with average dissimilarity judgments between Dutch political parties in 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
```

The two-dimensional successive solution is in figure 2. It has stress 232.3973565646. The simultaneous two-dimensional solution is in figure 3, with stress 128.8832581227.

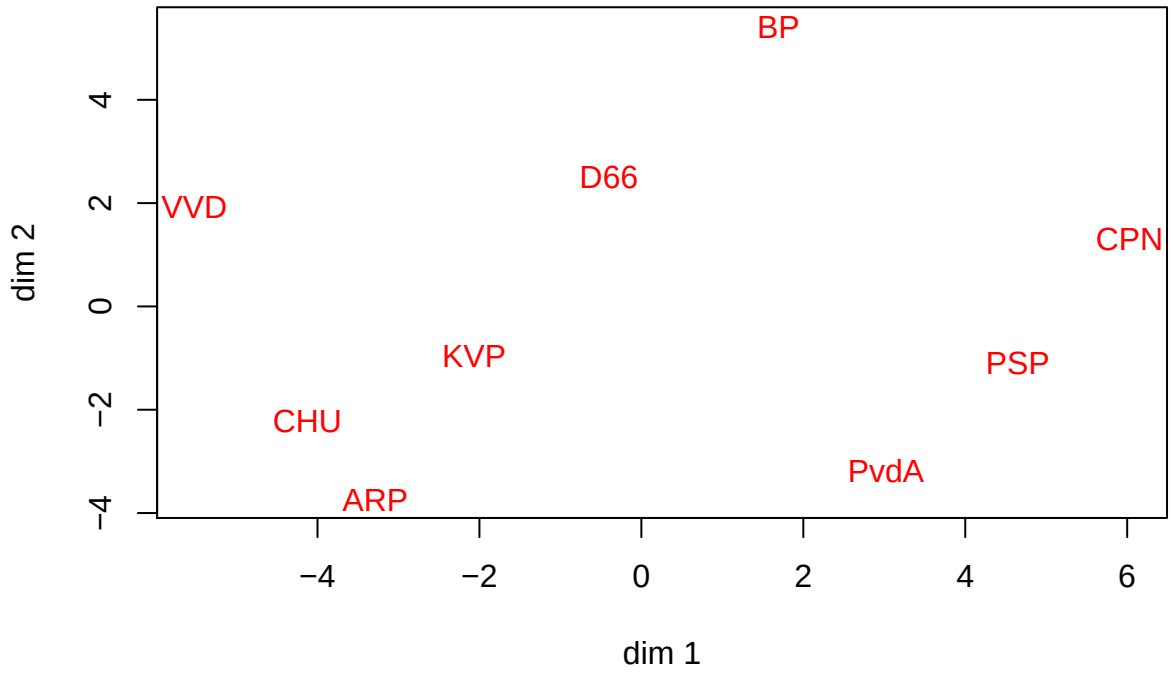

Figure 2: De Gruijter Data, Successive

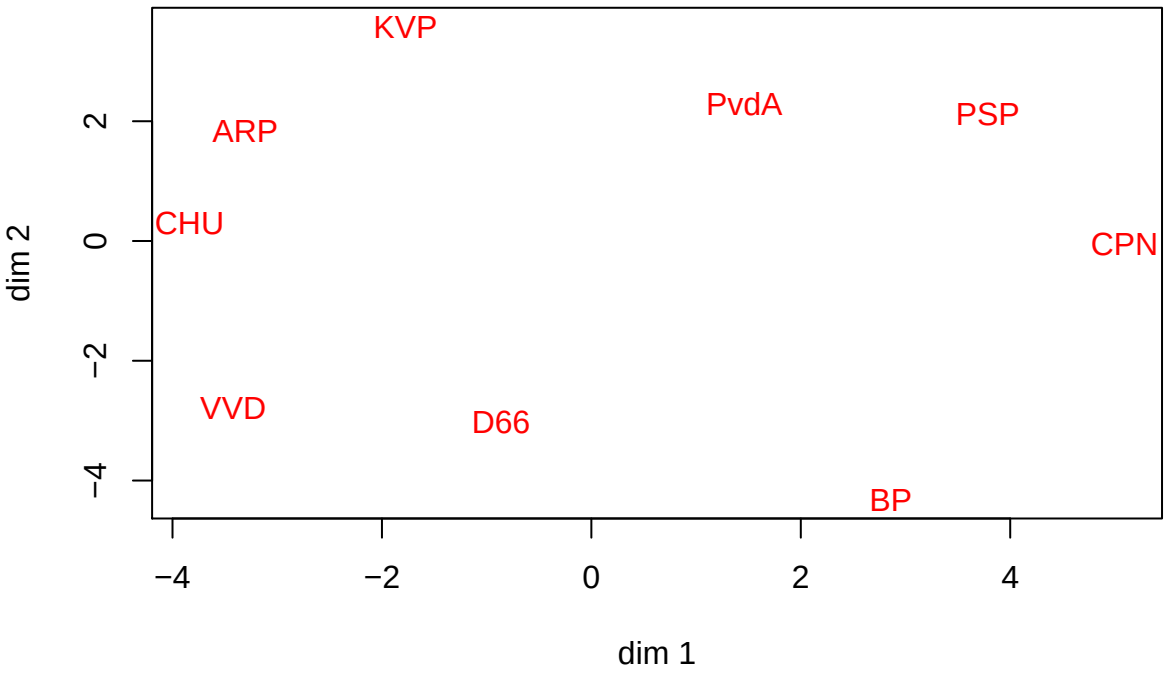

Figure 3: De Gruijter Data, Simultaneous

3.3 Additive Constant

In regularized MDS ϵ is a known constant. We can also design a version in which ϵ is estimated. The iteration for updating ϵ turns out to be

$$\epsilon^{(k+1)} = \epsilon^{(k)} \frac{\sum_{i=1}^n \sum_{j=1}^n w_{ij} \xi_{ij}(X_{ij}^{(k)})}{\sum_{i=1}^n \sum_{j=1}^n w_{ij}},$$

with

$$\xi_{ij}(X_{ij}^{(k)}) = \frac{\delta_{ij}}{\sqrt{d_{ij}^2(X^{(k)}) + \{\epsilon^{(k)}\}^2}}.$$

The De Gruijter data are suitable for illustrating the additive constant algorithm, because they are averaged dissimilarity judgments over a large number of students, and thus they are likely to exhibit regression to the mean.

The configuration

```
##              [,1]          [,2]
## KVP   1.795076832 -1.4883295209
## PvdA -1.449331623 -1.3662756501
## VVD   2.675413029  1.2333253971
## ARP   2.040024685 -1.3928429875
## CHU   2.317379999 -0.9477989531
## CPN  -3.992198052 -0.4130215421
## PSP  -2.954611015 -0.9587289598
## BP   -1.844521956  3.5848238952
```

