Gifi Update Notes

Jan de Leeuw

First created on September 30, 2019. Last update on May 08, 2022

```
library("MASS")
source ("gifiEngine.R")
source ("gifiUtilities.R")
source ("gifiCoding.R")
source ("gifiStructures.R")
source ("gifiMonotone.R")
source ("splineBasis.R")
source ("homals.R")
```

1 The data

The **data** for a gifiAnalysis is list of vectors (or factors), each of length n, possibly with NA's. Each vector can be numerical, character, or logical. Thus the **data** can be an R dataframe. Each element of the **data** can be thought of as a **variable**. All variables are functions defined on the same domain with n elements.

For a gifiAnalysis the **data** are partitioned into a number of **sets** of variables. Each set can contain both **active** and **passive** variables. The **active** variables participate in the actual computations of the gifiAnalysis, the **passive** (or **supplementary**) variables are used solely for post-processing and presentation purposes.

2 The gifi structure

Information about the variables is coded in **gifiVariables**. A **gifiVariable** is a structure with ten components, which are all objects that do not change during computation (parameters, if you like). These are:

- **data** - vector or factor of n numbers, or chars, possibly with NA's;
- **basis** - a matrix with a B-spline basis, $n \times k_i$;
- **degree** - degree of the spline d_i;

- **copies** - number of copies l_i;
- **ordinal** - true or false;
- **missing** - multiple, single, deleted, or average;
- **active** - true or false;
- **set** - integer code for set;
- **name** - name of variable;
- **type** - type (categorical, polynomial, or splinical).

A **gifi** is simply a list of **gifiVariables**.

3 Gifi loss

The **gifi** structure dictates the form of the loss function. **Gifi loss**, previously described in Gifi (1990) as **meet loss**, is defined as

$$\sigma(X, Z, A) = \frac{1}{mp} \sum_{j=1}^{m} \text{SSQ } (X - \sum_{i \in I_j} H_i Z_i A_i). \tag{1}$$

We use SSQ for the (unweighted) sum of squares of a numerical vector or matrix. The outer summation in (1) is over active sets, the inner summation over the active variables in the set (active sets are **gifiSets** with at least one active **gifiVariable**). Thus the index set $\mathcal{N} = \{1, 2, \cdots, N\}$, where N is the total number of active variables in the analysis, is partitined into the m index sets I_j, with $I_j \cap I_l = \emptyset$ and $\bigcup_{j=1}^{m} I_j = \mathcal{N}$.

In the loss function:

- X is $n \times p$, **object scores**, observations by dimensions;
- H_i is the **basis**, $n \times k_i$, observations by basis, known and fixed;
- Z_i are the **coefficients**, $k_i \times l_i$, basis by copies;
- A_i are the **loadings**, $l_i \times p$, copies by dimensions.

In addition we impose the constraints:

- X is **centered** $e'X = 0$ and **orthonormal** $X'X = I$.
- If a variable is **ordinal** there are inequality constraints on the non-missing elements of $T_i = H_i Z_i$ (or directly on the elements of Z_i).

4 The xGifi Structure

In the same way there is an **xGifiVariable** and an **xGifi**. Once again, an **xGifi** is a list of **xGifiVariables**.

The **xGifiVariable** has the information about the variable that changes during the computations of the gifiAnalysis, the estimates that are updated in each iteration. It has five components:

- **transform** $T_i = H_i Z_i$, orthogonalized, observations by copies, $n \times l_i$;
- **loadings** A_i, copies by dimensions, $l_i \times p$;
- **scores** $S_i = T_i A_i = H_i Z_i A_i$, observations by dimensions, $n \times p$;
- **quantifications** $Q_i = Z_i A_i$, degrees by dimensions, $k_i \times p$;
- **coefficients** Z_i, basis by copies, $k_i \times l_i$,

In previous versions of the gifiSystem a different terminology was used. The **quantifications** Q_i were restricted to be of **rank** less than or equal to l_i. There was no notion of **copies**, which assumes or implies a variable enters into the gifiAnalysis more than once, and there was no explicit decomposition $Q_i = Z_i A_i$ maintained. The notion of **copies** is more flexible than the original approach.

