VGAM Family Functions for Reduced-Rank Regression and Constrained Ordination

T. W. Yee

November 21, 2006

Beta Version 0.6-5

Department of Statistics,
University of Auckland,
New Zealand
yee@stat.auckland.ac.nz
http://www.stat.auckland.ac.nz/∼yee

Contents

[Important note: This document and code is not yet finished, but should be completed one day . . .]

1 Introduction

This document describes three classes of models that are variants or extensions of the VGLM and VGAM classes. These variants are summarized in Table 1. Their primary defining characteristic is that they operate on latent variables $\boldsymbol{\nu}$. The word "latent variable" has several shades of meaning in statistics but here it is defined as a linear combination of some explanatory variables $\boldsymbol{x}_2$, i.e., $\boldsymbol{\nu} = \mathbf{C}^T\boldsymbol{x}_2$ for some matrix of coefficients $\mathbf{C}$ is a vector of latent variables. This document does not explain all the technicalities behind each model in Table 1, therefore the interested reader is referred to the primary references for further details. Instead, this document focusses on the practical aspects of using the VGAM package to fit the models.

The three central functions in this document are quite similar to `vglm()` but with additional features. The `vglm()` function is similar in spirit to `glm()`, which is described in Chambers and Hastie (1993). The document "VGAM *Family Functions for Categorical Data*" is also relevant to this one, as regression models for categorical responses naturally produce a large number of parameters, therefore are good candidates for reduced-rank modelling.

Table 1: A summary of the models described in this document. The latent variables $\boldsymbol{\nu} = \mathbf{C}^T\boldsymbol{x}_2$, or $\nu = \boldsymbol{c}^T\boldsymbol{x}_2$ if $R = 1$. These models compare with VGLMs where $\boldsymbol{\eta} = \mathbf{B}_1^T\boldsymbol{x}_1 + \mathbf{B}_2^T\boldsymbol{x}_2$. Abbreviations: A = additive, C = constrained (preferred) or canonical, L = linear, O = ordination, Q = quadratic, RR = reduced-rank, VGLM = vector generalized linear model.

$\boldsymbol{\eta}$	Model	Purpose	S function	Reference
$\mathbf{B}_1^T\boldsymbol{x}_1 + \mathbf{A}\,\boldsymbol{\nu}$	RR-VGLM	CLO	`rrvglm()`	Yee and Hastie (2003)
$\mathbf{B}_1^T\boldsymbol{x}_1 + \mathbf{A}\,\boldsymbol{\nu} + \sum_{j=1}^{M}(\boldsymbol{\nu}^T\mathbf{D}_j\boldsymbol{\nu})\boldsymbol{e}_j$	QRR-VGLM	CQO	`cqo()`	Yee (2004)
$\mathbf{B}_1^T\boldsymbol{x}_1 + \sum_{r=1}^{R}\boldsymbol{f}_r(\nu_r)$	RR-VGAM	CAO	`cao()`	Yee (2006a)

1.1 Some Notation

Our data set comprises of the matrices $\mathbf{Y}$ and $\mathbf{X}$ which are $n \times q$ and $n \times p$ respectively. $\mathbf{Y}$ is the response and $\mathbf{X}$ is explanatory. There are n individuals (e.g., people or sites) under study, and i is used to index them. For each individual, a q-dimensional response vector $\boldsymbol{y}$ $(q \geq 1)$ and a p-dimensional covariate vector $\boldsymbol{x} = (x_1, \ldots, x_p)^T$ is observed, e.g., p environmental variables. It is convenient to partition $\boldsymbol{x}$ into $(\boldsymbol{x}_1^T, \boldsymbol{x}_2^T)^T$ later. In this document, often $q = M = S$, where S is the total number of species. The number of linear/additive predictors is always M. The index $s = 1, \ldots, S$ indexes species.

We have $\boldsymbol{\nu} = \mathbf{C}^T \boldsymbol{x}_2 = (\nu_1, \ldots, \nu_R)^T$. If $R = 1$ then we sometimes write $\nu = \boldsymbol{c}^T \boldsymbol{x}_2$. For models with an intercept term it is assumed that $x_1 = 1$, i.e., that of the first element of $\boldsymbol{x}$. The vector $\boldsymbol{e}_i$ is a vector of zeros but with a 1 in the ith position. The dimension of the reduced-rank regression is called the *rank*, R, and is usually 1 or 2 here. It is the number of axes in an ordination. The matrix $\mathbf{C}$ contains the *canonical* (or *constrained*) *coefficients* and are highly interpretable. They are sometimes called the *weights* or *loadings*.

2 What are RR-VGLMs?

To find out what RR-VGLMs are, we need to recall what VGLMs and VGAMs are.

2.1 VGLMs and VGAMs

VGLMs are defined as a model for which the conditional distribution of $\boldsymbol{Y}$ given $\boldsymbol{x}$ is of the form

$$f(\boldsymbol{y}|\boldsymbol{x}; \mathbf{B}) \;=\; h(\boldsymbol{y}, \eta_1, \ldots, \eta_M, \phi)$$

for some known function $h(\cdot)$, where $\mathbf{B} = (\boldsymbol{\beta}_1 \, \boldsymbol{\beta}_2 \, \cdots \, \boldsymbol{\beta}_M)$ is $p \times M$, ϕ is a dispersion parameter, and

$$\eta_j \;=\; \eta_j(\boldsymbol{x}) \;=\; \boldsymbol{\beta}_j^T \boldsymbol{x} \;=\; \beta_{(j)1} \, x_1 + \cdots + \beta_{(j)p} \, x_p \tag{1}$$

is the jth linear predictor. GLMs (McCullagh and Nelder, 1989) are a special case having only $M = 1$ linear predictor, and frequently M does not coincide with the dimension of $\boldsymbol{y}$. We have

$$\boldsymbol{\eta}(\boldsymbol{x}_i) \;=\; \begin{pmatrix} \eta_1(\boldsymbol{x}_i) \\ \vdots \\ \eta_M(\boldsymbol{x}_i) \end{pmatrix} \;=\; \boldsymbol{\eta}_i \;=\; \mathbf{B}^T \boldsymbol{x}_i. \tag{2}$$

Let $\boldsymbol{\eta}_0 = (\beta_{(1)1}, \cdots, \beta_{(M)1})^T$ be the vector of intercepts.

