Gifi Analysis of Multivariate Data

Jan de Leeuw

First created May 04, 2016. Last update October 28, 2021

Contents

}

}

Note

This book will be expanded/updated frequently. The directory deleeuwpdx.net/pubfolders/stress has a pdf version, the bib file, the complete Rmd file with the code chunks, and the R and C source code. Suggestions for improvement of text and code are welcome. All text and code are in the public domain and can be copied and used by anybody in any way they like. Attribution will be appreciated, but is not required.

Just as an aside: "above" in the text refers to anything that comes earlier in the book and "below" refers to anything that comes later. This always confuses me, so I had to write it down. I also number *all* displayed equations. Equations are displayed if and only if they are important, are referred to in the text, or mess up the line spacing.

Preface

In 1980 members of the Department of Data Theory at the University of Leiden taught a post-doctoral course in Nonlinear Multivariate Analysis. The course content was sort-of-published, in Dutch, as Gifi (1980). The course was repeated in 1981, and this time the sort-of-published version (Gifi (1981)) was in English.

The preface gives some details about the author.

> The text is the joint product of the members of the Department of Data Theory of the Faculty of Social Sciences, University of Leiden. 'Albert Gifi' is their 'nom de plume'. The portrait, however, of Albert Gifi shown here, is that of the real Albert Gifi to whose memory this book is dedicated, as a far too late recompense for his loyalty and devotion, during so any years, to the Cause he served.

Roughly ten years later a revised version of these course notes came out as an actual book in the *Wiley Series in Probabilty and Mathematical Statistics* (Gifi (1990)). This despite the fact that the contents of the book had very little to do with either probability or mathematical statistics. The book is organized around a series of computer programs for correspondence analysis, principal component analysis, and canonical analysis. The programs, written in FORTRAN, are called HOMALS, PRINCALS, PRIMALS, CRIMINALS, CANALS, OVERALS because they combine classical linear multivariate analysis with optimal transformation of the variables, using alternating least squares (or ALS). It serves, to some extent, as a manual for the programs, but it also discusses the properties of the techniques implemented in the programs, and it presents many detailed applications of these techniques.

Reviewers generally had some difficulties separating the wheat from the chaff.

> As the spirit of Albert Gifi has faded away, so has his whimsical approach to publishing, and his latest book is an idiosyncratic account of multivariate methods developed by the Leiden group during the 1970s. The names of their computer programs are distinguished by the ending ~ALS, thus we have OVERALS, PRINCALS, HOMALS, CANALS, MORALS, MANOVALS, CRIMINALS, PARTALS and PATHALS. Perhaps if you have a warped mind like this reviewer, you will turn rapidly to CRIMINALS. What can it be ? Surely it must give some illicit view of the truth about the world, a vision of the underworld of multivariate analysis ? Alas no ! It turns out only to be a synonym of Canonical Variate Analysis, sometimes known as Multiple Discriminant Analysis. Likewise HOMALS turns out to be Reciprocal Averaging, otherwise known as Correspondence Analysis. (Hill (1990))

This ambiguity and confusion are not too surprising. The Gifi book was a summary of the work of a large number of people, over a period of almost 20 years. Nevertheless, and perhaps because of this, it is somewhat of a *camel*, which we define for our purposes as a *horse designed by a committee*. Different chapters had different authors, and the common ideas behind the various techniques were not always clearly explained.

> In Gifi's MVA the criterion called "meet" loss plays a central role. Although the adoption of this criterion is one of the most important contributions of Gifi, the book would have been much more readable if this criterion had been introduced right at the outset and was followed throughout the rest of the book. (Takane (1992))

Nevertheless there is much original material in Gifi (1990), and the book has early applications of alternating least squares, majorization, coordinate descent, the delta method, and the bootstrap. And it emphasizes throughout the idea that statistics is about techniques, not about models. But, yes, the organization leaves much to be desired. An on demand printing of the first

and only edition is now available on Amazon for $ 492 – although of course used versions go for much less.

The book was published by a prestiguous publisher in a prestiguous series, but it is fair to say it never really caught on. It is not hard to understand why. The content, and the style, are unfamiliar to statisticians and mathematicians. There is no inference, no probability, and very little rigor. The content is in multivariate data analysis, which would be most at home these days, if anywhere, in a computer science department. The Gifi group did not have the resources of, say, Benzécri in France or Hayashi in Japan. The members were mostly active in psychometrics, a small and insular field, and they were from The Netherlands, a small country prone to overestimate its importance (Marvell (1653)). They also did not have the evangelical zeal necessary for creating and sustaining a large impact.

There have been some other major publication events in the Gifi saga. Around the same time as the Wiley book there was the publication of SPSS (1989). Starting in the late seventies the Gifi FORTRAN programs had been embedded in the SPSS system. The *SPSS Categories* manual was updated many times, in fact every time SPSS or IBM SPSS had a new release. Over the years other programs produced by the Department of Data Theory were added. A recent version is, for example, Meulman and Heiser (2012), corresponding to IBM SPSS 21. It acknowledges the contributions of some of the members of the Gifi team – but in IBM (2015), the version for IBM SPSS 23, these acknowledgements and the names of the authors have disappeared. Sic transit gloria mundi.

Michailidis and De Leeuw (1998) made an attempt to make the Gifi material somewhat more accessible by publishing a review article in a widely read mainstream statistical journal. Another such attempt is De Leeuw and Mair (2009a), in which the homals package for R is introduced. The homals package is basically a single monolithic R function that can do everything the Gifi programs can do, and then some. In both cases, however, the problem remained that the techniques, and the software, were too convoluted and too different from what both statisticians and users were accustomed to.

Van der Heijden and Van Buuren (1916) give an excellent, though somewhat wistful, historic overview of the Gifi project. It is too early for eulogies, however, and we refuse to give up. This book is yet another reorganization of the Gifi material, with many extensions. We take Yoshio Takane's advice

seriously, and we organize both the theory and the algorithms around what is called "meet-loss" in Gifi. In our software we separate the basic computational engine from its various applications that define the techniques of *Multivariate Analysis with Optimal Scaling (MVAOS)*. Hiding the core makes it possible to make the programs behave in much the same way as traditional MVA programs. The software is written in R (R Core Team (2016)), with some parts of the computational engine written in C.

The book itself is written in Rmarkdown, using bookdown (Xie (2016)) and knitr (Xie (2015)) to embed the computations and graphics, and to produce html and pdf versions that are completely reproducible. The book and all the files that go with it are in the public domain.

We would like to acknowledge those who have made substantial contributions to the Gifi project (and its immediate ancestors and offspring) over the years. Some of them are lost in the mists of time, some of them are no longer on this earth. They are, in alphabetical order, Bert Bettonvil, Jason Bond, Catrien Bijleveld, Frank Busing, Jacques Commandeur, Henny Coolen, Steef de Bie, Jan de Leeuw, John Gower, Patrick Groenen, Chris Haveman, Willem Heiser, Abby Israels, Judy Knip, Jan Koster, Pieter Kroonenberg, Patrick Mair, Adriaan Meester, Jacqueline Meulman, George Michailidis, Peter Neufeglise, Dré Nierop, Ineke Stoop, Yoshio Takane, Stef van Buuren, John van de Geer, Gerda van den Berg, Eeke van der Burg, Peter van der Heijden, Anita van der Kooij, Ivo van der Lans, Rien van der Leeden, Jan van Rijckevorsel, Renée Verdegaal, Peter Verboon, Susañña Verdel, and Forrest Young.

Chapter 1

Introduction

1.1 Some Dualisms

The type of multivariate analysis (MVA) we discuss in this book is sometimes called *descriptive* or *exploratory*, as opposed to *inferential* or *confirmatory*. It is located somewhere on the line between computational linear algebra and statistics, and it is probably close to data analysis, Big Data, machine learning, knowledge discovery, data mining, business analytics, or whatever other ill-defined label is used for the *mode du jour*.

In the days of Gifi (1990) there was a small-scale civil war between the mathematical statistical (confirmatory) approach to MVA and the data analytical (exploratory) approach. This is not a new conflict, because it has its roots in the Pearson-Yule debate (Mackenzie (1978)). The first shots in modern times were probably fired by Tukey (1962), but much additional polemic heat was generated in the 50 years since Tukey's famous paper. In order to stand our ground we were forced to participate, for example with De Leeuw (1984a), De Leeuw (1988a), De Leeuw (1990).

Here is what Gifi (1990) says, clearly with some intent to provoke.

> The statistical approach starts with a statistical model, usually based on the multinormal distribution. The model is assumed to be true, and within the model certain parametric hypotheses are constructed. The remaining free parameters are estimated and the hypotheses are tested. (Gifi (1990), p. 19)

> The data analytic approach does not start with a model, but looks for transformations and combinations of the variables with the explicit purpose of representing the data in a simple and comprehensive, and usually graphical, way. (Gifi (1990), p. 19)

Gifi's first chapter, in particular his section 1.5, outlines an approach to statistics that emphasizes *techniques* over *models*. A technique is a map of data into representations of some sort. Representations can be test statistics, confidence intervals, posterior distributions, tables, graphs. *Statistics studies the construction, properties, and performance of techniques.* Models are used to *gauge* techniques, that is to see how they perform on *synthetic data*, which are data described by equations or generated by sampling. Of course models can also be used to *inspire* techniques, but data analysis does not deal with the inspirational phase. The models themselves are a part of the client sciences, not of statistics. One of the key characteristics of a technique is its *stability*, which is studied by using data perturbations of various sorts. Small and unimportant perturbations of the data should lead to small and unimportant changes in the output. A large and important class of perturbations is based on sampling from a population, leading to sampling distributions, confidence intervals, hypothesis tests, and standard errors. In Gifi's branch of statistics the emphasis shifts from equations to algorithms, and from explanation to prediction.

In related philosophizing Breiman (2001) contrasted the *two cultures* of data modeling (98% of statisticians) and algorithmic modeling (2% of statistians), and implored statisticians to spend less time and energy in the first culture and more in the second.

> Reading a preprint of Gifi's book (1990) many years ago uncovered a kindred spirit. (Breiman (2001), p. 205)

This was written some time after the influential paper by Breiman and Friedman (1985), which introduced the Gifi-like ACE technique for multiple regression.

The emphasis in the data modeling culture is on explanation or information, the emphasis in the algorithmic modeling culture on prediction. There are various ways to present and evaluate this distinction. A good overview, from

the philosophy of science point of view, is Shmueli (2010). From the lofty heights of Academia we hear

> The two goals in analyzing data which Leo calls prediction and information I prefer to describe as "management" and "science." Management seeks *profit*, practical answers (predictions) useful for decision making in the short run. Science seeks *truth*, fundamental knowledge about nature which provides understanding and control in the long run. (Parzen, in the discussion of Breiman (2001), p. 224)

The emphasis on techniques was also shared by Cleveland (2014), who proposed a new curriculum for statistics departments with more emphasis on computing with data and tool evaluation. Another early ally was Laurie Davis, see the interesting papers by Davies (1995) and Davies (2008).

> The first chapter of Gifi (1990) contains an interesting discussion of statistical practice with special reference to multivariate data. The point of view taken there, with its emphasis on 'techniques', has points of contact with the present paper where we use Tukey's nomenclature and refer to 'procedure'. (Davies (2008), p. 192)

Of course currently the big discussion is if Data Science is actually statistics under a new name. And, more importantly, who should teach it. And, even more importantly, which department should receive the grant money. Parzen may believe that statisticians seek the Truth, whatever that is, but the current situation in Academia is that there is no truth if you do not consider profit. Statistics departments are typically small, and they feel threatened by gigantic Schools of Engineering looming over them (Association (2014), Yu (2014)). It is partly a question of scale: there are too many data to fit into statistics departments. Numerous new graduate Data Science programs are popping up, in many cases geared toward management and not so much toward science. Statistics departments are seriously considering changing their names, before the levies break and they are flooded by the inevitable rise of data.

We shall not pay much attention any more to these turf and culture wars, because basically they are over. Data analysis, in its multitude of disguises

and appearances, is the winner. Classical statistics departments are gone, or on their way out. They may not have changed their name, but their curricula and hiring practices are very different from what they were 20 or even 10 years ago.

> Neither do men put new wine into old bottles: else the bottles break, and the wine runneth out, and the bottles perish: but they put new wine into new bottles, and both are preserved. (Matthew 9:17)

Notwithstanding the monumental changes, inferential statistics remains an important form of stability analysis for data analysis techniques. Probabilistic models are becoming more and more important in many branches of science, and perturbing a probabilistic model is most naturally done by sampling. Thus huge parts of classical statistrucs are preserved, and not surprisingly these are exactly the parts useful in data analysis.

1.2 Quantifying Qualitative Data

One way of looking at *Multivariate Analysis with Optimal Scaling*, or *MVAOS*, is as an extension of classical linear multivariate analysis to variables that are binary, ordered, or even unordered categorical. In R terminology, classical MVA techniques can thus be applied if some or all of the variables in the dataframe are *factors*. Categorical variables are *quantified* and numerical variables are *transformed* to optimize the linear or bilinear least squares fit.

Least squares and eigenvalue methods for quantifying multivariate qualitative data were first introduced by Guttman (1941), although there were some bivariate predecessors in the work of Pearson, Fisher, Maung, and Hirschfeld (see De Leeuw (1983) or Gower (1990) for a historical overview). In this earlier work the emphasis was often on optimizing quadratic forms, or ratios of quadratic forms, and not so much on least squares, distance geometry, and graphical representations such as *biplots* (Gower and Hand (1996), Gower, Le Roux, and Gardner-Lubbe (2015), Gower, Le Roux, and Gardner-Lubbe (2016)). They were taken up by, among others, De Leeuw (1968a), by Benzécri and his students in France (see Cordier (1965)), and by Hayashi and

his students in Japan (see Tanaka (1979)). Early applications can be found in ecology, following an influential paper by Hill (1974). With increasing emphasis on software the role of graphical representations has increased and continues to increase.

In De Leeuw (1974) a first attempt was made to unify most classical descriptive multivariate techniques using a single least squares loss function and a corresponding *alternating least squares (ALS)* optimization method. His work then bifurcated to the *ALSOS project*, with Young and Takane at the University of North Carolina Chapell Hill, and the *Gifi project*, at the Department of Data Theory of Leiden University.

The ALSOS project was started in 1973-1974, when De Leeuw was visiting Bell Telephone Labs in Murray Hill. ALSOS stands for Alternating Least Squares with Optimal Scaling. The ALS part of the name was provided by De Leeuw (1968b) and the OS part by Bock (1960). At early meetings of the Psychometric Society some members were offended by our use of "Optimal Scaling", because they took it to imply that their methods of scaling were supposedly inferior to ours. But the "optimal" merely refers to optimality in the context of a specific least squares loss function.

Young, De Leeuw, and Takane applied the basic ALS and OS methodology to conjoint analysis, regression, principal component analysis, multidimensional scaling, and factor analysis, producing computer programs (and SAS modules) for each of the techniques. An overview of the project, basically at the end of its lifetime, is in Young, De Leeuw, and Takane (1980) and Young (1981).

The ALSOS project was clearly inspired by the path-breaking work of Kruskal (1964a) and Kruskal (1964b), who designed a general way to turn metric multivariate analysis techniques into non-metric ones. In fact, Kruskal applied the basic methodology developed for multidimensional scaling to linear models in Kruskal (1965), and to principal component analysis in Kruskal and Shepard (1974) (which was actually written around 1965 as well). In parallel developments closely related nonmetric methods were developed by Roskam (1968) and by Guttman and Lingoes (see Lingoes (1973)).

The Gifi project took its inspiration from Kruskal, but perhaps even more from Guttman (1941) (and to a lesser extent from the optimal scaling work of Fisher, see Gower (1990)). Guttman's quantification method, which later

became known as *multiple correspondence analysis*, was merged with linear and nonlinear principal component analysis in the HOMALS/PRINCALS techniques and programs (De Leeuw and Rijckevorsel (1980)). The MVAOS loss function that was chosen ultimately, for example in the work of Burg, De Leeuw, and Verdegaal (1988), had been used earlier by Carroll (1968) in multi-set canonical correlation analysis of numerical variables.

A project similar to ALSOS/Gifi was ACE, short for *Alternating Conditional Expectations*. The ACE method for regression was introduced by Breiman and Friedman (1985) and the ACE method for principal component analysis by Koyak (1987). Both techniques use the same ALS block relaxation methods, but instead of projecting on a cone or subspace of possible transformation, they apply a smoother (typically Friedman's supersmoother) to find the optimal transformation. This implies that the method is intended primarily for continuous variables, and that the convergence properties of the ACE algorithm are more complicated than those of a proper ALS algorithms.

An even more closely related project, by Winsberg and Ramsay, uses the cone of I-splines (integrated B-splines) to define the optimal transformations. The technique for linear models is in Winsberg and Ramsay (1980) and the one for principal component analysis in Winsberg and Ramsay (1983). Again, the emphasis on monotonic splines indicates that continuous variables play a larger role than in the ALSOS or Gifi system.

So generally there have been a number of projects over the last 50 years that differ in detail, but apply basically the same methodology (alternating least squares and optimal scaling) to generalize classical MVA techniques. Some of them emphasize transformation of continuous variables, some emphasize quantification of discrete variables. Some emphasize monotonicity, some smoothness. Usually these projects include techniques for regression and principal component analysis, but in the case of Gifi the various forms of correspondence analysis and canonical analysis are also included.

1.3 Beyond Gifi

The techniques discussed in Gifi (1990), and implemented in the corresponding computer programs, use a particular least squares loss function and minimize it by alternating least squares algorithms. All techniques use what

Gifi calls *meet loss*, which is basically the loss function proposed by Carroll (1968) for multiset canonical correlation analysis. Carroll's work was extended in Gifi by using optimal scaling to transform or quantify variables coded with indicators, and to use constraints on the parameters to adapt the basic technique, often called *homogeneity analysis*, to different classical MVA techniques.

There have been various extensions of the classical Gifi repertoire by adding techniques that do not readily fit into meet loss. Examples are path analysis (Coolen and De Leeuw (1987)), linear dynamic systems (Bijleveld and De Leeuw (1991)), and factor analysis (De Leeuw (2004)). But adding these techniques does not really add up to a new framework.

Somewhat more importantly, De Leeuw and Rijckevorsel (1988) discuss various ways to generalize meet loss by using *fuzzy coding*. Transformations are no longer step functions, and coding can be done with fuzzy indicators, such as B-spline bases. This makes it easier to deal with variables that have many ordered categories. Although this is a substantial generalization the basic framework remains the same.

One of the outgrowths of the Gifi project was the *aspect* approach, first discussed systematically by De Leeuw (1988c), and implemented in the R package `aspect` by Mair and De Leeuw (2010). In its original formulation it uses *majorization* to optimize functions defined on the space of correlation matrices, where the correlations are computed over transformed variables, coded by indicators. Thus we optimize aspects of the correlation matrix over transformations of the variables. The `aspect` software was recently updated to allow for B-spline transformations (De Leeuw (2015a)). Many different aspects were implemented, based on eigenvalues, determinants, multiple correlations, and sums of powers of correlation coefficients. Unformately, aspects defined in terms of canonical correlations, or generalized canonical correlations, were not covered. Thus the range of techniques covered by the `aspect` approach has multiple regression and principal component analysis in common with the range of the Gifi system, but is otherwise disjoint from it.

In De Leeuw (2004) a particular correlation aspect was singled out that could bridge the gap between the aspect approach and the Gifi approach, provided *orthoblocks* of transformations were introduced. This is combined with the notion of *copies*, introduced in De Leeuw (1984b), to design a new class of techniques that encompasses all of Gifi and that brings generalized canonical

correlation analysis in the aspect framework. Thus correlation aspects, and the majorization algorithms to optimize them, are now a true generalization of the Gifi system.

*** This is the system we discuss in this book.

Chapter 2

Coding and Transformations

2.1 Variables and Multivariables

In the multivariate analysis techniques presented in this book the data are measurements or classifications of n *objects* by m *variables*. Perhaps it is useful to insert some definitions here. A *variable* is a function that maps a domain of objects to a range of values. Domains are finite. The elements of the domain can be individuals, animals, plants, time points, locations, and so on. It is useful to distinguish the *codomain* (or *range*) of a variable and its *image*. The codomain of a variable can be the real numbers, but the image always is a finite set, the actual values the variable assumes on the domain. A *multivariable* is a sequence of variables defined on the same domain, with possibly different codomains. Multivariables are implemented in R as *dataframes*. Variables can have a finite codomain, which can be either ordered or unordered. This corresponds with a *factor* or an *ordered factor* in R. MVAOS techniques *quantify* factors, replacing the values in the image by real numbers. If the variables are real-valued to start with we replace real numbers by other real numbers and we *transform* instead of *quantify*. The distinction between quantification and transformation is somewhat fluid, because the image of a variable is always finite and thus, in a sense, all variables are categorical (a point also emphasized, for example, in Holland (1979)).

Although the variables in a multivariable have the same domain, there can be different numbers of missing data for different variables. We handle this in

the same way as R, by adding `NA` to the range of all variables. In this context it is also useful to define *latent* or *unobserved* variables. These are variables for which all values are missing, i.e. for which the image only contains `NA`. At first thought it seems somewhat perverse to have such completely missing variables, but think of examples such as principal components, factor scores, or error terms in linear models.

2.2 Induced Correlations and Aspects

If all categorical variables are quantified and all numerical variables are transformed we can compute the *induced correlation matrix* of the transformed and quantified variables. In the forms of MVAOS we consider in this book the statistics we compute, except for the transformations themselves, are usually functions of this induced correlation matrix. This means that they are functions of the second order relations between the variables, or, in order words, they are *joint bivariate*. Higher order moments and product moments are ignored. Different multivariate distributions with the same bivariate marginals will give the same MVAOS results.

2.3 Transformed Variables

The data are collected in the $n \times m$ matrix H, which codes the observations on the m variables. MVAOS does not operate on the data directly, but on *transformations* or *quantifications* of the variables. Choosing a transformation to minimize a loss function is known as *optimal scaling*. Clearly this so-called optimality is only defined in terms of a specific loss function, with specific constraints. Different constraints and different loss functions will lead to different optimal transformations.

Let us define the types of transformations we are interested in. The $n \times m$ matrix of *transformed variables* H has columns h_j, which are constrained by $h_j = G_j z_j$, where G_j is a given matrix defining the *basis* for variable j. In addition we require $h_j \in \mathcal{C}_j$ and $h_j \in \mathcal{S}$, where $\mathcal{C}_j$ is a *cone of transformations* and $\mathcal{S}$ is the unit sphere in $\mathbb{R}^n$. This will be discussed in more detail in later sections, but for the time being think of the example in which h_j is required to be a (centered and normalized) monotone polynomial function

of the image values of variable j. The whole of $\mathbb{R}^n$ and a single point in $\mathbb{R}^n$ are both special cases of these normalized cones. It is important, especially for algorithm construction, that the restrictions are defined for each variable separately. An exception to this rule is the *orthoblock*, using terminology from De Leeuw (2004), which requires that all or some of the columns of H are not only normalized but also orthogonal to each other. Clearly a normalized variable is an orthoblock of size one.

2.4 Bases

In earlier MVAOS work, summarized for example in Gifi (1990) or Michailidis and De Leeuw (1998), the basis matrices G_j were binary zero-one matrices, indicating category membership. These matrices are also known as *indicator matrices*. The same is true for the software in IBM SPSS Categories (Meulman and Heiser 2012) or in the R package homals (De Leeuw and Mair 2009a). In this paper we extend the current MVAOS software using *B-spline bases*, which provide a form of fuzzy non-binary coding suitable for both categorical and numerical variables (Rijckevorsel and De Leeuw 1988). B-spline basis were already discussed for some special cases in De Leeuw, Rijckevorsel, and Wouden (1981) and Gifi (1990), but corresponding easily accessible software was never released.

In this book we continue to use the term *indicators* for bases. Thus bases G_j must be non-negative, with rows that add up to one. If there is only one non-zero entry in each row, which of course is then equal to one, the indicator is *crisp*, otherwise it is *fuzzy*. B-spline bases are the prime example of fuzzy indicators, but other examples are discussed in Rijckevorsel and De Leeuw (1988). Only B-spline bases are implemented in our software, however.

Note that the identity matrix is a crisp indicator. This is of importance in connection with missing data and orthoblocks.

2.5 Copies and Rank

Within a block there can be more than one version of the same variable. These multiple versions are called *copies*. They were first introduced into the

Gifi framework by De Leeuw (1984b). Since MVAOS transforms variables, having more than one copy is not necessarily redundant, because different copies can and will be transformed differently. As a simple example of copies, think of using different monomials or orthogonal polynomials of a single variable x in a polynomial regression. The difference between copies and simply including a variable more than once is that copies have the same basis G_j.

In the Gifi algorithms copies of a variable are treated in exactly the same way as other variables. The notion of copies replaces the notion of the *rank of a quantification* used in traditional Gifi, which in turn generalizes the distinction between *single* and *multiple* quantifications. A single variable has only one copy in its block, a multiple variable has the maximum number of copies.

In our software the copies of a variable by definition have the same basis. It is possible, of course, to include the same variable multiple times, but with different bases. This must be done, however, at the input level. In terms of the structures defined in the software, a gifiVariable can have multiple copies but it only has one basis. If there is more than one basis for a variable, then we need to define an additional gifiVariable. Also note that copies of a variable are all in the same block. If you want different versions of a variable in different blocks, then that requires you to create different gifiVariables.

Defining copies is thus basically a coding problem. It can be handled simply by adding a variable multiple times to a data set, and giving each variable the same bases. In our algorithm we use the fact that copies belong to the same variable to create some special shortcuts and handling routines.

Ordinality restrictions on variables with copies require some special attention. In our current implementation we merely require the first copy to be ordinal with the data, the other copies are not restricted. Once again, if you want ordinal restrictions on all copies you need to create separate gifiVariables for each copy.

2.6 Orthoblocks

If a variable has more than one copy, then we require without loss of generality that the transformations are orthogonal.

2.7 Constraints

As discussed earlier, each variable has a cone of transformations associated with it, and we optimize over these transformations. In ALSOS and classical Gifi the three type of transformation cones considered are *nominal, ordinal,* and *numerical*. Our use of B-splines generalizes this distinction, because both numerical and nominal can be implemented using splines. What remains is the choice for the degree of the spline and the location of the knots.

Choice of degree and knots is basically up to the user, but the programs have some defaults. In most cases the default is to use crisp indicators with knots at the data points. Of course for truly categorical variables (i.e. for factors in R) crisp indicators are simply constructed by using the levels of the factor. We include some utilities to place knots at percentiles, or equally spaced on the range, or to have no interior knots at all (in which case we fit polynomials).

And finally the user decides, for all variables, if she wants the transformations (step functions, splines, and polynomials) to be monotonic with the data. Default is not requiring monotonicity.

Note that we require the spline to be monotonic in the non-missing data points – this does not mean the spline is monotonic outside the range of the data (think, for example, of a quadratic polynomial), it does not even mean the spline is monotonic between data points. This makes our spline transformations different from the integrated B-splines, or I-splines, used by Winsberg and Ramsay (1983), which are monotone on the whole real line. Because each variable has a finite image we are not really fitting a spline, we are fitting a number of discrete points that are required to be on a spline, and optionally to be monotonic with the data. In Winsberg and Ramsay (1983) the requirement is that the fitted points are on an I-spline, which automatically makes them monotonic with the data. Clearly our approach is the less restrictive one.

2.8 Missing Data

The utility `makeMissing()` expands the basis for the non-missing data in various ways. Option “m” (for “multiple”) is the default. It replaces the

basis with the direct sum of the non-missing basis and an identity matrix for the missing elements. Option "s" (for "single") adds a single binary column to the basis indicating which elements are missing. Option "a" (for "average") codes missing data by having all the elements in rows of the basis corresponding with missing data equal to one over the number of rows. With all three options the basis remains an indicator. Some of these options make most sense in the context of crisp indicators, where they are compared in Meulman (1982).

So suppose the data are

```
##      [,1]
## [1,] -0.50
## [2,]    NA
## [3,]  0.75
## [4,]  0.99
## [5,]    NA
```

Create a basis for the non-missing values with

```
mprint(basis <- bsplineBasis(x[which(!is.na(x))],1,c(-1,0,1)))
```

```
##      [,1]  [,2]  [,3]
## [1,]  0.50  0.50  0.00
## [2,]  0.00  0.25  0.75
## [3,]  0.00  0.01  0.99
```

The three different completion options for missing data give

```
mprint (makeMissing (x, basis, missing = "m"))
```

```
##      [,1]  [,2]  [,3]  [,4]  [,5]
## [1,]  0.50  0.50  0.00  0.00  0.00
## [2,]  0.00  0.00  0.00  1.00  0.00
## [3,]  0.00  0.25  0.75  0.00  0.00
## [4,]  0.00  0.01  0.99  0.00  0.00
## [5,]  0.00  0.00  0.00  0.00  1.00
```

```
mprint (makeMissing (x, basis, missing = "s"))
```

```
##      [,1]  [,2]  [,3]  [,4]
## [1,]  0.50  0.50  0.00  0.00
## [2,]  0.00  0.00  0.00  1.00
## [3,]  0.00  0.25  0.75  0.00
## [4,]  0.00  0.01  0.99  0.00
## [5,]  0.00  0.00  0.00  1.00
```

```
mprint (makeMissing (x, basis, missing = "a"))
```

```
##      [,1]  [,2]  [,3]
## [1,]  0.50  0.50  0.00
## [2,]  0.33  0.33  0.33
## [3,]  0.00  0.25  0.75
## [4,]  0.00  0.01  0.99
## [5,]  0.33  0.33  0.33
```

The default option for missing data in the previous version of the Gifi system was "missing data deleted", which involves weighting the rows in the loss functions by the number of non-missing data in that row. This leads to some complications, and consequently we have no option "d" in this version of Gifi.

2.9 Active and Passive Variables

If a variable is *passive* (or *supplementary*) it is incorporated in the analysis, but it does not contribute to the loss. Thus an analysis that leaves the passive variables out will give the same results for the active variables. Passive variables are transformed like all the others, but they do not contribute to the block scores, and thus not to the loss. They have category quantifications and scores, and can be used in the corresponding plots.

If all variables in a block are passive, then the whole block does not contribute to the loss. This happens specifically for singletons, if the single variable in the block is passive.