There are some alternative ways to write Gifi loss that will come in handy. If we concatenate the **transforms** $T_i = H_i Z_i$ horizontally and the **loadings** A_i vertically we can write

$$\sigma(X, Z, A) = \frac{1}{mp} \sum_{j=1}^{m} \text{SSQ } (X - T_j A_j). \tag{2}$$

We can also concatenate the **quantifications** Q_j vertically and write

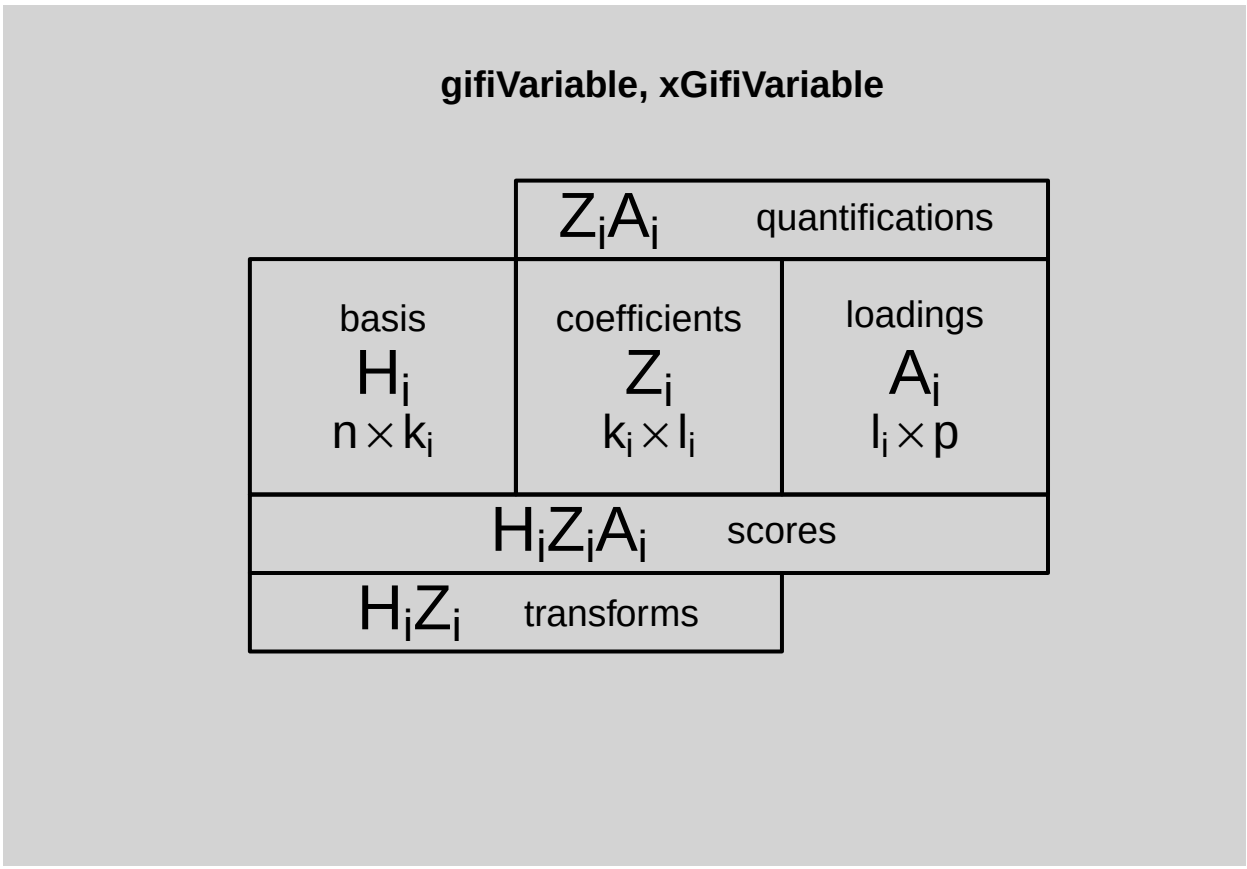
$$\sigma(X, Z, A) = \frac{1}{mp} \sum_{j=1}^{m} \text{SSQ } (X - H_j Q_j). \tag{3}$$

This is basically what is used in the original gifiSystem, with rank constraints of the Q_i.