VGAMs provide additive-model extensions to VGLMs, that is, (1) is generalized to

$$\eta_j(\boldsymbol{x}) \;=\; \beta_{(j)1} + f_{(j)2}(x_2) + \cdots + f_{(j)p}(x_p), \quad\quad j = 1, \ldots, M, \tag{3}$$

a sum of smooth functions of the individual covariates, just as with ordinary GAMs (Hastie and Tibshirani, 1990). The η_j in (3) are referred to as *additive predictors*. For identifiability, each component function is centered, i.e., $E[f_{(j)k}] = 0$.

In practice, it is very useful to consider constraints-on-the-functions. For VGAMs, we have

$$\begin{aligned} \boldsymbol{\eta}(\boldsymbol{x}) \;&=\; \boldsymbol{\eta}_0 + \boldsymbol{f}_2(x_2) + \cdots + \boldsymbol{f}_p(x_p) \\ &=\; \mathbf{H}_1 \, \boldsymbol{\beta}_1^* + \mathbf{H}_2 \, \boldsymbol{f}_2^*(x_2) + \cdots + \mathbf{H}_p \, \boldsymbol{f}_p^*(x_p) \end{aligned} \tag{4}$$

where $\mathbf{H}_1, \mathbf{H}_2, \ldots, \mathbf{H}_p$ are known full-column rank *constraint matrices*, $\boldsymbol{f}_k^*$ is a vector containing a possibly reduced set of component functions and $\boldsymbol{\beta}_1^*$ is a vector of unknown intercepts. With no constraints at all, $\mathbf{H}_1 = \mathbf{H}_2 = \cdots = \mathbf{H}_p = \mathbf{I}_M$ and $\boldsymbol{\beta}_1^* = \boldsymbol{\eta}_0$. Like the $\boldsymbol{f}_k$, the $\boldsymbol{f}_k^*$ are centered. For VGLMs, the $\boldsymbol{f}_k$ are linear so that

$$\mathbf{B}^T \;=\; \left(\mathbf{H}_1\boldsymbol{\beta}_1^* \;\; \cdots \;\; \mathbf{H}_p\boldsymbol{\beta}_p^*\right). \tag{5}$$

VGLMs are usually estimated by maximum likelihood estimation by Fisher scoring or Newton-Raphson.

2.2 (Partial) RR-VGLMs

Partition $\boldsymbol{x}$ into $(\boldsymbol{x}_1^T, \boldsymbol{x}_2^T)^T$ and $\mathbf{B} = (\mathbf{B}_1^T \; \mathbf{B}_2^T)^T$. In general, $\mathbf{B}$ is a dense matrix of full rank, i.e., rank $\min(M, p)$. Thus there are $M \times p$ regression coefficients to estimate. For some data sets both M and p are "too" large. For example, in a classification problem of the letters "A" to "Z" and "0" to "9" using a 16×16 grey-scale character digitization, we have $M + 1 = 26 + 10 = 36$ and $p = 1 + 16^2 = 257$ in a multinomial logit model. Then there would be $M \times p = 9252$ coefficients! Unless the data set was huge this would mean that the standard errors of each coefficient would be very large and some simplification would be necessary. Additionally, fitting the model in the first case would require a super computer because of massive memory and computational requirements.

One solution is based on a simple and elegant idea: replace $\mathbf{B}_2$ by a reduced-rank regression. This will cut down the number of regression coefficients enormously if the rank R is kept low. Ideally, the problem can be reduced down to one or two dimensions—a successful application of dimension reduction—and therefore can be plotted. The reduced-rank regression is applied to $\mathbf{B}_2$ because we want to make provision for some variables $\boldsymbol{x}_1$ that we want to leave alone. This means $\mathbf{B}_1$ remains unchanged. In practice, we often let $\boldsymbol{x}_1 = 1$ for the intercept term only.

The idea of reduced-rank regression was first proposed by Anderson (1951) and since then almost all applications of it has been to continuous responses (normal errors)—this is one reason why reduced-rank regression has never become as popular as it should be. In more detail, the new class, called RR-VGLMs has

$$\boldsymbol{\eta} \;=\; \mathbf{B}_1^T\boldsymbol{x}_1 + \mathbf{A}\,\mathbf{C}^T\boldsymbol{x}_2 \;=\; \mathbf{B}_1^T\boldsymbol{x}_1 + \mathbf{A}\,\boldsymbol{\nu}, \tag{6}$$

where $\mathbf{C} = (\boldsymbol{c}_{(1)}\, \boldsymbol{c}_{(2)} \cdots \boldsymbol{c}_{(R)})$ is $p_2 \times R$, $\mathbf{A} = (\boldsymbol{a}_{(1)}\, \boldsymbol{a}_{(2)} \cdots \boldsymbol{a}_{(R)}) = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_M)^T$ is $M \times R$. Both $\mathbf{A}$ and $\mathbf{C}$ are of full-column rank. The effect is that it replaces the dense matrix $\mathbf{B}_2$ by the product of two thin matrices $\mathbf{C}$ and $\mathbf{A}^T$. Of course, $R \leq \min(M, p_2)$ but ideally we want $R \ll \min(M, p_2)$. One can think of (6) as a reduced-rank regression of the coefficients of $\boldsymbol{x}_2$ after having adjusted for the variables in $\boldsymbol{x}_1$.

Strictly speaking, we call models given by (6) *partial RR-VGLMs* because only a part of the regressors have a reduced-rank representation. But we drop the "partial" for convenience. Let $\dim(\boldsymbol{x}_1) = p_1$ and $\dim(\boldsymbol{x}_2) = p_2$ so that $p_1 + p_2 = p$. To distinguish between variables belonging to $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, the argument `Norrr` receives a formula with terms that are left untouched by reduced-rank regression, i.e., those for $\boldsymbol{x}_1$. By default, `Norrr` $\sim$ `1` for $\boldsymbol{x}_1 = 1$. Here's another example.

```
rrvglm(y ~ x2 + x3 + x4 + x5, family = famfun(parallel = TRUE ~ x2 - 1),
       Norrr = ~ x2 + x3)
```

This means that only variables `x4` and `x5` are represented by reduced-rank regression; that is, $\boldsymbol{x}_2 = (X_4, X_5)^T$, and $\boldsymbol{x}_1 = (1, X_2, X_3)^T$ are left alone. The variable X_2 receives a parallelism constraint.