2.10 Interactive Coding

One of the major contributions of *Analyse des Données* is the emphasis on *coding*, which in our context can be defined as choosing how to represent the raw data of an experiment in an actual data frame (and, to a lesser extent, how to choose blocks, number of copies, dimensionality, degrees, and knots). In the section we discuss one important coding variation. Suppose we have n observations on two factors, one with p levels and one with q levels. Then the data can be coded as n observations on one factor with $p \times q$ levels, and we can construct a corresponding crisp indicator. The same reasoning applies to more than two categorical variables, which we can always code *interactively*. It also applies to bases for numerical variables, where we can define an interactive basis by using products of columns from the bases of each of the variables.

If $G = \{g_{is}\}$ and $H = \{h_{it}\}$ are two indicators of dimensions $n \times m_g$ and $n \times m_h$, then the $n \times m_g m_h$ matrix with elements {g_{is}h_{it}} is again an indicator: the elements are non-negative, and rows add up to one.

```
mprint (x <- bsplineBasis (1:9/10, 1, .5))
```

```
##       [,1]  [,2]  [,3]
##  [1,]  1.00  0.00  0.00
##  [2,]  0.75  0.25  0.00
##  [3,]  0.50  0.50  0.00
##  [4,]  0.25  0.75  0.00
##  [5,]  0.00  1.00  0.00
##  [6,]  0.00  0.75  0.25
##  [7,]  0.00  0.50  0.50
##  [8,]  0.00  0.25  0.75
##  [9,]  0.00  0.00  1.00
```

```
mprint (y <- makeIndicator (c (rep (1, 5), rep (2, 4))))
```

```
##       [,1]  [,2]
##  [1,]  1.00  0.00
##  [2,]  1.00  0.00
```

```
##  [3,]  1.00  0.00
##  [4,]  1.00  0.00
##  [5,]  1.00  0.00
##  [6,]  0.00  1.00
##  [7,]  0.00  1.00
##  [8,]  0.00  1.00
##  [9,]  0.00  1.00
```

```
mprint (makeColumnProduct (list (x, y)))
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]
##  [1,]  1.00  0.00  0.00  0.00  0.00  0.00
##  [2,]  0.75  0.00  0.25  0.00  0.00  0.00
##  [3,]  0.50  0.00  0.50  0.00  0.00  0.00
##  [4,]  0.25  0.00  0.75  0.00  0.00  0.00
##  [5,]  0.00  0.00  1.00  0.00  0.00  0.00
##  [6,]  0.00  0.00  0.00  0.75  0.00  0.25
##  [7,]  0.00  0.00  0.00  0.50  0.00  0.50
##  [8,]  0.00  0.00  0.00  0.25  0.00  0.75
##  [9,]  0.00  0.00  0.00  0.00  0.00  1.00
```

Chapter 3

Aspects

3.1 Definition

An *aspect* is a real valued function ϕ defined on the compact convex set $\mathcal{R}^{m\times m}$ of correlation matrices of order m. Note that a *correlation matrix* is a positive semi-definite matrix with ones on the diagonal.

In De Leeuw (2004) a class of MVAOS techniques is defined by optimizing aspects, using majorization algorithms. Optimization is over a set $\mathcal{R}$ of correlation matrices, usually the correlation matrices that correspond with admissible transformations of the data. See De Leeuw (1988b) and De Leeuw, Michailidis, and Wang (1999) for additional results on aspects. Software in R that optimizes general aspects is discussed by Mair and De Leeuw (2010).

The aspect optimization algorithm is based on majorization, and assumes that the aspect that is maximized is a convex function on the space of correlation matrices (or, equivalently, that the aspect is concave and minimized). Some examples of interesting convex aspects are:

- The sum of the p largest eigenvalues of the correlation matrix (as in principal component analysis).
- The squared multiple correlation (SMC) of one variable with the others (as in multiple regression).
- The sum of some SMC's over some or all variables (as in path analysis).

There are also some convex aspects are not directly associated with a standard multivariate technique.

- The sum of the p^{th} powers of the correlation coefficients, with $p \geq 1$.
- The sum of the p^{th} powers of the absolute values of the correlation coefficients, with $p \geq 1$.
- Any norm on the space of correlation matrices.

Another interesting aspect, related to multinormal maximum likelihood estimation, is

$$\phi(R) = \min_{\Gamma \in \mathcal{G}} \log \mathbf{det}(\Gamma) + \mathbf{tr}\ R\Gamma^{-1},$$

where $\mathcal{G}$ is some (possibly parametrized) subset of the correlation matrices. For instance, $\mathcal{G}$ could be all matrices satisfying some factor analysis or structural equations model. To compute ϕ we have to calculate the multinormal maximum likelihood estimate of the model for given R. The aspect ϕ is concave in R, so in our framework we minimize it over $R \in \mathcal{R}$.

3.2 Stationary Equations

The stationary equations when optimizing a differentiable aspect ϕ over the centered and standardized transformations in x_j are

$$\sum_{\ell=1}^{m} \frac{\partial \phi}{\partial r_{j\ell}} \mathbf{E}(x_\ell | x_j) = \lambda_j x_j$$

3.3 Bilinearizability

Chapter 4

Pattern Constraints and Gifi Loss

4.1 Aspects from Patterns

MVAOS is a linear multivariate technique in the sense that it makes linear combinations of transformed variables, and it is a nonlinear multivariate technique in the sense that these transformations are generally nonlinear. The coefficients of the linear combinations are collected in a matrix A, which we call the *pattern*. There are L linear combinations of the m variable blocks, and consequently there are mL submatrices $A_{j\ell}$. L is the *number of equation blocks*. Constraints on the pattern largely define the technique. The typical situation is that either $A_{j\ell}$ is free to vary over all matrices of the appropriate dimensions, or $A_{j\ell}$ is equal to a fixed matrix, usually either the identity or zero. But more complicated constraints on the $A_{j\ell}$ are sometimes also necessary.

An *MVAOS System* is a bilinear homogeneous system in the transformed variables H and the pattern A of the form $HA = 0$. There is no assumption that for actual data this system has a non-trivial solution. We will look for approximate solutions, using a least squares loss function. Thus we define *Gifi Multivariate Analysis*, or *MVAOS*, as the minimization of the loss function

$$\sigma(H, A) = \sum_{\ell=1}^{L} \mathbf{SSQ}\ (\sum_{j=1}^{m} H_j A_{j\ell}), \tag{4.1}$$

over H and A, under suitable restrictions. Here **SSQ**() is the (unweighted) sum of squares. The usual restriction on A is that for each block of equations ℓ there is at least one block of variables j such that $A_{j\ell} = I$.

If we write the MVAOS system simply as $HA = 0$ the loss function becomes $\sigma(H, A) = \mathbf{tr}\ A'R(H)A$, with $R(H) =: H'H$ the *induced correlation matrix* of the transformed variables in H.

In order to make MVAOS systems less mysterious we give three examples, choosing the names of the parameters to fit the problem. This is also an opportunity to sprinkle some more acronyms around. The first is multivariate linear regression (MLR). Its MVAOS system is

$$\begin{bmatrix} Y & X \end{bmatrix} \begin{bmatrix} I \\ -B \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix},$$

which means we minimize $\mathbf{SSQ}(Y - XB)$ over B and possibly over transformations of the columns of X and Y. If we require that $\mathbf{rank}(B) = p$, with p less than the minimum of the number of rows and columsn of B, then this becomes reduced rank regression (RRR). The second example is principal component analysis (PCA). This has the same MVAOS system as MLR, but the minimization over X is over all orthoblocks, i.e. all X such that $X'X = I$. The final example for now is exploratory factor analysis (EFA). Its MVAOS system is

$$\begin{bmatrix} Y & F & U \end{bmatrix} \begin{bmatrix} I \\ -A \\ -\Delta \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix},$$

and we minimize $\mathbf{SSQ}(Y - FA - U\Delta)$, with the constraint that $(F \mid U)$ is an orthoblock and that Δ is diagonal.

4.2 Gifi Loss

For embedding the previous Gifi work in our new framework we define a specific class of MVAOS systems, called *Gifi systems*. They are of the form

$$\begin{bmatrix} X & H_1 & \cdots & H_m \end{bmatrix} \begin{bmatrix} I & 0 & \cdots & 0 \\ -A_1 & I & \cdots & 0 \\ 0 & -A_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I \\ 0 & 0 & \cdots & -A_m \end{bmatrix}$$

and thus *Gifi loss* is

$$\sigma(X, H, A) = \sum_{j=1}^{m} \mathbf{SSQ}\ (X - H_j A_j). \tag{4.2}$$

In (4.2) the matrix X is an orthoblock, which contains the *object scores*. Note that in Gifi loss each variable block j corresponds with a unique submatrix A_j, except for the object scores block, which contributes to all equation blocks. In Gifi systems the A_j are generally unconstrained.

There is some additional terminology that is more or less specific to Gifi loss. The *variable scores* are $V_k =: h_k a_k' = G_k z_k a_k'$, and the *block scores* are $U_j =: H_j A_j = \sum_{k \in \mathcal{K}_j} V_k$. The *category quantifications* are $Y_k =: z_k a_k'$, so that $V_k = G_k Y_k$. Note that both variable scores and category quantifications, as defined here, are of rank one. In other words, their columns as well as their rows are proportional to each other.

A Gifi solution can be associated with various sets of *loadings*, i.e. correlations between observed variables and constructed (or latent) variables, in this case object scores. Since both X and H_j are centered and normalized the *variable loadings* for block j are simply the cross-product $X'H_j$. Because optimal A_j is the linear least squares solution for given X and H_j we have $H_j'(X - H_j A_j) = 0$, which means the loadings are equal to the covariances between transformed variables and block scores. Each block has a *discrimination matrix*, defined as $\Delta_j =: A_j' H_j' H_j A_j = A_j' H_j' X = X' H_j H_j^+ X$, with H_j^+ the Moore-Penrose inverse. The diagonal $\Lambda_j =: \mathbf{diag}(\Delta_j)$ of the discrimination matrix, the *discrimination measures*, are the variances of the block scores. Thus the

block loadings, the correlations between transformed variables H_j and block scores U_j are equal to the correlations between the object scores and the block scores, and are given by $H_j'U_j\Lambda_j^{-\frac{1}{2}} = X'U_j\Lambda_j^{-\frac{1}{2}}$.

Loss function (4.2) can be interpreted geometrically. Zero loss, i.e. solvability of the Gifi system, means that the scores x_i for object i coincide with the j block scores u_{ij}, which consequently coincide with each other. This is why Gifi analysis is also called *homogeneity analysis*, because we transform and combine the variables in such a way that the block scores are as homogeneous as possible. If we plot the n object scores x_i and the n block scores u_{ij} in a p-dimensional plot, then we want to make the squared distances $\mathbf{SSQ}(x_i - u_{ij})$ summed over all i and j to be as small as possible.

4.3 Associated Eigenvalue Problems

Associated with the problem of minimizing loss function (4.2) are some eigenvalue and singular value problems defined by the matrices H_j. This has been discussed in detail in Gifi (1990), and there are some more recent discussions in A. Tenenhaus and Tenenhaus (2011) and Van der Velden and Takane (2012).

We begin the section with some definitions, which are more or less standard in MVAOS. First $H =: (H_1 \mid H_2 \mid \cdots \mid H_m)$, and $C =: H'H$. The matrix C, which is called the *Burt matrix* in correspondence analysis, is a $p \times p$ block matrix, with $m \times m$ blocks define by $C_{j\ell} =: H_j'H_\ell$. We also use separate notation $D_j =: C_{jj} = H_j'H_j$ for the diagonal blocks in C, and for their direct sum $D =: D_1 \oplus \cdots \oplus D_m$. Finally A stacks the A_j on top of each other.

The stationary equations for minimizing σ over $X'X = I$ and A, for given H, are

$$H'X = DA, \tag{4.3}$$

$$HA = XM, \tag{4.4}$$

with $X'X = I$, and M an $r \times r$ symmetric matrix of Lagrange multipliers. It follows that

$$CA = DAM, \tag{4.5}$$

as well as

$$HD^+H'X = XM, \tag{4.6}$$

with $X'X = I$ and D^+ the Moore-Penrose inverse of D. It follows that $X = KT$, where K are eigenvectors corresponding with the r largest eigenvalues of HD^+H and L is an arbitrary rotation matrix. In the same way $A = LT$, where L are the eigenvectors corresponding with the r largest eigenvalues of D^+C. The non-zero eigenvalues of HD^+H' and D^+C are the same, both are equal to the squares of the singular values of $HD^{-\frac{1}{2}}$, with $D^{-\frac{1}{2}}$ the symmetric square root of D^+.

The result can be made a bit more intuitive by defining the orthogonal projectors $P_j =: H_jD_j^+H_j'$ and their average $P_\star$. Then X can be chosen as the normalized eigenvectors of $P_\star$ and, if λ_s are the corresponding ordered eigenvalues,

$$\min_{X'X=I}\min_{A}\sigma(X,A,H) = \sum_{s=1}^{r}(1-\lambda_s(P_\star)). \tag{4.7}$$

The eigenvalues in Λ are all between zero and one.

In MVAOS the fit of block j is called the *discrimination matrix*. It is defined as $\Delta_j =: X'P_jX = A_j'D_jA_j$. Note that the average discrimination measure $\Delta_\star$ is equal to the diagonal matrix Λ.

4.4 History

The history of loss function (4.1) is simple. Although numerous special cases have been used over the years, in its general form it only occurs, as far as we know, in De Leeuw (2004). It was designed to bridge the gap between (4.2) and linear systems such as RRR, MIMIC, EFA, and LISREL/EQS. It mainly differs from (4.2) in its systematic use of copies and orthoblocks.

The history of (4.2), on the other hand, is complicated. It is easiest to start with the special case in which all variables are numerical (in our system that means no internal knots and degree equal to one). In that case MVAOS is a form of *Generalized Canonical Correlation Analysis (GCCA)*, which extends canonical correlation analysis (CCA) to two or more blocks of variables.

The various GCCA techniques proposed over the years for computing p-dimensional solutions are either *simultaneous* or *successive*. In a successive

algorithm the loss function is only defined for $p = 1$. It is first optimized over all one-dimensional solutions. And then, for a subsequent dimension q, the one-dimensional criterion is optimized over all solutions that are orthogonal to solutions $1, \cdots, q-1$. In a simultaneous technique the loss function is defined for all p, and the solution is computed by minimizing over all p-dimensional solutions. In a successive solution the first p dimensions of a $p+1$ dimensional solution are the p dimensional soution, i.e. successive solutions are *nested*. Simultaneous solutions are generally not nested. On the other hand the successive p dimensional solution is usually not the best possible p dimensional solution.

GCCA starts with Horst (1961b), Horst (1961a). The techniques proposed by Horst are successive, which means his loss functions are only defined for one-dimensional solutions, specifically one-dimensional block scores $u_j = H_j a_j$. In Horst (1961a) four different techniques are proposed, with different loss functions, all defined as functions of the *induced correlation matrix* of the block scores. For our purposes, the interesting one is his method 2, in which the largest eigenvalue of the induced correlation matrix is maximized. Horst (1961b) uses a different criterium, the sum of the correlation coefficients, which is not related to the Gifi loss function in any simple way.

In a small paper, hidden away in a large proceedings volume, Carroll (1968) proposed a successive method maximizing $\sum_{j=1}^m \mathbf{cor}^2(x, H_j a_j)$ over a_j and the auxilary variable x. This turns out to be equivalent to method 2 of Horst. The work of Horst and Carroll was extended by Kettenring (1971) (in greater detail in Kettenring (1969)), who introduced several additional criteria, and baptized the Horst-Carroll method MAXVAR. In later work, it was shown by Gifi (1980) that minimizing $\sum_{s=1}^p \sum_{j=1}^m \mathbf{cor}^2(x_s, H_j a_{js})$ over $X'X = I$ and A gives the same result as successive MAXVAR. Also see M. Tenenhaus and Young (1985). We need one important qualification, using terminology introduced by Dauxois and Pousse (1976), which is that the successive method should use *weak orthogonality* $\sum_{j=1}^m a'_{js} H_j H_j a_{jt} = \delta^{st}$, with δ^{st} the Kronecker delta, and not *strong orthogonality*, which says that $a'_{js} H_j H_j a_{jt} = \delta^{st}$ for all j, s, t. More recently Kiers, Cléroux, and Ten Berge (1994) have shown that simultaneous/successive MAXVAR also optimizes various measures of correlation defined on matrix space.

The most important contribution of Gifi, however, is the switch from correlations and quadratic forms to least squares loss functions and Euclidean

distances, ultimately leading to the loss function (4.2). Undoubtedly this was partly due to the heavily geometrical approach to MVA we were taught by John van de Geer, the father of Albert Gifi (Van de Geer (1971)). Van de Geer was influenced in turn by Coombs (1964), who introduced another basically geometric approach for the representation of data. On the computational side there was the influence of multidimensional scaling, with its emphasis on distance, breaking through in in the late sixties and early seventies. Shepard, Kruskal, Gnanadesikan, and Kettenring all worked at Bell Telephone Laboratories in Murray Hill, and both De Leeuw and Benzécri had visiting positions there around that time.

In the classical Gifi system (Gifi (1990), Michailidis and De Leeuw (1998)) a slightly different parametrization of Gifi loss, and a correspondingly different ALS algorithm, were used. The loss function used by Gifi is

$$\sigma(X,Y) = \frac{1}{m}\sum_{j=1}^{m}\mathbf{SSQ}\ (X - \sum_{\ell\in K_j} G_\ell Y_\ell), \tag{4.8}$$

where the G_ℓ are known spanning matrices for the cones of transformations, and the Y_ℓ are matrices of *category quantifications*. Loss function (4.8) is geared more towards quantification of discrete categorical variables.

Because of the full rank decompositions $Y_{j\ell} = Z_{j\ell}A_{j\ell}$ it follows that (4.2) and (4.8) are essentially the same. Simply define $H_j = G_j Z_j$. We feel that the alternative parametrization in terms of H_j and A_j has some conceptual and computational advantages.

Chapter 5

Algorithm

5.1 Block Relaxation

Our task is to minimize $\sigma(H, A)$ over H and A, suitably constrained. Write the constraints as $H \in \mathcal{H}$ and $A \in \mathcal{A}$. The strategy we use is block relaxation ((**deleeuw_B_15?**)). Thus we iterate as follows.

0. Set $k = 0$ and start with some $H^{(0)}$.
1. $A^{(k)} \in \underset{A \in \mathcal{A}}{\textbf{argmin}}\ \sigma(H^{(k)}, A)$.
2. $H^{(k+1)} \in \underset{H \in \mathcal{H}}{\textbf{argmin}}\ \sigma(H, A^{(k)})$.
3. If converged stop. Else $k \leftarrow k + 1$ and go to step 1.

It is assumed that step 1, updating A for given H, can be carried out simply by some form of linear least squares. We assume that for each ℓ there is at least one j such that $A_{j\ell} = I$. Note that this is the case for MLR, PCA, EFA, and for all Gifi Systems.

Step 2 is somewhat more intricate, because of the cone restrictions. In partitioned form we can write the loss function as

$$\sigma(H, A) = \sum_{i=1}^{m} \textbf{tr}\ H_i' \sum_{j=1}^{m} H_j \sum_{\ell=1}^{L} A_{j\ell} A_{i\ell}'$$

$$B_{ij}(A) = \sum_{\ell=1}^{L} A_{j\ell} A_{i\ell}'$$

5.2 Majorization

$$\textbf{tr}\ H'HG = \textbf{tr}\ (\tilde{H}+(H-\tilde{H}))'(\tilde{H}+(H-\tilde{H}))G \geq \textbf{tr}\ \tilde{H}'\tilde{H}G+2\textbf{tr}\ \tilde{H}'(H-\tilde{H})G$$

$$\textbf{tr}\ H'\tilde{H}G(\tilde{H})$$

5.3 Alternating Least Squares

The standard way to minimize loss function (4.8) is implemented in the `OVERALS` program Meulman and Heiser (2012). It is also the one used in the `homals` package (De Leeuw and Mair 2009a).

In this paper the algorithm is different because we use the loss function (4.2). We still use ALS, which means in this case that we cycle through three substeps in each iteration. We update A for given X and H, we then update X for given H and A, and finally we update H for given X and A. Algorithm A goes as follows.

0. Set $k = 0$ and start with some $X^{(0)}, H^{(0)}, A^{(0)}$.
1. $X^{(k+1)} = \textbf{ortho}(\textbf{center}(H^{(k)}A^{(k)})$.
2. For $j = 1, \cdots, m$ compute $A_j^{(k+1)} = \{H_j^{(k)}\}^+ X^{(k+1)}$.
3. For $j = 1, \cdots, m$ and $s = 1, \cdots p_j$ compute $h_{js}^{(k+1)} = \textbf{proj}_{\mathcal{K}_{js}\cap\mathcal{S}}((X^{(k+1)} - \sum_{t<s} h_{jt}^{(k+1)}\{a_{jt}^{(k+1)}\}' - \sum_{t>s} h_{jt}^{(k)}\{a_{jt}^{(k+1)}\}')a_s^{(k+1)})$.
4. If converged stop. Else $k \leftarrow k+1$ and go to step 1.

In step 1 we use superscript + for the Moore-Penrose inverse. In step 2 the center operator does column centering, the ortho operator finds an orthonormal basis for the column space of its argument.

The complicated part is step 4, the *optimal scaling*, i.e. the updating of H_j for given X and A_j. We cycle through the variables in the block, each time projecting a single column on the cone of admissible transformations of the variable, and then normalizing the projection to length one. The *target*, i.e. the vector we are projecting, is complicated, because the other variables in the same block must be taken into account.

In order to simplify the optimal scaling computations within an iteration we can use majorization (**deleeuw_B_15?**). This has the additional benefit that the optimal scaling step becomes embarassingly parallel. We expand the loss for block j around a previous solution $\tilde{H}_j$.

$$\mathbf{SSQ}(X-H_jA_j)=\mathbf{SSQ}(X-\tilde{H}_jA_j)-2\mathbf{tr}\ (H_j-\tilde{H}_j)'(X-\tilde{H}_jA_j)A_j'+\mathbf{tr}\ A_j'(H_j-\tilde{H}_j)'(H_j-\tilde{H}_j)A_j.$$

Now

$$\mathbf{tr}\ (H_j-\tilde{H}_j)A_jA_j'(H_j-\tilde{H}_j)'\leq\kappa_j\ \mathbf{tr}\ (H_j-\tilde{H}_j)'(H_j-\tilde{H}_j),$$

where κ_j is the largest eigenvalue of $A_j'A_j$. Thus

$$\mathbf{SSQ}(X-H_jA_j)\leq\mathbf{SSQ}(X-\tilde{H}_jA_j)+\kappa_j\ \mathbf{SSQ}(H_j-U_j)-\frac{1}{\kappa_j}\ \mathbf{SSQ}((X-\tilde{H}_jA_j)A_j'),$$

where U_j is the *target*

$$U_j=\tilde{H}_j+\frac{1}{\kappa_j}(X-\tilde{H}_jA_j)A_j'. \tag{3}$$

It follows we can update the optimal scaling of the variables by projecting the columns of U_j on their respective cones and then normalizing. See De Leeuw (1975) for results on normalized cone regression. This can be done for all variables in the block separately, without taking any of the other variables in the block (or in any of the other blocks) into account. Thus the optimal scaling is easy to parallellize. The resulting algorithm B is as follows.

0. Set $k=0$ and start with some $X^{(0)},H^{(0)},A^{(0)}$.
1. $X^{(k+1)}=\mathbf{ortho}(\mathbf{center}(H^{(k)}A^{(k)})$.
2. For $j=1,\cdots,m$ compute $A_j^{(k+1)}=\{H_j^{(k)}\}^+X^{(k+1)}$.
3. For $j=1,\cdots,m$ compute $U_j^{(k+1)}=H_j^{(k)}+\frac{1}{\kappa_j}(X^{(k+1)}-H_j^{(k)}A_j^{(k+1)})\{A_j^{(k+1)}\}'$ and for $s=1,\cdots p_j$ compute $h_{js}^{(k+1)}=\mathbf{proj}_{\mathcal{K}_{js}\cap\mathcal{S}}(u_{js}^{(k+1)})$.
4. If converged stop. Else $k\leftarrow k+1$ and go to step 1.

5.4 Implementation Details

If we follow the ALS strategy strictly the **ortho**() operator should be implemented using Procrustus rotation (Gibson 1962). Thus if $Z=K\Lambda L'$ is

the singular value decomposition of X, then $\textbf{ortho}(Z) = KL'$. Note, however, that any other basis for the column space of Z merely differs from the Procrustus basis by a rotation. And this rotation matrix will carry unmodified into the upgrade of A_j in step 2 of the algorithm, and thus after steps 1 and 2 the loss will be the same, no matter which rotation we select. In our algorithm we use the QR decomposition to find the basis, using the Gram-Schmidt code from De Leeuw (2015b).

In actual computation we column-center the basis and compute a full rank QR decomposition, using the code in De Leeuw (2015b). Thus $G_\ell = Q_\ell R_\ell$,

We implement the cone restrictions by the constraints $h_{js} = G_{js} z_s$ in combination with $T_{js} h_{js} \geq 0$. Thus the transformed variables must be in the intersection of the subspace spanned by the columns of the *transformation basis* G_{js} and the polyhedral convex cones of all vectors h such that $T_{js} h \geq 0$. We suppose that all columns of the G_{js} add up to zero, and we require, in addition, the normalization $SSQ(h_{js}) = 1$.

We use the code described in De Leeuw (2015c) to generate B-spline bases. Note that for coding purposes binary indicators are B-splines of degree zero, while polynomials are B-splines without interior knots. We include the utility functions to generate lists of knots. There is `knotsQ()` for knots at the quantiles, `knotsR()` for knots equally spaced on the range, `knotsD()` for knots at the data points, and `knotsE()` for no interior knots. Also note that binary indicators can be created for qualitative non-numerical variables, for which B-splines are not defined. We have added the option using degree -1 to bypass the B-spline code and generate an indicator matrix, using the utility `makeIndicator()`. Note that `'makeIndicator(foo)` is equivalent to `bsplineBasis(foo, degree = 0, innerknots = sort(unique(foo)))`. Throughout we first orthonormalize the basis matrices G_{js}, using the Gram-Schmidt code from De Leeuw (2015b).

The matrices T_{js} in the homogeneous linear inequality restrictions that define the cones $\mathcal{K}_{js}$ can be used to define monotonicity or convexity of the resulting transformations. In the current implementation we merely allow for monotonicity, which means the T_{js} do not have to be stored. The transformations for each variable can be restricted to be increasing, or they can be unrestricted. By using splines without interior knots we allow in addition for polynomial transformations, which again can be restricted to be either monotonic or not. Note that it is somewhat misleading to say we are fitting

monotone splines or polynomials, we are mainly requiring monotonicity at the data points.

If there are multiple copies of a variable in a block then requiring the transformation to be ordinal means that we want the transformation of the first copy to be monotonic. The transformations of the other copies are not constrained to be monotonic. If you want all copies to be transformed monotonically, you have to explicitly introduce them as separate variables.

For variables with copies there is yet another complication. For copies we have $H_jA_j = G_j(Z_jA_j) = G_jY_j$. If we require monotonicity in MVAOS we constrain a column of H_j (in fact, the first one) to be monotonic. In classic Gifi, in which the G_j are binary indicators, we constrain the first column of Y_j, which automatically implies the first column of G_jY_j is monotonic as well. In previous Gifi work with B-splines, we also constrained the first column of Y_j, which again implied the first column of G_jY_j was monotnic as well. But in our current MVAOS implementation monotonicity of the first column of H_j does not imply monotonicity of the first column of H_jA_j, even if the basis G_j is a binary indicator. This discrepancy between the old and the new Gifi only comes into play for ordinal variables with multiple copies.

Missing data are incorporated in the definition of the cones of transformations by using a G_{js} which is the direct sum of a spline basis for the non-missing and an identity matrix for the missing data. This is called *missing data multiple* in Gifi (1990). There are no linear inequality restrictions on the quantifications of the missing data.

5.5 Wrappers

The `homals()` implementation in De Leeuw and Mair (2009a) is a single monolithic program in R, which specializes to the various MVAOS techniques by a suitable choice of its parameters. This approach has some disadvantages. If we want principal component analysis, we already know all blocks are singletons. If we want multiple correspondence analysis we know each variable has p copies. If we want multiple regression, we know there are two blocks, and one is a singleton. So it is somewhat tedious to specify all parameters all of the time. Also, some of the output, graphical and otherwise, is specific to a particular technique. For regression we want residuals and fitted

values, in canonical analysis we want block scores and loadings. And, more generally, we may want the output in a form familiar from the classical MVA techniques. It is indeed possible to transform the `homals()` output to more familar forms (De Leeuw (2009)), but this requires some extra effort.

In this book we go back to the original approach of Gifi (1990) and write separate programs for nonlinear versions principal component analysis, multiple regression, canonical analysis, discriminant analysis, and so on.

These programs, now written in R and no longer in FORTRAN, are wrappers for the main computational core, the program `gifiEngine()`. The wrappers, which have the familiar names `morals()`, `corals()`, `princals()`, `homals()`, `criminals()`, `overals()`, `primals()`, and `canals()`, create a gifi object from the data and parameters, and then pass this to `gifiEngine()`. Computations are itereated to convergence, and result are stored in a xGifi object. Then the output is transformed to a format familiar from the corresponding technique from classical MVA. Each wrapper `foo` returns a structure of class `foo`.

This modular approach saves code, because both `makeGifi()` and `gifiEngine()` are common to all programs. It also makes it comparatively easy to add new wrappers not currently included, possibly even contributed by others.

Although we like the above quotation from Hill (1990), it is not quite accurate. Our current generation of wrappers can use B-spline bases, it can use an arbitrary number of copies of a variable, and each copy can be either categorical, ordinal, polynomial, or splinical. Thus, even more so than the original gifi programs, we have a substantial generalization of the classical techniques, not merely a sequence of synonyms.