And finally we can concatenate the **bases** horizontally and make Z_j the direct sum of the Z_i in set j. Thus Z_j is block diagonal, with the Z_i as diagonal blocks. Then

$$\sigma(X, Z, A) = \frac{1}{mp} \sum_{j=1}^{m} \text{SSQ } (X - H_j Z_j A_j). \tag{4}$$

The following diagram may be helpful. Or not.



5 Sanity

Before the analysis starts the gifiVariables and gifiSets are checked for sanity. If on eof the following conditions is violated the analysis stops with the corresponding error message.

- A variable cannot be completely missing.
- A gifiAnalysis needs more than one active set.
- A gifiAnalysis needs more than one active variable.
- Sets need to be coded using consecutive integers.
- Each basis is non-negative, with rows that add up to one.
- A passive variable must be the only variable in its set.

6 The Gifi engine

We minimize Gifi loss using **alternating least squares**, combined with majorization. The `gifiEngine` function The alternation is over X, the Z_i and the A_i.

The orthonomality constraints on X are simple to handle, and minimizing over X for fixed scores $S_j = H_iZ_iA_i$ is an orthogonal **Procrustus** problem. The engine has the option to use QR decomposition instead of Procrustus, which also gives a valid ALS method.

Because A is unconstrained, minimizing over the A_j for given X and $T_j = H_j Z_j$ is a simple least squares problem, which can be solved for each set separately. Because our least squares problems are often singular or ill-conditioned, we use the Moore-Penrose inverse function ginv() from the MASS package.

The most complicated part is solving for the Z_i for given A and X, and that is where majorization comes in. We outline the majorization theory for minimizing gifiLoss in an appendix. There are three possible ways to majorize discussed there, but only one of them is implemented in the gifiEngine.

7 Active and Passive Variables

Passive variables are handled in two stages. In the first stage we minimize the loss function (1) over X, Z_i, and A_i. Then in the second stage we minimize

$$\text{SSQ } (X - H_i Z_i A_i) \tag{5}$$

separately for each passive variable, using the optimal X from the first stage.

Fitting passive variables is actually implemented in the main iteration loop of the gifiEngine, so we do not really need two stages. Each passive variable defines its own set, in which it is the only variable. And these passive sets do not contribute to the loss and to the alternating least squares step that updates X.

8 Choice of Basis

Note that in earlier versions of the gifiSystem all data were **categorical**. The **basis** always was an **indicator matrix**. In this version the basis is either categorical, or a B-spline basis (De Boor (2001)). For a B-spline basis the user has to give a vector of **interior knots**. The **exterior knots** are always chosen to be the maximum and minimum of the data values, which implies that for B-spline basis the data must be numerical. There can be missing data, because the basis is only constructed for the non-missing data. After basis construction the various missing data options are used to complete the basis to a non-negative basis-matrix in which all rows add up to one. For categorical data, which do not have to be numerical, an indicator matrix is used, which is also non-negative with rows adding up to one.

In actual computation we use a **reduced basis**, which is obtained from the basis matrix by columnwise centering and then leaving off the last column. Columnwise centering has the effect that all rows now add up to zero and the basis is singular. Leaving off the last column can eliminate that singularity, and still gives a basis for the space of centered vectors in the space spanned by the original basis. Moreover, a centered vector is a linear combination of the original basis vectors with increasing coefficients if and only if it also is a linear combination of the reduced basis vectors with increasing coefficients. This is important for our treatment of ordinal variables.

9 Missing Data

For each variable there are four options for handling missing data. They are defined in terms of the bases of the variables, and they preserve the non-negativity and unit-row-sum property of the bases.

- **single** add a single 0/1 column to the basis with 1 for missing.
- **multiple** add a 0/1 column to the basis for each missing observation, i.e. append an identity matrix.
- **average** for missing observations, use a basis row with all elements equal to the mean of the non-missing rows.
- **random** for missing observations, use basis rows by sampling with replacement from the non-missing rows.