```
## D66    1.412768101   1.7488483213
```

is given in figure 4. It requires 175 iterations and has stress 16.2605927675. Allowing estimation of an additive constant does not change the configuration very much, but stress is much lower than before. We do see the three Christian Democrat parties KVP, ARP, and CHU moving much closer together.

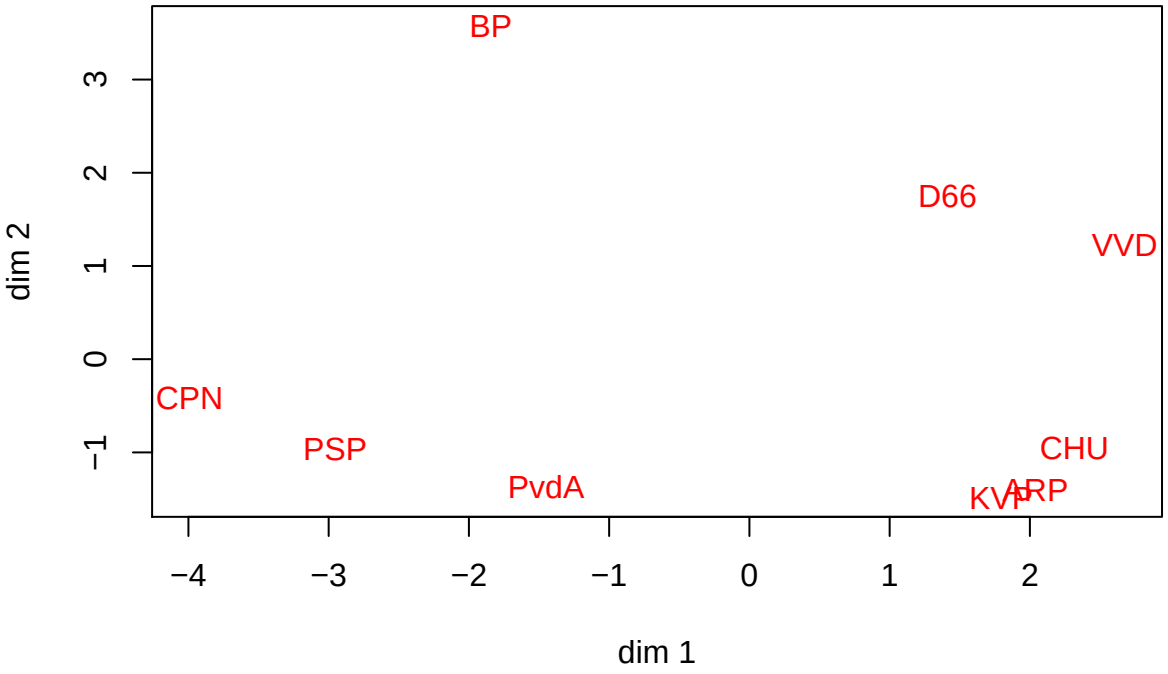

Figure 4: De Gruijter Data, Additive Constant

3.4 MDS with Restrictions

Our examples so far can also be analyzed using the framework of De Leeuw and Heiser (1980). If all X_{ij} are the same

$$\rho(X, X^{(k)}) = \mathbf{tr}\ X'VX - 2\mathbf{tr}\ X'B(X^{(k)})X^{(k)} + \mathbf{tr}\ (X^{(k)})'H(X^{(k)})X^{(k)},$$

with

$$H(X^{(k)}) := \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}\xi_{ij}^2(X^{(k)})A_{ij}.$$

Thus a `smacof` step minimizes

$$\mathbf{tr}\ (X - V^+B(X^{(k)})X^{(k)})'V(X - V^+B(X^{(k)})X^{(k)}).$$

Thus if there are restrictions on the possible configurations, say $X \in \Omega$, then we first compute the Guttman transform and then its (weighted) least squares projection on the constraint set Ω.

For all problems in which the common configuration is partitioned into two parts $\begin{bmatrix} X_1 & | & X_2 \end{bmatrix}$ we have separation of the loss components

$$\rho(X, X^{(k)}) = \mathbf{tr}\ (X_1 - V^+B(X^{(k)})X_1^{(k)})'V(X_1 - V^+B(X^{(k)})X_1^{(k)}) + \mathbf{tr}\ (X_2 - V^+B(X^{(k)})X_2^{(k)})'V(X_2 - V^+B(X^{(k)})X_2^{(k)}).$$

For the regularized MDS and additive constant problems X is of the form $\begin{bmatrix} X_1 & | & \epsilon I \end{bmatrix}$, for nested MDS we have $\begin{bmatrix} X_1 & | & X_2 \end{bmatrix}$, with X_1 the fixed lower-dimensional solution. Thus for the regularized MDS problem and for the additive constant problem $X_1^{(k+1)} = V^+B(X^{(k)})X_1^{(k)}$, and for the additiive constant problem

$$\epsilon^{(k+1)} = \frac{\mathbf{tr}\ VV^+B(X^{(k)})X_2^{(k)}}{\mathbf{tr}\ V} = \epsilon^{(k)}\frac{\mathbf{tr}\ B(X^{(k)})}{\mathbf{tr}\ V}.$$

For nested MDS simply $X_2^{(k+1)} = V^+B(X^{(k)})X_2^{(k)}$.

4 Partitioned Dissimilarities

4.1 Piecewise MDS

Consider the following situation. The dissimilarities are partitioned into, say, m groups, and the configurations X_ℓ are constrained as $X_\ell = X\Lambda_\ell$ with Λ_ℓ diagonal. The groups of dissimilarities are arbitrary. Perhaps dissimilarities come from different sources, or have different degrees of precision.

We shall treat the general case in which both X and the Λ_ℓ must be estimated. The case in which the Λ_ℓ are fixed and known is included as a special case in the general treatment. In iteration k we must minimize

$$\sum_{\ell=1}^m \sum_{(i,j)\in\mathcal{I}_\ell} w_{ij}\ \mathbf{tr}\ (X\Lambda_\ell - \xi_{ij}(X_\ell^{(k)})X_\ell^k)'A_{ij}(X\Lambda_\ell - \xi_{ij}(X_\ell^{(k)})X_\ell^k), \tag{8}$$

which becomes

$$\sum_{\ell=1}^m \mathbf{tr}\ \Lambda_\ell X'V_\ell X\Lambda_\ell - 2\sum_{\ell=1}^m \mathbf{tr}\ \Lambda_\ell X'B_\ell(X_\ell^{(k)})X_\ell^{(k)} + \sum_{\ell=1}^m \mathbf{tr}\ (X_l^{(k)})'H_\ell X_l^{(k)}.$$

The stationary equations are

$$\sum_{\ell=1}^m V_\ell X\Lambda_\ell^2 = \sum_{\ell=1}^m B_\ell(X_\ell^{(k)})X_\ell^{(k)}\Lambda_\ell \tag{9}$$

and, in the case we are also minimizing over Λ_ℓ,

$$\Lambda_\ell = \frac{\mathbf{diag}(X'B_\ell(X_\ell^{(k)})X_\ell^{(k)})}{\mathbf{diag}(X'V_\ell X)} \tag{10}$$

There is little hope to solve these two equations analytically, so we resort to alternating least squares. It is easy enough to solve (10) for the Λ_ℓ for given X. To solve (9) for X for given Λ_ℓ consider column s from both sides of the equation. This gives

$$x_s = \left[\sum_{\ell=1}^{m} \Lambda_{ls}^2 V_\ell\right]^+ u_s,$$

where u_s is column s of the matrix on the right-hand side of (9). Note that if at some point a diagonal element of Λ_ℓ is zero, it will remain zero in subsequent iterations. Thus we can impose zeroes in the Λ_ℓ by fiLambdang them in the initial estimate.

The De Gruijter data show the familar bending or horseshoe effect, where the left right scale is bended to bring the extreme left and right closer than they would otherwise be. We first analyze the data with the `psmacof` function, using a single common dimension and with another single dimension for the largest dissimilarities. The cutoff is the median of the δ_{ij}, which is 6.35. Thus there are two groups: for the smaller dissimilarities below the median both only the first dimension gets unit weight, for the larger dissimilarities both dimensions have unit weight. The Λ_ℓ are fixed, and there is no need to minimize over them.

After 143 iterations we convergence on the configuration