2.3 Why Reduced-Rank Regression?

There are a number of reasons why reduced-rank regression is a good idea:

1. If $R \ll \min(M, p_2)$ then a more parsimonious model results. The resulting number of parameters is often much less than the full model (the difference is $(M-R)(p_2-R)$, which is substantial when $R \ll \min(M, p_2)$).

2. The reduced-rank approximation provides a vehicle for a low dimensional view of the data, e.g., the biplot. This is illustrated later.

3. It allows for a flexible nonparametric generalization—called RR-VGAMs. These are described in Section 8.

4. It is readily interpretable. One can think of $\boldsymbol{\nu}$ as a vector of R latent variables—linear combinations of the original predictor variables that give more explanatory power. They often can be thought of as a proxy for some underlying variable behind the mechanism of the process generating the data. For some models, such as the cumulative logit model (McCullagh, 1980), this argument is natural and well-known. In fields such as plant ecology the idea is an important one.

Unfortunately, much of the standard theory of reduced-rank regression (see, for example, Reinsel and Velu (1998)) and its ramifications are not applicable to RR-VGLMs in general. This is due to technical reasons. Another complication in inference of RR-VGLMs is that the solution to a lower rank problem is not nested within a higher rank problem.

2.4 Normalizations

The factorization (6) is not unique because $\boldsymbol{\eta}_i = \mathbf{B}_1^T \boldsymbol{x}_1 + \mathbf{A}\,\mathbf{M}\,\mathbf{M}^{-1}\,\boldsymbol{\nu}_i$ for any nonsingular matrix $\mathbf{M}$. The following lists some common uniqueness constraints.

1. Restrict $\mathbf{A}$ to the form

$$\mathbf{A} \;=\; \begin{pmatrix} \mathbf{I}_R \\ \widetilde{\mathbf{A}} \end{pmatrix}, \quad \text{say.} \tag{7}$$

   (Actually, it may be necessary to represent $\mathbf{I}_R$ in rows other than the first R.) This is used when derivatives are used to fit the model. This type of constraint is called a *Corner* constraint and corresponds to `Corner=TRUE` in `rrvglm()`.

2. Another normalization of $\mathbf{A}$, which makes direct comparisons with other statistical methods possible, is based on the singular value decomposition (SVD)

$$\mathbf{A}\mathbf{C}^T = (\mathbf{U}\mathbf{D}^{\alpha})(\mathbf{D}^{1-\alpha}\mathbf{V}^T) \tag{8}$$

   for some specified $0 \leq \alpha \leq 1$. For the alternating method of estimation, $\alpha = \frac{1}{2}$ is the default as it scales both sides symmetrically. The parameter α is `Alpha=0.5` in `rrvglm.control()`.

3. It is often possible to choose $\mathbf{M}$ so that $\widehat{\mathrm{Var}}(\mathbf{M}^{-1}\,\widehat{\boldsymbol{\nu}})$ is diagonal and its trace is 1.

4. For the stereotype model described in the next section, we could choose $\mathbf{M}$ so that the columns of $\mathbf{C}$, $\boldsymbol{c}_{(r)}$, $r = 1, \ldots, R$, are orthogonal with respect to the within-group covariance matrix $\mathbf{W}$—this type of normalization is similar to linear discriminant analysis.

5. Sometimes we want to choose $\mathbf{M}$ so that the latent variables are uncorrelated, i.e., $\widehat{\mathrm{Var}}(\widehat{\boldsymbol{\nu}}_i)$ is diagonal.

2.5 Reduced-Rank Multinomial Logit Model (RR-MLM)

The *multinomial logit model* (MLM) is a popular regression model for categorical data. It appears under other names, such as the *multiple logistic regression model* or *polytomous logistic regression model*. The MLM is given by

$$p_j \;=\; P(Y=j|\boldsymbol{x}) \;=\; \frac{\exp\{\eta_j(\boldsymbol{x})\}}{\sum_{\ell=1}^{M+1}\exp\{\eta_\ell(\boldsymbol{x})\}}, \quad j=1,\ldots,M+1,$$

where $\eta_j(\boldsymbol{x}) = \beta_j^T\boldsymbol{x}$. and $\boldsymbol{y}_i$ is a $(M+1)$-vector of counts for a categorical response variable taking levels $1,2,\ldots,M+1$. The MLM is used for a nominal response and is particularly useful for exploring how the relative chances of falling into the response categories depend upon the covariates: $p_j(\boldsymbol{x})/p_k(\boldsymbol{x}) = \exp\{\eta_j(\boldsymbol{x})-\eta_k(\boldsymbol{x})\}$. Identifiability constraints, e.g., $\eta_{M+1}(\boldsymbol{x})\equiv 0$, are required by the model. This implies

$$\log\left(\frac{p_j}{p_{M+1}}\right) \;=\; \eta_j, \quad j=1,\ldots,M.$$

VGAM fits the multinomial logit model using the family function `multinomial()`. It uses the last column of the response matrix as baseline, or if the response is a factor, the last level. The special case of $M=1$ corresponds to logistic regression. For further details, see the article "VGAM *Family Functions for Categorical Data*".

The reduced-rank version of the MLM, the RR-MLM, was proposed by Anderson (1984). He called it the *stereotype model*. Note that Anderson (1984) defined his rank-1 stereotype model to have ordered constraints on the parameters of the $\mathbf{A}$ matrix; VGAM ignores this as it would be very difficult to estimate the model with such a constraint. Greenland (1994) attempted to popularize the 1-dimensional stereotype model.

A typical RR-MLM can be fitted like

```
fit = rrvglm(cbind(y1,y2,y3,y4) ~ x2 + x3 + x4 + x5, family = multinomial,
             data = mydata, Norrr = ~ 1 + x2, Rank = 2)
```

3 Summary of RR-VGLM Functions Written

The function `rrvglm()` should operate on all VGAM family functions with $M>2$, although reduced-rank regression does not make sense for many of them. It returns an object with class `"rrvglm"` which is, not surprisingly, related to a `"vglm"` object.

Table 2 gives the current list of S functions that have been written. The function `rrvglm.control()` is the control function for `rrvglm()`. It calls either `valt.control()`, `optim.control()` for R or `nlminbcontrol()` for S-PLUS, which are the control functions of `valt()`, `optim()`, and `nlminb()`.