10 Ordinal Restrictions

Monotone splines De Leeuw (2017)

If a variable has more than one copy (in original Gifi terminology this means the variable a **multiple quantification**) then we require only the first r copies are constrained ordinally..

Because of the indeterminacy in the product $S_i = H_i Z_i A_i$, the requirement that r copies are monotonic with the data is actually equivalent to the requirement that there are r monotone directions in the column space of the basis (and these directions can be highly correlated).

Busing

11 Wrappers

The following wrappers, and their defining properties, are included:

- homals: one variable per set, all variables ndim copies
- princals: one variable per set, all variables one copy
- primals: one variable per set, all variables one copy, ndim = 1
- canals: two sets, all variables one copy
- morals: two sets, one set has a single variable with one copy, ndim = 1
- criminals: two sets, one set has a single variable with ndim copies
- addals: two sets, one set one variable with one copy, other sets variables have ndim copies, ndim = 1

For wrappers such as canals, princals, morals, addals, criminals the numerical output (the results returned by the wrapper functions) should look as if ordinary regression, PCA, CDA, etc. was applied to the transformed data.

For this the results in Appendix 1 are helpful. We use the fact that the classical multivariate techniques can all be formulated as finding eigenvalues and eigenvectors of the generalized eigenvalue problem $Cx = m\lambda Dx$, where $C = T'T$ and $D = \bigoplus_{j=1}^{m} T_j$. Here T is the horizontal concatenation of the $T_j = H_j Z_j$.

Thus, after convergence of the gifiAnalysis, each wrapper recovers corresponding classical multivariate analysis results in their familiar form.

The wrappers far from exhaust all possible gifiAnalyses. Using the two functions makeGifi() and gifiEngine() users can write their own wrappers, or do perform analyses that deviate from the standard wrappers (such as analyses with variables having different numbers of copies within a set). Here is an example,

```
data (sleeping, package="homals")
nobs <- length (sleeping[[1]])
nvar <- length (sleeping)
ndim <- 2
set.seed <- NULL
x <- ortho (center (matrix (rnorm (nobs * ndim), nobs, ndim)))
g <-
  makeGifi(
    sleeping,
    knots = list(c(-5,0,5),c(500, 1000, 1500, 2000, 2500),100*1:7,NULL),
    degrees = c (2, 2, 2, -1),
    copies =  c (1, 1, 1, 2),
    ordinal = c (0, 0, 0, 0),
    missing = rep ("r", 4),
    active = rep (TRUE, 4),
    names = names (sleeping),
    sets = c(1, 1, 2, 1)
  )
f <- xGifi(g, x)
h <-
  gifiEngine(
    g,
    ndim = ndim,
    use_qr = FALSE,
    itmax = 1000,
    eps = 1e-6,
    seed = 123,
    verbose = TRUE
  )
```

12 Appendix: Three problems

Consider the function, called **meet-loss** by Gifi,

$$\sigma(Y,B)=\sum_{j=1}^{m}\text{SSQ }(X-T_jA_j). \tag{6}$$

Here the T_j are given $n\times k_j$ matrices, and we want to minimize σ over the $n\times p$ matrix X and the m $k_j\times p$ matrices A_j. Obviously we need some sort of normalization to make this problem interesting, because otherwise $X=0$ and $A_j=0$ trivially solve the minimization problem. The three problems we discuss in this appendix use three different normalizations.

First we simplify the notation. Define $D_j=T_j'T_j$, and the direct sum $D=\bigoplus_{j=1}^m D_j$. Thus D is block diagonal, with the D_j as the diagonal blocks. Also define

$$T=\begin{bmatrix}T_1 & | & \cdots & | & T_m\end{bmatrix},$$

and

$$A=\begin{bmatrix}A_1\\ \vdots\\ A_m\end{bmatrix}.$$

Then

$$\sigma(X,A)=m\text{ tr }X'X-2\text{ tr }X'TA+\text{tr }A'DA.$$

12.1 First Problem

The first problem we discuss is minimization of σ over X constrained by $mX'X=I$, with no constraints on A.