```
##                [,1]          [,2]
## KVP    1.0813664617 -2.2452423226
## PvdA  -2.5518200495 -1.2956894362
## VVD    4.0227369242 -0.2114471130
## ARP    2.6197105314 -1.3322490334
## CHU    4.9836790756  0.5617603512
## CPN   -5.6188008950  1.2897569555
## PSP   -3.8704005788  1.4107253694
## BP     0.5384787791  5.2408076168
## D66   -1.2049502488 -3.4184223877
```

which is plotted in figure 5 and has stress 267.5183648725.

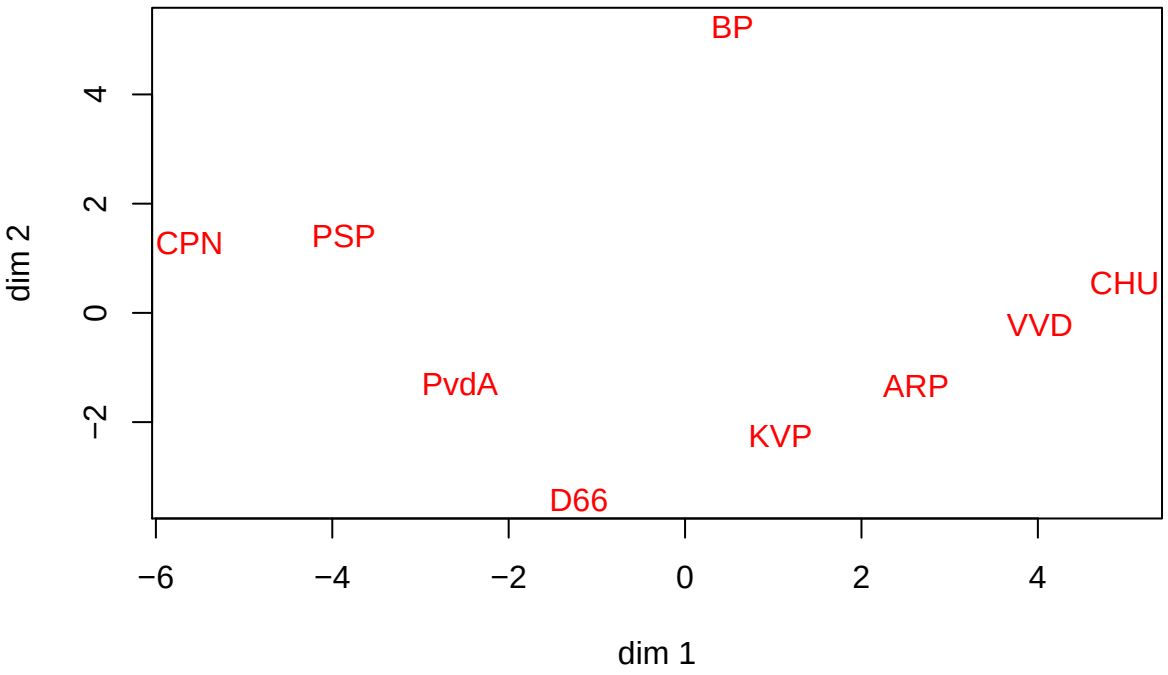

Figure 5: De Gruijter Data, Piecewise 1 + 1

In the next `psmacof` analysis, of the same data, we have two common dimensions and one extra dimension for the large dissimilarities. We also allow optimization over the Λ_ℓ.

The configuration is

```
##               [,1]          [,2]          [,3]
## KVP   0.7186125933 -3.7503571612  3.8641674845
## PvdA -1.8106869909 -2.1504566111 -0.4253679371
## VVD   2.0428672942  2.3164246876 -3.8180733343
## ARP   2.5691370975 -2.8077967715 -0.2075461529
## CHU   3.1007780979 -0.9687218503 -1.9774713207
## CPN  -2.8390433755  2.5465240162  3.0219949858
## PSP  -4.7186654342 -0.3376902107  0.5033059889
## BP    1.4759116538  3.1106773394  3.1092401795
## D66  -0.5389109361  2.0413965617 -4.0702498937
```

and we plot it in figure 6. It requires 1815 iterations and has stress 27.6841203481.

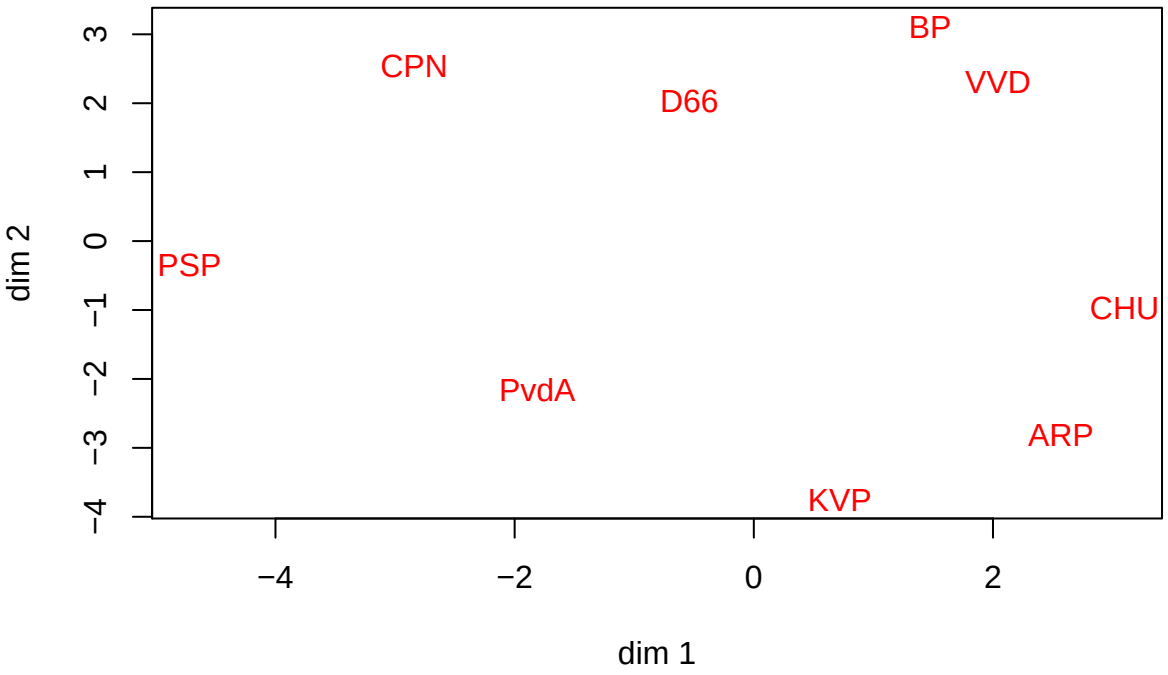

Figure 6: De Gruijter Data, Piecewise 2 + 1

The algorithm becomes only slightly more complicated if we do not require the Λ_ℓ to be diagonal, but allow them to be full matrices.

4.2 INDSCAL/PARAFAC

An interesting special case of (8), where the dissimilarity groups are defined by different individuals or occasions or time-points, is

$$\sum_{\ell=1}^{m}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij\ell}\ \mathbf{tr}\ (X\Lambda_\ell - \xi_{ij\ell}(X_\ell^{(k)})X_\ell^k)'A_{ij}(X\Lambda_\ell - \xi_{ij\ell}(X_\ell^{(k)})X_\ell^k),$$

with

$$\xi_{ij\ell}(X_\ell) = \frac{\delta_{ij\ell}}{d_{ij}(X_\ell)}$$

Now piecewise MDS with m groups is the `smacof` algorithm for INDSCAL, in which we optimize over the Λ_ℓ and use the same common configuration for each group. Again, this is easily extended to matrices Λ_ℓ which are not restricted to be diagonal, thus giving the `smacof` algorithm for IDIOSCAL.

We use the color data of Helm (1959) to illustate the `ismacof` function. There are 12 individuals, the first ten with normal color vision (N), the last two with color deficiency (CD). For individual N6 and individual CD2 the experiment was replicated.

After 494 iterations we find stress 2542.2372780397. In figure 7 we give the common configuration X, and in figure 12. Note in figure 12 the color deficient subjects are in blue, the color normal subjects are in red. In a real analysis of these data we would probably conclude

that two dimensions was not enough to show important variation. The stress measures for each of the 16 individuals, including two replications, are