5.6 Structures

The computations are controlled by the arguments to the wrappers. These arguments are used to construct three structures: the gifi, the gifiBlock, and the gifiVariable. A gifi is just a list of gifiBlocks, and a gifiBlock is a list of gifiVariables. This reflects the partitioning of the variables into blocks. A gifiVariable contains a great deal of information about the variable. The function `makeGifiVariable()` is a constructor that returns a structure

of class `gifiVariable`. The contents of a `gifiVariable` remain the same throughout the computations.

```
    return (structure (
      list (
        data = data,
        basis = basis,
        qr = qr,
        copies = copies,
        degree = degree,
        ties = ties,
        missing = missing,
        ordinal = ordinal,
        active = active,
        name = name,
        type = type
      ),
      class = "gifiVariable"
    ))
```

There are three corresponding structures containing initial and intermediate results, and eventually output, the xGifi, xGifiBlock, and xGifiVariable. Again, an xGifi is a list of xGifiBlocks, and an xGifiBlock is a list of xGifiVariables. The constructor for an xGifiVariable returns an object of class `xGifiVariable`, which contains the elements that are updated in each iteration during the computations. There is an `xGifiVariable` for both active and passive variables.

```
 return (structure (
    list(
      transform = transform,
      weights = weights,
      scores = scores,
      quantifications = quantifications
    ),
    class = "xGifiVariable"
  ))
```

Chapter 6

Multiple Correspondence Analysis and homals()

6.1 Introduction

Suppose all basis matrices $G_{j\ell}$ in block j are the same, say equal to G_j. Then the block scores $H_j A_j$ are equal to $G_j Z_j A_j$, which we can write simply as $G_j Y_j$. Thus loss must be minimized over X and the Y_j.

If all G_j are binary indicators of categorical variables, and the m blocks are all of span one, then MVAOS is *multiple correspondence analysis* (MCA). The block scores $G_j Y_j$ are k_j different points in $\mathbb{R}^p$, with k_j the number of categories of the variable, which is usually much less than n. The plot connecting the block scores to the object scores is called the *star plot* of the variable. If k_j is much smaller than n a star plot will connect all object scores to their category centroids, and the plot for a block (i.e. a variable) will show k_j stars. Since loss σ is equal to the sum of squared distances between object scores and block scores, we quantify or transform variables so that stars are small.

In our MVAOS MCA function `homals()` we allow for B-spline bases and for monotonicity restrictions. The input data (as for all MVAOS programs) needs to be numeric, and we included a small utility function `makeNumeric()` that can be used on data frames, factors, and character variables to turn them into numeric matrices. All other arguments to the function have default

values.

```
homals <-
  function (data,
            knots = knotsD (data),
            degrees = rep (-1, ncol (data)),
            ordinal = rep (FALSE, ncol (data)),
            ndim = 2,
            ties = "s",
            missing = "m",
            names = colnames (data, do.NULL = FALSE),
            itmax = 1000,
            eps = 1e-6,
            seed = 123,
            verbose = FALSE)
```

The output is a structure of class `homals`, i.e. a list with a class attribute`homals`. The list consists of transformed variables (in xhat), their correlation (in rhat), the objectscores (in objectscores), the blockscores (in blockscores, which is itself a list of length number of variables), the discrimination matrices (in dmeasures, a list of length number of variables), their average (in lambda), the weights (in a), the number of iterations (in ntel), and the loss function value (in f).

```
    return (structure (
      list (
        transform = v,
        rhat = corList (v),
        objectscores = h$x,
        scores = y,
        quantifications = z,
        dmeasures = d,
        lambda = dsum / ncol (data),
        weights = a,
        loadings = o,
        ntel = h$ntel,
        f = h$f
```

```
      ),
      class = "homals"
    ))
```

Note that in MCA we have $H_jA_j = G_jY_j$. In previous Gifi publications the Y_j are called *category quantifications*. Our current `homals()` does not output the categaory quantifications directly, only the block scores G_jY_j. If the G_j are binary indicators, the Y_j are just the distinct rows of G_jY_j. There is also some indeterminacy in the representation H_jA_j, which we resolve, at least partially, by using the QR decomposition $H_j = Q_jR_j$ to replace H_j by Q_j, and use $H_jA_j = Q_j(R_jA_j)$. One small problem with this is that we may have $r_j =: \mathbf{rank}(H_j) < r$, in which case there are only r_j copies in Q_j. This happens, for example, in the common case in which variable j is binary and takes only two values.

6.2 Equations

6.3 Examples

6.3.1 Hartigan's Hardware

Our first example are semi-serious data from Hartigan (1975) (p. 228), also analyzed in Gifi (1990) (p. 128-135). A number of screws, tacks, nails, and bolts are classified by six variables. The data are

```
##        thread head indentation bottom length brass
## tack        N    F           N      S      1     N
## nail1       N    F           N      S      4     N
## nail2       N    F           N      S      2     N
## nail3       N    F           N      F      2     N
## nail4       N    F           N      S      2     N
## nail5       N    F           N      S      2     N
## nail6       N    C           N      S      5     N
## nail7       N    C           N      S      3     N
## nail8       N    C           N      S      3     N
```

```
## screw1      Y    O          T      S      5      N
## screw2      Y    R          L      S      4      N
## screw3      Y    Y          L      S      4      N
## screw4      Y    R          L      S      2      N
## screw5      Y    Y          L      S      2      N
## bolt1       Y    R          L      F      4      N
## bolt2       Y    O          L      F      1      N
## bolt3       Y    Y          L      F      1      N
## bolt4       Y    Y          L      F      1      N
## bolt5       Y    Y          L      F      1      N
## bolt6       Y    Y          L      F      1      N
## tack1       N    F          N      S      1      Y
## tack2       N    F          N      S      1      Y
## nailb       N    F          N      S      1      Y
## screwb      Y    O          L      S      1      Y
```

We can do a simple MCA, using all the default values.

```
h <- homals (makeNumeric(hartigan))
```

After 54 iterations we find a solution with loss 0.5157272962. The object scores are plotted in figure 2.

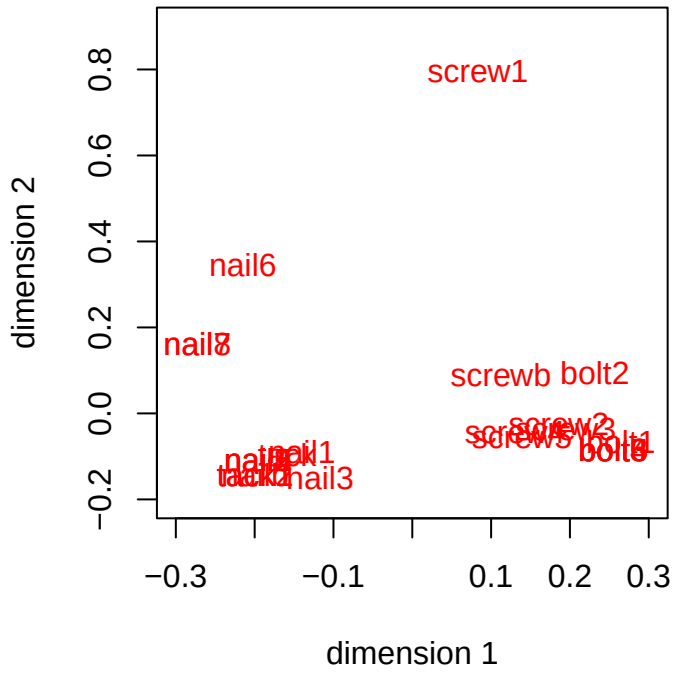


Figure 2: Hartigan Data, Object Scores

The star plots, produced by the utility `starPlotter()` are in figure 3.

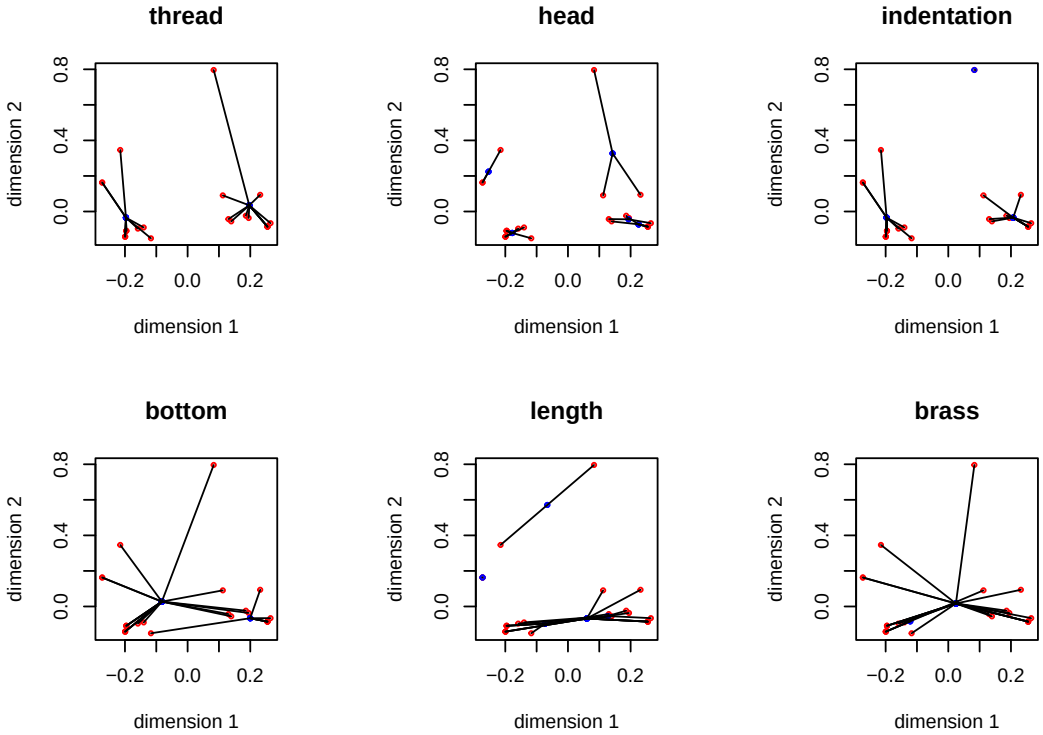


Figure 3: Hartigan Data, Star Plots

The discriminations matrices Δ_j are

```
##      [,1]  [,2]
## [1,]  0.93  0.16
## [2,]  0.16  0.03
##      [,1]  [,2]
## [1,]  0.96  0.04
## [2,]  0.04  0.64
##      [,1]  [,2]
## [1,]  0.94  0.07
## [2,]  0.07  0.66
##      [,1]  [,2]
## [1,]  0.39 -0.13
## [2,] -0.13  0.04
##      [,1]  [,2]
## [1,]  0.29 -0.19
```

```
## [2,] -0.19  0.82
##      [,1]  [,2]
## [1,]  0.07  0.05
## [2,]  0.05  0.03
```

and their average Λ is

```
##      [,1]  [,2]
## [1,]  0.60  0.00
## [2,]  0.00  0.37
```

Note that the loss was 0.5157272962, which is one minus the average of the trace of Λ. The induced correlations are

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]
##  [1,]  1.00  1.00  0.01  0.98 -0.20  0.46  0.03  0.41 -0.22
##  [2,]  1.00  1.00 -0.00  0.98 -0.21  0.46  0.03  0.41 -0.22
##  [3,]  0.01 -0.00  1.00  0.08  0.38  0.18 -0.59  0.40 -0.02
##  [4,]  0.98  0.98  0.08  1.00  0.00  0.50 -0.10  0.43 -0.21
##  [5,] -0.20 -0.21  0.38  0.00  1.00  0.14 -0.66  0.05  0.09
##  [6,]  0.46  0.46  0.18  0.50  0.14  1.00 -0.17  0.28 -0.29
##  [7,]  0.03  0.03 -0.59 -0.10 -0.66 -0.17  1.00 -0.00 -0.10
##  [8,]  0.41  0.41  0.40  0.43  0.05  0.28 -0.00  1.00  0.23
##  [9,] -0.22 -0.22 -0.02 -0.21  0.09 -0.29 -0.10  0.23  1.00
```

Of the six variables, three are binary. Thus they only have a single transformed variable associated with them, which is just the standardization to mean zero and sum of squares one. The total number of transformed variables is consequently 9. The eigenvalues of the induced correlation matrix (divided by the number of variables, not the number of transformed variables) are

```
## [1]  0.60  0.37  0.21  0.13  0.10  0.07  0.02  0.00 -0.00
```

Note that the two dominant eigenvalues are again equal to the diagonal elements of Λ.

###GALO

The second example is somewhat more realistic. In the GALO dataset (Peschar (1975)) data on 1290 school children in the sixth grade of an elementary school in 1959 in the city of Groningen (Netherlands) were collected. The variables are gender, IQ (categorized into 9 ordered categories), advice (teacher categorized the children into 7 possible forms of secondary education, i.e., Agr = agricultural; Ext = extended primary education; Gen = general; Grls = secondary school for girls; Man = manual, including housekeeping; None = no further education; Uni = pre- University), SES (parent's profession in 6 categories) and school (37 different schools). The data have been analyzed previously in many Gifi publications, for example in De Leeuw and Mair (2009a). For our MCA we only make the first four variables, school is treated as passive

We use this example to illustrate some of the constraints on transformations. Two copies are used for all variables (although gender effectively only has one, of course). IQ is treated as ordinal, using a piecewise linear spline with knots at the nine data points.

```
galo_knots <- knotsD(galo)
galo_degrees <- c(-1,1,-1,-1,-1)
galo_ordinal <- c(FALSE, TRUE, FALSE, FALSE,FALSE)
galo_active <-c (TRUE, TRUE, TRUE, TRUE, FALSE)
```

```
h <- homals (galo, knots = galo_knots, degrees = galo_degrees, ordinal = galo_
```

We first give transformations for the active variables (and their copies) in figure 4 . We skip gender, because transformation plots for binary variables are not very informative. We give two transformation plots for IQ, first using H and then using HA. This illustrates the point made earlier, that transformation plots of block scores for ordinal variables with copies need not be monotone. It also illustrates that additional copies of an ordinal variable are not scaled to be monotone. Note that the plots for advice and SES are made with the utility `stepPlotter()`. Because the degree of the splines for those variables is zero, these transformation plots show step functions, with the steps at the knots, which are represented by vertical lines.

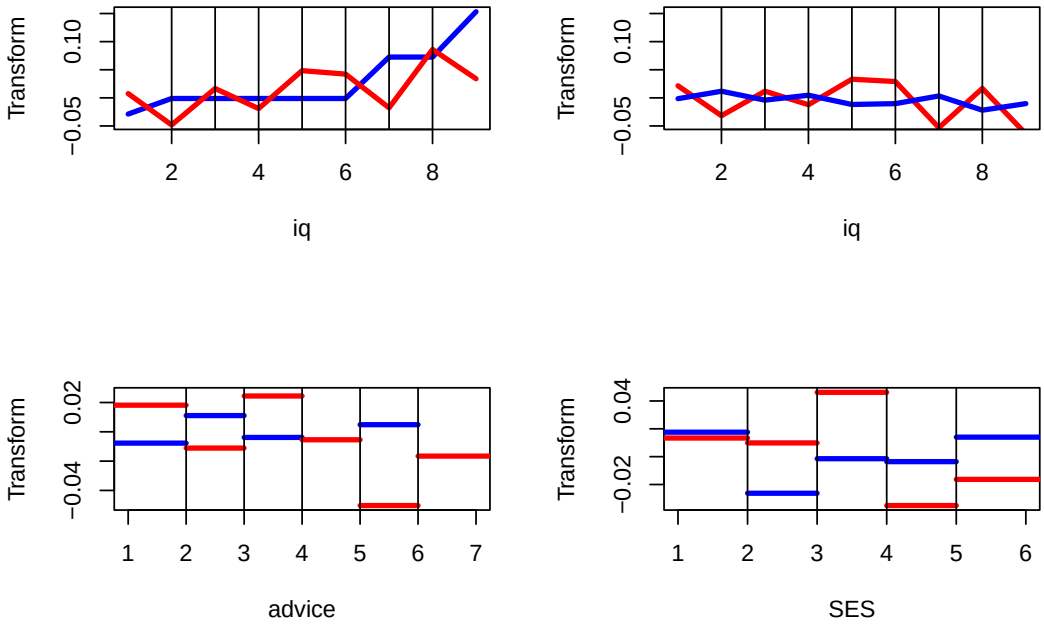
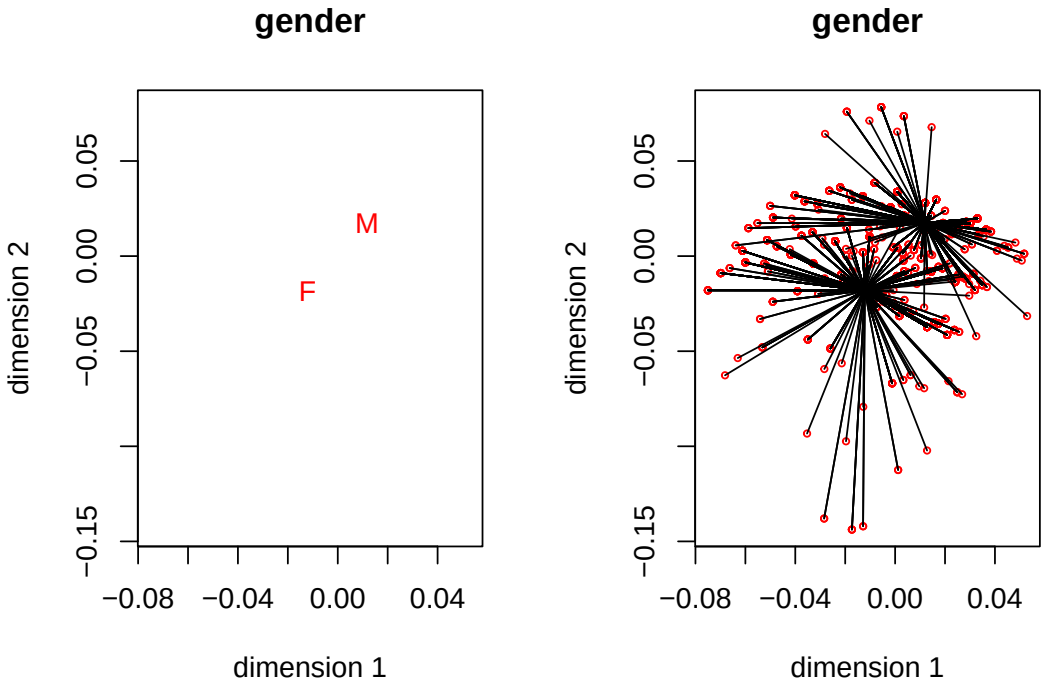
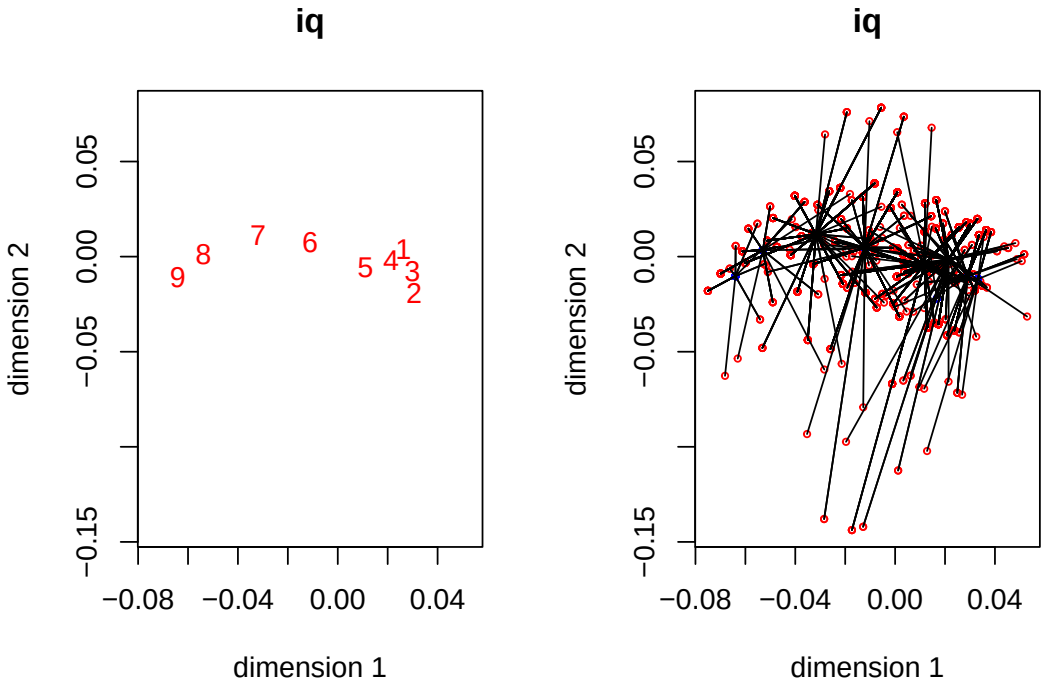


Figure 4: Galo Data, Transformations

The four star plots for the active variables, together with the four category quantification plots, are in figure 5. Note that `homals()` does not compute category quantifications, we have to compute them from the `homals()` output. Also note that for gender, advice and SES the object scores are connected to the category centroids of the variables. For IQ object scores are connected to points on the line connecting adjacent category quantifications. See De Leeuw and Rijckevorsel (1988) for category plots using forms of fuzzy coding (of which B-splines are an example).


gender
dimension 2
dimension 1
M
F
−0.08
−0.04
0.00
0.04
0.05
0.00
−0.05
−0.15

gender
dimension 2
dimension 1
−0.08
−0.04
0.00
0.04
0.05
0.00
−0.05
−0.15


iq
dimension 2
dimension 1
9
8
7
6
5
4
1
3
2
−0.08
−0.04
0.00
0.04
0.05
0.00
−0.05
−0.15

iq
dimension 2
dimension 1
−0.08
−0.04
0.00
0.04
0.05
0.00
−0.05
−0.15

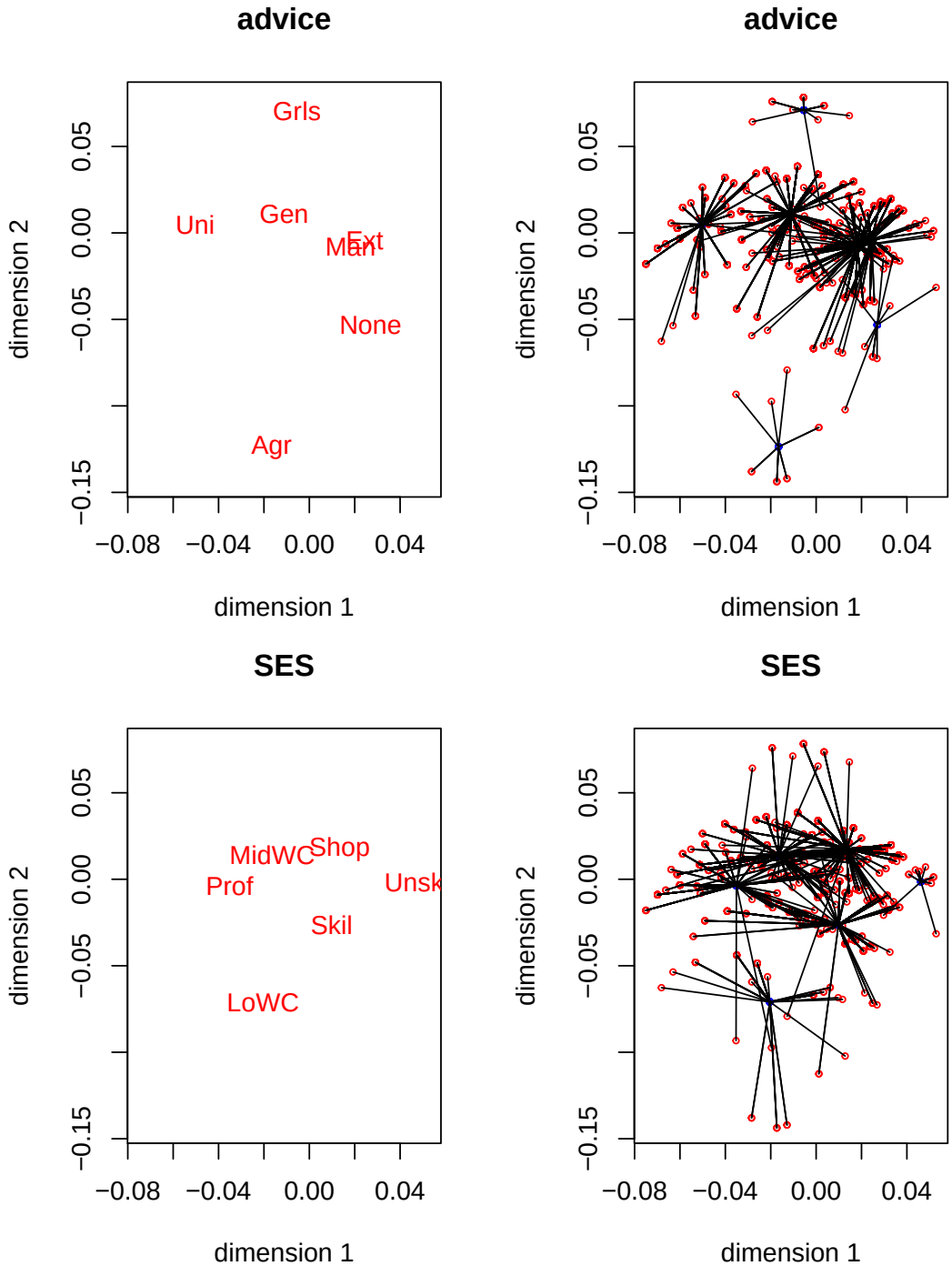


Figure 5: Galo Data, Category Quantifications and Star Plots

For this analysis we need 52 iterations to obtain loss 0.5425100382. The average discrimination matrix over the four active variables is

```
##      [,1]  [,2]
## [1,]  0.54  0.00
## [2,]  0.00  0.38
```

while the eigenvalues of the induced correlation matrix of the active variables and their copies, divided by four, are

```
## [1]  0.54  0.38  0.26  0.21  0.18  0.13  0.05
```

The category quantifications for the passive variable indicating the 37 schools are in figure 6.

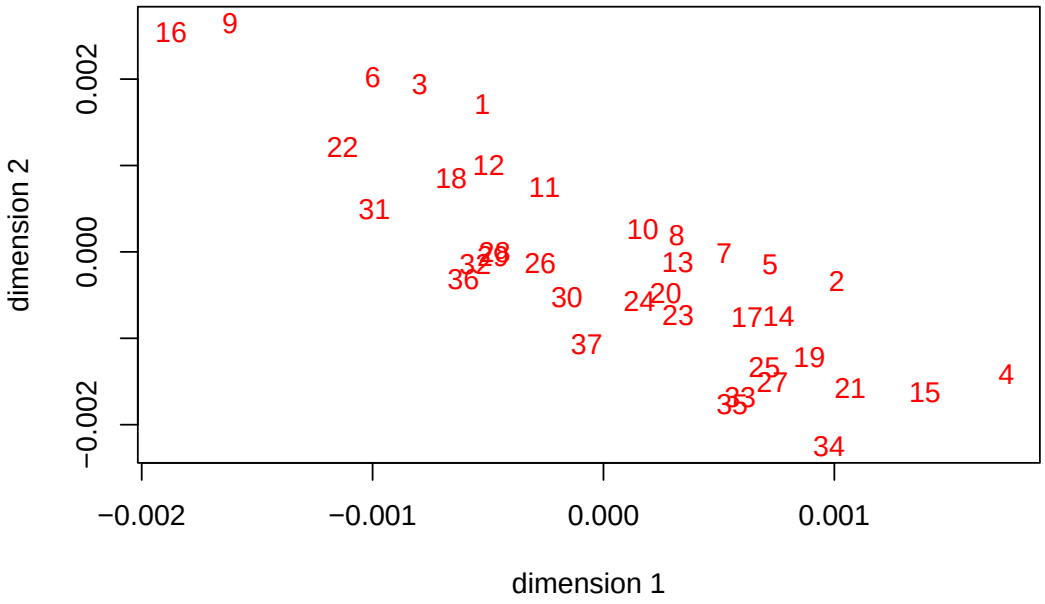


Figure 6: Galo Data, Schools as Passive

If we look at the scale of the plot we see all schools are pretty close to the origin. The discrimination matrices are consequently also small. In 1959 schools were pretty much the same.

```
##      [,1]    [,2]
## [1,]  0.0011 -0.0014
## [2,] -0.0014  0.0022
```

6.3.2 Thirteen Personality Scales

Our next example is a small data block from the `psych` package (Revelle 2015) of five scales from the Eysenck Personality Inventory, five from a Big Five inventory, a Beck Depression Inventory, and State and Trait Anxiety measures.

```
epi<- read.csv("data/epi.bfi.csv")
epi_knots <- knotsQ(epi)
epi_degrees <- rep (0, 13)
epi_ordinal <- rep (FALSE, 13)
```

We perform a two-dimensional MCA, using degree zero and inner knots at the three quartiles for all 13 variables.