If $mX'X=I$ then

$$\min_B\sigma(X,A)=p-\text{tr }X'TD^+T'X,$$

where the minimum is attained for $A=D^+T'X$, with D^+ the Moore-Penrose inverse. It follows that problem 1 is solved by choosing X equal to the eigenvectors corresponding with the p largest eigenvalues of $TD^+T'X=mX\Phi$, normalize them by $mX'X=I$, and then setting $A=D^+T'X$.

12.2 The Second Problem

The second problem has no constraints on X, but requires $A'DA=I$. If $A'DA=I$ then

$$\min_Y\sigma(X,A)=p-\frac{1}{m}\text{tr }A'CA,$$

where $C=T'T$. The minimum is attained for $X=\frac{1}{m}TA$. Problem 2 is solved by finding the eigenvectors corresponding with the p largest eigenvalues of $CA=mDA\Psi$, normalize them by $A'DA=I$, and then setting $X=\frac{1}{m}TA$.

12.3 The Third Problem

The third problem is to use **both** $mX'X = I$ and $A'DA = I$. Then

$$\sigma(X, A) = 2(p - \text{tr } X'TA).$$

Problem 3 is solved by finding the left and right singular vectors corresponding with the largest singular values of $TA = mX\Lambda$ and $T'X = DA\Lambda$, both appropriately normalized.

It follows that $\Phi = \Psi = \Lambda^2$, and the solutions to the three problems are essentially the same, except for the scaling of dimensions. If A with $A'DA = I$ and X with $mX'X = I$ solve the singular value problem 3, then X and $D^+T'X = A\Lambda$ solve problem 1, and A and $\frac{1}{m}TA = X\Lambda$ solve problem 2.

13 Appendix: Majorization

In this appendix we consider the problem of minimizing

$$\sigma(Z_j) = \text{SSQ } (X - H_jZ_jA_j)$$

over Z_j, where H_j has the H_i for $i \in I_j$ concatenated horizontally, the A_j have the A_i concatenated vertically, and the Z_j is the direct sum of the Z_i. Also define the transforms $T_j = H_jZ_j$, which have the $T_i = H_iZ_i$ concatenated horizontally, and the quantifications $Q_j = Z_jA_j$, which have the $Q_i = Z_iA_i$ concatenated vertically.

We start with expanding σ around the current best solution $\tilde{Z}_j$. Define the **residuals** $\tilde{R}_j = X - H_j\tilde{Z}_jA_j$. Then

$$\begin{aligned}\sigma(Z_j) &= SSQ(X - H_j\{\tilde{Z}_j + (Z_j - \tilde{Z}_j)\}A) = SSQ(\tilde{R}_j - H(Z_j - \tilde{Z}_j)A_j)\\ &= \sigma(\tilde{Z}_j) - 2 \text{ tr } (Z_j - \tilde{Z}_j)'H_j'\tilde{R}_jA_j' + \text{tr } (Z_j - \tilde{Z}_j)'H_j'H_j(Z - \tilde{Z})A_jA_j'\end{aligned}$$

From here we can go at least three ways.

13.1 First Majorization

The first majorization uses the result that for any positive semi-definite F and G of the same order we have tr $FG \leq \lambda_{\max}(F)$tr G, where $\lambda_{\max}(F)$ is the largest eigenvalue of F. The proof is simple, using the eigen-decomposition $F = K\Lambda K'$.

$$\text{tr } FG = \text{tr } K\Lambda K'G = \text{tr } \Lambda K'GK \leq \lambda_{\max}(F)\text{tr } K'GK = \lambda_{\max}(F)\text{tr } G.$$

We apply this result to our minimization problem. Suppose κ_j is the largest eigenvalue of $A_j'A_j$. Then

$$\sigma(Z_j) \leq \sigma(\tilde{Z}_j) - 2 \text{ tr } (H_jZ_j - H_j\tilde{Z}_j)'\tilde{R}_jA_j' + \kappa_j \text{ tr } (H_jZ_j - H_j\tilde{Z}_j)'(H_jZ_j - H_j\tilde{Z}_j)$$

This gives the majorization function that must be minimized over Z. By completing the square this can be done by minimizing SSQ $(H_jZ_j - \tilde{U}_j)$, where $\tilde{U}_j = H_j\tilde{Z}_j + \kappa_j^{-1}\tilde{R}_jA_j'$.