```
##      N1      N2      N3      N4      N5     N6a     N6b      N7      N8      N9
##   57.26  208.61   88.45   76.56  111.57   94.32   48.39  116.07  148.45  125.86
##     N10     CD1    CD2a    CD2b     CD3     CD4
## 162.94  202.58  279.83  240.64  387.40  193.32
```

and thus stress is much larger for color deficient individuals.

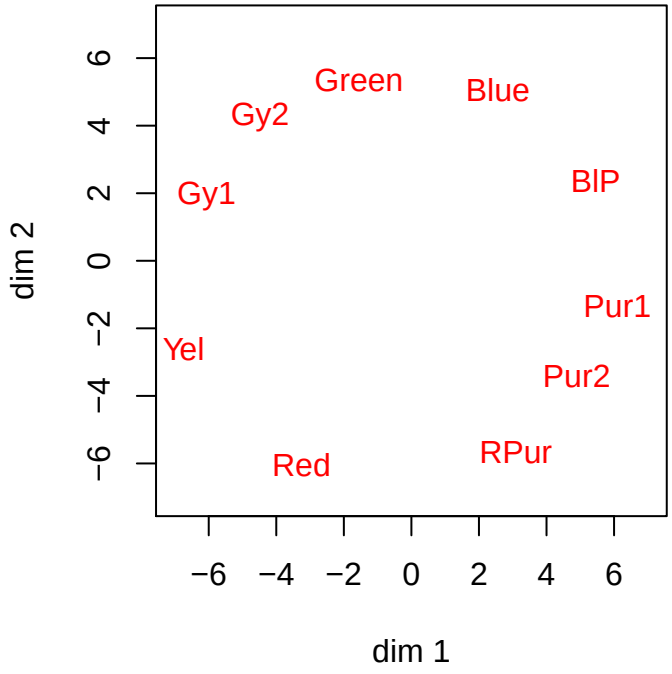

Figure 7: Helm Data, Configuration

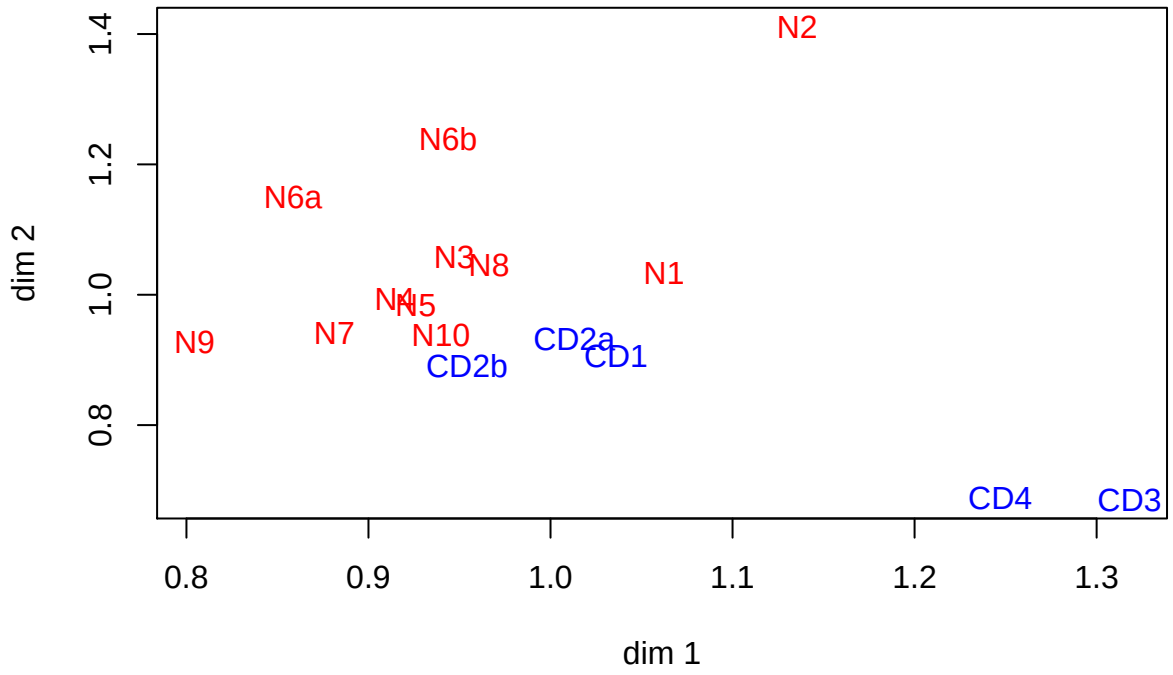

Figure 8: Helm Data, Subject Weights

5 Modeling Asymmetry

5.1 Slide-vector

The slide vector model for asymmetric dissimilarities was proposed, but never published, by Kruskal around 1973. It was discussed in detail by Zielman and Heiser (1993). It assumes, in our notation, that $X_{ij} = X + \frac{1}{2}(e_i - e_j)z'$ and $X_{ji} = X + \frac{1}{2}(e_j - e_i)z' = X - \frac{1}{2}(e_i - e_j)z'$, so that $\frac{1}{2}(X_{ij} + X_{ji}) = X$ and $(X_{ij} - X_{ji}) = (e_i - e_j)z'$. In terms of the squared distances

$$d_{ij}^2(X_{ij}) = \mathbf{tr}\ X_{ij}'A_{ij}X_{ij} = (x_i - x_j + z)'(x_i - x_j + z).$$

Define

$$T := \begin{bmatrix} X \\ z \end{bmatrix}$$

and u_{ij} is a vector of length $n+1$ with zeroes, except for elements i and $n+1$, which are $+1$, and element j, which is -1. Thus $T'u_{ij} = (x_i - x_j + z)$ and $d_{ij}^2(X_{ij}) = u_{ij}'TT'u_{ij}$.

It follows that a `smacof` step is

$$T^{(k+1)} = \left\{\sum_{i=1}^n\sum_{j=1}^n w_{ij}u_{ij}u_{ij}'\right\}^+ \left\{\sum_{i=1}^n\sum_{j=1}^n w_{ij}\xi_{ij}(X_{ij}^{(k)})u_{ij}u_{ij}'\right\}T^{(k)}, \tag{11}$$

with

$$\xi_{ij}(X_{ij}^{(k)}) = \frac{\delta_{ij}}{d_{ij}(X_{ij}^{(k)})}.$$

This can be clarified somewhat, by using

$$u_{ij}u_{ij}' = \begin{bmatrix} A_{ij} & (e_i - e_j) \\ (e_i - e_j)' & 1 \end{bmatrix},$$

so that, for example,

$$\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}u_{ij}u_{ij}' = \begin{bmatrix} V & \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(e_i - e_j) \\ \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(e_i - e_j)' & \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} \end{bmatrix}.$$

Note that if $W = \{w_{ij}\}$ is symmetric, then $\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(e_i - e_j) = 0$, which simplifies the matrix generalized inverse in (11).

We apply the function `ssmacof` to the brand switching of nine green and seven blended bottled teas given by Taniola and Yadohisa (2016). The frequencies of switching n_{ij} are transformed to asymmetric dissimilarities with $\delta_{ij} = \sqrt{n_{ii} + n_{jj} - 2n_{ij}}$.