```
h <- homals(epi, knots = epi_knots, degrees = epi_degrees, ordinal = epi_ordinal)
```

We have convergence in 271 iterations to loss 0.7472905867. The object scores are in figure 7.

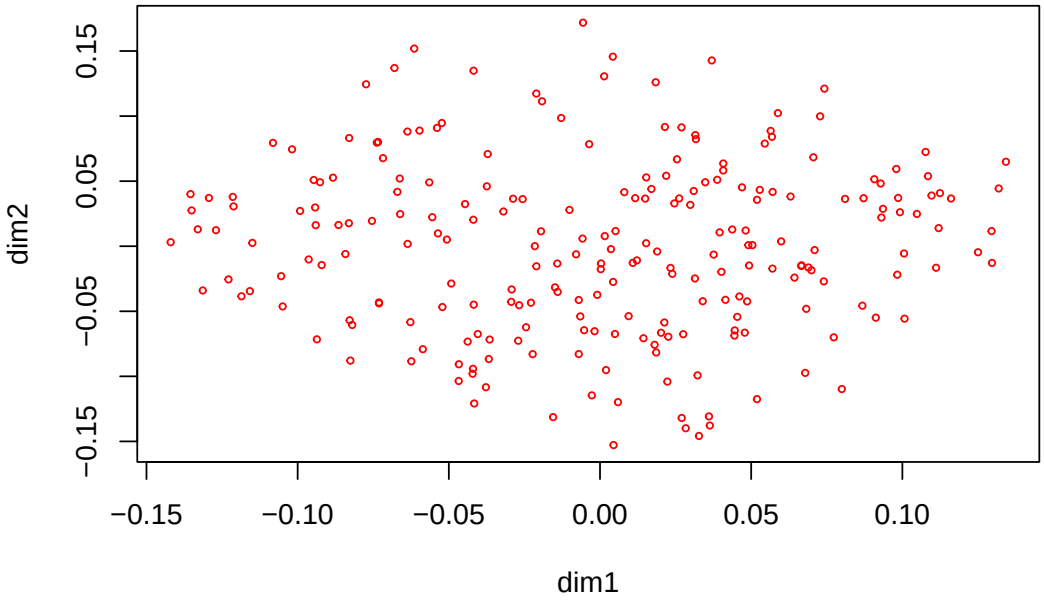
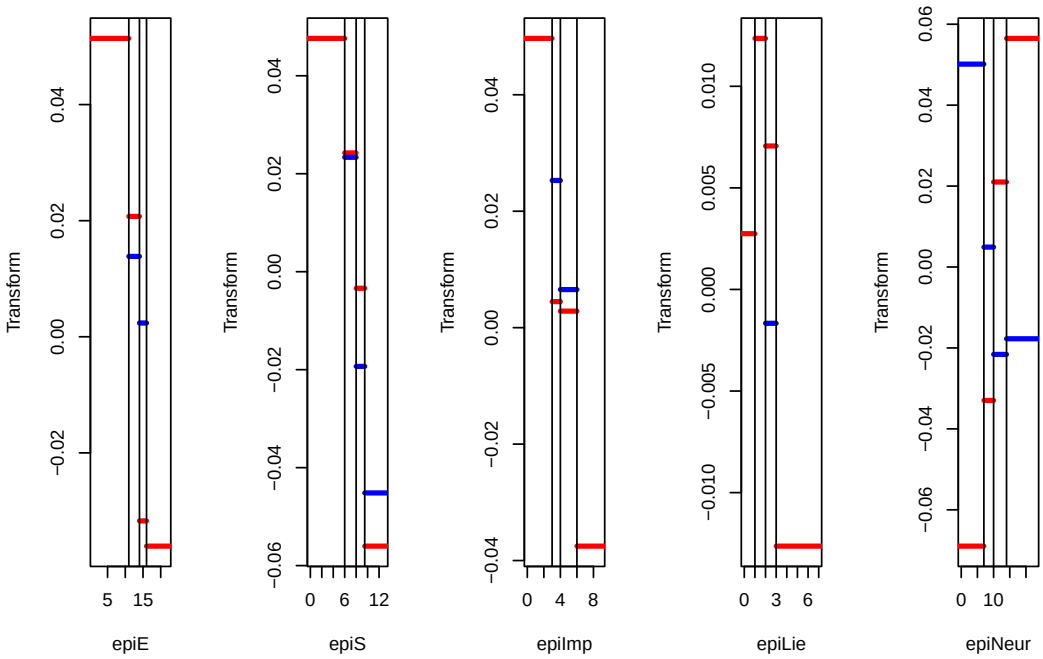
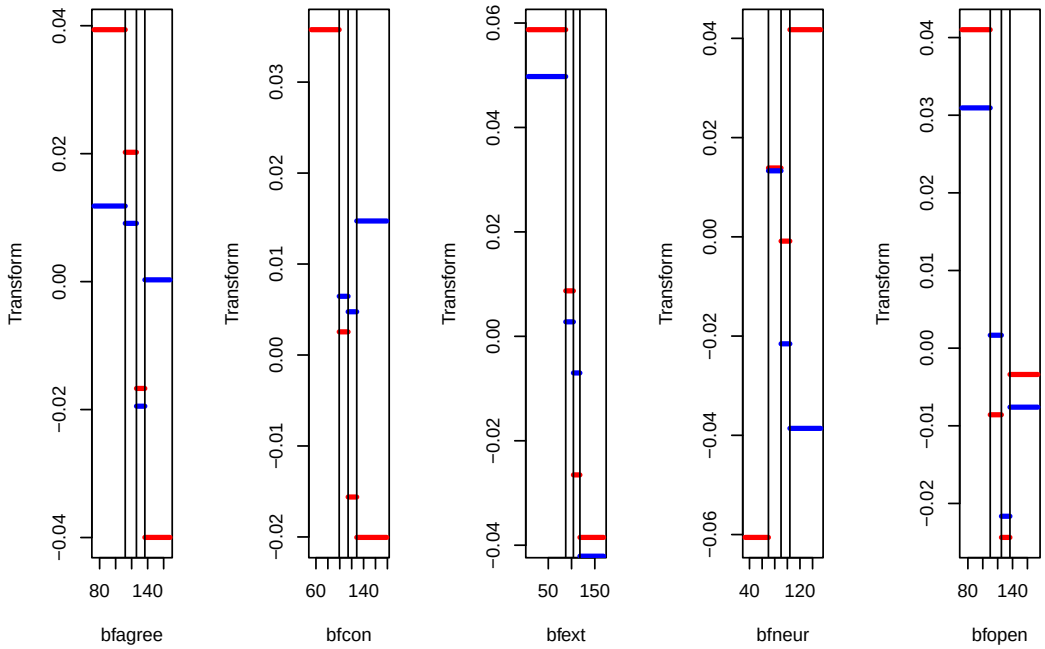


Figure 7: Personality Scales, Object Scores, Multiple Nominal, Degree Zero

Figure 8 has the $G_j Y_j$ for each of the thirteen variables, with the first dimension in red, and the second dimension in blue.




Transform
bfagree
bfcon
bfext
bfneur
bfopen

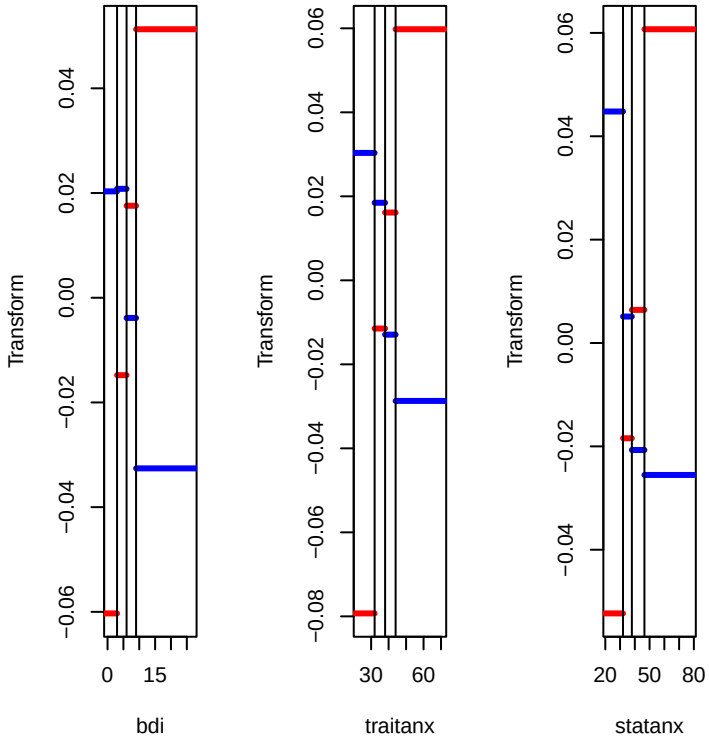
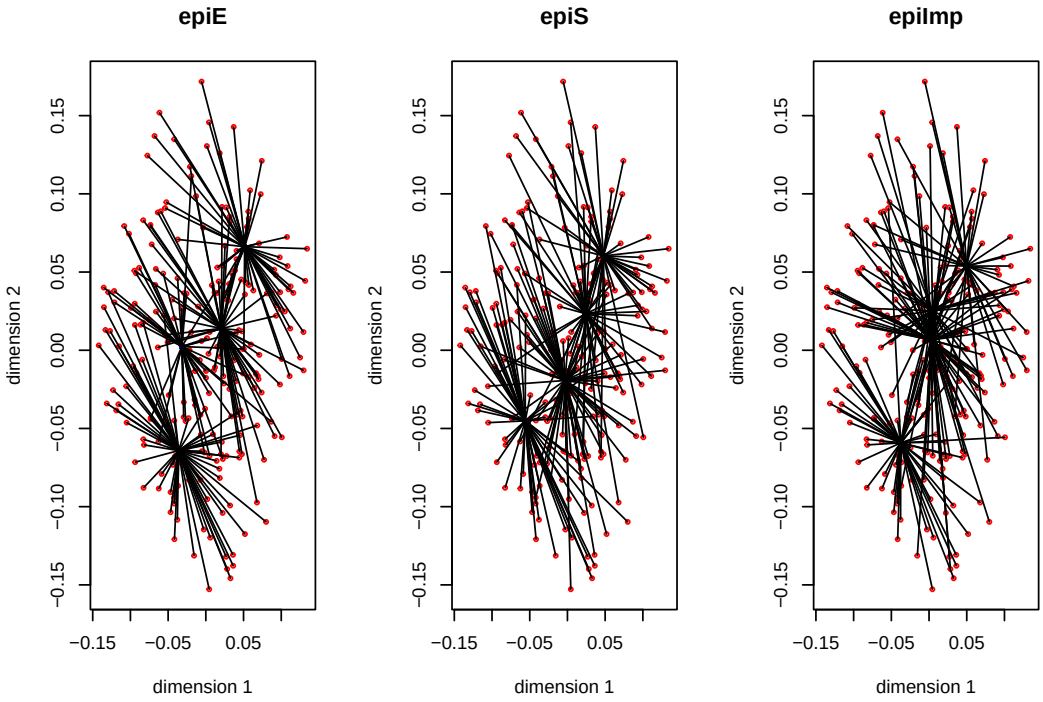
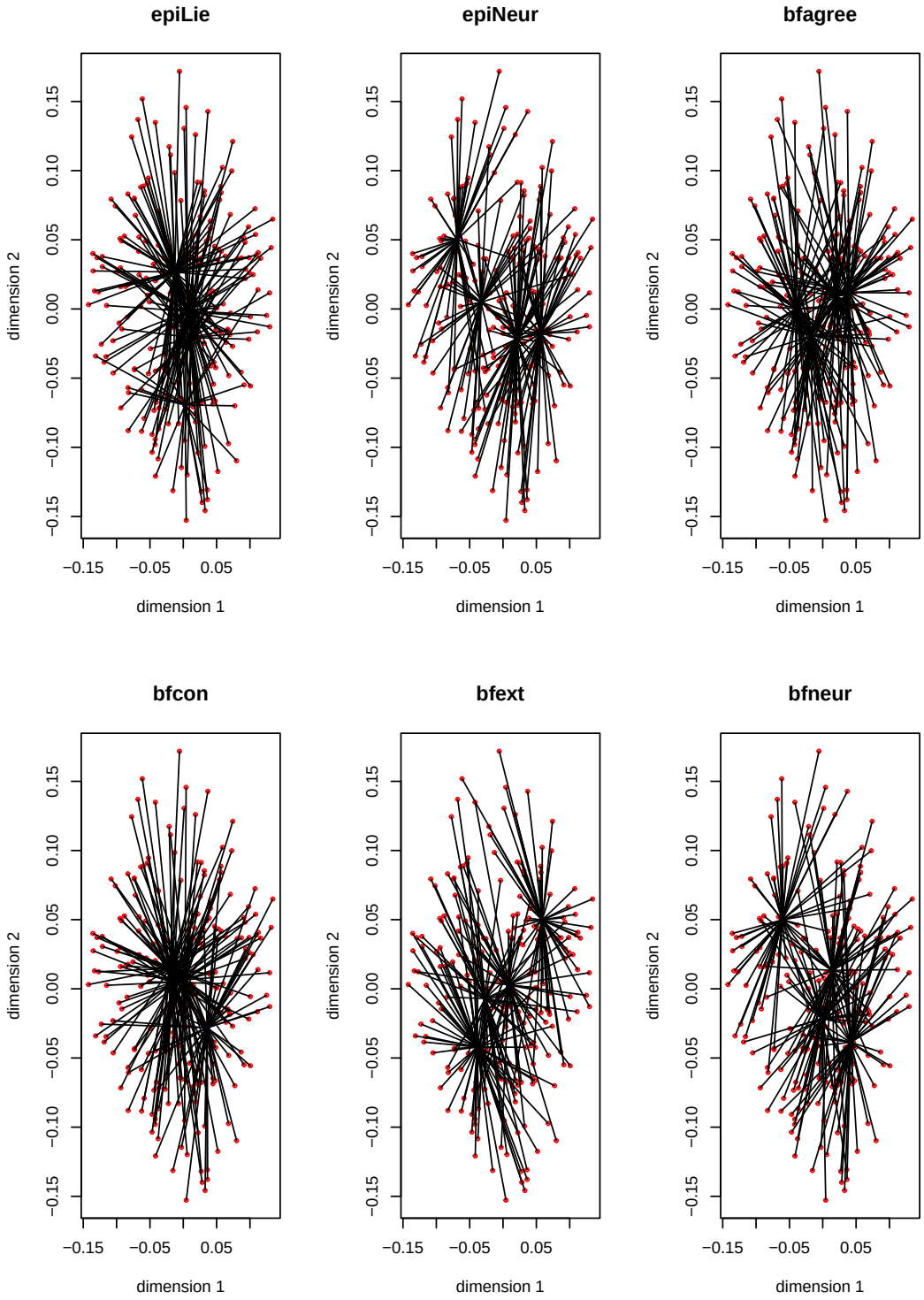
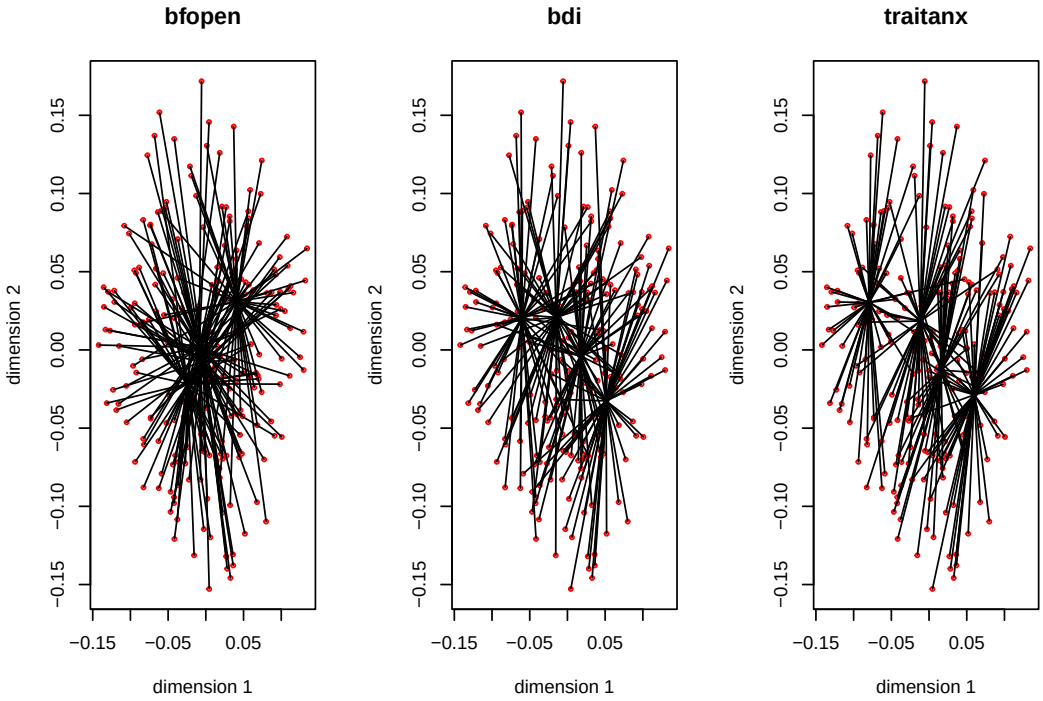

Transform
bdi
traitanx
statanx

Figure 8: Personality Scales, Transformations, Multiple Nominal, Degree Zero

The thirteen star plots are in figure 8.




epiLie
epiNeur
bfagree
bfcon
bfext
bfneur
dimension 1
dimension 2
−0.15
−0.05
0.05
0.00
0.10
0.15
−0.10


bfopen
bdi
traitanx
dimension 2
dimension 1
−0.15
−0.10
−0.05
0.00
0.05
0.10
0.15

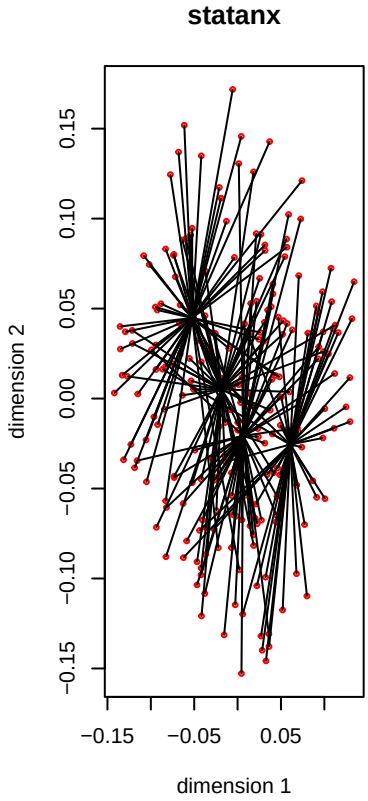


Figure 9: Personality Scales, Star Plots, Multiple Nominal, Degree Zero

Now change the degree to two for all variables, i.e. fit piecewise quadratic polynomials which are differentiable at the knots. We still have two copies for each variable, and these two copies define the blocks.

```
epi_degrees <- rep (2, 13)
h <- homals (epi, knots = epi_knots, degrees = epi_degrees, ordinal = epi_ordinal)
```

We have convergence in 560 iterations to loss 0.7179023033. The object scores are in figure 10 and the transformation plots in figure 11.

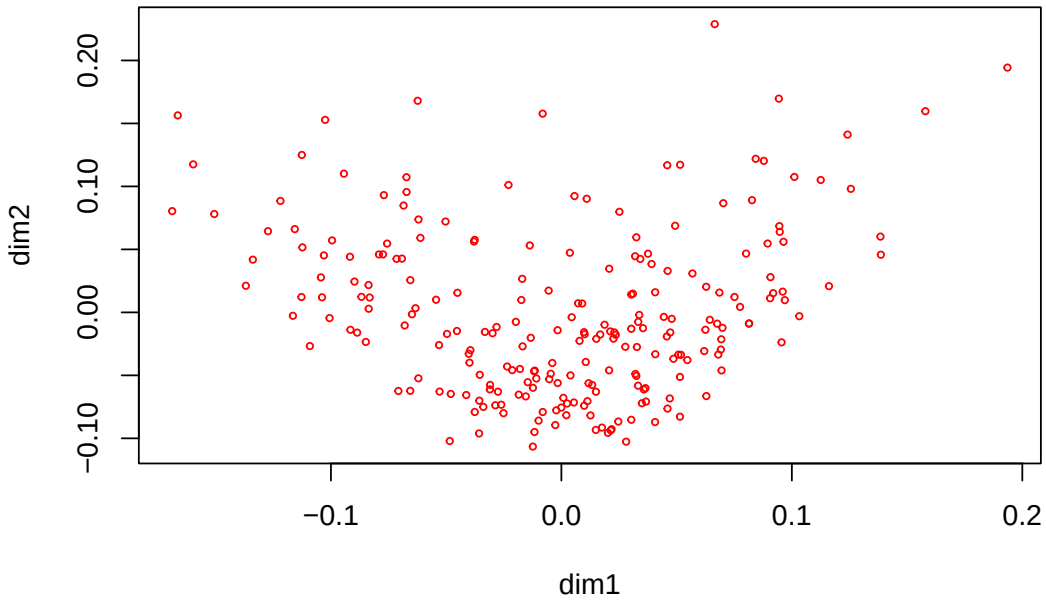
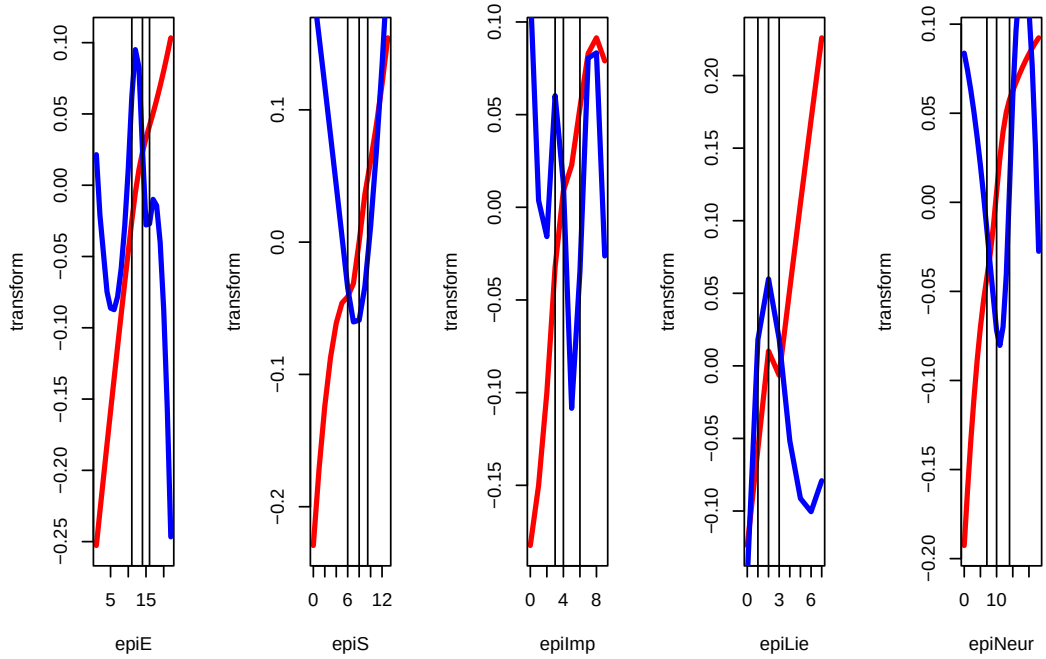


Figure 10: Personality Scales, Object Scores, Multiple Nominal, Degree Two



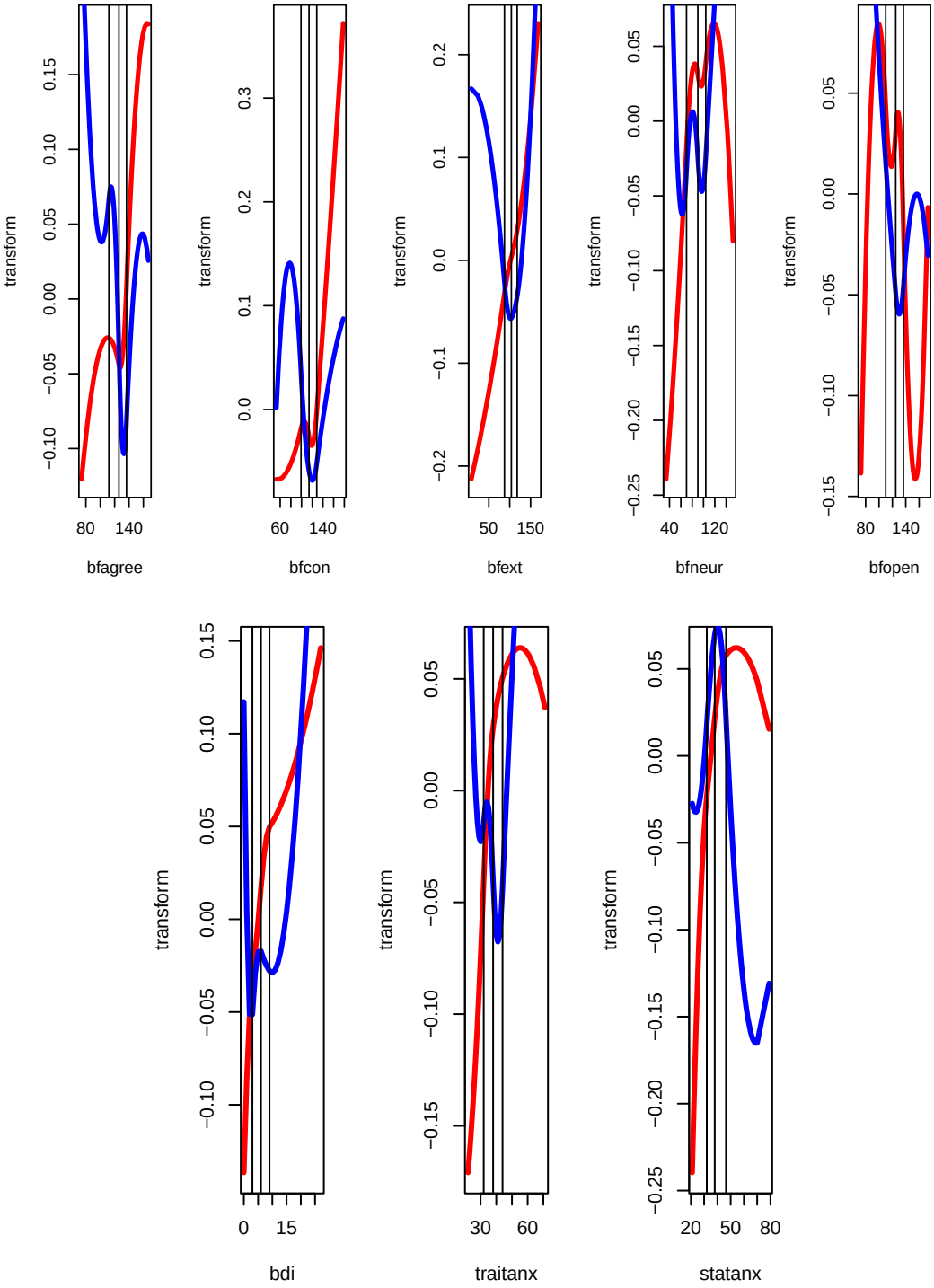

transform
bfagree
bfcon
bfext
bfneur
bfopen
bdi
traitanx
statanx

Figure 11: Personality Scales, Transformations, Multiple Nominal, Degree Two

#Correspondence Analysis and corals()

##Introduction

Ordinary correspondence analysis (OCA) is the special case of MCA in which there are only two variables, and both variables have the maximum number of copies. Consequently the `homals()` wrapper can be used to compute a CA. Because input and output can be organized a bit differently for OCA we have written the separate wrapper `corals()`.

Note that `corals()` is not really intended for routine OCA computation. There are many packages in R which do that job much more efficiently. We mention, for example, `anacor` (De Leeuw and Mair (2009b)) and `ca` (Nenadic and Greenacre (2007)). However, `corals()` can be used for a number of cases which the usual OCA packages do not cover.

In `corals()`, as in the other packages, the default input is a single non-negative matrix F. Although any non-negative matrix will do, the most common, and the most natural, input is an $r \times c$ *cross table* with bivariate frequencies. Suppose the frequencies add up to the total number of observations n. Then `gifiEngine()`, which is called by `corals()`, requires input in the form of an $n \times 2$ matrix. Thus a 2×2 table with 1000 observations becomes a 1000×2 matrix. The utility `preCorals()` makes the conversion, but of course the representation is embarrassingly inefficient, both in terms of memory and in terms of computation. After the computations are done, the utility `postCorals()` restores transformations and scores to the appropriate row and column dimensions.

Here are the arguments and their defaults.

```
## function (data, ftype = TRUE, xknots = NULL, yknots = NULL, xdegree = -1,
##     ydegree = -1, xordinal = FALSE, yordinal = FALSE, xties = "s",
##     yties = "s", xmissing = "m", ymissing = "m", xname = "X",
##     yname = "Y", ndim = 2, itmax = 1000, eps = 1e-06, seed = 123,
##     verbose = FALSE)
## NULL
```

If dtype is `FALSE`, then data is a matrix of dimension $n \times 2$, with n the number of observations. This takes us back to the input format of `homals()` with

two variables. If xknots and yknots are kept to their default `NULL` then they are replaced in `corals()` by the quartiles of the two variables.

The redeeming feature of `corals()` is that, unlike the other classical OCA packages, it can handle numerical variables, it can incorporate higher order splines, it can impose monotonicity restrictions, and it can deal with missing data in one of both of the variables. If there are supplementary variables then it makes more sense to use `homals()`.

##Equations

The usual stationary equations for OCA, using the category quantifications Y_1 and Y_2 are

$$\begin{aligned} C_{12}Y_2 &= D_1Y_1\Lambda, \\ C_{21}Y_1 &= D_2Y_2\Lambda, \end{aligned}$$

with normalization $Y_1'D_1Y_1 = I$ and $Y_2'D_2Y_2 = I$.

In the output of `gifiEngine()` the category quantifications $\tilde{Y}_1$ and $\tilde{Y}_2$ and the object scores X satisfy

$$\begin{aligned} G_1\tilde{Y}_1 + G_2\tilde{Y}_2 &= 2X\tilde{\Lambda}, \\ D_1^{-1}G_1'X &= \tilde{Y}_1, \\ D_2^{-1}G_2'X &= \tilde{Y}_2, \end{aligned}$$

with normalization $X'X = I$. It follows that

$$\begin{aligned} C_{12}\tilde{Y}_2 &= D_1\tilde{Y}_1(2\tilde{\Lambda} - I), \\ C_{21}\tilde{Y}_1 &= D_2\tilde{Y}_2(2\tilde{\Lambda} - I), \end{aligned}$$

and thus for the discrimination matrices $\tilde{Y}_1'D_1\tilde{Y}_1 = \tilde{Y}_2'D_2\tilde{Y}_2 = X'P_1X = X'P_2X = \tilde{\Lambda}$. The two sets of quantities from OCA and `corals()` are related by $\Lambda = 2\tilde{\Lambda} - I$, $Y_1 = \tilde{Y}_1\tilde{\Lambda}^{-\frac{1}{2}}$ and $Y_2 = \tilde{Y}_2\tilde{\Lambda}^{-\frac{1}{2}}$.

In classical OCA there is no direct equivalent of the object scores X. Also we typically do not use the decomposition $H_jA_j = G_jZ_jA_j = G_jY_j$, with $H_j'H_j = Z_j'D_jZ_j = I$. From `corals()` we get the loadings $H_j'X$, the correlations between the object scores and transformed copies, which for singleton blocks are always equal to the weights A_j. But since the decomposition $Y_j = Z_jA_j$ is not unique these are of limited use. The correlations between

X and the $G_j\tilde{Y}_j$ are more interesting. Since $X'G_j\tilde{Y}_j = \tilde{\Lambda}$, we see these correlations are equal to $\tilde{\Lambda}^{\frac{1}{2}}$.

##Examples

###Glass

We start with a classical OCA example that was also used by Gifi (1990) (p 277-280) and by De Leeuw and Mair (2009b). Data are from Glass (1954). Occupational status of the fathers is crossed with occupational status of the son, for 3497 British families. The row (father) and column (son) categories are

- PROF professional and high administrative
- EXEC managerial and executive
- HSUP higher supervisory
- LSUP lower supervisory
- SKIL skilled manual and routine nonmanual
- MEMI semi-skilled manual
- UNSK unskilled manual

```
data (glass, package = "anacor")
names <- c("PROF","EXEC","HSUP","LSUP","SKIL","MEMI","UNSK")
glass <- as.matrix (glass)
```

We apply apply `corals()` with the default options. Thus we only compute two dimensions and use crisp indicators.