Table 5 lists generic functions which are also applicable to RR-VGLMs.

3.1 `valt.control()`

`valt()` implements the alternating algorithm. This algorithm (see (6)) fixes $\mathbf{C}$ and solves for $\mathbf{A}$ and $\mathbf{B}_1$, and then fixes $\mathbf{C}$ and solves for $\mathbf{A}$ and $\mathbf{B}_1$; this can be iterated until convergence. The most important arguments to `valt.control()` are `Linesearch` and `Maxit`.

3.2 optim.control() and nlminbcontrol()

In R, optim() is used for the derivative-based algorithm, while in S-PLUS, nlminb() is used. The functions optim.control() and nlminbcontrol() pass on parameters to the optimizing functions.

Table 2: A summary of VGAM functions and code for RR-VGLMs. See also Table 5 for several functions that are applicable to RR-VGLMs too.

Function	Purpose
rrvglm()	Fits RR-VGLMs.
rrar()	RR-autoregressive model (for vglm() only) family function
grc()	Goodman's RC association model (for two-way tables)
biplot.rrvglm()	Biplot for RR-VGLMs ($R = 2$ only)
lvplot.rrvglm()	Same as biplot.rrvglm().
printsummary.rrvglm()	Prints the summary
summary.rrvglm()	Summary function for 'all' RR-VGLMs
nlminbcontrol()	Control function for nlminb() in S-PLUS
rrvglm.optim.control()	Control function for optim() in R
rrvglm.control()	Control function for rrvglm()
valt.control()	Control function for valt()
valt()	Alternating algorithm
rrr.init.expression	Setup expression
rrr.end.expression	Cleanup expression
rrr.alternating.expression	Implements the alternating algorithm
rrr.derivative.expression	Implements the derivative algorithm
rrr.normalize()	Implements the various normalizations

4 Other RR-VGLM Topics

4.1 Normalizations

Currently rrr.normalize() implements the various normalizations. This is actually not good because the code is not modular.

VGAM family functions based on the alternating method have the arguments Corner=TRUE, Svd=FALSE, Uncor=FALSE. If set true, they apply the corner, SVD and uncorrelated normalizations described in Section 2.4. Note that if Svd=TRUE is set then Corner=FALSE is set internally. For corner constraints Index.corner=1:Rank is the default, which is (7); these give the rows for which $\mathbf{I}_R$ occupy. The argument Alpha=0.5 is the α parameter for the SVD method in (8).

For the derivative-based algorithm only corner constraints are supported.

4.2 Output

Suppose fit is an "rrvglm" object. Then, using a combination of (5) and (6),

(i) fit@constraints holds the $\mathbf{H}_k$ and $\widehat{\mathbf{A}}$ (these are better extracted with constraints(fit)),

(ii) `coef(fit)` holds the $\hat{\boldsymbol{\beta}}_k^*$ and $\mathrm{vec}(\widehat{\mathbf{C}}^T)$,

(iii) `coef(fit, matrix=TRUE)` is the estimate of $\mathbf{B} = (\mathbf{B}_1^T \; \mathbf{A}\mathbf{C}^T)^T$,

(iv) `Coef(fit)` returns $\widehat{\mathbf{A}}$, $\widehat{\mathbf{C}}$, $\widehat{\mathbf{B}}_1$ etc.

The function `lv.rrvglm()` returns a $n \times R$ matrix of latent variables with the ith row equalling $\hat{\boldsymbol{\nu}}_i^T$. These are plotted in `lvplot.rrvglm()` (rank-2 only), which is equivalent to `biplot.rrvglm()`.

4.3 Implementation Details

The S expression `rrr.init.expression` contains essential code that is inserted into `vglm.fit()` to give `rrvglm.fit()`. Furthermore, one of `rrr.alternating.expression` and `rrr.derivative.expression` is also used: these perform, e.g., an alternating algorithm iteration between successive Newton-Raphson/Fisher scoring iterations.

The function `valt.control()` allows a RR-VGLM family function to have arguments that affect `valt()`. It has the same type of effect that `vglm.control()` has on `vglm()`. Similarly, `rrvglm.optim.control()` and `nlminbcontrol()` for the derivative based algorithm. Arguments usually start with an uppercase letter to avoid conflict with the control parameters associated with those of `vglm()`.

4.4 Convergence

The alternating algorithm can be very slow at converging but VGAM allows a line search in the alternating method which may improve convergence substantially. It is invoked by `Linesearch=TRUE`.

Unknown elements in $\mathbf{C}$ are chosen randomly using `rnorm()`. One should use `set.seed()` before calling `rrvglm()` to control this if reproducibility is required.

4.5 Latent Variable Plots and Biplots

Biplots are available for RR-VGLMs and provide a graphical summary of the rank-2 approximation to $\mathbf{B}_2$. These are based on Equation (6), and show that the k-j element of $\mathbf{B}_2$ is the inner-product of the kth row of $\mathbf{C}$ and the jth row of $\mathbf{A}$. The rows of $\mathbf{C}$ are usually represented by arrows, and the rows of $\mathbf{A}$ by points (labelled by `object@misc$predictors.names`). Currently, `lvplot.rrvglm()` is equivalent to `biplot.rrvglm()`.

If `fit2` is a rank-2 RR-VGLM, then `lvplot(fit2)` will produce a scatter plot of the fitted latent variables $\hat{\nu}_{i2}$ versus $\hat{\nu}_{i1}$ (see (6)). A convex hull can be overlaid on each group. By default, all the observations belong to one group.

4.6 Miscellaneous Notes

1. `summary.rrvglm()` computes asymptotic standard errors. If the matrix `@cov.unscaled` is not positive-definite (this would indicate an ill-posed model, e.g., one where the intercepts are part of $\boldsymbol{x}_2$ rather than $\boldsymbol{x}_1$), then a warning is given. In such cases the user should try

   (a) `numerical=TRUE` to use numerical derivatives,

   (b) fitting another model,

(c) increasing the rank.

Alternatively, there are the options `omit13` and `kill.all` in `summary.rrvglm()` that allow an 'inferior' `@cov.unscaled` to be returned, for example, one where $\mathbf{A}$ is fixed or one where $\mathbf{C}$ is fixed. Then the standard errors are too small.