If we majorize in this way for every set we obtain the majorizing function

$$\sum_{j=1}^{m}\sum_{i\in I_j}\text{SSQ }(H_iZ_i - \tilde{U}_i),$$

with $\tilde{U}_i = H_i\tilde{Z}_i + \kappa_j^{-1}\tilde{R}_jA_i'$. Note that $\tilde{R}_j$, and thus $\tilde{U}_i$, depends on the current best scores of the other variables in the set. Nevertheless it is fine from the algorithmic point of view to update the $\tilde{R}_j$ after all Z_i have been adjusted (think about the distinction between the Gauss-Jordan and the Gauss-Seidel method for iteratively solving linear equations).

13.2 Second Majorization

Alternatively, we can use another inequality. If F and G are two positive semi-definite matrices, then for any E for which the matrix product $E'FEG$ is defined tr $E'FEG \leq \lambda_{\max}(F)\lambda_{\max}(G)\text{SSQ}(E)$. The proof uses $e = \text{vec}(E)$ and the Kronecker product.

$$\text{tr }E'FEG = e'(F\otimes G)e \leq \lambda_{\max}(F\otimes G)e'e = \lambda_{\max}(F)\lambda_{\max}(G)\text{SSQ}(E).$$

If μ_j is the largest eigenvalue of $H_j'H_j$ then

$$\sigma(Z_j) \leq \sigma(\tilde{Z}_j) - 2\text{ tr }(Z_j - \tilde{Z}_j)'H_j'\tilde{R}_jA_j' + \kappa\mu\text{ SSQ }(Z_j - \tilde{Z}_j)$$

and this majorization function can be minimized by minimizing $\text{SSQ}(Z_j - \tilde{V}_j)$, with $\tilde{V}_j = \tilde{Z}_j + \kappa_j^{-1}\mu_j^{-1}\ H_j'\tilde{R}_jA_j'$.

The second majorization minimizes

$$\sum_{j=1}^{m}\sum_{i\in I_j}\text{SSQ }(Z_i - \tilde{V}_i).$$

with $\tilde{V}_i = \tilde{Z}_i + \kappa_j^{-1}\mu_j^{-1}\ H_i'\tilde{R}_jA_i'$.

13.3 Third majorization

There is a third majorization, which also may be useful. We now expand around $\tilde{Q}_i = \tilde{Z}_i\tilde{A}_i$. Using the same reasoning as in the other two majorizations, we arrive at

$$\sum_{j=1}^{m}\sum_{i\in I_j}\text{SSQ }(Z_iA_i - \tilde{Y}_i).$$

with $\tilde{Y}_i = \tilde{Z}_i\tilde{A}_i + \mu^{-1}H_i'\tilde{R}_j$ and $\tilde{R}_j = X - H_j\tilde{Z}_j\tilde{A}_j$.

13.4 Comparison

The first majorization makes most sense when the ordinal constraints are on the columns of $T_i = H_i Z_i$, the second one when they are on the columns of Z_i. Of course in general T_i will be much larger than Z_i, so the second and third majorization optimize in much smaller spaces. The third majorization is particularly useful if there are no ordinal constraints, because minimizing over $Z_i A_i$ is just a singular value problem, which is usually small (the minimum of the number of columns in the basis and the dimensionality). For both the second and third majorization we have to compute the μ_j, but they do not change over iterations, so that only has to be done once.

14 Appendix: Copy Expansion

If we adopt the modification suggsted in this appendix we may have to change the basic structures. If variable i has l **copies** then $S = HZ$ can be written as

$$S = \sum_{s=1}^{l} H z_s a_s,$$

or, in words, having a variable with l **copies** is exactly the same as having l variables with one copy, where all l variables have the same basis. In that sense the notion of **copies** (or the older notion of **rank**) is superfluous, and our gifiStructures could reflect that.