```
##       DG  IG1 IG2 IG3 KaG SaG SG1 SG2 7G  ABl CBl1 CBl2 CBl3 KaBl KBl1 KBl2
## DG    32   0   0   0   0   0   0   0   0   0    0    0    0    0    1    0
## IG1    2 283   0  18   2   2   1  40   0   0    0    5   18    0   17    2
## IG2    2  41 177   9   0   2   1  13   0   0    0    2   10    0    6    0
## IG3    0   0   0  18   2   0   3   9   0   0    1    2    8    0    9    0
## KaG    0   0   0   0  58   0   1   3   0   0    0    1    1    6    0    1
## SaG    2   0   0   0   0  24   0   0   0   0    0    0    0    0    2    0
## SG1    0   0   0   0   0   1  14   0   0   0    0    1    2    0    8    0
## SG2    3   0   0   0   0   3   8 246   0   0    2    2   26    0   21    1
## 7G     0   4   5   0   0   0   0   3  56   0    0    1    1    0    2    0
## ABl    0  15  11   5   0   0   2  11   4 137    2    5   16    2    5    2
## CBl1   0   0   0   0   0   0   0   0   0   0   22    0    0    0    3    1
## CBl2   0   0   0   0   0   3   0   0   0   0    4  114    0    0   13    7
## CBl3   2   0   0   0   0   5   0   0   0   0    7   22  201    0   23    5
## KaBl   0   0   0   0   0   0   0   2   0   0    0    3    2   28    1    2
## KBl1   0   0   0   0   0   0   0   0   0   0    0    0    0    0  192   13
## KBl2   0   0   0   0   0   0   0   0   0   0    0    0    0    0    0    5
```

The slide vector algorithm takes 1199 iterations to find a solution with stress 2844.4928948188. The slide vector z is , and the configuration

```
##       [,1]     [,2]
## DG   +1.6003  +3.5210
## IG1  +14.8848 +6.2248
## IG2  +10.5817 -7.4227
## IG3  +1.7787  -1.3772
## KaG  -5.6537  +1.7343
## SaG  -1.3980  +2.4636
## SG1  +0.4404  +0.4537
## SG2  +2.1264  +14.7160
## 7G   +5.6544  +0.1990
```

```
## ABl  +1.7900  -11.4184
## CBl1 -2.6166  -1.9803
## CBl2 -5.6584  -8.6451
## CBl3 -9.7128  +9.4961
## KaBl +0.2821  -3.9134
## KBl1 -13.3137 -3.2594
## KBl2 -0.7857  -0.7921
```

is plotted in figure 9 (green teas in blue, blended teas in red).

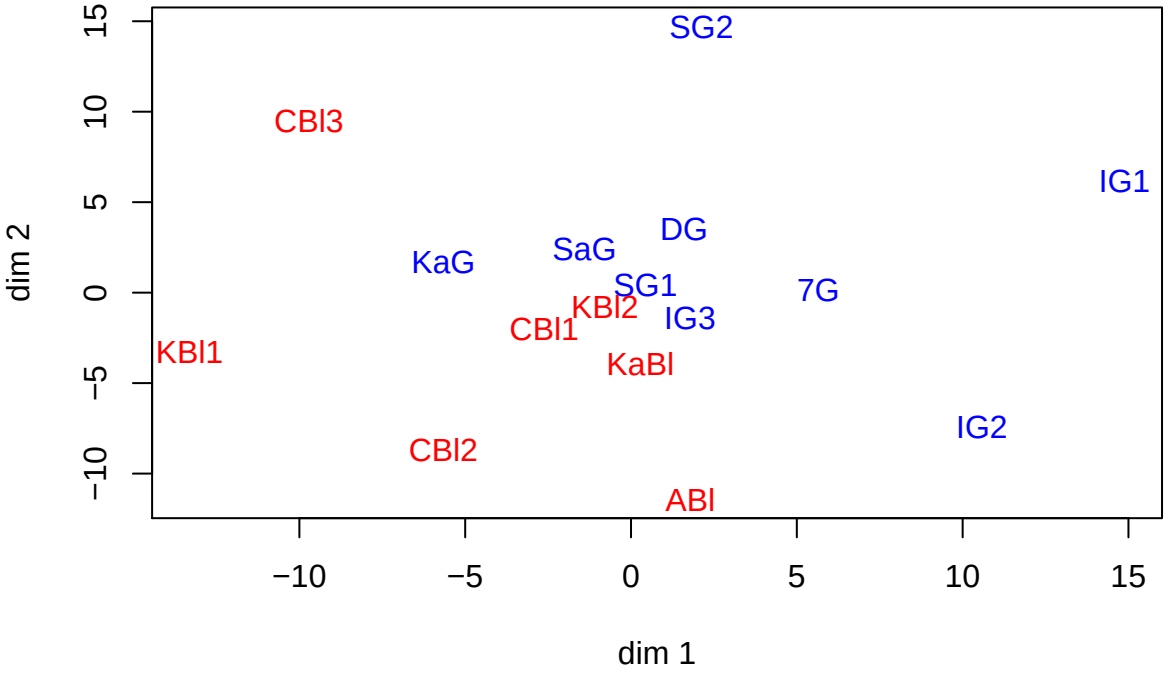

Figure 9: Tea Data, Slide Vector, Configuration

5.2 Asymmetric Scaling of Dimensions

Another possible set of constraints that can be used for asymmetric dissimilarities is $X_{ij} = X\Lambda_j$. Thus $X_{ji} = X\Lambda_i$, and $X_{ij} + X_{ji} = X(\Lambda_i + \Lambda_j)$ while $X_{ij} - X_{ji} = X(\Lambda_i - \Lambda_j)$. In terms of the squared distances

$$d_{ij}^2(X_{ij}) = \sum_{s=1}^{p} \lambda_{js}^2 (x_{is} - x_{js})^2.$$

Not surprisingly, the stationary equation are very similar to equations (9) and (10) in the section on piecewise MDS. We can use the iterative algorithm

$$\sum_{j=1}^{m} V_j X^{(k+1)} (\Lambda_j^{(k)})^2 = \sum_{j=1}^{m} B_j(X_{ij}^{(k)}) X^{(k)} (\Lambda_j^{(k)})^2,$$

$$\Lambda_j^{(k+1)} = \frac{\mathbf{diag}((X^{(k+1)})' B_j(X_{ij}^{(k)}) X_{ij}^{(k)})}{\mathbf{diag}((X^{(k+1)})' V_j X^{(k+1)})}.$$

Here

$$V_j = \sum_{i=1}^{n} w_{ij} A_{ij},$$
$$B_j(X_{ij}^{(k)}) = \sum_{i=1}^{n} w_{ij} \xi_{ij}(X_{ij}^{(k)}) A_{ij}.$$

The solution, implemented in `zsmacof` is virtually the same as the solution for X and Λ from (9) and (10). It uses alternating least squares in exactly the same way.

We apply the method to the tea data. The algorithm now takes 1542 iterations to find a solution with stress 2396.1180470101. The weights Λ_j are