```
h <- corals(glass)
```

Minimum loss is 0.3017408039, attained after 88 iterations. The two discrimination matrices are both equal to

```
##       [,1]  [,2]
## [1,]  0.76 -0.00
## [2,] -0.00  0.63
```

which means the corresponding canonical correlations are 0.5256290124, 0.2674055567. The maximum correlation between SES of father and son is 0.5256290124.

The category quantifications for fathers are

```
##       [,1]     [,2]
## [1,]  -0.0594   0.0384
## [2,]  -0.0312  -0.0214
## [3,]  -0.0098  -0.0198
## [4,]  -0.0009  -0.0119
## [5,]   0.0049  -0.0003
## [6,]   0.0099   0.0118
## [7,]   0.0112   0.0163
```

and for sons

```
##       [,1]     [,2]
## [1,]  -0.0652   0.0436
## [2,]  -0.0295  -0.0221
## [3,]  -0.0138  -0.0146
## [4,]  -0.0005  -0.0144
## [5,]   0.0045  -0.0023
## [6,]   0.0090   0.0122
## [7,]   0.0106   0.0153
```

We did not require the first dimension to be increasing, it just came out that way. We plot category quantifications in figure 12.

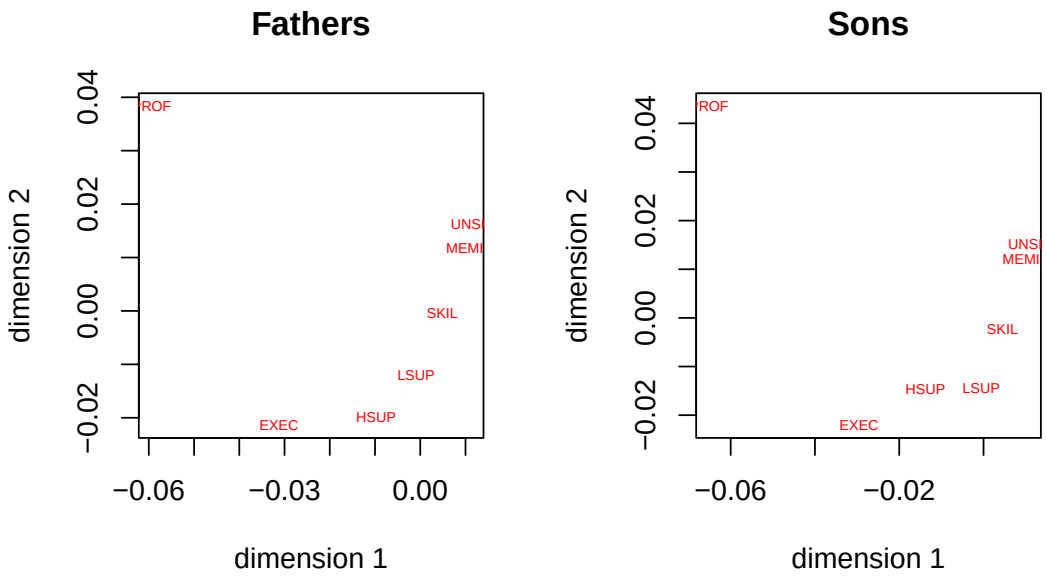


Figure 12: Glass Data, Category Quantifications

The 3497 objectscores can take only 49 different values, of which only 47 actually occur in the data. They are plotted in figure 27. Point labels are first letters of the two corresponding SES categories, first letter for the fathers, second letter for the sons.

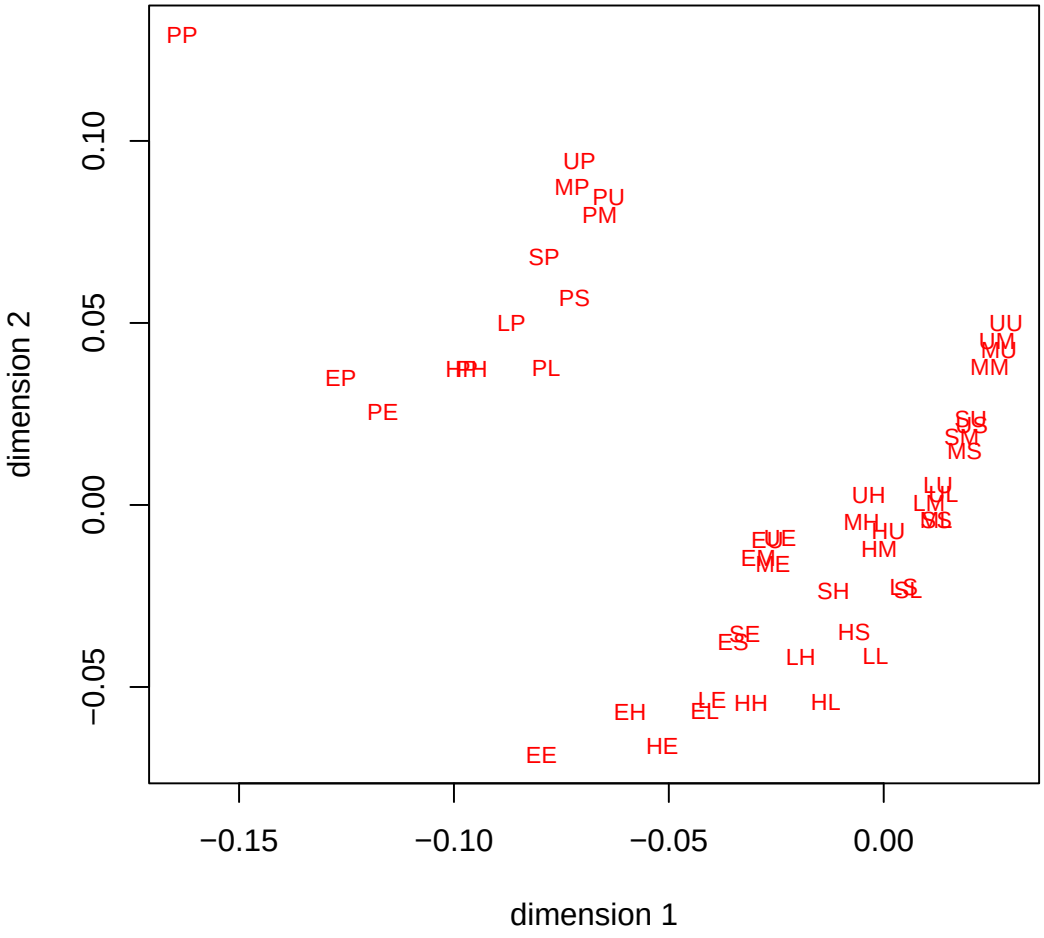
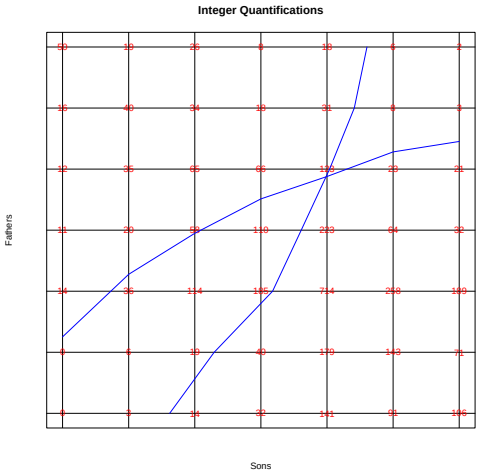
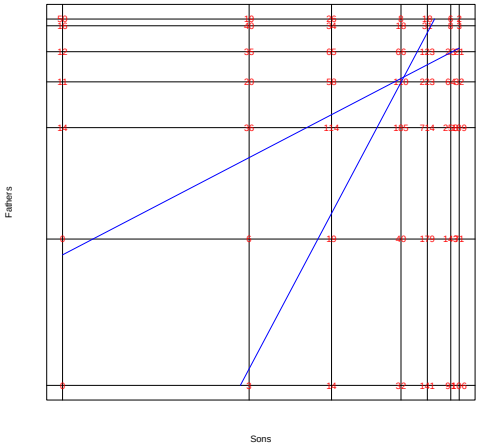


Figure 13: Glass Data, Object Scores

Next, we look at *regression plots*, made with the utility `regressionPlotter()`. One-dimensional category quantifications for rows and columns are used to locate row and column categories on the horizontal and vertical axes. Frequencies from the table are used to label the intersections of the corresponding vertical and horizontal lines. We then compute the regression lines, using row and column averages of the category quantifications, for these transformed variables. In the first plot we see what happens if we use equally spaced scores for the categories of both fathers and sons. Regressions are not quite linear. Then we use the first dimension of the OCA quantifications, which linearizes the regressions. And in the third plot we use the second dimension, which again linearizes the regressions, but permutes the rows

and colums of the table.


Integer Quantifications
Fathers
Sons


First Corals Dimension
Fathers
Sons

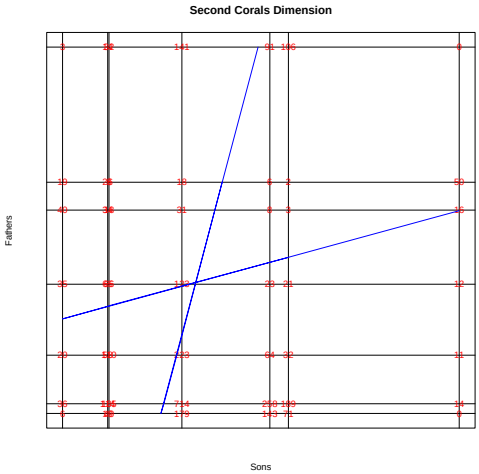

Second Corals Dimension
Fathers
Sons

Figure 14: Glass Data, Regression Plots

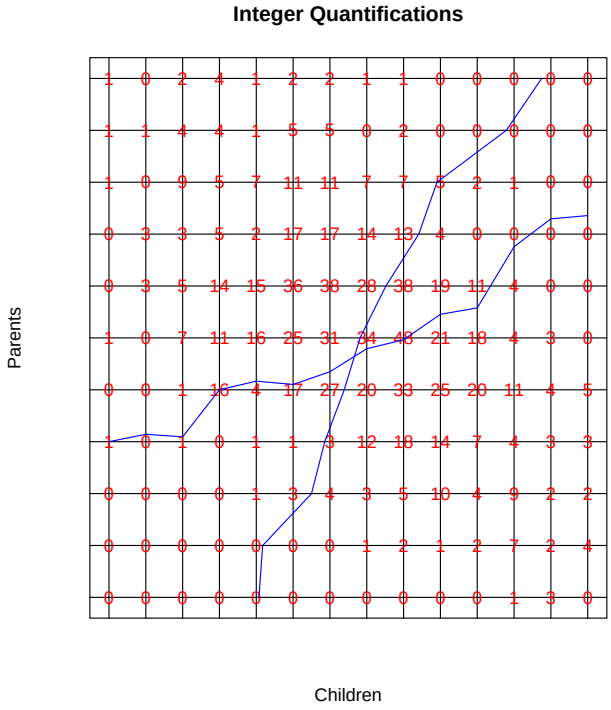
###Galton

To illustrate some of the additional `corals()` options we use the classical father-child RFF height data of Galton (1889). It has mid-parent height in the rows and mid-adult-child height in the columns.

```
data (galton, package = "anacor")
galton <- as.matrix (galton)
galton <- galton[nrow (galton):1, ]
galton
```

```
##            below 62.2 62.2 63.2 64.2 65.2 66.2 67.2 68.2 69.2 70.2 71.2 72.
## below 64.5          1    0    2    4    1    2    2    1    1    0    0
## 64.5                1    1    4    4    1    5    5    0    2    0    0
## 65.5                1    0    9    5    7   11   11    7    7    5    2
## 66.5                0    3    3    5    2   17   17   14   13    4    0
## 67.5                0    3    5   14   15   36   38   28   38   19   11
## 68.5                1    0    7   11   16   25   31   34   48   21   18
## 69.5                0    0    1   16    4   17   27   20   33   25   20    1
## 70.5                1    0    1    0    1    1    3   12   18   14    7
## 71.5                0    0    0    0    1    3    4    3    5   10    4
## 72.5                0    0    0    0    0    0    0    1    2    1    2
## above 72.5          0    0    0    0    0    0    0    0    0    0    0
##            73.2 above 73.2
## below 64.5    0          0
## 64.5          0          0
## 65.5          0          0
## 66.5          0          0
## 67.5          0          0
## 68.5          3          0
## 69.5          4          5
## 70.5          3          3
## 71.5          2          2
## 72.5          2          4
## above 72.5    3          0
```

The regression plots from a one-dimensional `corals()`, with default options, in the familiar before and after format, are in figure 15.


Integer Quantifications
Parents
Children

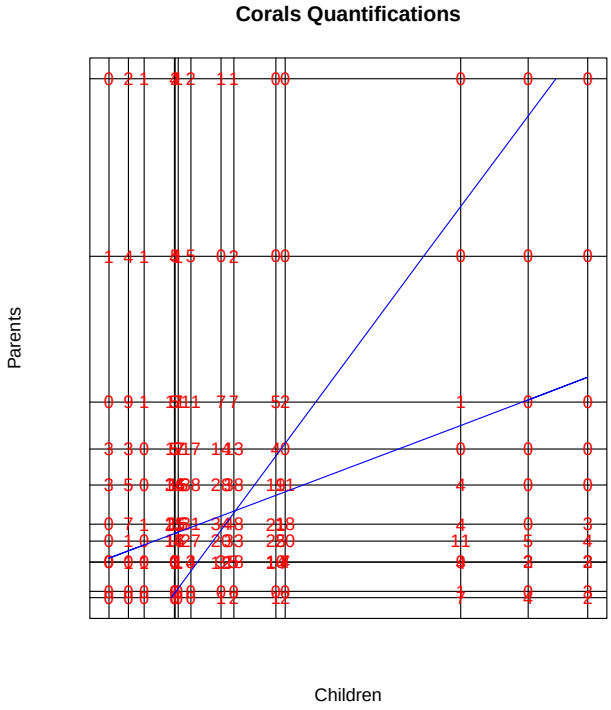
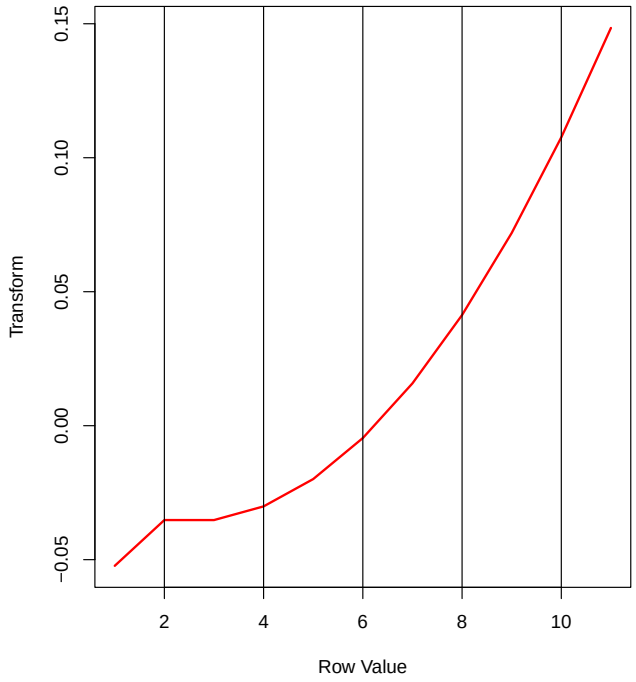

Corals Quantifications
Parents
Children

Figure 15: Galton Data, Regression Plots

We see some deviations from monotonicity and the ends of the scale, where some columns of the table are interchanged. This is presumably because of the small number of observations in the extreme categories. We repeat the analysis with ordinal transformations of degree 2 (i.e. piecewise quadratics, differentiable at the knots, and monotone at the data points) and equally spaced knots.

```
galton_knots = c(2, 4, 6, 8, 10)
h <- corals(
  galton,
  ndim = 1,
  xord = TRUE,
  yord = TRUE,
  xdeg = 2,
  ydeg = 2,
  xknots = galton_knots,
  yknots = galton_knots
)
```

The transformations of the variables are in figure 16. They show some clear deviations from linearity.


Transform
0.15
0.10
0.05
0.00
−0.05
2
4
6
8
10
Row Value

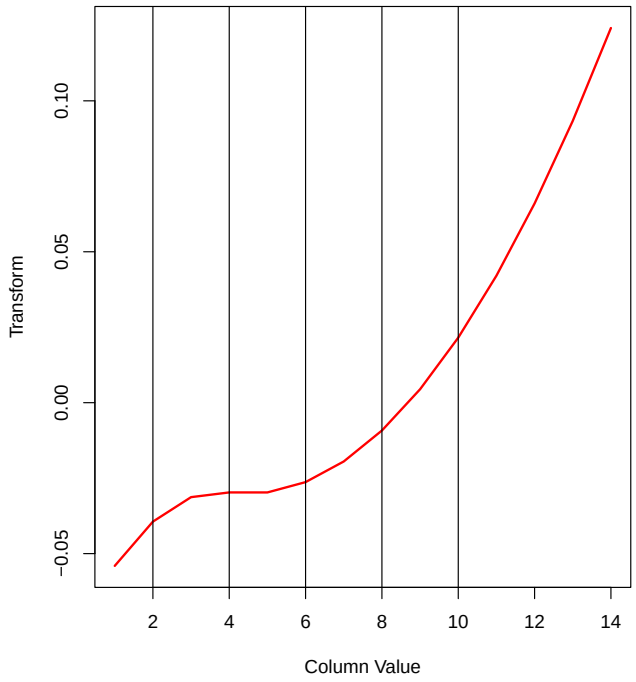

Transform
0.10
0.05
0.00
−0.05
2
4
6
8
10
12
14
Column Value

Figure 16: Galton Data, Transformation Plots

Chapter 7

Canonical Correspondence Analysis and coranals()

7.1 Introduction

7.2 Equations

Canonical analysis of GX and H.

7.3 Examples

Chapter 8

Nonlinear Principal Component Analysis and princals()

8.1 Introduction

princals, principals, Shepard-Kruskal, mdrace, history

8.2 Equations

Suppose all m blocks each contain only a single variable. Then the Burt matrix is the correlation matrix of the H_j, which are all $n \times 1$ matrices in this case. It follows that MVAOS maximizes the sum of the r largest eigenvalues of the correlation matrix over transformations, i.e. MVAOS is *nonlinear principal component analysis* (De Leeuw 2014).

8.3 Examples

8.3.1 Thirteen Personality Scales

We use the same data as before for an NLPCA with all blocks of rank one, all variables ordinal, and splines of degree 2.

```
epi_copies <- rep (1, 13)
epi_ordinal <- rep (TRUE, 13)
h <- princals(epi, epi_knots, epi_degrees, epi_ordinal, epi_copies)
```

In 19 iterations we find minimum loss 0.7330408597. The object scores are in figure 17 and the transformation plots in figure 18. NLPCA maximizes the sum of the two largest eigenvalues of the correlation matrix of the variables. Before transformation the eigenvalues are 4.0043586779, 2.6702003226, 1.9970911525, 0.8813983145, 0.6571462887, 0.6299946278, 0.524689638, 0.4657021776, 0.3457514762, 0.3403360653, 0.2767530816, 0.1835448669, 0.0230333104, after transformation they are 4.1956970306, 2.7452518707, 1.6036670387, 0.8209125803, 0.718259977, 0.676961949, 0.5185327518, 0.4544125071, 0.4197680221, 0.3519542253, 0.2932653658, 0.1700269788, 0.0312897028. The sum of the first two goes from 6.6745590006 to 6.9409489014.

```
plot(h$objectscores, xlab = "dim1", ylab = "dim2", col = "RED", cex = .5)
```

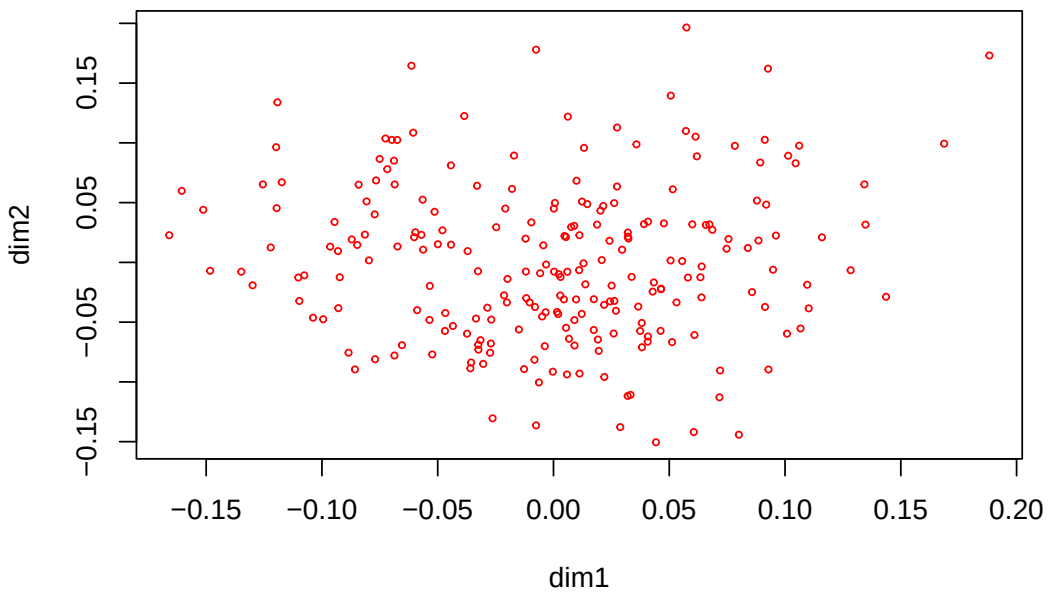
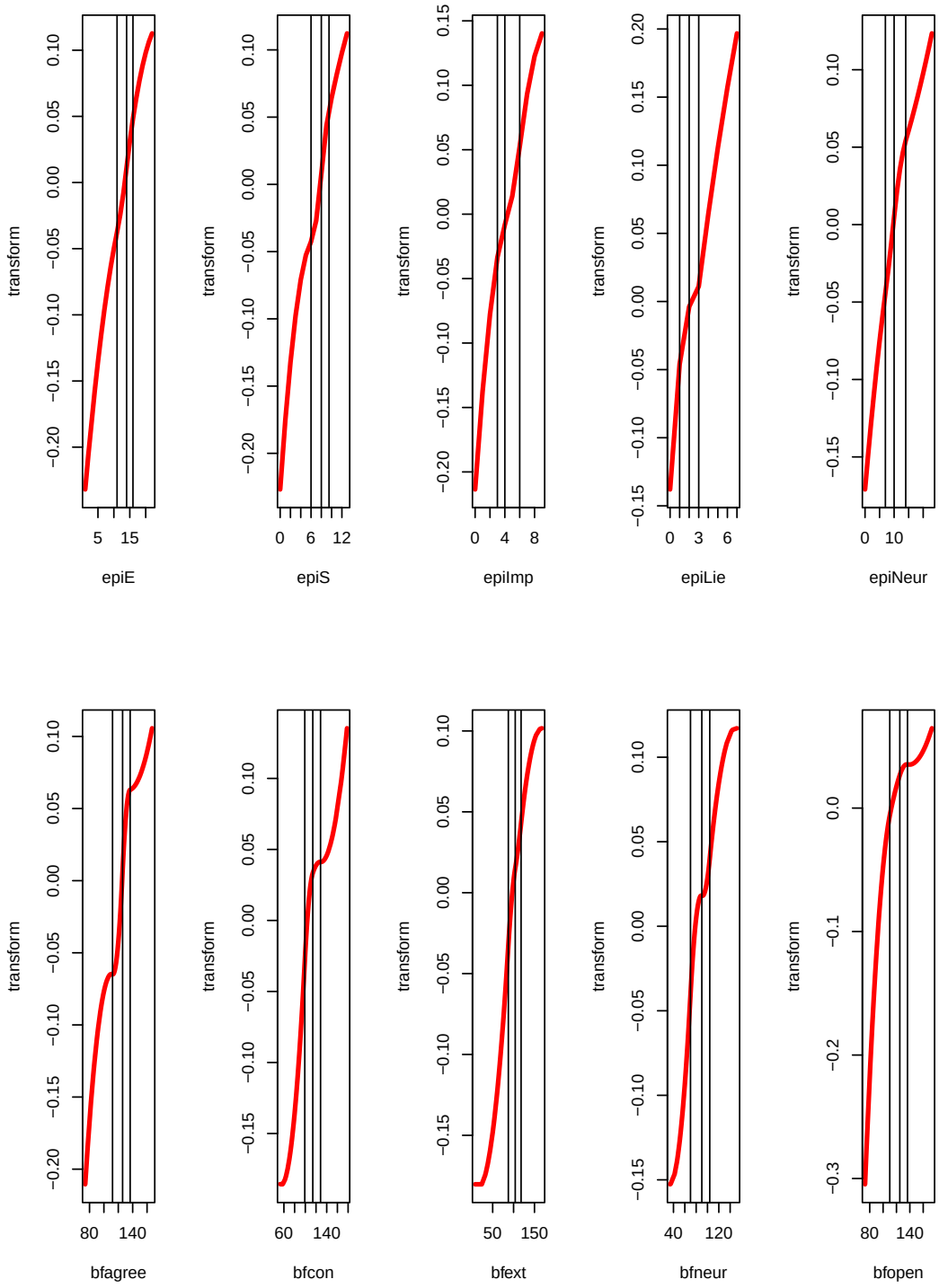


Figure 17: Personality Scales, Object Scores, Single Ordinal, Degree Two


transform
epiE
epiS
epiImp
epiLie
epiNeur
bfagree
bfcon
bfext
bfneur
bfopen

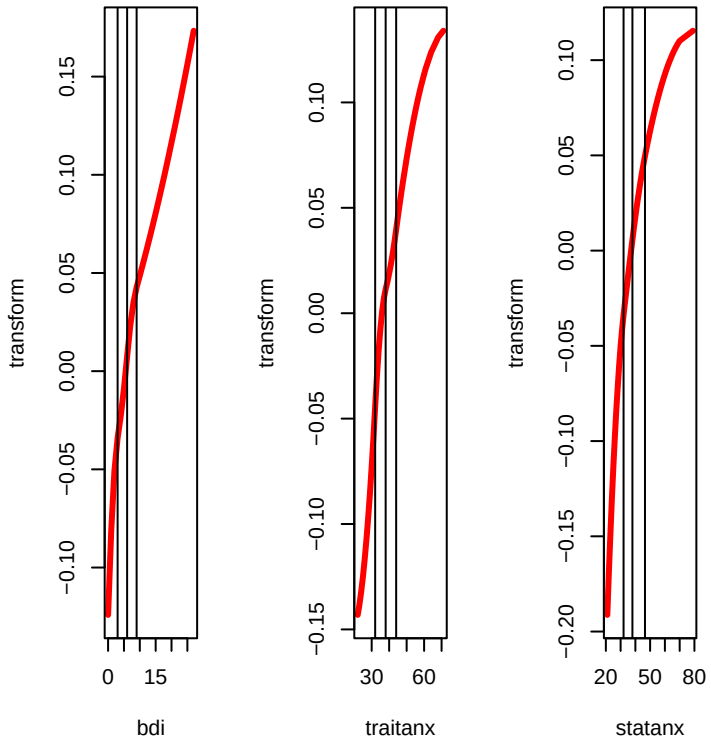


Figure 18: Personality Scales, Transformations, Single Ordinal, Degree Two

We repeat the analysis with ordinal variables of degree two, without interior knots. Thus we the transformation plots will be quadratic polynomials that are monotone over the range of the data.