The main reason to use **copies** is to save storage (only one basis for all copies). In addition we must make sure the different coefficients z_s corresponding with the same basis H are, and remain, linearly independent. One alternative way of handling copies is to define a gifiCopy, and define a gifiVariable as a list of gifiCopies. Or we can leave the gifiVariable as is, and treat each copy as a different xGifiVariable.

```
data(hartigan, package = "homals")
h <- homals (hartigan)
```

0.5157418 in 172

```
hortigan <-
  data.frame(
    hartigan[[1]],
    hartigan[[1]],
    hartigan[[2]],
    hartigan[[2]],
    hartigan[[3]],
    hartigan[[3]],
    hartigan[[4]],
    hartigan[[4]],
```

```
    hartigan[[5]],
    hartigan[[5]],
    hartigan[[6]],
    hartigan[[6]]
  )
x <- ortho (center (matrix(rnorm(48), 24, 2)))
g <-
  makeGifi(
    hortigan,
    knots = rep (NULL, 12),
    degrees = rep(-1, 12),
    ordinal = rep (0, 12),
    copies = rep(1, 12),
    missing = rep ("r", 12),
    active = rep (TRUE, 12),
    names = names(hortigan),
    sets = c(1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6)
  )
f <- xGifi(g, x)
h <-
  gifiEngine(
    g,
    ndim = 2,
    use_qr = FALSE,
    itmax = 1000,
    eps = 1e-6,
    seed = NULL,
    verbose = FALSE
  )
```

0.5157504 in 250

15 Appendix: Upgrades

In the future I will use basically the same approach using a variable structure (will not change much from the gifiVariable) and an engine to implement some upgrades. First, we will add **burtEngine**, which will allow us to add (generalized) correspondence analysis. Then, gifiEngine will give birth to **nextEngine**, which implements the theory of De Leeuw (2004). We add **orthoBlock** to the possible types of a variable and require a pattern matrix to code the SEM. This allows us to add pathals, dynamals, lisrals, lineals, MIMIC, factals etc. Then, we will add **aspectEngine**, which implements De Leeuw (1988).

15.1 Appendx: Burt Engine

The gifiEngine operates on a list of gifiVariables that are generated from a dataframe. The burtEngine works on a burtTable, which is the crossproduct of the horizontal concatenation of all bases. We can generate the burtTable from a gifiStructure, but there is also an option to generate it from a single frequency table (to do generalized correspondence analysis).

We must maximize tr $Q'CQ$ over all Q satisfying $Q'DQ = I$. Now

$$\text{tr } Q'CQ \geq \text{tr } \tilde{Q}'C\tilde{Q} + 2 \text{ tr } (Q - \tilde{Q})'C\tilde{Q} = 2 \text{ tr } Q'C\tilde{Q} - \text{tr } \tilde{Q}'C\tilde{Q}.$$

Thus majorization theory tells us to maximize tr $Q'C\tilde{Q}$ over $Q'DQ = I$. This is a weighted procrustus problem with solution $Q = D^{-\frac{1}{2}}KL'$, using the SVD $D^{-\frac{1}{2}}C = K\Lambda L'$. Thus K is an orthonormal basis for the column space of $D^{-\frac{1}{2}}C$. Any other orthonormal basis for this space is a rotation of K and consequently gives the same value of tr $Q'CQ$. Also note that $D^{\frac{1}{2}}$ can be the symmetric square root, but it can also be a (triangular) Cholesky factor. In any case, it does not change during computations, so it only has to be computed once.

References

De Boor, C. 2001. *A Practical Guide to Splines*. Revised Edition. New York: Springer-Verlag.

De Leeuw, J. 1988. "Multivariate Analysis with Optimal Scaling." In *Proceedings of the International Conference on Advances in Multivariate Statistical Analysis*, edited by S. Das Gupta and J. K. Ghosh, 127–60. Calcutta, India: Indian Statistical Institute.

———. 2004. "Least Squares Optimal Scaling of Partially Observed Linear Systems." In *Recent Developments in Structural Equation Models*, edited by K. van Montfort, J. Oud, and A. Satorra. Dordrecht, Netherlands: Kluwer Academic Publishers.

———. 2017. "Computing and Fitting Monotone Splines." 2017.

Gifi, A. 1990. *Nonlinear Multivariate Analysis*. New York, N.Y.: Wiley.