2. `summary.rrvglm()` only works with corner constraints (7), i.e., if `object@control$Corner=TRUE`. In the output, the elements of $\widetilde{\mathbf{A}}$ in (7) come first. They are labelled with the prefix `"I(lv.mat)"` or something similar. The elements are enumerated going down each column starting from the first column, i.e., $\mathrm{vec}(\widetilde{\mathbf{A}})$. Then comes the fitted coefficients $\hat{\boldsymbol{\beta}}_k^*$ corresponding to $\mathbf{B}_1$, followed by $\mathrm{vec}(\mathbf{C}^T)$. Actually, $\mathbf{B}_1$ and $\mathrm{vec}(\mathbf{C}^T)$ are intermingled, depending on the order of the original formula.

3. ϕ is estimated if `summary(..., dispersion=0)` is invoked. Then

$$\hat{\phi} \;=\; \frac{1}{nM-p^*}\sum_{i=1}^{n}\left(\boldsymbol{z}_i-\widehat{\mathbf{B}}_1^T\boldsymbol{x}_{1i}-\widehat{\mathbf{A}}\,\widehat{\mathbf{C}}^T\boldsymbol{x}_{2i}\right)^T\mathbf{W}_i\left(\boldsymbol{z}_i-\widehat{\mathbf{B}}_1^T\boldsymbol{x}_{1i}-\widehat{\mathbf{A}}\,\widehat{\mathbf{C}}^T\boldsymbol{x}_{2i}\right) \quad (9)$$

where $p^* = \mathrm{dim}(\hat{\boldsymbol{\beta}}_1^{*T},\ldots,\hat{\boldsymbol{\beta}}_{p_1}^{*T},\mathrm{vec}(\widetilde{\mathbf{A}})^T)^T$ is the total number of parameters to be estimated.

4. For an example of a reduced-rank autoregressive model for time-series data see Ahn and Reinsel (1988).

5 RR-VGLM Tutorial Examples

In this section we illustrate some of the models available.

5.1 Stereotype model

The data frame `car.all` is available in S-PLUS (type `data(car.all)` in R) We first preprocess the data.

```
> data(car.all)
> cars = car.all
> y = cars[["Country"]]
> cars = cars[y == "USA" | y == "Japan" | y == "Germany" |
+     y == "Japan/USA", ]
> cars = cars[, c("Country", "Length", "Width", "Weight",
+     "HP", "Disp.", "Price")]
> cars = na.omit(cars)
> cars[, -1] = scale(cars[, -1])
> temp = as.character(cars[, 1])
> temp[temp == "Germany"] = "G"
> temp[temp == "Japan"] = "J"
> temp[temp == "Japan/USA"] = "T"
> temp[temp == "USA"] = "U"
> cars[, 1] = as.factor(temp)
> cars[1:5, ]
```

```
              Country     Length       Width     Weight         HP
Acura Integra       J -0.3090048 -0.54606484 -0.5694623 -0.1047130
Acura Legend        J  0.6535452  0.05765902  0.4535964  0.6271814
Audi 100            G  0.7910523  0.66138288 -0.1439423 -0.1047130
Audi 80             G -0.3777584 -0.54606484 -0.6237840 -0.6414355
BMW 325i            G -0.4465120 -1.14978870 -0.2163713  0.8223532
                    Disp.      Price
Acura Integra -0.80031904 -0.5102931
Acura Legend   0.02117533  1.0068293
Audi 100      -0.33319479  1.2602752
Audi 80       -0.65534945  0.3128139
BMW 325i      -0.15600973  0.9938017
```

Now we can fit a rank-1 model.

```
> set.seed(301)
> cars[, -1] = scale(cars[, -1])
> fit1 = rrvglm(Country ~ Length + Width + Weight + HP +
+     Disp. + Price, multinomial, Svd = TRUE, Uncor = TRUE,
+     data = cars)
> fit1

Call:
rrvglm(formula = Country ~ Length + Width + Weight + HP + Disp. +
    Price, family = multinomial, data = cars, Svd = TRUE, Uncor = TRUE)

Coefficients:

Residual Deviance: 114.9055
Log-likelihood: -57.45273
```

Now look at some of the constraint matrices.

```
> constraints(fit1)[1:2]

$`(Intercept)`
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1

$Length
          [,1]
[1,] 9.570348
[2,] 7.417516
[3,] 5.190612
```

The intercept constraint matrix is often $\mathbf{I}_M$ and the rest are $\widehat{\mathbf{A}}$. Now $\widehat{\mathbf{B}} = (\hat{\boldsymbol{\eta}}_0, \widehat{\mathbf{B}}_2^T)^T$ is:

```
> coef(fit1, matrix = TRUE)
```

```
            log(mu[,1]/mu[,4]) log(mu[,2]/mu[,4]) log(mu[,3]/mu[,4])
(Intercept)          -2.673038          0.3338085         -0.2982763
Length               -3.570319         -2.7671825         -1.9364125
Width                -1.910480         -1.4807207         -1.0361753
Weight                2.417909          1.8740050          1.3113869
HP                    1.032084          0.7999188          0.5597653
Disp.                -9.053950         -7.0172804         -4.9105362
Price                 8.661599          6.7131882          4.6977393
```

Biplots are available for rank-2 models. Running

```
> fit2 = rrvglm(Country ~ Length + Width + Weight + HP +
+     Disp. + Price, multinomial, Rank = 2, Svd = TRUE, Uncor = TRUE,
+     Linesearch = TRUE, data = cars)
```

first, we can obtain the two plots in Figure 1 easily.

```
> par(mfrow = c(2, 1), mar = c(5, 4, 1, 1) + 0.1)
> grps = unclass(ordered(cars$Country))
> lvplot(fit2, scores = TRUE, spch = grps, C = FALSE, A = FALSE,
+     scol = grps)
> biplot(fit2, gap = 0.1, scale = 0.5, xlim = c(-3, 5), ylim = c(-4,
+     4))
```