```
h <- princals(epi, knotsE(epi), epi_degrees, epi_ordinal)
```

In 21 iterations we find minimum loss 0.7392792203. The object scores are in figure 19 and the transformation plots in figure 20. The eigenvalues are now 4.0845452813, 2.6942027568, 1.8268475832, 0.8731782338, 0.6698533738, 0.6503449076, 0.5406240035, 0.4597013756, 0.3666352518, 0.3470225888, 0.2847374711, 0.1783404951, 0.0239666775, with sum of the first two equal to 6.7787480381.

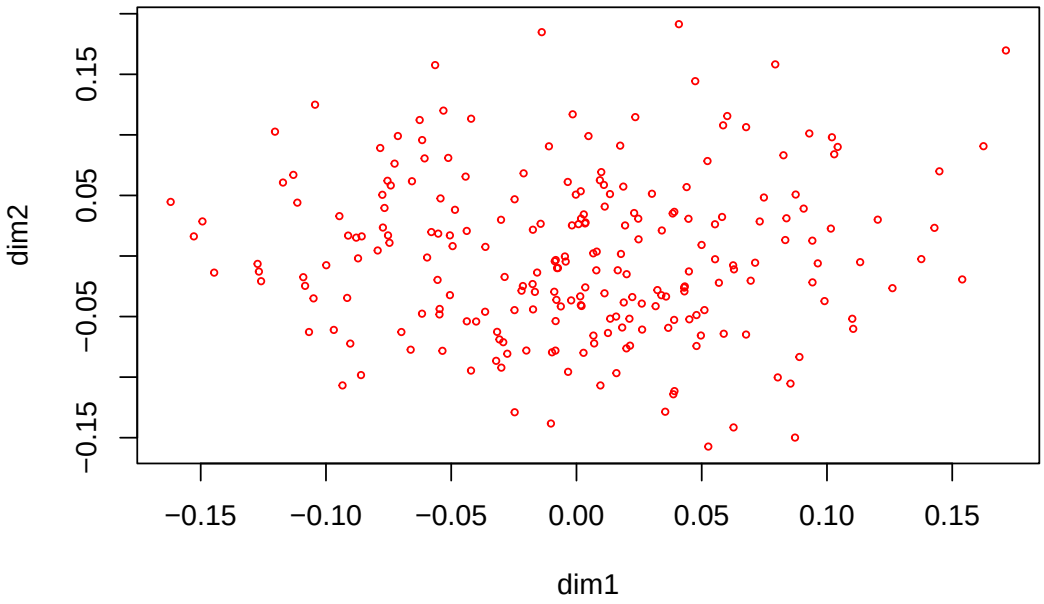
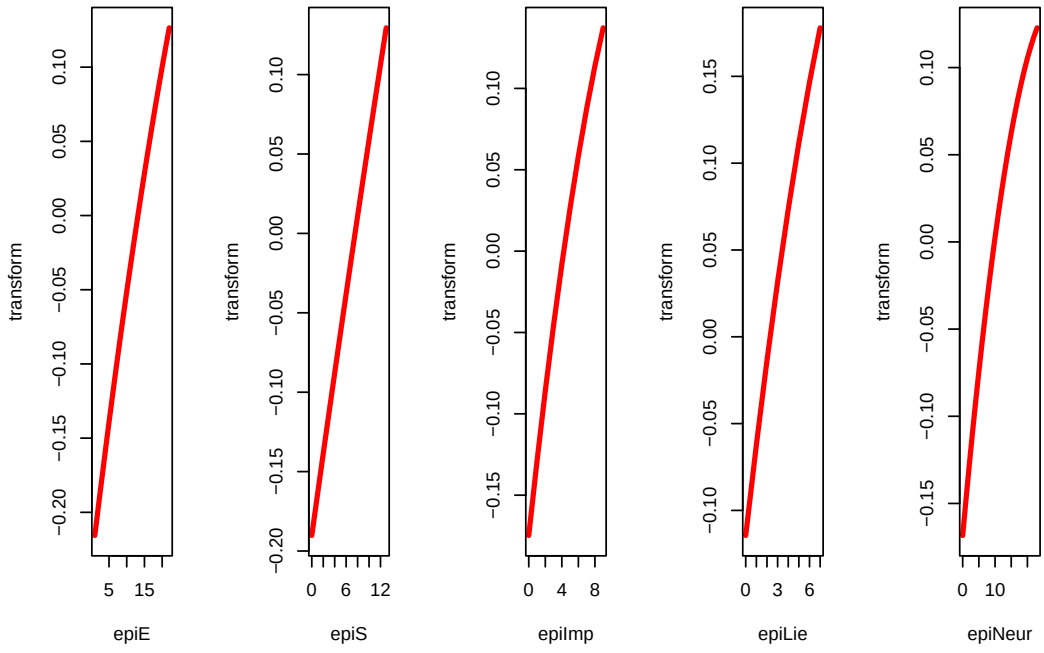
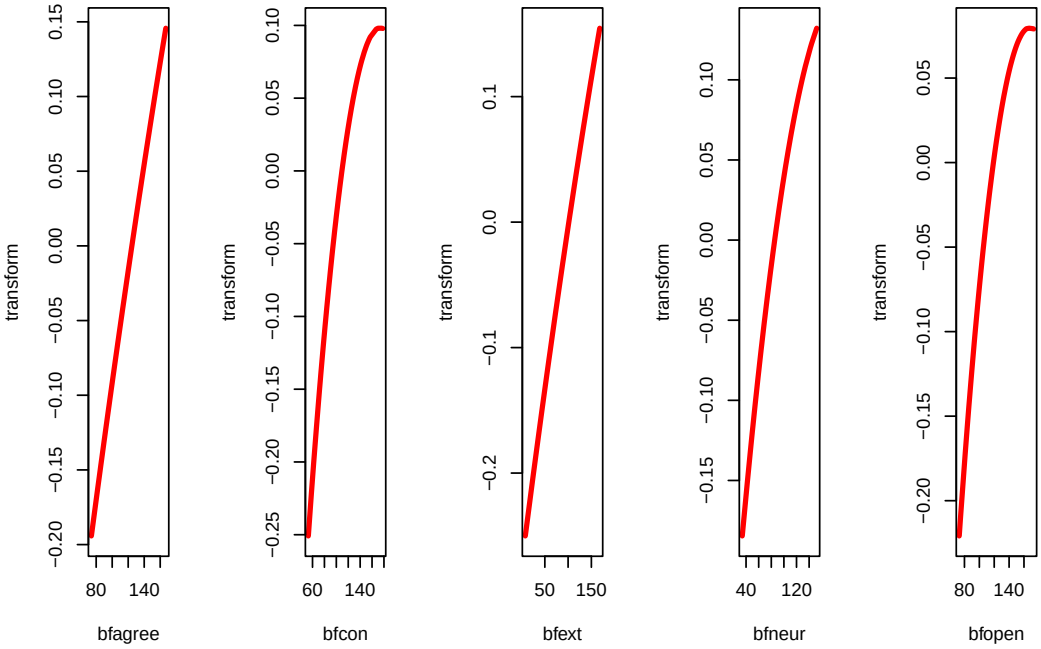


Figure 19: Personality Scales, Object Scores, Single Numerical, Degree Two




transform
bfagree
bfcon
bfext
bfneur
bfopen

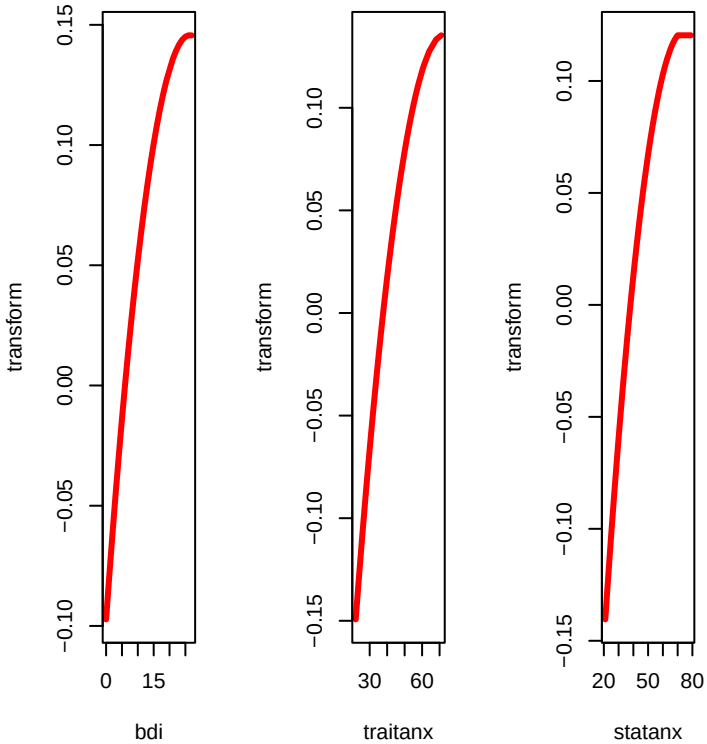

transform
bdi
traitanx
statanx

Figure 20: Personality Scales, Transformations, Single Numerical, Degree Two

Chapter 9

Canonical Analysis and canals()

9.1 Equations

If there are only two blocks the generalized eigenvalue problem for the Burt matrix becomes

$$\begin{bmatrix} D_1 & C_{12} \\ C_{21} & D_2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = 2\lambda \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix},$$

which we can rewrite as

$$\begin{aligned} C_{12}a_2 &= (2\lambda - 1)D_1a_1, \\ C_{21}a_1 &= (2\lambda - 1)D_2a_2, \end{aligned}$$

from which we see that MVAOS maximizes the sum of the r largest canonical correlations between H_1 and H_2. See also Van der Velden (2012).

9.2 Examples

Chapter 10

Multiple Regression and morals()

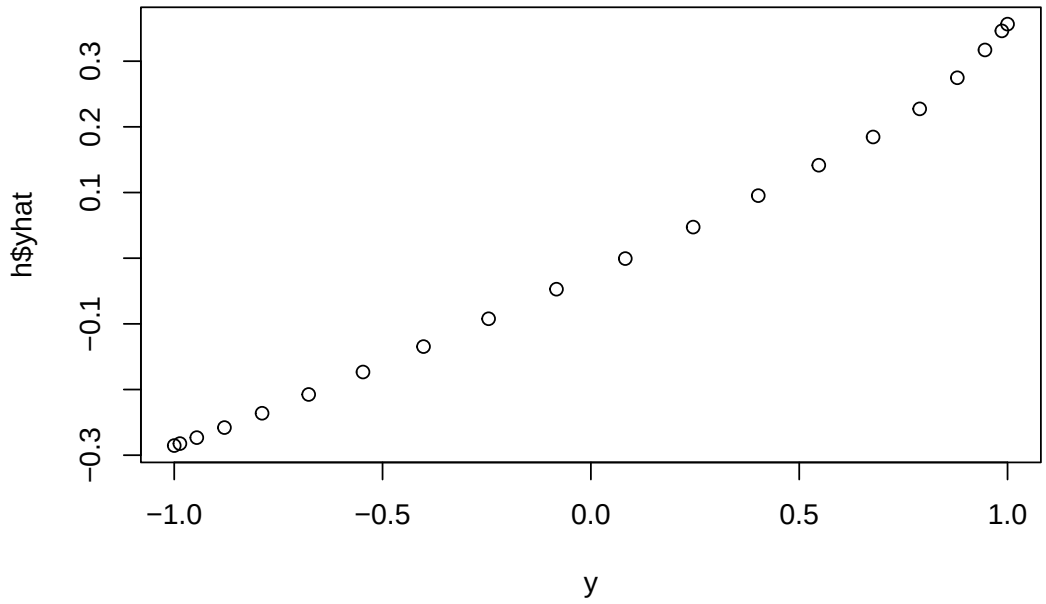
If the second block only contains a single copy of a single variable then we choose transformations that maximize the multiple correlation of that variable and the variables in the first block.

10.1 Equations

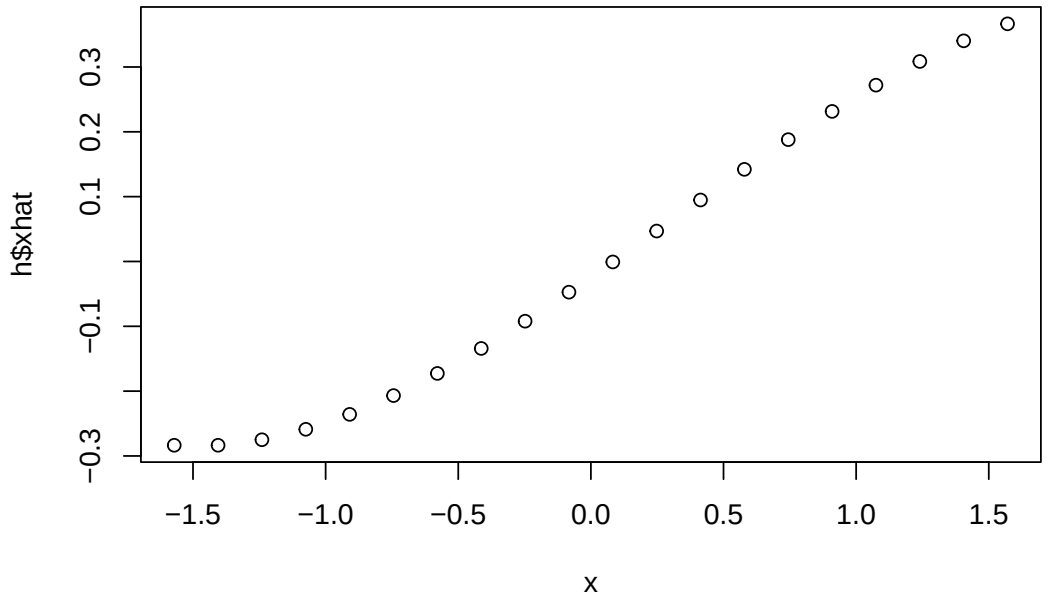
10.2 Examples

10.2.1 Polynomial Regression

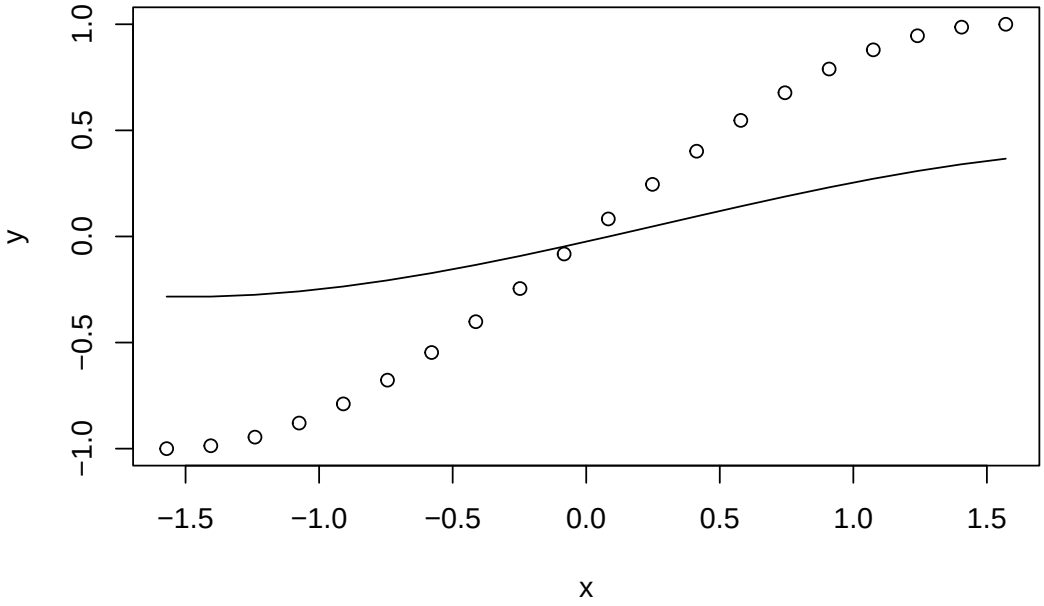
```
x <- center(as.matrix (seq (0, pi, length = 20)))
y <- center(as.matrix (sin (x)))
h<- morals (x, y, xknots = knotsE(x), xdegrees = 3, xordinal = TRUE)
plot(y, h$yhat)
```




```
plot(x, h$xhat)
```




```
plot (x, y)
lines (x, h$ypred)
```



10.2.2 Gases with Convertible Components

We analyze a regression example, using data from Neumann, previously used by Willard Gibbs, and analyzed with regression in a still quite readable article by Wilson (1926). Wilson's analysis was discussed and modified using splines in Gifi (1990) (pages 370-376). In the regression analysis in this section we use two copies of temperature, with spline degree zero, and the first copy ordinal. For pressure and the dependent variable density we use a single ordinal copy with spline degree two.

```
data (neumann, package = "homals")
xneumann <- neumann[, 1:2]
yneumann <- neumann[, 3, drop = FALSE]
xdegrees <- c(0,2)
```

```
h <- morals (xneumann, yneumann, xdegrees = c(0,2), xcopies = c(2,1))
```

In 58 iterations we find minimum loss 0.026805848, corresponding with a multiple correlation of 0.8956511807. The object scores are in figure 21 plotted against the original variables (not the transformed variables), and the transformation plots in are figure 22.

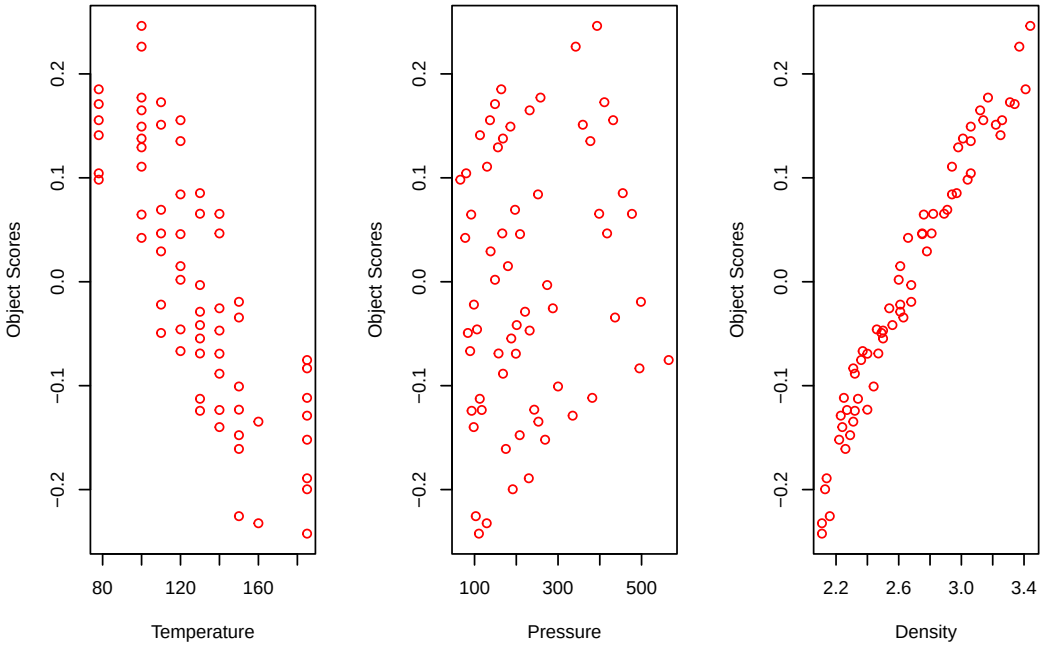


Figure 21: Gases with Convertible Components, Objects Scores

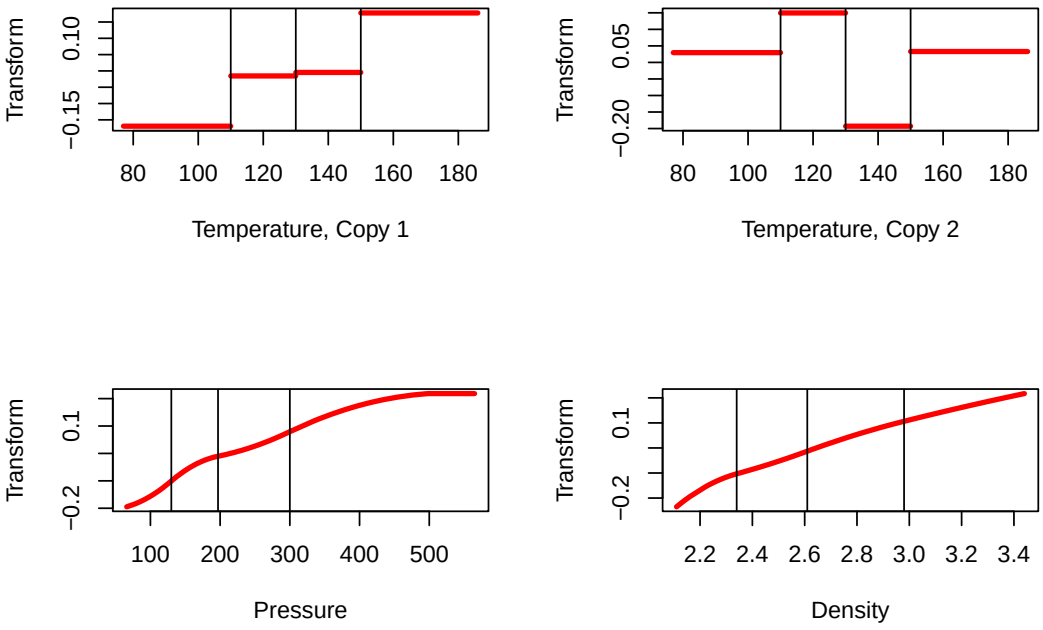


Figure 22: Gases with Convertible Components, Transformations

10.3 Conjoint Analysis and addals()

Chapter 11

Discriminant Analysis and criminals()

11.1 Equations

If the second block contains more than one copy of a single variable and we use binary indicator coding for that variable, then we optimize the eigenvalue (between/within ratio) sums for a canonical discriminant analysis.

11.2 Examples

11.2.1 Iris data

The next example illustrates (canonical) discriminant analysis, using the obligatory Anderson-Fisher iris data. Since there are three species of iris, we use two copies for the species variable. The other four variables are in the same block, they are transformed using piecewise linear monotone splines with five knots.

```
data(iris, package="datasets")
iris_vars <- names(iris)
iris_knots <- knotsQ(iris[,1:4])
```

```
x <- as.matrix(iris[,1:4])
y <- as.matrix(iris[[5]])
```

```
h <- criminals (x, y, xdegrees = 1)
```

In 191 iterations we find minimum loss 0.0302260098. The object scores are in figure 23 plotted against the original variables (not the transformed variables), and the transformation plots are in figure 24.

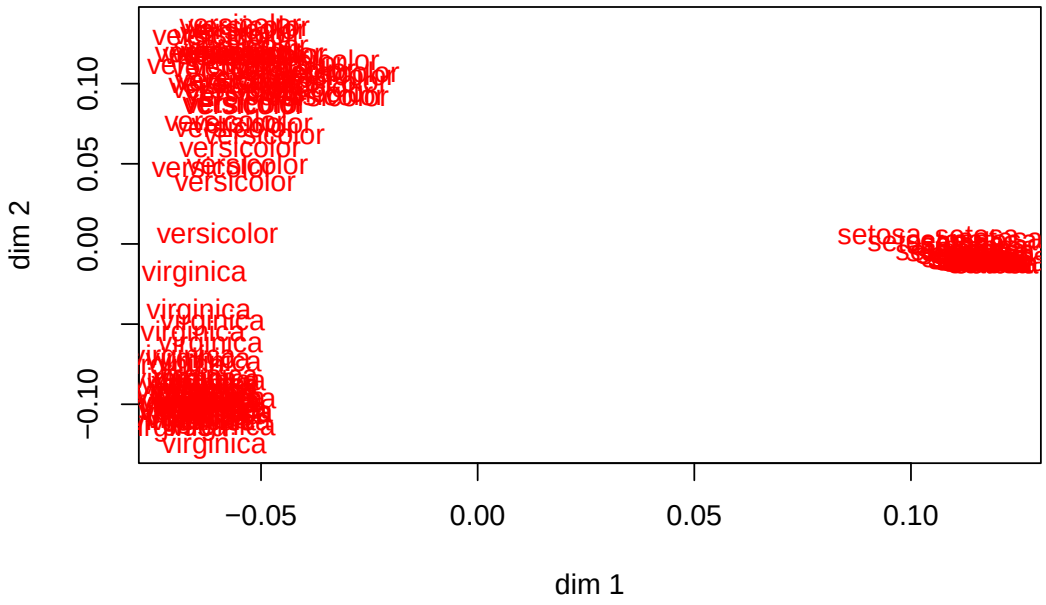


Figure 23: Iris Data, Objects Scores

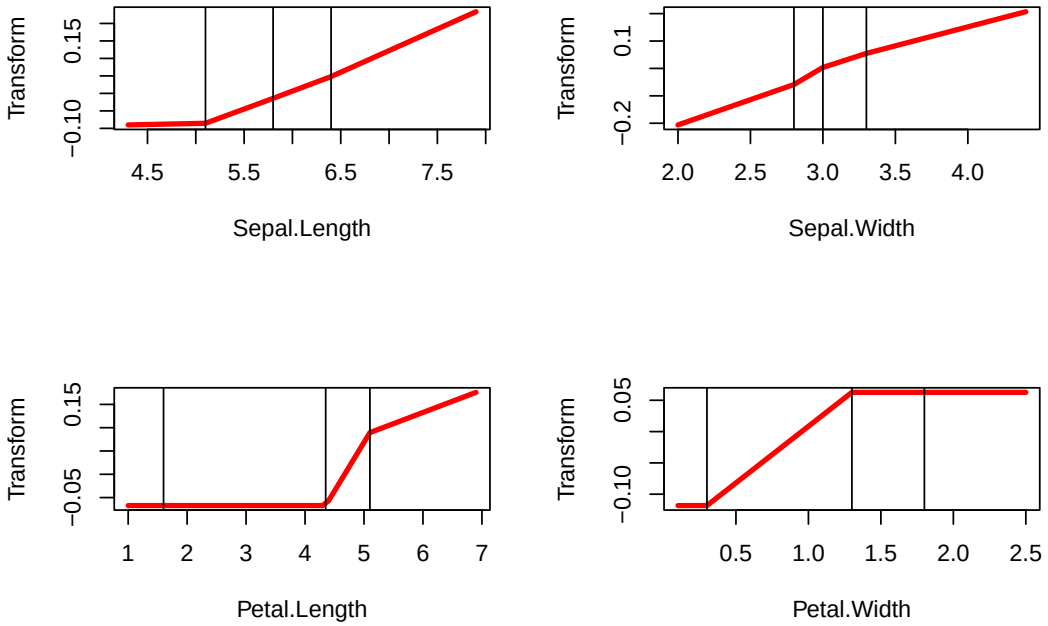


Figure 24: Iris Data, Transformations

Discriminant analysis decomposes the total dispersion matrix T into a sum of a between-groups dispersion B and a within-groups dispersion W, and then finds directions in the space spanned by the variables for which the between-variance is largest relative to the total variance. MVAOS optimizes the sum of the r largest eigenvalues of $T^{-1}B$. Before optimal transformation these eigenvalues for the iris data are `r`, after transformation they are `r`.

Chapter 12

Multiblock Canonical Correlation and overals()

12.1 Equations

12.2 Examples

12.2.1 Thirteen Personality Scales

This is the same example as before, but now we group the five scales from the Eysenck Personality Inventory and the five from the Big Five inventory into blocks. The remaining three variables define three separate blocks. No copies are used, and we use monotone cubic splines with the interior knots at the quartiles.