```
##        [,1]    [,2]
##  [1,] +1.2115 +1.0674
##  [2,] +1.0153 +0.9645
##  [3,] +0.8786 +1.4545
##  [4,] +1.0770 +1.1366
##  [5,] +0.8805 +1.3140
##  [6,] +1.0944 +1.1450
##  [7,] +1.1469 +1.1243
##  [8,] +1.4716 +0.8998
##  [9,] +1.0446 +1.2479
## [10,] +1.0328 +0.9526
## [11,] +1.1162 +1.0369
## [12,] +1.3433 +0.8488
## [13,] +0.8730 +1.1972
## [14,] +1.1687 +1.0219
## [15,] +0.9285 +0.9481
## [16,] +1.0838 +1.0586
```

and they are plotted in figure 10.

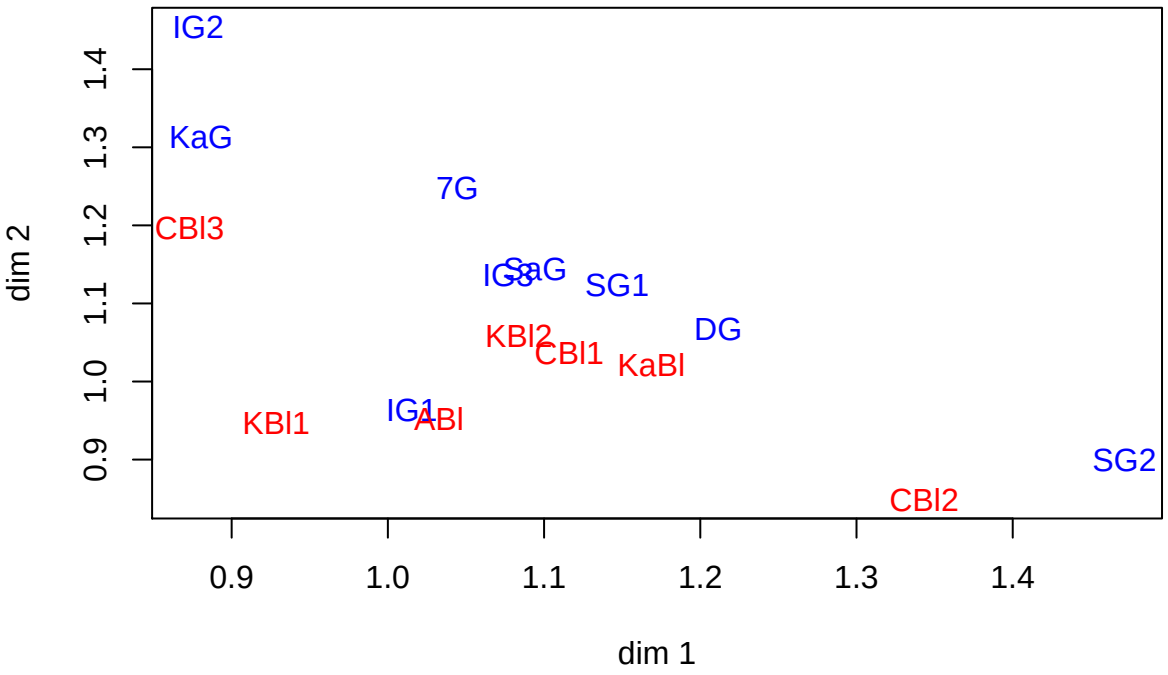

Figure 10: Tea Data, Scaled Dimensions, Weights

The configuration is

```
##       [,1]     [,2]
## DG   -0.2577  +3.3303
## IG1  +10.5474 +11.5414
## IG2  +12.4243 -0.7794
## IG3  +1.9434  -1.0281
## KaG  -6.8271  -0.5935
## SaG  -2.6243  +1.0595
## SG1  +0.0416  +0.2046
## SG2  -1.6144  +14.2417
## 7G   +4.5461  +1.9746
## ABl  +7.0853  -8.7582
## CBl1 -2.2465  -2.9573
## CBl2 -0.7856  -10.4176
## CBl3 -11.8659 +6.0329
## KaBl +1.0778  -3.8971
## KBl1 -10.7461 -8.5591
## KBl2 -0.6983  -1.3947
```

It is plotted in figure 11.

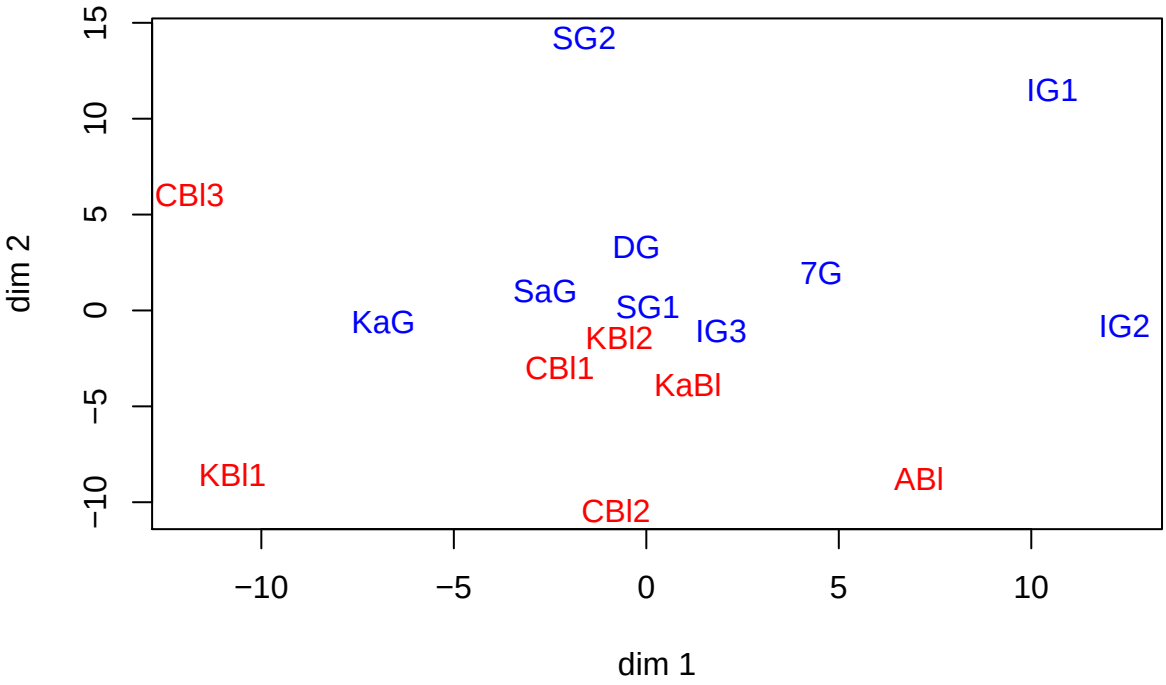

Figure 11: Tea Data, Scaled Dimensions, Configuration

This particular approach to asymmetry may be most interesting in unfolding, where we have

$$d_{ij}^2(X_{ij}, Y_{ij}) = \sum_{s=1}^{p} \lambda_{js}^2 (x_{is} - y_{js})^2,$$

and where the asymmetric treatment of rows and columns is more natural.

6 Appendix: Code

The code for the six different versions of anarchic MDS is both redundant and relatively inefficient. All six programs have the same structure and, for example, share the same voluminous code for debugging I/O.

6.1 auxilary.R

```
mprint <- function (x, d = 2, w = 5, f = "") {
  print (noquote (formatC (
    x, di = d, wi = w, fo = "f", flag = f
  )))
}


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

direct_sum <- function (x) {
  n <- length (x)
  nr <- sapply (x, nrow)
  nc <- sapply (x, ncol)
  s <- matrix (0, sum(nr), sum(nc))
  k <- 0
  l <- 0
  for (j in 1:n) {
    s[k + (1:nr[j]), l + (1:nc[j])] <- x[[j]]
    k <- k + nr[j]
    l <- l + nc[j]
  }
  return(s)
}



repMatrix <- function (x, m) {
  z <- array (0, c(dim (x), m))
  for (j in 1:m)
    z[, , j] <- x
  return (z)
}

slideMatrix <- function (x) {
  n <- nrow (x)
  unit <- function (i, n) ifelse (i == 1:n, 1, 0)
  s <- matrix (0, n + 1, n + 1)
  for (i in 1:n) {
    for (j in 1:n) {
      u <- c (unit (i, n) - unit (j, n), 1)
```

```
      s <- s + x[i, j] * outer (u, u)
    }
  }
  return (s)
}

slideDistance <- function (x, z) {
  n <- nrow (x)
  d <- matrix (0, n, n)
  for (i in 1:n) {
    for (j in 1:n) {
      d[i, j] <- sqrt (sum ((x[i, ] - x[j, ] + z) ^ 2))
    }
  }
  return (d)
}