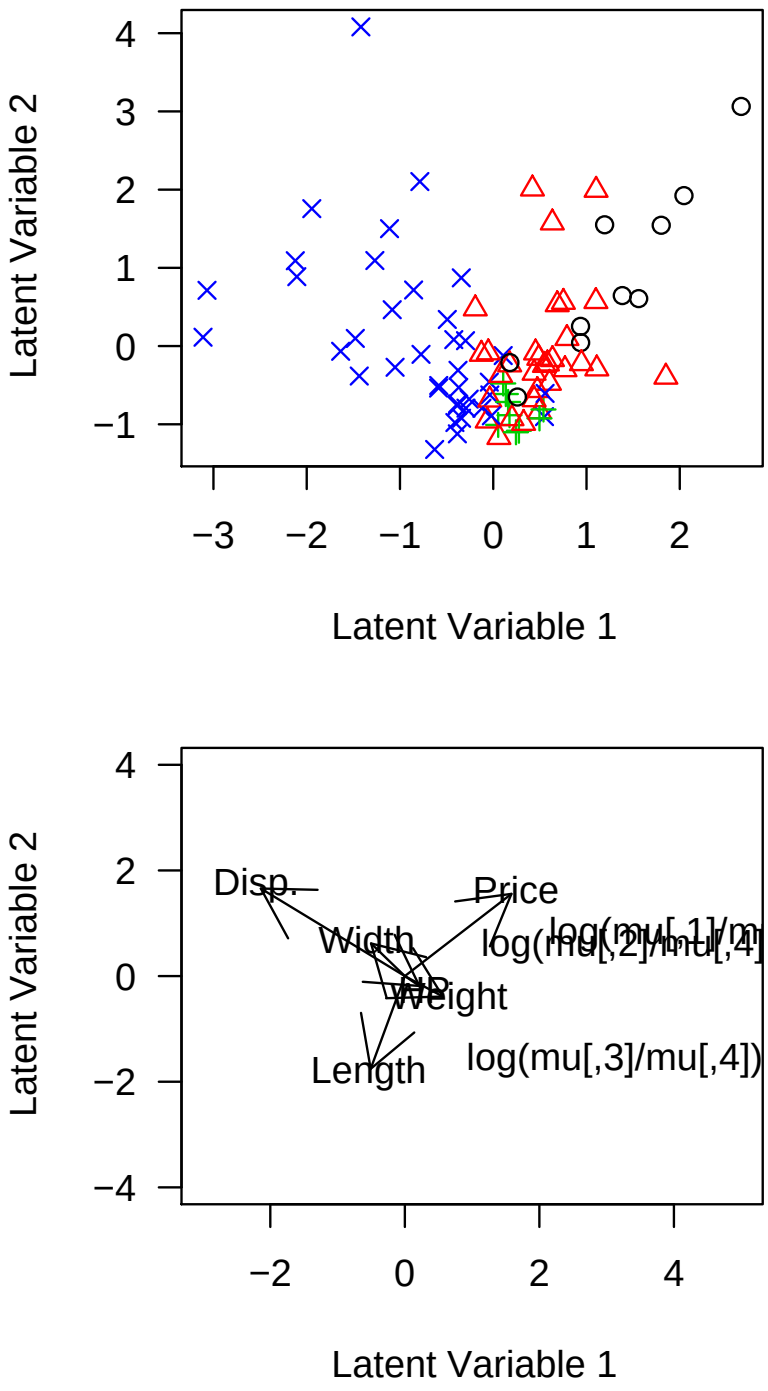

Figure 1: Car data: latent variable plots (biplots) of `fit2`.

Now corner constraints are the default, and these are necessary for a summary of the object.

```
> fit3 = rrvglm(Country ~ Length + Width + Weight + HP +
+     Disp. + Price, multinomial, Rank = 1, cars)
> print(summary(fit3))

Call:
rrvglm(formula = Country ~ Length + Width + Weight + HP + Disp. +
    Price, family = multinomial, data = cars, Rank = 1)

Pearson Residuals:
                       Min       1Q    Median         3Q    Max
log(mu[,1]/mu[,4]) -3.5445 -0.16507 -0.035991 -0.0053429 4.8180
log(mu[,2]/mu[,4]) -4.6939 -0.49349 -0.066701  0.5011074 2.1657
log(mu[,3]/mu[,4]) -3.7784 -0.25807 -0.197650 -0.0539636 1.9870

Coefficients:
                  Value Std. Error  t value
I(lv.mat):1     0.77505   0.084341  9.18949
I(lv.mat):2     0.54236   0.126586  4.28457
(Intercept):1 -2.67306   1.047357 -2.55219
(Intercept):2  0.33381   0.562960  0.59295
(Intercept):3 -0.29828   0.575931 -0.51791
Length        -3.57030   1.519037 -2.35037
Width         -1.91048   1.554015 -1.22938
Weight         2.41791   1.657847  1.45846
HP             1.03207   1.218558  0.84696
Disp.         -9.05395   2.424809 -3.73388
Price          8.66159   1.929831  4.48826

Number of linear predictors:  3

Names of linear predictors:
log(mu[,1]/mu[,4]), log(mu[,2]/mu[,4]), log(mu[,3]/mu[,4])

Dispersion Parameter for multinomial family:   1

Residual Deviance: 114.9055 on 256 degrees of freedom

Log-likelihood: -57.45273 on 256 degrees of freedom

Number of Iterations: 6
```

Just to show that prediction works, one can try

```
> index = 1:4
> max(abs(predict(fit3, cars[index, ]) - predictors(fit3)[index,
+     ]))

[1] 1.598721e-14
```

5.2 Goodman's RC Association Model

Suppose $\mathbf{Y} = [(y_{ij})]$ is a $n \times M$ matrix of counts. Goodman's RC(R) association model (Goodman, 1981) fits a reduced-rank type model to $\mathbf{Y}$ by firstly assuming that Y_{ij} has a Poisson distribution, and that

$$\log \mu_{ij} \;=\; \mu + \alpha_i + \gamma_j + \sum_{r=1}^{R} c_{ir}\, a_{jr}, \quad i = 1, \ldots, n; \;\; j = 1, \ldots, M, \tag{10}$$

where $\mu_{ij} = E(Y_{ij})$ is the mean of the i-j cell. Here, R is the rank, which is usually chosen so that $R \ll \min(n, M)$. Note that (10) is saturated when $R = \min(n, M)$. Note that Equation (4.3) in Yee and Hastie (2003) is wrong; it is corrected by (10).