```
epi_knots <- lapply (epi, function (x) fivenum (x)[2:4])
epi_degrees <- rep (3, 13)
epi_blocks <- c(1,1,1,1,1,2,2,2,2,2,3,4,5)
```

```
h <- overals(epi, epi_blocks, epi_copies, epi_knots, epi_degrees, epi_ordinal)
```

In 191 iterations we find minimum loss 0.0302260098. The object scores are in figure 25 and the transformation plots in figure 26.

Figure 25: Personality Scales, Multi-Set, Objects Scores

Figure 26: Personality Scales, Multi-Set, , Transformations

Chapter 13

Code

13.1 R Code

13.1.1 Driver

```
source ("rcode/gifiEngine.R")
source ("rcode/gifiUtilities.R")
source ("rcode/gifiWrappers.R")
source ("rcode/gifiStructures.R")
source ("rcode/aspectEngine.R")
source ("rcode/theAspects.R")
source ("rcode/matrix.R")
source ("rcode/coneRegression.R")
source ("splineBasis.R")
source ("coding.R")
```

13.1.2 Engine

```
gifiEngine <-
  function (gifi,
            ndim,
```

```
          itmax,
          eps,
          seed,
          verbose) {
  set.seed (seed)
  nobs <- nrow (as.matrix (gifi[[1]][[1]]$data))
  nsets <- length (gifi)
  nvars <- sum (sapply (gifi, length))
  itel <- 1
  if (nvars == 1)
    stop("a gifiAnalysis needs more than one variable")
  x <- matrix (rnorm (nobs * ndim), nobs, ndim)
  x <- gsRC (center (x))$q
  xGifi <- xGifi (gifi, x)
  fold <- 0
  asets <- 0
  for (i in 1:nsets) {
    gifiSet <- gifi[[i]]
    xGifiSet <- xGifi[[i]]
    nvarset <- length (gifiSet)
    ha <- matrix (0, nobs, ndim)
    activeCount <- 0
    for (j in 1:nvarset) {
      if (gifiSet[[j]]$active) {
        activeCount <- activeCount + 1
        ha <- ha + xGifiSet[[j]]$scores
      }
    }
    if (activeCount > 0) {
      asets <- asets + 1
      fold <- fold + sum ((x - ha) ^ 2)
    }
  }
  fold <- fold / (asets * ndim)
  repeat {
    xz <- matrix(0, nobs, ndim)
    fnew <- fmid <- 0
```

```
      for (i in 1:nsets) {
        gifiSet <- gifi[[i]]
        xGifiSet <- xGifi[[i]]
        nvarset <- length (gifiSet)
        hh <- matrix (0, nobs, 0)
        activeCount <- 0
        for (j in 1:nvarset) {
          if (gifiSet[[j]]$active) {
            activeCount <- activeCount + 1
            hh <- cbind (hh, xGifiSet[[j]]$transform)
          }
        }
        if (activeCount == 0)
          next
        lf <- lsRC (hh, x)
        aa <- lf$solution
        rs <- lf$residuals
        kappa <- max (eigen (crossprod (aa))$values)
        fmid <- fmid + sum (rs ^ 2)
        target <- hh + tcrossprod (rs, aa) / kappa
        hh <- matrix (0, nobs, 0)
        scopies <- 0
        for (j in 1:nvarset) {
          gifiVar <- gifiSet[[j]]
          jdata <- gifiVar$data
          jbasis <- gifiVar$basis
          jcopies <- gifiVar$copies
          jdegree <- gifiVar$degree
          jties <- gifiVar$ties
          jmissing <- gifiVar$missing
          jordinal <- gifiVar$ordinal
          ja <- aa[scopies + 1:jcopies, , drop = FALSE]
          jtarget <- target[, scopies + 1:jcopies, drop = FALSE]
          hj <-
            gifiTransform (
              data = jdata,
              target = jtarget,
```

```
            basis = jbasis,
            copies = jcopies,
            degree = jdegree,
            ordinal = jordinal,
            ties = jties,
            missing = jmissing
          )
        hj <- gsRC(normalize (center (hj)))$q
        sc <- hj %*% ja
        xGifi[[i]][[j]]$transform <- hj
        xGifi[[i]][[j]]$weights <- ja
        xGifi[[i]][[j]]$scores <- sc
        xGifi[[i]][[j]]$quantifications <-
          lsRC(jbasis, sc)$solution
        activeCount <- 0
        if (gifiSet[[j]]$active) {
          activeCount <- activeCount + 1
          hh <- cbind (hh, hj)
        }
        scopies <- scopies + jcopies
      }
      if (activeCount > 0) {
        ha <- hh %*% aa
        xz <- xz + ha
        fnew <- fnew + sum ((x - ha) ^ 2)
      }
    }
    fmid <- fmid / (asets * ndim)
    fnew <- fnew / (asets * ndim)
    if (verbose)
      cat(
        "Iteration: ",
        formatC (itel, width = 3, format = "d"),
        "fold: ",
        formatC (
          fold,
          digits = 8,
```

```
            width = 12,
            format = "f"
          ),
          "fmid: ",
          formatC (
            fmid,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "fnew: ",
          formatC (
            fnew,
            digits = 8,
            width = 12,
            format = "f"
          ),
          "\n"
        )
       if (((itel == itmax) || ((fold - fnew) < eps)) && (itel > 1))
        break
      itel <- itel + 1
      fold <- fnew
      x <- gsRC (center (xz))$q
    }
    return (list (
      f = fnew,
      ntel = itel,
      x = x,
      xGifi = xGifi
    ))
  }

gifiTransform <-
  function (data,
            target,
            basis,
```

```
          copies,
          degree,
          ordinal,
          ties,
          missing) {
  nobs <- nrow (as.matrix (data))
  h <- matrix (0, nobs, copies)
  if (degree == -1) {
    if (ordinal) {
      h[, 1] <-
        coneRegression (
          data = data,
          target = target[, 1],
          type = "c",
          ties = ties,
          missing = missing
        )
    }
    else {
      h[, 1] <-
        coneRegression (
          data = data,
          target = target[, 1],
          basis = basis,
          type = "s",
          missing = missing
        )
    }
  }
  if (degree >= 0) {
    if (ordinal) {
      h[, 1] <-
        coneRegression (
          data = data,
          target = target[, 1],
          basis = basis,
          type = "i",
```

```
            ties = ties,
            missing = missing
          )
      }
      else {
        h[, 1] <-
          coneRegression (
            data = data,
            target = target[, 1],
            basis = basis,
            type = "s",
            ties = ties,
            missing = missing
          )
      }
    }
    if (copies > 1) {
      for (l in 2:copies)
        h[, l] <-
          coneRegression (
            data = data,
            target = target[, l],
            basis = basis,
            type = "s",
            ties = ties,
            missing = missing
          )
    }
    return (h)
  }
```

13.1.3 Aspect Engine

```
aspectEngine <-
  function (gifi,
```

```
          afunc,
          eps = 1e-6,
          itmax = 100,
          verbose = 1,
          monotone = FALSE,
          ...) {
  nsets <- length (gifi)
  for (i in 1:nsets) {
    gifiSet <- gifi[[i]]
    nvars <- length (gifiSet)
    for (j in 1:nvars) {
      gifiVar <- gifiSet[[j]]
      q <- gifiVar$qr$q
    }
  }
  itel <- 1
  tdata <- matrix (0, n, m)
  for (j in 1:m) {
    tdata[, j] <- bd$x[[j]]
  }
  tdata <- apply (tdata, 2, function (z)
    z - mean (z))
  tdata <- apply (tdata, 2, function (z)
    z / sqrt (sum (z ^ 2)))
  corr <- crossprod (tdata)
  af <- afunc(corr, ...)
  fold <- af$f
  g <- af$g
  repeat {
    for (j in 1:m) {
      target <- drop (tdata[, -j] %*% g[-j, j])
      k <- bd$b[[j]]
      v <- bd$v[[j]]
      u <- drop (crossprod(k, target))
      s0 <- sum(target * tdata[, j])
      if (ordinal[j]) {
        ns <- nnls(v, u)
```

```
      rr <- residuals(ns)
      tt <- drop(k %*% rr)
    } else
      tt <- drop (k %*% u)
    tt <- tt - mean (tt)
    sq <- sum(tt ^ 2)
    if (sq > 1e-15) {
      tt <- tt / sqrt (sq)
      tdata[, j] <- tt
    }
    s1 <- sum(target * tdata[, j])
    if (verbose > 1)
      cat (
        "**** Variable",
        formatC(j, width = 3, format = "d"),
        "Before",
        formatC(
          s0,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "After",
        formatC(
          s1,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "\n"
      )
    if (!monotone) {
      corr <- cor (tdata)
      af <- afunc (corr, ...)
      fnew <- af$f
      g <- af$g
    }
```

```
    }
    if (monotone) {
      corr <- cor (tdata)
      af <- afunc (corr, ...)
      fnew <- af$f
      g <- af$g
    }
    if (verbose > 0)
      cat(
        "Iteration: ",
        formatC (itel, width = 3, format = "d"),
        "fold: ",
        formatC (
          fold,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "fnew: ",
        formatC (
          fnew,
          digits = 8,
          width = 12,
          format = "f"
        ),
        "\n"
      )
    if ((itel == itmax) || ((fnew - fold) < eps))
      break
    itel <- itel + 1
    fold <- fnew
  }
  return (list (
    tdata = tdata,
    f = fnew,
    r = corr,
    g = g
```

```
    ))
  }
```

13.1.4 Some Aspects

```
maxeig <- function (r, p) {
  e <- eigen (r)
  f <- sum (e$values[1:p])
  g <- tcrossprod(e$vectors[,1:p])
  return (list (f = f, g = g))
}

maxcor <- function (r, p) {
  f <- sum (r ^ p)
  g <- p * (r ^ (p - 1))
  return (list (f = f, g = g))
}

maxabs <- function (r, p) {
  f <- sum (abs(r) ^ p)
  g <- p * (abs(r) ^ (p - 1)) * sign(r)
  return (list (f = f, g = g))
}

maxdet <- function (r) {
  f <- -log(det (r))
  g <- -solve(r)
  return (list (f = f, g = g))
}

maxsmc <- function (r, p) {
  beta <- solve (r[-p,-p], r[p,-p])
  f <- sum (beta * r[p,-p])
  h <- rep (1, nrow (r))
  h[-p] <- -beta
```

```
  g <- -outer (h, h)
  return (list (f = f, g = g))
}

maxsum <- function (r, p) {
  f <- sum (sqrt (r ^ 2 + p))
  g <- r / sqrt (r ^ 2 + p)
  return (list (f = f, g = g))
}

maximage <- function (r) {
  n <- nrow(r)
  f <- 0
  g <- matrix (0, n, n)
  for (p in 1:n) {
    beta <- solve (r[-p,-p], r[p,-p])
    f <- f + sum (beta * r[p,-p])
    h <- rep (1, nrow (r))
    h[-p] <- -beta
    g <- g - outer (h, h)
  }
  return (list (f = f, g = g))
}

maxfac <- function (r, p) {
  fa <- factanal (NULL, p, covmat = r, rotation = "none")
  s <- tcrossprod (fa$loadings) + diag (fa$unique)
  g <- - solve (s)
  f <- -log(det (s)) + sum (g * r)
  return (list (f = f, g = g))
}
```

13.1.5 Structures

```
makeGifiVariable <-
  function (data,
            knots,
            degree,
            ordinal,
            ties,
            copies,
            missing,
            name) {
    there <- which (!is.na (data))
    notthere <- which (is.na (data))
    nmis <- length (notthere)
    nobs <- length (data)
    if (length (there) == 0)
      stop ("a gifiVariable cannot be completely missing")
    work <- data[there]
    if (degree == -1) {
      type <- "categorical"
      basis <- makeIndicator (work)
      if (ncol (basis) == 1) {
        stop ("a gifiVariable must have more than one category")
      }
      if (ncol (basis) == 2) {
        type <- "binary"
      }
    }
    if (degree >= 0) {
      if (length (knots) == 0)
        type <- "polynomial"
      else
        type <- "splinical"
      basis <- bsplineBasis (work, degree, knots)
    }
     if (nmis > 0)
      basis <- makeMissing (data, basis, missing)
    copies <- min (copies, ncol (basis) - 1)
    qr <- gsRC (center (basis))
```

```
    if (qr$rank == 0)
      stop ("a gifiVariable cannot be completely zero")
    return (structure (
      list (
        data = data,
        basis = basis,
        qr = qr,
        copies = copies,
        degree = degree,
        ties = ties,
        ordinal = ordinal,
        name = name,
        type = type
      ),
      class = "gifiVariable"
    ))
  }

makeGifiSet <-
  function (data,
            knots,
            degrees,
            ordinal,
            ties,
            copies,
            missing,
            names) {
    nvars <- ncol (data)
    varList <- as.list (1:nvars)
    for (i in 1:nvars) {
      varList[[i]] <-
        makeGifiVariable (
          data = data[, i],
          knots = knots[[i]],
          degree = degrees[i],
          ordinal = ordinal[i],
          ties[i],
```

```
          copies = copies[i],
          missing = missing[i],
          name = names[i]
        )
    }
    return (structure (varList, class = "gifiSet"))
  }

makeGifi <-
  function (data,
            knots,
            degrees,
            ordinal,
            ties,
            copies,
            missing,
            names,
            sets) {
    nsets <- max (sets)
    setList <- as.list (1:nsets)
    for (i in 1:nsets) {
      k <- which (sets == i)
      setList [[i]] <-
        makeGifiSet (
          data = data[, k, drop = FALSE],
          knots = knots[k],
          degrees = degrees[k],
          ordinal = ordinal[k],
          ties = ties[k],
          copies = copies[k],
          missing = missing[k],
          names = names[k]
        )
    }
    return (structure (setList, class = "gifi"))
  }
```

```
xGifiVariable <- function (gifiVariable, ndim) {
  basis <- gifiVariable$basis
  nbas <- ncol (basis)
  nobs <- length (gifiVariable$data)
  copies <- gifiVariable$copies
  transform <- matrix (0, nobs, copies)
  transform[, 1] <- drop(basis %*% (1:nbas))
  if (copies > 1) {
    for (i in 2:copies)
      transform[, i] <- drop (basis %*% rnorm (nbas))
  }
  transform <- gsRC (normalize (center (transform)))$q
  nbas <- ncol (transform)
  weights <- matrix (rnorm (nbas * ndim), nbas, ndim)
  scores <- transform %*% weights
  quantifications <- lsRC (basis, scores)$solution
  return (structure (
    list(
      transform = transform,
      weights = weights,
      scores = scores,
      quantifications = quantifications
    ),
    class = "xGifiVariable"
  ))
}

xGifiSet <- function (gifiSet, ndim) {
  nvars <- length (gifiSet)
  varList <- as.list (1:nvars)
  for (i in 1:nvars) {
    varList[[i]] <- xGifiVariable (gifiSet[[i]], ndim)
  }
  return (structure (varList, class = "xGifiSet"))
}

xGifi <- function (gifi, ndim) {
```

```
  nsets <- length (gifi)
  setList <- as.list (1:nsets)
  for (i in 1:nsets) {
    setList[[i]] <- xGifiSet (gifi[[i]], ndim)
  }
  return (structure (setList, class = "xGifi"))
}
```

13.1.6 Wrappers

```
homals <-
  function (data,
            knots = knotsD (data),
            degrees = -1,
            ordinal = FALSE,
            ndim = 2,
            ties = "s",
            missing = "m",
            names = colnames (data, do.NULL = FALSE),
            active = TRUE,
            itmax = 1000,
            eps = 1e-6,
            seed = 123,
            verbose = FALSE)  {
    nvars <- ncol (data)
    g <- makeGifi (
      data = data,
      knots = knots,
      degrees = reshape (degrees, nvars),
      ordinal = reshape (ordinal, nvars),
      ties = reshape (ties, nvars),
      copies = rep (ndim, ncol (data)),
      missing = reshape (missing, nvars),
      active = reshape (active, nvars),
      names = names,
```

```
    sets = 1:nvars
  )
  h <- gifiEngine(
    gifi = g,
    ndim = ndim,
    itmax = itmax,
    eps = eps,
    seed = seed,
    verbose = verbose
  )
  a <- v <- z <- d <- y <- o <- as.list (1:ncol(data))
  dsum <- matrix (0, ndim, ndim)
  nact <- 0
  for (j in 1:nvars) {
    jgifi <- h$xGifi[[j]][[1]]
    v[[j]] <- jgifi$transform
    a[[j]] <- jgifi$weights
    y[[j]] <- jgifi$scores
    z[[j]] <- jgifi$quantifications
    cy <- crossprod (y[[j]])
    if (g[[j]][[1]]$active) {
      dsum <- dsum + cy
      nact <- nact + 1
    }
    d[[j]] <- cy
    o[[j]] <- crossprod (h$x, v[[j]])
  }
  return (structure (
    list (
      transform = v,
      rhat = corList (v),
      objectscores = h$x,
      scores = y,
      quantifications = z,
      dmeasures = d,
      lambda = dsum / nact,
      weights = a,
```

```
        loadings = o,
        ntel = h$ntel,
        f = h$f
      ),
      class = "homals"
    ))
  }

corals <-
  function (data,
            ftype = TRUE,
            xknots = NULL,
            yknots = NULL,
            xdegree = -1,
            ydegree = -1,
            xordinal = FALSE,
            yordinal = FALSE,
            xties = "s",
            yties = "s",
            xmissing = "m",
            ymissing = "m",
            xname = "X",
            yname = "Y",
            ndim = 2,
            itmax = 1000,
            eps = 1e-6,
            seed = 123,
            verbose = FALSE) {
    if (ftype) {
      xy <- preCorals (as.matrix(data))
      x <- xy[, 1, drop = FALSE]
      y <- xy[, 2, drop = FALSE]
    } else {
      x <- data[, 1, drop = FALSE]
      y <- data[, 2, drop = FALSE]
    }
    if (is.null(xknots))
```

```
    xknots <- knotsD(x)
  if (is.null(yknots))
    yknots <- knotsD(y)
  g <- makeGifi (
    data = cbind (x, y),
    knots = c(xknots, yknots),
    degrees = c(xdegree, ydegree),
    ordinal = c(xordinal, yordinal),
    ties = c(xties, yties),
    copies = c(ndim, ndim),
    missing = c(xmissing, ymissing),
    active = c(TRUE, TRUE),
    names = c(xname, yname),
    sets = c(1, 2)
  )
  h <- gifiEngine(
    gifi = g,
    ndim = ndim,
    itmax = itmax,
    eps = eps,
    seed = seed,
    verbose = verbose
  )
  xg <- h$xGifi[[1]][[1]]
  yg <- h$xGifi[[2]][[1]]
  return (structure (
    list(
      burt = crossprod (cbind(g[[1]][[1]]$basis, g[[2]][[1]]$basis)),
      objectscores = h$x,
      xtransform = postCorals (x, xg$transform),
      ytransform = postCorals (y, yg$transform),
      rhat = cor (cbind (xg$transform, yg$transform)),
      xweights = xg$weights,
      yweights = yg$weights,
      xscores = postCorals (x, xg$scores),
      yscores = postCorals (y, yg$scores),
      xdmeasure = crossprod (xg$scores),
```

```
        ydmeasure = crossprod (yg$scores),
        xquantifications = xg$quantifications,
        yquantifications = yg$quantifications,
        xloadings = crossprod (xg$transform, h$x),
        yloadings = crossprod (yg$transform, h$x),
        lambda = (crossprod (xg$scores) + crossprod (yg$scores)) / 2,
        ntel = h$ntel,
        f = h$f
      ),
      class = "corals"
    ))
  }

coranals <- function () {

}

morals <-
  function (x,
            y,
            xknots = knotsQ(x),
            yknots = knotsQ(y),
            xdegrees = 2,
            ydegrees = 2,
            xordinal = TRUE,
            yordinal = TRUE,
            xties = "s",
            yties = "s",
            xmissing = "m",
            ymissing = "m",
            xnames = colnames (x, do.NULL = FALSE),
            ynames = "Y",
            xactive = TRUE,
            xcopies = 1,
            itmax = 1000,
            eps = 1e-6,
            seed = 123,
```

```
          verbose = FALSE) {
  npred <- ncol (x)
  nobs <- nrow (x)
  xdegrees <- reshape (xdegrees, npred)
  xordinal <- reshape (xordinal, npred)
  xties <- reshape (xties, npred)
  xmissing <- reshape (xmissing, npred)
  xactive <- reshape (xactive, npred)
  xcopies <- reshape (xcopies, npred)
  g <- makeGifi (
    data = cbind (x, y),
    knots = c (xknots, yknots),
    degrees = c (xdegrees, ydegrees),
    ordinal = c (xordinal, yordinal),
    sets =  c (rep(1, npred), 2),
    copies = c (xcopies, 1),
    ties = c (xties, yties),
    missing = c (xmissing, ymissing),
    active = c (xactive, TRUE),
    names = c (xnames, ynames)
  )
  h <- gifiEngine(
    gifi = g,
    ndim = 1,
    itmax = itmax,
    eps = eps,
    seed = seed,
    verbose = verbose
  )
  xhat <- matrix (0, nobs, 0)
  for (j in 1:npred)
    xhat <- cbind (xhat, h$xGifi[[1]][[j]]$transform)
  yhat <- h$xGifi[[2]][[1]]$transform
  rhat <- cor (cbind (xhat, yhat))
  qxy <- lsRC(xhat, yhat)$solution
  ypred <- xhat %*% qxy
  yres <- yhat - ypred
```

```
    smc <- sum (yhat * ypred)
    return (structure (
      list (
        objscores = h$x,
        xhat = xhat,
        yhat = yhat,
        rhat = rhat,
        beta = qxy,
        ypred = ypred,
        yres = yres,
        smc = smc,
        ntel = h$ntel,
        f = h$f
      ),
      class = "morals"
    ))
  }

princals <-
  function (data,
            knots = knotsQ (data),
            degrees = 2,
            ordinal = TRUE,
            copies = 1,
            ndim = 2,
            ties = "s",
            missing = "m",
            names = colnames (data, do.NULL = FALSE),
            active = TRUE,
            itmax = 1000,
            eps = 1e-6,
            seed = 123,
            verbose = FALSE)  {
    aname <- deparse (substitute (data))
    nvars <- ncol (data)
    nobs <- nrow (data)
    g <- makeGifi (
```

```
    data = data,
    knots = knots,
    degrees = reshape (degrees, nvars),
    ordinal = reshape (ordinal, nvars),
    sets =  1:nvars,
    copies = reshape (copies, nvars),
    ties = reshape (ties, nvars),
    missing = reshape (missing, nvars),
    active = reshape (active, nvars),
    names = names
  )
  h <- gifiEngine(
    gifi = g,
    ndim = ndim,
    itmax = itmax,
    eps = eps,
    seed = seed,
    verbose = verbose
  )
  a <- v <- z <- d <- y <- o <- as.list (1:nvars)
  dsum <- matrix (0, ndim, ndim)
  for (j in 1:nvars) {
    jgifi <- h$xGifi[[j]][[1]]
    v[[j]] <- jgifi$transform
    a[[j]] <- jgifi$weights
    y[[j]] <- jgifi$scores
    z[[j]] <- jgifi$quantifications
    cy <- crossprod (y[[j]])
    dsum <- dsum + cy
    d[[j]] <- cy
    o[[j]] <- crossprod (h$x, v[[j]])
  }
  return (structure (
    list (
      transform = v,
      rhat = corList (v),
      objectscores = h$x,
```

```
        scores = y,
        quantifications = z,
        dmeasures = d,
        lambda = dsum / ncol (data),
        weights = a,
        loadings = o,
        ntel = h$ntel,
        f = h$f
      ),
      class = "princals"
    ))
  }

criminals <-
  function (x,
            y,
            xknots = knotsQ(x),
            yknots = knotsD(y),
            xdegrees = 2,
            ydegrees = -1,
            xordinal = TRUE,
            yordinal = FALSE,
            xcopies = 1,
            xties = "s",
            yties = "s",
            xmissing = "m",
            ymissing = "m",
            xactive = TRUE,
            xnames = colnames (x, do.NULL = FALSE),
            ynames = "Y",
            ndim = 2,
            itmax = 1000,
            eps = 1e-6,
            seed = 123,
            verbose = FALSE) {
    aname <- deparse (substitute (data))
    npred <- ncol (x)
```

```
nobs <- nrow (x)
g <- makeGifi (
  data = cbind (x, y),
  knots = c(xknots, yknots),
  degrees = c (reshape (xdegrees, npred), ydegrees),
  ordinal = c (reshape (xordinal, npred), yordinal),
  sets =  c (rep(1, npred), 2),
  copies = c (reshape (xcopies, npred), length (unique (y))),
  ties = c (reshape (xties, npred), yties),
  missing = c (reshape (xmissing, npred), ymissing),
  active = c (reshape (xactive, npred), TRUE),
  names = c(xnames, ynames)
)
h <- gifiEngine(
  gifi = g,
  ndim = ndim,
  itmax = itmax,
  eps = eps,
  seed = seed,
  verbose = verbose
)
x <- matrix (0, nobs, 0)
for (j in 1:npred)
  x <- cbind (x, h$xGifi[[1]][[j]]$transform)
y <- as.vector(y)
g <- ifelse (outer (y, unique (y), "=="), 1, 0)
d <- colSums (g)
v <- crossprod (x)
u <- crossprod (g, x)
b <- crossprod (u, (1 / d) * u)
w <- v - b
e <- eigen (v)
k <- e$vectors
l <- sqrt (abs (e$values))
l <- ifelse (l < 1e-7, 0, 1 / l)
f <- eigen (outer(l, l) * crossprod (k, b %*% k))
a <- k %*% (l * f$vectors)
```

```
    z <- x %*% a
    u <- (1 / d) * crossprod (g, x)
    return (structure (
      list(
        objectscores = h$x,
        xhat = x,
        loadings = a,
        scores = z,
        groupmeans = u,
        bwratios = f$values,
        ntel = h$ntel,
        f = h$f
      ),
      class = "criminals"
    ))
  }

canals <-
  function (x,
            y,
            xknots = knotsQ(x),
            yknots = knotsQ(y),
            xdegrees = rep(2, ncol(x)),
            ydegrees = rep(2, ncol(y)),
            xordinal = rep (TRUE, ncol (x)),
            yordinal = rep (TRUE, ncol (y)),
            xcopies = rep (1, ncol (x)),
            ycopies = rep (1, ncol (y)),
            ndim = 2,
            itmax = 1000,
            eps = 1e-6,
            seed = 123,
            verbose = FALSE) {
    h <- gifiEngine(
      data = cbind (x, y),
      knots = c(xknots, yknots),
      degrees = c(xdegrees, ydegrees),
```

```
    ordinal = c(xordinal, yordinal),
    sets = c(rep(1, ncol(x)), rep(2, ncol (y))),
    copies = c(xcopies, ycopies),
    ndim = ndim,
    itmax = itmax,
    eps = eps,
    seed = seed,
    verbose = verbose
  )
  x <- h$h[, 1:sum(xcopies)]
  y <- h$h[, -(1:sum(xcopies))]
  u <- crossprod (x)
  v <- crossprod (y)
  w <- crossprod (x, y)
  a <- solve (chol (u))
  b <- solve (chol (v))
  s <- crossprod (a, w %*% b)
  r <- svd (s)
  xw <- a %*% (r$u)
  yw <- b %*% (r$v)
  xs <- x %*% xw
  ys <- y %*% yw
  xl <- crossprod (x, xs)
  yl <- crossprod (y, ys)
  return (structure (
    list(
      xhat = x,
      yhat = y,
      rhat = cor (cbind (x, y)),
      cancors = r$d,
      xweights = xw,
      yweights = yw,
      xscores = xs,
      yscores = ys,
      xloadings = xl,
      yloadings = yl,
      ntel = h$ntel,
```

```
        f = h$f
      ),
      class = "canals"
    ))
  }


overals <-
  function (data,
            sets,
            copies,
            knots = knotsQ (data),
            degrees = rep (2, ncol (data)),
            ordinal = rep (TRUE, ncol (data)),
            ndim = 2,
            itmax = 1000,
            eps = 1e-6,
            seed = 123,
            verbose = FALSE)  {
    h <- gifiEngine(
      data = data,
      knots = knots,
      degrees = degrees,
      ordinal = ordinal,
      sets = sets,
      copies = copies,
      ndim = ndim,
      itmax = itmax,
      eps = eps,
      seed = seed,
      verbose = verbose
    )
    xhat <- h$h
    rhat <- cor (xhat)
    a <- h$a
    y <- xhat
    for (j in 1:ncol(data)) {
```

```
      k <- (1:ndim) + (j - 1) * ndim
      y[, k] <- xhat[, k] %*% a[k,]
    }
    return (structure (
      list (
        xhat = xhat,
        rhat = rhat,
        objscores = h$x,
        quantifications = y,
        ntel = h$ntel,
        f = h$f
      ),
      class = "overals"
    ))
  }


primals <- function () {

}

addals <- function () {

}

pathals <- function () {

}
```

13.1.7 Splines

```
bsplineBasis <- function (x, degree, innerknots, lowknot = min(x,innerknots),
    innerknots <- unique (sort (innerknots))
    knots <- c(rep(lowknot, degree + 1), innerknots, rep(highknot, degree + 1)
    n <- length (x)
```

```
    m <- length (innerknots) + 2 * (degree + 1)
    nf <- length (innerknots) + degree + 1
    basis <- rep (0,  n * nf)
    res <- .C("splinebasis", d = as.integer(degree),
        n = as.integer(n), m = as.integer (m), x = as.double (x), knots = as.double (
    basis <- matrix (res$basis, n, nf)
    basis <- basis[,which(colSums(basis) > 0)]
    return (basis)
}

knotsQ <- function (x, n = rep (5, ncol (x))) {
  do <- function (i) {
    y <- quantile (x[, i], probs = seq(0, 1, length = max (2, n[i])))
    return (y[-c(1, length(y))])
  }
  lapply (1:ncol(x), function (i)
    do (i))
}

knotsR <- function (x, n = rep (5, ncol (x))) {
  do <- function (i) {
    y <- seq (min(x[, i]), max(x[, i]), length = max (2, n[i]))
    return (y[-c(1, length(y))])
  }
  lapply (1:ncol(x),  function (i)
    do (i))
}

knotsE <- function (x) {
  lapply (1:ncol(x), function (i)
    numeric(0))
}

knotsD <- function (x) {
  do <- function (i) {
    y <- sort (unique (x[, i]))
    return (y[-c(1, length(y))])
```

```
  }
  lapply (1:ncol(x),  function (i)
    do (i))
}
```

13.1.8 Gram-Schmidt

```
gsRC <- function (x, eps = 1e-10) {
  n <- nrow (x)
  m <- ncol (x)
  h <-
    .C(
      "gsC",
      x = as.double(x),
      r = as.double (matrix (0, m, m)),
      n = as.integer (n),
      m = as.integer (m),
      rank = as.integer (0),
      pivot = as.integer (1:m),
      eps = as.double (eps)
    )
  rank = h$rank
  return (list (
    q = matrix (h$x, n, m)[, 1:rank, drop = FALSE],
    r = matrix (h$r, m, m)[1:rank, , drop = FALSE],
    rank = rank,
    pivot = h$pivot
  ))
}

lsRC <- function (x, y, eps = 1e-10) {
  n <- nrow (x)
  m <- ncol (x)
  h <- gsRC (x, eps)
  l <- h$rank
```

```
  p <- order (h$pivot)
  k <- 1:l
  q <- h$q
  a <- h$r[, k, drop = FALSE]
  v <- h$r[, -k, drop = FALSE]
  u <- crossprod (q, y)
  b <- solve (a, u)
  res <- drop (y - q %*% u)
  s <- sum (res ^ 2)
  b <- rbind(b, matrix (0, m - l, ncol(y)))[p, , drop = FALSE]
  if (l == m) {
    e <- matrix(0, m, 1)
  } else {
    e <- rbind (-solve(a, v), diag(m - l))[p, , drop = FALSE]
  }
  return (list (
    solution = b,
    residuals = res,
    minssq = s,
    nullspace = e,
    rank = l,
    pivot = p
  ))
}

nullRC <- function (x, eps = 1e-10) {
  h <- gsRC (x, eps = eps)
  rank <- h$rank
  r <- h$r
  m <- ncol (x)
  t <- r[, 1:rank, drop = FALSE]
  s <- r[, -(1:rank), drop = FALSE]
  if (rank == m)
    return (matrix(0, m, 1))
  else {
    nullspace <- rbind (-solve(t, s), diag (m - rank))[order(h$pivot), , drop = FALSE
    return (gsRC (nullspace)$q)
```

```
  }
}

ginvRC <- function (x, eps = 1e-10) {
  h <- gsRC (x, eps)
  p <- order(h$pivot)
  q <- h$q
  s <- h$r
  z <- crossprod (s, (solve (tcrossprod(s), t(q))))
  return (z[p, , drop = FALSE])
}
```

13.1.9 Cone regression

```
dyn.load ("lib/pava.so")

amalgm <- function (x, w = rep (1, length (x))) {
  n <- length (x)
  a <- rep (0, n)
  b <- rep (0, n)
  y <- rep (0, n)
  lf <-
    .Fortran (
      "AMALGM",
      n = as.integer (n),
      x = as.double (x),
      w = as.double (w),
      a = as.double (a),
      b = as.double (b),
      y = as.double (y),
      tol = as.double(1e-15),
      ifault = as.integer(0)
    )
  return (lf$y)
}
```

```
isotone <-
  function (x,
            y,
            w = rep (1, length (x)),
            ties = "s") {
    there <- which (!is.na (x))
    notthere <- which (is.na (x))
    xthere <- x[there]
    f <- sort(unique(xthere))
    g <- lapply(f, function (z)
      which(x == z))
    n <- length (x)
    k <- length (f)
    if (ties == "s") {
      w <- sapply (g, length)
      h <- lapply (g, function (z)
        y[z])
      m <- sapply (h, sum) / w
      r <- amalgm (m, w)
      s <- rep (0, n)
      for (i in 1:k)
        s[g[[i]]] <- r[i]
    s[notthere] <- y[notthere]
    }
    if (ties == "p") {
      h <- lapply (g, function (z)
        y[z])
      m <- rep (0, n)
      s <- rep (0, n)
      for (i in 1:k) {
        ii <- order (h[[i]])
        g[[i]] <- g[[i]][ii]
        h[[i]] <- h[[i]][ii]
      }
      m <- unlist (h)
      r <- amalgm (m, w)
      s[there] <- r[order (unlist (g))]
```