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

amat <- function (i, j, n) {
  u <- unit (i, n) - unit (j, n)
  return (outer (u, u))
}
```

6.2 asmacof.R

```
asmacof <-
  function (delta,
            f,
            w = 1 - diag (nrow (delta)),
            p = 2,
            xold = torgerson (delta, p),
            itmax = 1000,
            eps = 1e-10,
            verbose = FALSE) {
    dold <- as.matrix (dist (xold))
    eold <- sqrt (dold ^ 2 + f ^ 2)
    sold <- sum (w * (delta - eold) ^ 2)
    v <- -(w + t(w))
    diag (v) <- -rowSums (v)
    m <- ginvRC (v)
    itel <- 1
```

```
    repeat {
      h <- w * delta / (eold + diag (nrow (eold)))
      b <- -(h + t(h))
      diag (b) <- -rowSums(b)
      xnew <- m %*% b %*% xold
      dnew <- as.matrix (dist (xnew))
      enew <- sqrt (dnew ^ 2 + f ^ 2)
      snew <- sum (w * (delta - enew) ^ 2)
      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
      itel <- itel + 1
      sold <- snew
      dold <- dnew
      eold <- enew
      xold <- xnew
    }
    return (list (x = xnew, stress = snew, itel = itel))
  }
```

6.3 bsmacof.R

```
bsmacof <-
  function (delta,
            w = 1 - diag (nrow (delta)),
```

```
          p = 2,
          itmax = 1000,
          eps = 1e-10,
          verbose = FALSE) {
  n <- nrow (delta)
  xold <- torgerson (delta, p)
  dold <- as.matrix (dist (xold))
  fold <- 1
  eold <- sqrt (dold ^ 2 + (fold - fold * diag (n)) ^ 2)
  sold <- sum (w * (delta - eold) ^ 2)
  v <- -(w + t(w))
  diag (v) <- -rowSums (v)
  m <- ginvRC (v)
  itel <- 1
  repeat {
    h <- w * delta / (eold + diag (n))
    b <- -(h + t(h))
    diag (b) <- -rowSums(b)
    xnew <- m %*% b %*% xold
    dnew <- as.matrix (dist (xnew))
    emid <- sqrt (dnew ^ 2 + (fold - fold * diag (n)) ^ 2)
    smid <- sum (w * (delta - emid) ^ 2)
    fnew <- fold * sum (w * (delta / (eold + diag (n)))) / sum (w)
    enew <- sqrt (dnew ^ 2 + (fnew - fnew * diag (n)) ^ 2)
    snew <- sum (w * (delta - enew) ^ 2)
    if (verbose)
      cat(
        "Iteration: ",
        formatC (itel, width = 3, format = "d"),
        "sold: ",
        formatC (
          sold,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "smid: ",
        formatC (
          smid,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "snew: ",
```

```
          formatC (
            snew,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "fnew: ",
          formatC (
            fnew,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "\n"
        )
      if ((itel == itmax) | (sold - snew < eps))
        break
      itel <- itel + 1
      sold <- snew
      dold <- dnew
      eold <- enew
      xold <- xnew
      fold <- fnew
    }
    return (list (
      x = xnew,
      stress = snew,
      f = fnew,
      itel = itel
    ))
  }
```

6.4 psmacof.R

```
psmacof <-
  function (delta,
            pattern,
            p = 2,
            lambda = matrix (1, dim (pattern)[3], p),
            xold = torgerson (delta, p),
            w = 1 - diag (nrow (delta)),
            itmax = 1000,
            eps = 1e-10,
            do_lambda = TRUE,
```

```
        verbose = FALSE) {
  m <- dim (pattern)[3]
  n <- nrow (delta)
  xnew <- xold
  dold <- array (0, c(n, n, m))
  v <- array (0, c (n, n, m))
  eold <- matrix (0, n, n)
  for (j in 1:m) {
    dold[, , j] <- as.matrix (dist (xold %*% diag (lambda [j,])))
    eold <- eold + pattern[, , j] * dold[, , j] ^ 2
    vj <- -pattern[, , j] * (w + t(w))
    diag (vj) <- -rowSums(vj)
    v[, , j] <- vj
  }
  eold <- sqrt (eold)
  sold <- sum (w * (delta - eold) ^ 2)
  itel <- 1
  repeat {
    g <- matrix (0, n, p)
    b <- array (0, c(n, n, m))
    for (j in 1:m) {
      h <- w * delta / (dold[, , j] + diag (n))
      bj <- -pattern[, , j] * (h + t(h))
      diag (bj) <- -rowSums(bj)
      b[, , j] <- bj
      g <- g + bj %*% xold %*% diag (lambda[j,] ^ 2)
    }
    for (s in 1:p) {
      mm <- matrix (0, n, n)
      for (j in 1:m) {
        mm <- mm + (lambda [j, s] ^ 2) * v[, , j]
        xnew[, s] <- ginv (mm) %*% g[, s]
      }
    }
    if (do_lambda) {
      for (j in 1:m) {
        ua <- colSums (xnew * b[, , j] %*% xold %*% diag (lambda[j,]))
        ub <- colSums (xnew * v[, , j] %*% xnew)
        lambda[j,] <- ua / ub
      }
    }
    dnew <- array (0, c(n, n, m))
    enew <- matrix (0, n, n)
    for (j in 1:m) {
```

```
        dnew[, , j] <- as.matrix (dist (xnew %*% diag (lambda [j,])))
        enew <- enew + pattern[, , j] * dnew[, , j] ^ 2
      }
      enew <- sqrt (enew)
      snew <- sum (w * (delta - enew) ^ 2)
      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
      itel <- itel + 1
      sold <- snew
      dold <- dnew
      xold <- xnew
    }
    return (list (
      x = xnew,
      stress = snew,
      lambda = lambda,
      itel = itel
    ))
  }
```

6.5 ismacof.R

```
ismacof <-
  function (delta,
```

```
          p = 2,
          xold = torgerson (rowMeans (delta, dims = 2), p),
          w = repMatrix (1 - diag (dim (delta)[1]), dim (delta)[3]),
          lambda = matrix (1, dim (delta)[3], p),
          itmax = 1000,
          eps = 1e-10,
          verbose = FALSE) {
m <- dim (delta)[3]
n <- nrow (delta[, , 1])
xnew <- xold
dold <- array (0, c(n, n, m))
v <- array (0, c (n, n, m))
for (j in 1:m) {
  dold[, , j] <- as.matrix (dist (xold %*% diag (lambda [j,])))
  vj <- -(w[, , j] + t(w[, , j]))
  diag (vj) <- -rowSums(vj)
  v[, , j] <- vj
}
svec <- colSums (w * (delta - dold) ^ 2, dims = 2)
sold <- sum (svec)
itel <- 1
repeat {
  g <- matrix (0, n, p)
  b <- array (0, c(n, n, m))
  for (j in 1:m) {
    h <- w[, , j] * delta[, , j] / (dold[, , j] + diag (n))
    bj <- -(h + t(h))
    diag (bj) <- -rowSums(bj)
    b[, , j] <- bj
    g <- g + bj %*% xold %*% diag (lambda[j,] ^ 2)
  }
  for (s in 1:p) {
    mm <- matrix (0, n, n)
    for (j in 1:m) {
      mm <- mm + (lambda [j, s] ^ 2) * v[, , j]
      xnew[, s] <- ginv (mm) %*% g[, s]
    }
  }
  for (j in 1:m) {
    ua <- colSums (xnew * b[, , j] %*% xold %*% diag (lambda[j,]))
    ub <- colSums (xnew * v[, , j] %*% xnew)
    lambda[j,] <- ua / ub
  }
  dnew <- array (0, c(n, n, m))
```