In (10) the parameters α_i and γ_j are called the *row* and *column scores* respectively. Identifiability constraints are needed for these, such as corner constraints, e.g., $\alpha_1 = \gamma_1 = 0$. The parameters a_{ir} and c_{jr} also need constraints, e.g., $a_{1r} = c_{1r} = 0$ for $r = 1, \ldots, R$. We can write (10) as

$$\log \mu_{ij} \;=\; \mu + \alpha_i + \gamma_j + \delta_{ij},$$

where the $n \times M$ matrix $\boldsymbol{\Delta} = [(\delta_{ij})]$ of interaction terms is approximated by the reduced rank quantity $\sum_{r=1}^{R} c_{ir}\, a_{jr}$.

Goodman's RC(R) association model fits within the VGLM framework by letting

$$\boldsymbol{\eta}_i \;=\; \log \boldsymbol{\mu}_i \tag{11}$$

where $\boldsymbol{\mu}_i = E(\boldsymbol{Y}_i)$ is the mean of the ith row of $\mathbf{Y}$. Then Goodman's RC(R) association model models the matrix $(\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_n)^T$ using reduced-rank regression; Yee and Hastie (2003) gives details on how (10) fits into the reduced-rank VGLM framework (6). To save a long story, VGAM can fit (10), but requires a considerable amount of setting up of indicator variables etc. before calling `rrvglm()`. For this reason, the function `grc()` has been written to fit Goodman's RC model easily. It accepts a matrix as its first argument, and any other argument (except `summary`; see below) is fed into `rrvglm.control()`. This means, for example, that the default rank is $R = 1$.

We now present a numerical example with the artificial data set given in Table 3. We use corner constraints $\alpha_1 = \gamma_1 = 0$ and fit a rank-2 model.

```
> y = matrix(c(4, 6, 4, 2, 1, 6, 5, 7, 1, 1, 4, 8, 9, 4,
+     3, 2, 2, 5, 7, 6), 4, 5, byrow = TRUE)
> dimnames(y) = list(letters[1:nrow(y)], LETTERS[1:ncol(y)])
> y

  A B C D E
a 4 6 4 2 1
```

Table 3: An artificial 4×5 table of data.

	A	B	C	D	E
a	4	6	4	2	1
b	6	5	7	1	1
c	4	8	9	4	3
d	2	2	5	7	6

```
b 6 5 7 1 1
c 4 8 9 4 3
d 2 2 5 7 6
```

```
> options(contrasts = c("contr.treatment", "contr.poly"))
> g2 = grc(y, Rank = 2)
```

Before looking at the estimates, we can look at the model matrices.

```
> cbind(g2@x)
```

```
  (Intercept) Row2 Row3 Row4 Col2 Col3 Col4 Col5 b c d
a           1    0    0    0    1    1    1    1 0 0 0
b           1    1    0    0    1    1    1    1 1 0 0
c           1    0    1    0    1    1    1    1 0 1 0
d           1    0    0    1    1    1    1    1 0 0 1
```

```
> cbind(model.matrix(g2))
```

```
    (Intercept) Row2 Row3 Row4 Col2 Col3 Col4 Col5       b:1      b:2
a:1           1    0    0    0    0    0    0    0 0.0000000 0.000000
a:2           1    0    0    0    1    0    0    0 0.0000000 0.000000
a:3           1    0    0    0    0    1    0    0 0.0000000 0.000000
a:4           1    0    0    0    0    0    1    0 0.0000000 0.000000
a:5           1    0    0    0    0    0    0    1 0.0000000 0.000000
b:1           1    1    0    0    0    0    0    0 0.0000000 0.000000
b:2           1    1    0    0    1    0    0    0 1.0000000 0.000000
b:3           1    1    0    0    0    1    0    0 0.0000000 1.000000
b:4           1    1    0    0    0    0    1    0 0.9455561 3.040363
b:5           1    1    0    0    0    0    0    1 0.2457475 3.144984
c:1           1    0    1    0    0    0    0    0 0.0000000 0.000000
c:2           1    0    1    0    1    0    0    0 0.0000000 0.000000
c:3           1    0    1    0    0    1    0    0 0.0000000 0.000000
c:4           1    0    1    0    0    0    1    0 0.0000000 0.000000
c:5           1    0    1    0    0    0    0    1 0.0000000 0.000000
d:1           1    0    0    1    0    0    0    0 0.0000000 0.000000
d:2           1    0    0    1    1    0    0    0 0.0000000 0.000000
d:3           1    0    0    1    0    1    0    0 0.0000000 0.000000
d:4           1    0    0    1    0    0    1    0 0.0000000 0.000000
d:5           1    0    0    1    0    0    0    1 0.0000000 0.000000
          c:1      c:2       d:1      d:2
a:1 0.0000000 0.000000 0.0000000 0.000000
a:2 0.0000000 0.000000 0.0000000 0.000000
a:3 0.0000000 0.000000 0.0000000 0.000000
a:4 0.0000000 0.000000 0.0000000 0.000000
a:5 0.0000000 0.000000 0.0000000 0.000000
b:1 0.0000000 0.000000 0.0000000 0.000000
b:2 0.0000000 0.000000 0.0000000 0.000000
b:3 0.0000000 0.000000 0.0000000 0.000000
b:4 0.0000000 0.000000 0.0000000 0.000000
```

```
b:5 0.0000000 0.000000 0.0000000 0.000000
c:1 0.0000000 0.000000 0.0000000 0.000000
c:2 1.0000000 0.000000 0.0000000 0.000000
c:3 0.0000000 1.000000 0.0000000 0.000000
c:4 0.9455561 3.040363 0.0000000 0.000000
c:5 0.2457475 3.144984 0.0000000 0.000000
d:1 0.0000000 0.000000 0.0000000 0.000000
d:2 0.0000000 0.000000 1.0000000 0.000000
d:3 0.0000000 0.000000 0.0000000 1.000000
d:4 0.0000000 0.000000 0.9455561 3.040363
d:5 0.0000000 0.000000 0.2457475 3.144984
```