```
      s[notthere] <- y[notthere]
    }
    if (ties == "t") {
      w <- sapply (g, length)
      h <- lapply (g, function (x)
        y[x])
      m <- sapply (h, sum) / w
      r <- amalgm (m, w)
      s <- rep (0, n)
      for (i in 1:k)
        s[g[[i]]] <- y[g[[i]]] + (r[i] - m[i])
      s[notthere] <- y[notthere]
    }
    return (s)
  }

coneRegression <-
  function (data,
            target,
            basis = matrix (data, length(data), 1),
            type = "i",
            ties = "s",
            missing = "m",
            itmax = 1000,
            eps = 1e-6) {
    itel <- 1
    there <- which (!is.na (data))
    notthere <- which (is.na (data))
    nmis <- length (notthere)
    solution <- rep(0, length (data))
    wdata <- data[there]
    wtarget <- target[there]
    wbasis <- basis [there, ]
    if (type == "s")  {
      solution  <- drop (basis %*% lsRC (basis, target)$solution)
    }
     if ((type == "c") && (missing != "a")) {
```

```
      solution[there] <- isotone (x = wdata, y = wtarget, ties = ties)
      if (nmis > 0) {
        if (missing == "m")
          solution[notthere] <- target[notthere]
        if (missing == "s")
          solution[notthere] <- mean (target[notthere])
      }
     }
    if ((type == "i")  || ((type == "c") && (missing == "a"))) {
      solution <-
        dykstra (
          target = target,
          basis = basis,
          data = data,
          ties = ties,
          itmax = itmax,
          eps = eps
        )
    }
    return (solution)
  }

dykstra <- function (target, basis, data, ties, itmax, eps) {
  x0 <- target
  itel <- 1
  a <- b <- rep (0, length (target))
  fold <- Inf
  repeat {
    x1 <- drop (basis %*% lsRC (basis, x0 - a)$solution)
    a <- a + x1 - x0
    x2 <- isotone (data, x1 - b, ties = ties)
    b <- b + x2 - x1
    fnew <- sum ((target - (x1 + x2) / 2) ^ 2)
    xdif <- max (abs (x1 - x2))
    if ((itel == itmax) || (xdif < eps))
      break
    itel <- itel + 1
```

```
    x0 <- x2
    fold <- fnew
  }
  return ((x1 + x2) / 2)
}
```

13.1.10 Coding

```
dyn.load("lib/coding.so")

decode <- function(cell, dims) {
  if (length(cell) != length(dims)) {
    stop("Dimension error")
  }
  if (any(cell > dims) || any (cell < 1)) {
    stop("No such cell")
  }
  .Call("DECODE", as.integer(cell), as.integer(dims))
}


encode <- function(ind, dims) {
  if (length(ind) > 1) {
    stop ("Dimension error")
  }
  if ((ind < 1) || (ind > prod(dims))) {
    stop ("No such cell")
  }
  .Call("ENCODE", as.integer(ind), as.integer(dims))
}
```

13.1.11 Utilities

```
makeNumeric <- function (x) {
  do <- function (y) {
    u <- unique (y)
    return (drop (ifelse (outer (y, u, "=="), 1, 0) %*% (1:length (u))))
  }
  if (is.vector (x)) {
    return (do (x))
  }
  else {
    return (apply (x, 2, do))
  }
}

center <- function (x) {
  do <- function (z) {
    z - mean (z)
  }
  if (is.matrix (x))
    return (apply (x, 2, do))
  else
    return (do (x))
}

normalize <- function (x) {
  do <- function (z) {
    z / sqrt (sum (z ^ 2))
  }
  if (is.matrix (x))
    return (apply (x, 2, do))
  else
    return (do (x))
}

makeMissing <- function (data, basis, missing) {
  there <- which (!is.na (data))
```

```
  notthere <- which (is.na (data))
  nmis <- length (notthere)
  nobs <- length (data)
  ndim <- ncol (basis)
  if (missing == "m") {
    abasis <- matrix (0, nobs, ndim + nmis)
    abasis [there, 1:ndim] <- basis
    abasis [notthere, ndim + 1:nmis] <- diag(nmis)
    basis <- abasis
  }
  if (missing == "a") {
    abasis <- matrix (0, nobs, ndim)
    abasis [there,] <- basis
    abasis [notthere,] <- 1 / ndim
    basis <- abasis
  }
  if (missing == "s") {
    abasis <- matrix (0, nobs, ndim + 1)
    abasis [there, 1:ndim] <- basis
    abasis [notthere, ndim + 1] <- 1
    basis <- abasis
  }
  return (basis)
}

makeIndicator <- function (x) {
  return (as.matrix(ifelse(outer(
    x, sort(unique(x)), "=="
  ), 1, 0)))
}

reshape <- function (x, n) {
  if (length (x) == 1)
    return (rep (x, n))
  else
    return (x)
}
```

```
aline <- function (a) {
  abline (0, a[2] / a[1])
}

aperp <- function (a, x) {
  abline (x * (sum (a ^ 2) / a[2]),-a[1] / a[2])
}

aproj <- function (a, h, x) {
  mu <- (h - sum (a * x)) / (sum (a ^ 2))
  return (x + mu * a)
}

stepPlotter <- function (x, y, knots, xlab) {
  y <- as.matrix (y)
  plot (x,
        y[, 1],
        type = "n",
        xlab = xlab,
        ylab = "Transform")
  nknots <- length (knots)
  knots <- c(min(x) - 1, knots, max(x) + 1)
  for (i in 1:(nknots + 1)) {
    ind <- which ((x >= knots [i]) & (x < knots[i + 1]))
    lev <- median (y [ind, 1])
    lines (rbind (c(knots[i], lev), c (knots[i + 1], lev)), col = "RED", lwd = 3)
    if (ncol (y) == 2) {
      lev <- median (y [ind, 2])
      lines (rbind (c(knots[i], lev), c (knots[i + 1], lev)), col = "BLUE", lwd = 3)
    }
  }
}

starPlotter <- function (x, y, main = "") {
  plot(
    x,
    xlab = "dimension 1",
```

```
    ylab = "dimension 2",
    col = "RED",
    cex = .5,
    main = main
  )
  points(y, col = "BLUE", cex = .5)
  for (i in 1:nrow(x))
    lines (rbind (x[i, ], y[i, ]))
}

regressionPlotter <-
  function (table,
            x,
            y,
            xname = "Columns",
            yname = "Rows",
            main = "",
            lines = TRUE,
            cex = 1.0,
            ticks = "n") {
    if (ticks != "n") {
      ticks <- NULL
    }
    nr <- nrow (table)
    nc <- ncol (table)
    sr <- rowSums (table)
    sc <- colSums (table)
    rc <- sum (table)
    x <- x - sum (sr * x) / rc
    y <- y - sum (sc * y) / rc
    x <- x / sqrt (sum (sr * (x ^ 2)) / rc)
    y <- y / sqrt (sum (sc * (y ^ 2)) / rc)
    ar <- drop ((table %*% y) / sr)
    ac <- drop ((x %*% table) / sc)
    plot (
      0,
      xlim = c (min(y), max(y)),
```

```
      ylim = c (min(x), max(x)),
      xlab = xname,
      ylab = yname,
      main = main,
      xaxt = ticks,
      yaxt = ticks,
      type = "n"
    )
    if (lines) {
      for (i in 1:nr)
        abline (h = x[i])
      for (j in 1:nc)
        abline (v = y[j])
    }
    for (i in 1:nr) {
      for (j in 1:nc) {
        text(y[j],
             x[nr - i + 1],
             as.character(table[i, j]),
             cex = cex,
             col = "RED")
      }
    }
    lines (y, ac, col = "BLUE")
    lines (ar, x, col = "BLUE")
  }

corList <- function (x) {
  m <- length (x)
  n <- nrow (x[[1]])
  h <- matrix (0, n, 0)
  for (i in 1:m) {
    h <- cbind (h, x[[i]])
  }
  return (cor (h))
}
```

```
preCorals <- function (x) {
  n <- sum (x)
  r <- nrow (x)
  s <- ncol (x)
  v <- numeric (0)
  for (i in 1:r)
    for (j in 1:s)
      v <- c(v, rep(c(i, j), x[i, j]))
  return (matrix (v, n, 2, byrow = TRUE))
}

postCorals <- function (ff, x) {
  y <- matrix(0, max(ff), ncol (x))
  for (i in 1:nrow (x))
    y[ff[i],] <- x[i,]
  return (y)
}

preCoranals <- function (x, y) {
  n <- sum (x)
  m <- ncol (y)
  r <- nrow (x)
  s <- ncol (x)
  v <- numeric (0)
  for (i in 1:r)
    for (j in 1:s)
      v <- c(v, rep(c(y[i,], j), x[i, j]))
  return (matrix (v, n, m + 1, byrow = TRUE))
}

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

burtTable <- function (gifi) {
```

```
  nsets <- length (gifi)
  nobs <- length(gifi[[1]][[1]]$data)
  hh <- matrix (0, nobs, 0)
  hl <- list ()
  for (i in 1:nsets) {
    gifiSet <- gifi[[i]]
    nvars <- length (gifiSet)
    hi <- matrix(0, nobs, 0)
    for (j in 1:nvars) {
      gifiVariable <- gifiSet[[j]]
      hi <- cbind (hi, gifiVariable$basis)
    }
    hl <- c (hl, list (crossprod (hi)))
    hh <- cbind (hh, hi)
  }
  return (list (cc = crossprod (hh), dd = directSum (hl)))
}

interactiveCoding <- function (data) {
  cmin <- apply (data, 2, min)
  cmax <- apply (data, 2, max)
  if (!all(cmin == 1))
    stop ("data must be start at 1")
  nobs <- nrow(data)
  h <- numeric(0)
  for (i in 1:nobs)
    h <- c(h, decode (data[i, ], cmax))
  return (h)
}

makeColumnProduct <- function (x) {
  makeTwoColumnProduct <- function (a, b) {
    n <- nrow (a)
    ma <- ncol (a)
    mb <- ncol (b)
    ab <- matrix (0, n, ma * mb)
    k <- 1
```

```
    for (i in 1:ma) {
      for (j in 1:mb) {
        ab[, k] <- a[, i] * b[, j]
        k <- k + 1
      }
    }
    return (ab)
  }
  if (!is.list(x)) {
    x <- list (x)
  }
  m <- length (x)
  z <- matrix (1, nrow(x[[1]]), 1)
  for (k in 1:m)
    z <- makeTwoColumnProduct (z, x[[k]])
  return (z)
}


profileFrequencies <- function (data) {
  h <- interactiveCoding (data)
  cmax <- apply (data, 2, max)
  u <- unique (h)
  m <- length (u)
  g <- ifelse (outer (h, u, "=="), 1, 0)
  n <- colSums (g)
  h <- matrix (0, m, ncol (data))
  for (j in 1:m)
    h[j, ] <- encode (u[j], cmax)
  return (list (h = h, n = n))
}


directSum <- function (x) {
  m <- length (x)
  nr <- sum (sapply (x, nrow))
  nc <- sum (sapply (x, ncol))
```

```
  z <- matrix (0, nr, nc)
  kr <- 0
  kc <- 0
  for (i in 1:m) {
    ir <- nrow (x[[i]])
    ic <- ncol (x[[i]])
    z[kr + (1:ir), kc + (1:ic)] <- x[[i]]
    kr <- kr + ir
    kc <- kc + ic
  }
  return (z)
}
```

13.2 C Code

13.2.1 Splines

```
double bs (int nknots, int nspline, int degree, double x, double * knots);
int mindex (int i, int j, int nrow);

void splinebasis (int *d, int *n, int *m, double * x, double * knots, double * basis)
    int mm = *m, dd = *d, nn = *n;
    int k = mm - dd - 1, i , j, ir, jr;
    for (i = 0; i < nn; i++) {
        ir = i + 1;
        if (x[i] == knots[mm - 1]) {
           basis [mindex (ir, k, nn) - 1] =  1.0;
            for (j = 0; j <  (k - 1);  j++) {
                jr = j + 1;
                basis [mindex (ir, jr, nn) - 1] = 0.0;
            }
        } else {
            for (j = 0; j < k ; j++) {
                jr = j + 1;
```

```
                basis [mindex (ir, jr, nn) - 1] = bs (mm, jr, dd + 1, x[i], kn
            }
        }
    }
}

int mindex (int i, int j, int nrow) {
    return (j - 1) * nrow + i;
}

double bs (int nknots, int nspline, int updegree, double x, double * knots) {
    double y, y1, y2, temp1, temp2;
     if (updegree == 1) {
        if ((x >= knots[nspline - 1]) && (x < knots[nspline]))
            y = 1.0;
        else
            y = 0.0;
     }
     else {
        temp1 = 0.0;
        if ((knots[nspline + updegree - 2] - knots[nspline - 1]) > 0)
            temp1 = (x - knots[nspline - 1]) / (knots[nspline + updegree - 2]
        temp2 = 0.0;
        if ((knots[nspline + updegree - 1] - knots[nspline]) > 0)
            temp2 = (knots[nspline + updegree - 1] - x) / (knots[nspline + upd
        y1 = bs(nknots, nspline, updegree - 1, x, knots);
        y2 = bs(nknots, nspline + 1, updegree - 1, x, knots);
        y =  temp1 * y1 + temp2 * y2;
    }
    return y;
}
```

13.2.2 Gram-Schmidt

```c
#include <math.h>

#define MINDEX(i, j, n) (((j)-1) * (n) + (i)-1)

void gsC(double *, double *, int *, int *, int *, int *, double *);

void gsC(double *x, double *r, int *n, int *m, int *rank, int *pivot,
         double *eps) {
  int i, j, ip, nn = *n, mm = *m, rk = *m, jwork = 1;
  double s = 0.0, p;
  for (j = 1; j <= mm; j++) {
    pivot[j - 1] = j;
  }
  while (jwork <= rk) {
    for (j = 1; j < jwork; j++) {
      s = 0.0;
      for (i = 1; i <= nn; i++) {
        s += x[MINDEX(i, jwork, nn)] * x[MINDEX(i, j, nn)];
      }
      r[MINDEX(j, jwork, mm)] = s;
      for (i = 1; i <= nn; i++) {
        x[MINDEX(i, jwork, nn)] -= s * x[MINDEX(i, j, nn)];
      }
    }
    s = 0.0;
    for (i = 1; i <= nn; i++) {
      s += x[MINDEX(i, jwork, nn)] * x[MINDEX(i, jwork, nn)];
    }
    if (s > *eps) {
      s = sqrt(s);
      r[MINDEX(jwork, jwork, mm)] = s;
      for (i = 1; i <= nn; i++) {
        x[MINDEX(i, jwork, nn)] /= s;
      }
      jwork += 1;
    }
    if (s <= *eps) {
```

```
      ip = pivot [rk - 1];
      pivot[rk - 1] = pivot[jwork - 1];
      pivot[jwork - 1] = ip;
      for (i = 1; i <= nn; i++) {
        p = x[MINDEX(i, rk, nn)];
        x[MINDEX(i, rk, nn)] = x[MINDEX(i, jwork, nn)];
        x[MINDEX(i, jwork, nn)] = p;
      }
      for (j = 1; j <= mm; j++) {
        p = r[MINDEX(j, rk, mm)];
        r[MINDEX(j, rk, mm)] = r[MINDEX(j, jwork, mm)];
        r[MINDEX(j, jwork, mm)] = p;
      }
      rk -= 1;
    }
  }
  *rank = rk;
}
```

13.2.3 Coding

```
#include <R.h>
#include <Rinternals.h>

SEXP DECODE( SEXP, SEXP );
SEXP ENCODE( SEXP, SEXP );

SEXP
  DECODE( SEXP cell, SEXP dims )
  {
    int             aux = 1, n = length(dims);
    SEXP            ind;
    PROTECT( ind = allocVector( INTSXP, 1 ) );
    INTEGER( ind )[0] = 1;
    for( int i = 0; i < n; i++ ) {
```

```
      INTEGER( ind )[0] += aux * ( INTEGER( cell )[i] - 1 );
      aux *= INTEGER(dims)[i];
    }
    UNPROTECT( 1 );
    return (ind);
  }

SEXP
  ENCODE( SEXP ind, SEXP dims )
  {
    int             n = length(dims), aux = INTEGER(ind)[0], pdim = 1;
    SEXP            cell;
    PROTECT( cell = allocVector( INTSXP, n ) );
    for ( int i = 0; i < n - 1; i++ )
      pdim *= INTEGER( dims )[i];
    for ( int i = n - 1; i > 0; i-- ){
      INTEGER( cell )[i] = ( aux - 1 ) / pdim;
      aux -= pdim * INTEGER( cell )[i];
      pdim /= INTEGER( dims )[i - 1];
      INTEGER( cell )[i] += 1;
    }
    INTEGER( cell )[0] = aux;
    UNPROTECT( 1 );
    return cell;
  }
```

Chapter 14

Backmatter

14.1 Key Names and Symbols

14.2 References

Association, American Statistical. 2014. "Discovery with Data: Leveraging Statistics with Computer Science to Transform Science and Society." 2014. http://www.amstat.org/policy/pdfs/BigDataStatisticsJune2014.pdf.

Bijleveld, C. C. J. H., and J. De Leeuw. 1991. "Fitting Longitudinal Reduced-Rank Regression Models by Alternating Least Squares." *Psychometrika* 56 (3): 433–47.

Bock, R. D. 1960. "Methods and Applications of Optimal Scaling." Psychometric Laboratory Report 25. Chapell Hill, N.C.: L.L. Thurstone Psychometric Laboratory, University of North Carolina.

Breiman, L. 2001. "Statistical Modeling: The Two Cultures." *Statistical Science* 16 (3): 199–231.

Breiman, L., and J. H. Friedman. 1985. "Estimating Optimal Transformations for Multiple Regression and Correlation." *Journal of the American Statistical Association* 80: 580–619.

Burg, E. Van der, J. De Leeuw, and R. Verdegaal. 1988. "Homogeneity Analysis with K Sets of Variables: An Alternating Least Squares Approach with Optimal Scaling Features." *Psychometrika* 53: 177–97.

Carroll, J. D. 1968. "A Generalization of Canonical Correlation Analysis to

Three or More Sets of Variables." In *Proceedings of the 76th Annual Convention of the American Psychological Association*, 227–28. Washington, D.C.: American Psychological Association.

Cleveland, W. S. 2014. "Data Science: An Action Plan for Expanding the Technical Areas of the Field of Statistics." *Statistical Analysis and Data Mining* 7: 414–17.

Coolen, H., and J. De Leeuw. 1987. "Least Squares Path Analysis with Optimal Scaling." Research Report RR-87-03. Leiden, The Netherlands: Department of Data Theory FSW/RUL.

Coombs, C. H. 1964. *A Theory of Data*. Wiley.

Cordier, B. 1965. "L'Analyse Factorielle des Correspondances." {Th\'ese de Troisieme Cycle}, Université de Rennes; Faculté des Sciences.

Dauxois, J., and A. Pousse. 1976. "Les Analyses Factorielles en Calcul des Probabilités et en Statistique: Essai d'Étude Synthetique." PhD thesis, Université Paul-Sabatier, Toulouse, France.

Davies, P. L. 1995. "Data Features." *Statistica Neerlandica* 49 (2): 185–245.

———. 2008. "Approximating Data." *Journal of the Korean Statistical Society* 37: 191–211.

De Leeuw, J. 1968a. "Canonical Discriminant Analysis of Relational Data." Research Note 007-68. Department of Data Theory FSW/RUL.

———. 1968b. "Nonmetric Discriminant Analysis." Research Note 06-68. Department of Data Theory, University of Leiden.

———. 1974. *Canonical Analysis of Categorical Data*. Leiden, The Netherlands: Psychological Institute, Leiden University.

———. 1975. "A Normalized Cone Regression Approach to Alternating Least Squares Algorithms." Department of Data Theory FSW/RUL.

———. 1983. "On the Prehistory of Correspondence Analysis." *Statistica Neerlandica* 37: 161–64.

———. 1984a. "Models of Data." *Kwantitatieve Methoden* 5: 17–30.

———. 1984b. "The Gifi System of Nonlinear Multivariate Analysis." In *Data Analysis and Informatics*, edited by E. Diday et al. Vol. III. Amsterdam: North Holland Publishing Company.

———. 1988a. "Models and Techniques." *Statistica Neerlandica* 42: 91–98.

———. 1988b. "Multivariate Analysis with Linearizable Regressions." *Psychometrika* 53: 437–54.

———. 1988c. "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.

———. 1990. "Data Modeling and Theory Construction." In *Operationalization and Research Strategy*, edited by J. J. Hox and J. De Jong-Gierveld. Amsterdam, The Netherlands: Swets; Zeitlinger.

———. 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.

———. 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.

———. 2009. "Regression, Discriminant Analysis, and Canonical Analysis with homals." Preprint Series 562. Los Angeles, CA: UCLA Department of Statistics.

———. 2014. "History of Nonlinear Principal Component Analysis." In *The Visualization and Verbalization of Data*, edited by J. Blasius and M. Greenacre. Chapman; Hall.

———. 2015a. "Aspects of Correlation Matrices."

———. 2015b. "Exceedingly Simple b-Spline Code."

———. 2015c. "Regression with Linear Inequality Restrictions on Predicted Values."

De Leeuw, J., and P. Mair. 2009a. "Homogeneity Analysis in r: The Package Homals." *Journal of Statistical Software* 31 (4): 1–21.

———. 2009b. "Simple and Canonical Correspondence Analysis Using the R Package Anacor." *Journal of Statistical Software* 31 (5): 1–18.

De Leeuw, J., G. Michailidis, and D. Y. Wang. 1999. "Correspondence Analysis Techniques." In *Multivariate Analysis, Design of Experiments, and Survey Sampling*, edited by S. Ghosh, 523–47. Marcel Dekker.

De Leeuw, J., and J. L. A. Van Rijckevorsel. 1980. "HOMALS and PRINCALS: Some Generalizations of Principal Components Analysis." In *Data Analysis and Informatics*. Amsterdam: North Holland Publishing Company.

———. 1988. "Beyond Homogeneity Analysis." In *Component and Correspondence Analysis*, edited by J. L. A. Van Rijckevorsel and J. De Leeuw, 55–80. Chichester, England: Wiley.

De Leeuw, J., J. L. A. Van Rijckevorsel, and H. Van der Wouden. 1981. "Nonlinear Principal Component Analysis Using b-Splines." *Methods of Operations Research* 43: 379–94.

Galton, F. 1889. *Natural Inheritance*. MacMillan; Co.

Gibson, W. A. 1962. "On the Least Squares Orthogonalization of an Oblique Transformation." *Psychometrika* 27: 193–95.

Gifi, A. 1980. *Niet-Lineaire Multivariate Analyse [Nonlinear Multivariate Analysis]*. Leiden, The Netherlands: Department of Data Theory FSW/RUL.

———. 1981. *Nonlinear Multivariate Analysis*. Leiden, The Netherlands: Department of Data Theory FSW/RUL.

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

Glass, D. V. 1954. *Social Mobility in Britain*. Free Press.

Gower, J. C. 1990. "Fisher's Optimal Scores and Multiple Correspondence Analysis." *Biometrics* 46: 947–61.

Gower, J. C., and D. J. Hand. 1996. *Biplots*. Monographs on Statistics and Applied Probability 54. Chapman; Hall.

Gower, J. C., N. J. Le Roux, and S. Gardner-Lubbe. 2015. "Biplots: Quantitative Data." *WIREs Computational Statistics* 7: 42–62. https://doi.org/10.1002/wics.1338.

———. 2016. "Biplots: Qualitative Data." *WIREs Computational Statistics* 8: 82–111. https://doi.org/10.1002/wics.1377.

Guttman, L. 1941. "The Quantification of a Class of Attributes: A Theory and Method of Scale Construction." In *The Prediction of Personal Adjustment*, edited by P. Horst, 321–48. New York: Social Science Research Council.

Hartigan, J. 1975. *Clustering Algorithms*. Wiley.

Heiser, W. J. 1995. "Convergent Computing by Iterative Majorization: Theory and Applications in Multidimensional Data Analysis." In *Recent Advantages in Descriptive Multivariate Analysis*, edited by W. J. Krzanowski, 157–89. Oxford: Clarendon Press.

Hill, M. O. 1974. "Correspondence Analysis: a Neglected Multivariate Method." *Applied Statistics* 23: 340–54.

———. 1990. "Review of A. Gifi, Multivariate Analysis." *Journal of Ecology* 78 (4): 1148–49.

Holland, P. W. 1979. "The Tyranny of Continuous Models in a World of Discrete Data." *IHS-Journal* 3: 29–42.

Horst, P. 1961a. "Generalized Canonical Correlations and their Applications to Experimental Data." *Journal of Clinical Psychology* 17: 331–47.

———. 1961b. "Relations Among m Sets of Measures." *Psychometrika* 26: 129–49.

IBM. 2015. *IBM SPSS Categories 23*. IBM Corporation.

Kettenring, J. R. 1969. "Canonical Analysis of Several Sets of Variables." Institute of Statistics Mimeo Series 615. University of North Carolina at Chapell Hill.

———. 1971. "Canonical Analysis of Several Sets of Variables." *Biometrika* 58: 433–51.

Kiers, H. A. L, R. Cléroux, and J. M. F Ten Berge. 1994. "Generalized Canonical Analysis Based on Optimizing Matrix Correlations and a Relation with IDIOSCAL." *Computational Statistics and Data Analysis* 18: 331–40.

Koyak, R. 1987. "On Measuring Internal Dependence in a Set of Random Variables." *Annals of Statistics* 15: 1215–28.

Kruskal, J. B. 1964a. "Multidimensional Scaling by Optimizing Goodness of Fit to a Nonmetric Hypothesis." *Psychometrika* 29: 1–27.

———. 1964b. "Nonmetric Multidimensional Scaling: a Numerical Method." *Psychometrika* 29: 115–29.

———. 1965. "Analysis of Factorial Experiments by Estimating Monotone Transformations of the Data." *Journal of the Royal Statistical Society* B27: 251–63.

Kruskal, J. B., and R. N. Shepard. 1974. "A Nonmetric Variety of Linear Factor Analysis." *Psychometrika* 39: 123–57.

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

Lingoes, J. C. 1973. *The Guttman-Lingoes Nonmetric Program Series*. Mathesis Press.

Mackenzie, D. 1978. "Statistical Theory and Social Interests: A Case-Study." *Social Studies of Science* 8: 35–83.

Mair, P., and J. De Leeuw. 2010. "A General Framework for Multivariate Analysis with Optimal Scaling: The r Package Aspect." *Journal of Statistical Software* 32 (9): 1–23. http://www.stat.ucla.edu/~deleeuw/janspubs/2010/articles/mair_deleeuw_A_10.pdf.

Marvell, A. 1653. "The Character of Holland." https://ebooks.adelaide.edu.au/m/marvell/andrew/poems/poem58.html.

Meulman, J. J. 1982. *Homogeneity Analysis of Incomplete Data*. Leiden, The Netherlands: DSWO Press.

Meulman, J. J., and W. J. Heiser. 2012. *IBM SPSS Categories 21*. IBM Corporation.

Michailidis, G., and J. De Leeuw. 1998. "The Gifi System for Descriptive Multivariate Analysis." *Statistical Science* 13: 307–36.

Nenadic, O., and M. Greenacre. 2007. "Correspondence Analysis in R, with two- and three-dimensional graphics: The ca package." *Journal of Statistical Software* 20 (3): 1–13.

Peschar, J. L. 1975. *School, Milieu, Beroep*. Groningen, The Netherlands: Tjeek Willink.

R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

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

Rijckevorsel, J. L. A. Van, and J. De Leeuw, eds. 1988. *Component and Correspondence Analysis*. Wiley.

Roskam, E. E. 1968. "Metric Analysis of Ordinal Data in Psychology." PhD thesis, University of Leiden.

Shmueli, G. 2010. "To Explain or to Predict ?" *Statistical Science* 25: 289–310.

SPSS. 1989. *SPSS Categories*. SPSS Inc.

Takane, Y. 1992. "Review of Albert Gifi, Nonlinear Multivariate Analysis." *Journal of the American Statistical Association* 87: 587–88.

Tanaka, Y. 1979. "Review of the Methods of Quantification." *Environmental Health Perspectives* 32: 113–23.

Tenenhaus, A., and M. Tenenhaus. 2011. "Regularized Generalized Canonical Correlation Analysis." *Psychometrika* 76: 257–84.

Tenenhaus, M., and F. W. Young. 1985. "An Analysis and Synthesis of Multiple Correspondence Analysis, Optimal Scaling, Dual Scaling, Homogeneity Analysis and Other Methods for Quantifying Categorical Multivariate Data." *Psychometrika* 50: 91–119.

Tukey, J. W. 1962. "The Future of Data Analysis." *Annals of Mathematical Statistics* 33: 1–79.

Van de Geer, J. P. 1971. *Introduction to Multivariate Analysis for the Social Sciences*. San Francisco, CA: Freeman.

Van der Heijden, P. G. M., and S. Van Buuren. 1916. "Looking Back at the Gifi System of Nonlinear Multivariate Analysis." *Journal of Statistical Software* 73 (4).

Van der Velden, M. 2012. "On Generalized Canonical Correlation Analysis."

In *Proceedings 58th World Statistical Congress, 2011, Dublin*, 758–65. The Hague: International Statiatical Instutute.

Van der Velden, M., and Y. Takane. 2012. "Generalized Canonical Correlation Analysis with Missing Values." *Computational Statistics* 27: 551–71.

Wilson, E. B. 1926. "Empiricism and Rationalism." *Science* 64: 47–57.

Winsberg, S., and J. O. Ramsay. 1980. "Monotone Transformations to Additivity." *Biometrika* 67: 669–74.

———. 1983. "Monotone Spline Transformations for Dimension Reduction." *Psychometrika* 48: 575–95.

Xie, Y. 2015. *Dynamic Documents with R and knitr*. Second Edition. CRC Press.

———. 2016. *Bookdown: Authoring Books with r Markdown.* https://github.com/rstudio/bookdown.

Young, F. W. 1981. "Quantitative Analysis of Qualitative Data." *Psychometrika* 46: 357–88.

Young, F. W., J. De Leeuw, and Y. Takane. 1980. "Quantifying Qualitative Data." In *Similarity and Choice. Papers in Honor of Clyde Coombs*, edited by E. D. Lantermann and H. Feger. Bern: Hans Huber. http://www.stat.ucla.edu/~deleeuw/janspubs/1980/chapters/young_deleeuw_takane_C_80.pdf.

Yu, B. 2014. "Let Us Own Data Science." *IMS Bulletin* 43: 1, 13–16.