```
    for (j in 1:m) {
      dnew[, , j] <- as.matrix (dist (xnew %*% diag (lambda [j,])))
    }
    svec <- colSums (w * (delta - dnew) ^ 2, dims = 2)
    snew <- sum (svec)
    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
    itel <- itel + 1
    sold <- snew
    dold <- dnew
    xold <- xnew
  }
  return (list (
    x = xnew,
    stress = snew,
    svec = svec,
    lambda = lambda,
    itel = itel
  ))
}
```

6.6 ssmacof.R

```
ssmacof <-
  function (delta,
            p = 2,
            zold = rep (0, p),
            xold = torgerson ((delta + t (delta)) / 2, p),
            w = 1 - diag (nrow (delta)),
            itmax = 1000,
            eps = 1e-10,
            verbose = FALSE) {
    n <- nrow (delta)
    dold <- slideDistance (xold, zold)
    vmat <- slideMatrix (w)
    sold <- sum (w * (delta - dold) ^ 2)
    itel <- 1
    repeat {
      bnew <- slideMatrix (w * delta / (dold + diag (n)))
      tnew <- ginv (vmat) %*% bnew %*% rbind (xold, zold)
      xnew <- tnew[1:n, ]
      znew <- drop (tnew[n + 1, ])
      dnew <- slideDistance (xnew, znew)
      snew <- sum (w * (delta - dnew) ^ 2)
      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
      itel <- itel + 1
      sold <- snew
      dold <- dnew
      xold <- xnew
      zold <- znew
    }
    return (list (
      x = xnew,
      z = znew,
      stress = snew,
      itel = itel
    ))
  }
```

6.7 zsmacof.R

```
zsmacof <-
  function (delta,
            p = 2,
            lold = matrix (1, nrow (delta), p),
            xold = torgerson ((delta + t (delta)) / 2, p),
            w = 1 - diag (nrow (delta)),
            itmax = 1000,
            eps = 1e-10,
            verbose = FALSE) {
    n <- nrow (delta)
    xnew <- xold
    lnew <- lold
    dold <- dnew <- matrix (0, n, n)
    for (i in 1:n) {
      for (j in 1:n) {
        dold [i, j] <- sqrt (sum (((xold [i, ] - xold [j, ]) * lold [j, ]) ^ 2))
      }
    }
    v <- array (0, c (n, n, n))
    for (j in 1:n) {
      vv <- matrix (0, n, n)
      for (i in 1:n) {
        vv <- vv + w[i, j] * amat (i, j, n)
      }
      v[, , j] <- vv
    }
    sold <- sum (w * (delta - dold) ^ 2)
```

```
  itel <- 1
  repeat {
    u <- array (0, c (n, p, n))
    for (j in 1:n) {
      bb <- matrix (0, n, n)
      for (i in 1:n) {
        if (i == j) next
        bb <- bb + w [i, j] * (delta [i, j] / dold [i , j]) * amat (i, j, n)
      }
      u[, , j] <- ginv (v[, , j]) %*% bb %*% xold %*% diag (lold [j, ])
    }
    for (s in 1:p) {
      vv <- matrix (0, n, n)
      xs <- rep (0, n)
      for (j in 1:n) {
        vv <- vv + v[, , j] * (lold [j, s] ^ 2)
        xs <- xs + v[, , j] %*% u[, s, j] * lold [j, s]
      }
      xnew [, s] <- ginv (vv) %*% xs
    }
    for (i in 1:n) {
      for (j in 1:n) {
        dnew [i, j] <- sqrt (sum (((xnew [i, ] - xnew [j, ]) * lold [j , ]) ^ 2))
      }
    }
    smid <- sum (w * (delta - dnew) ^ 2)
    for (j in 1:n) {
      ua <- colSums (xnew * v[, , j] %*% u[, , j])
      ub <- colSums (xnew * v[, , j] %*% xnew)
      lnew[j, ] <- ua / ub
    }
    for (i in 1:n) {
      for (j in 1:n) {
        dnew [i, j] <- sqrt (sum (((xnew [i, ] - xnew [j, ]) * lnew [j , ]) ^ 2))
      }
    }
    snew <- sum (w * (delta - dnew) ^ 2)
    if (verbose)
      cat(
        "Iteration: ",
        formatC (itel, width = 3, format = "d"),
        "sold: ",
        formatC (
          sold,
```

```
            digits = 8,
            width = 12,
            format = "f"
          ),
          "smid: ",
          formatC (
            smid,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "snew: ",
          formatC (
            snew,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "\n"
        )
      if ((itel == itmax) | (sold - snew < eps))
        break
      itel <- itel + 1
      sold <- snew
      dold <- dnew
      lold <- lnew
      xold <- xnew
    }
    return (list (
      x = xnew,
      d = dnew,
      lambda = lnew,
      stress = snew,
      itel = itel
    ))
  }
```

References

Borg, I., and P.J.F. Groenen. 2005. *Modern Multidimensional Scaling: Theory and Applications*. Second Edition. 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://deleeuwpdx.net/janspubs/1977/chapters/deleeuw_C_77.pdf.

———. 1984. "Differentiability of Kruskal's Stress at a Local Minimum." *Psychometrika* 49: 111–13. http://deleeuwpdx.net/janspubs/1984/articles/deleeuw_A_84f.pdf.

———. 1988. "Convergence of the Majorization Method for Multidimensional Scaling." *Journal of Classification* 5: 163–80. http://deleeuwpdx.net/janspubs/1988/articles/deleeuw_A_88b.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://deleeuwpdx.net/janspubs/1994/chapters/deleeuw_C_94c.pdf.

———. 2005. "Unidimensional Scaling." In *The Encyclopedia of Statistics in Behavioral Science*, edited by B.S. Everitt and D.C, 4:2095–7. New York, N.Y.: Wiley. http://deleeuwpdx.net/janspubs/2005/chapters/deleeuw_C_05h.pdf.

De Leeuw, J., and W. J. Heiser. 1980. "Multidimensional Scaling with Restrictions on the Configuration." In *Multivariate Analysis, Volume V*, edited by P.R. Krishnaiah, 501–22. Amsterdam, The Netherlands: North Holland Publishing Company. http://deleeuwpdx.net/janspubs/1980/chapters/deleeuw_heiser_C_80.pdf.

De Leeuw, J., and W.J. Heiser. 1977. "Convergence of Correction Matrix Algorithms for Multidimensional Scaling." In *Geometric Representations of Relational Data*, edited by J.C. Lingoes, 735–53. Ann Arbor, Michigan: Mathesis Press. http://deleeuwpdx.net/janspubs/1977/chapters/deleeuw_heiser_C_77.pdf.

De Leeuw, J., and P. Mair. 2009. "Multidimensional Scaling Using Majorization: SMACOF in R." *Journal of Statistical Software* 31 (3): 1–30. http://deleeuwpdx.net/janspubs/2009/articles/deleeuw_mair_A_09c.pdf.

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

Groenen, P.J.F., and M. Van de Velden. 2016. "Multidimensional Scaling by Majorization: A Review." *Journal of Statistical Software* 73 (8): 1–26. doi:10.18637/jss.v073.i08.

Helm, C.E. 1959. "A Multidimensional Ratio Scaling Analysis of Perceived Color Relations." Technical Report. Princeton, NJ: Educational Testing Service.

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

Revelle, William. 2015. *Psych: Procedures for Psychological, Psychometric, and Personality Research*. Evanston, Illinois: Northwestern University. http://CRAN.R-project.org/package=psych.

Taniola, K., and H. Yadohisa. 2016. "Discriminant Coordinates for Asymmetric Dissimilarity

Data Based on Radius Model." *Behaviormetrika* 43 (1): 1–17.

Zielman, B., and W.J. Heiser. 1993. "Analysis of Asymmetry by a Slide-vector." *Psychometrika* 58 (1): 101–14.