The first one, g2@x, is the model matrix corresponding to the lm-type, whereas model.matrix(g2) is the 'large' model matrix corresponding to the vlm-type. Now the estimates are as follows.

```
> print(g2)

Call:
rrvglm(formula = as.formula(str2), family = poissonff, data = .grc.df,
    control = myrrcontrol, constraints = cms)


Coefficients:


Residual Deviance: 0.5504491
Log-likelihood: -31.99336
```

```
> coef(g2, matrix = TRUE)

             log(E[A])  log(E[B])  log(E[C])  log(E[D])  log(E[E])
(Intercept)  1.2254942  1.2254942  1.2254942  1.2254942  1.2254942
Row2         0.5622138  0.5622138  0.5622138  0.5622138  0.5622138
Row3         0.3565643  0.3565643  0.3565643  0.3565643  0.3565643
Row4        -0.6639087 -0.6639087 -0.6639087 -0.6639087 -0.6639087
Col2         0.0000000  0.6011756  0.0000000  0.0000000  0.0000000
Col3         0.0000000  0.0000000  0.2940680  0.0000000  0.0000000
Col4         0.0000000  0.0000000  0.0000000 -0.6591283  0.0000000
Col5         0.0000000  0.0000000  0.0000000  0.0000000 -1.1777399
b            0.0000000 -0.7777065 -0.1325454 -1.1383513 -0.6079725
c            0.0000000 -0.1433544  0.2243264  0.5464841  0.6702738
d            0.0000000 -0.4263387  0.8000238  2.0292357  2.4112899
```

From the output, it can be seen that $\hat{\mu} = 1.2255$, $\hat{\alpha}_2 = 0.5622$, $\hat{\gamma}_2 = 0.6012$ etc. By default, grc() has the first row of $\mathbf{A}$ consisting of structural zeros, which has the effect of zeroing the first column of $\mathbf{\Delta}$. One can see this here by looking at, e.g.,

```
> constraints(g2)[["b"]]

          [,1]     [,2]
[1,] 0.0000000 0.000000
[2,] 1.0000000 0.000000
[3,] 0.0000000 1.000000
[4,] 0.9455561 3.040363
[5,] 0.2457475 3.144984
```

which is $\widehat{\mathbf{A}}$. Also,

$$\widehat{\mathbf{C}} = \begin{pmatrix} 0.0000 & 0.0000 \\ -0.7777 & -0.1326 \\ -0.1433 & 0.2243 \\ -0.4264 & 0.8000 \end{pmatrix}.$$

The fitted values can be obtained by

```
> fitted(g2)
```

```
          A        B        C        D        E
a 3.405849 6.213161 4.570224 1.761853 1.048913
b 5.975741 5.008704 7.023282 0.990276 1.001998
c 4.864960 7.689684 8.169869 4.346688 2.928799
d 1.753450 2.088450 5.236625 6.901183 6.020291
```

Note that the function `biplot()` can return quantities such the estimated $\mathbf{A}$ and $\mathbf{C}$:

```
> i = Coef(g2)
> rbind(A = 0, i@C) %*% t(i@A)
```

```
  log(E[A])  log(E[B])  log(E[C])  log(E[D])  log(E[E])
A         0  0.0000000  0.0000000  0.0000000  0.0000000
b         0 -0.7777065 -0.1325454 -1.1383513 -0.6079725
c         0 -0.1433544  0.2243264  0.5464841  0.6702738
d         0 -0.4263387  0.8000238  2.0292357  2.4112899
```

The latter is $\widehat{\mathbf{\Delta}}$, which is the bottom portion of `coef(g2, matrix=TRUE)` above. Now

```
> svd(rbind(0, i@C) %*% t(i@A))
```

```
$d
[1] 3.584733e+00 1.005117e+00 1.283493e-16 0.000000e+00

$u
           [,1]       [,2]        [,3] [,4]
[1,]  0.0000000  0.0000000  0.00000000    1
[2,]  0.3285847 -0.9440891  0.02698058    0
[3,] -0.2503684 -0.1146129 -0.96134256    0
[4,] -0.9106853 -0.3091273  0.27403016    0

$v
            [,1]       [,2]       [,3] [,4]
[1,]  0.00000000  0.0000000  0.0000000    1
[2,]  0.04703541  0.8779550  0.2559655    0
[3,] -0.23105945 -0.1471324 -0.6144152    0
[4,] -0.65803005  0.3828198 -0.4293540    0
[5,] -0.71511933 -0.2469737  0.6104348    0
```

which verifies that $\widehat{\mathbf{\Delta}}$ is of rank 2. One idea of determining the rank R is to fit a high rank model and plot the singular values as in a scree plot. A sharp dropoff is a suggested way of guessing the best R.

Here is a summary.

```
> print(summary(g2))

Call:
rrvglm(formula = as.formula(str2), family = poissonff, data = .grc.df,
    control = myrrcontrol, constraints = cms)

Pearson Residuals:
  log(E[A]) log(E[B])  log(E[C])  log(E[D])  log(E[E])
a  0.321947 -0.085517 -0.2667330  0.1794159 -0.0477589
b  0.009924 -0.003889 -0.0087852  0.0097716 -0.0019956
c -0.392154  0.111905  0.2904285 -0.1662876  0.0416047
d  0.186191 -0.061205 -0.1034035  0.0376157 -0.0082696

Coefficients:
                Value Std. Error   t value
I(lv.mat)1:1  0.94557    4.82952  0.195790
I(lv.mat)1:2  0.24574    3.85305  0.063779
I(lv.mat)2:1  3.04036    3.77775  0.804809
I(lv.mat)2:2  3.14499    3.20025  0.982732
(Intercept)   1.22549    0.48190  2.543028
Row2          0.56221    0.64964  0.865424
Row3          0.35656    0.53060  0.672000
Row4         -0.66391    0.99316 -0.668482
Col2          0.60118    0.56856  1.057368
Col3          0.29407    0.68741  0.427790
Col4         -0.65913    0.90481 -0.728468
Col5         -1.17774    0.98394 -1.196963
b:1          -0.77771    0.82623 -0.941271
b:2          -0.13255    1.00334 -0.132104
c:1          -0.14335    0.74182 -0.193247
c:2           0.22433    0.49299  0.455036
d:1          -0.42634    1.32336 -0.322164
d:2           0.80002    1.29461  0.617963

Number of linear predictors:  5

Names of linear predictors:
log(E[A]), log(E[B]), log(E[C]), log(E[D]), log(E[E])

(Default) Dispersion Parameter for poissonff family:   1

Residual Deviance: 0.55045 on 2 degrees of freedom

Log-likelihood: -31.99336 on 2 degrees of freedom

Number of Iterations: 6
```

The standard errors are 'correct' because it treats the elements of $\widetilde{\mathbf{A}}$ as unknown parameters.

Finally, a biplot can be obtained (see Figure 2.)

```
> biplot(g2, xlim = c(-1.5, 1.5))
```

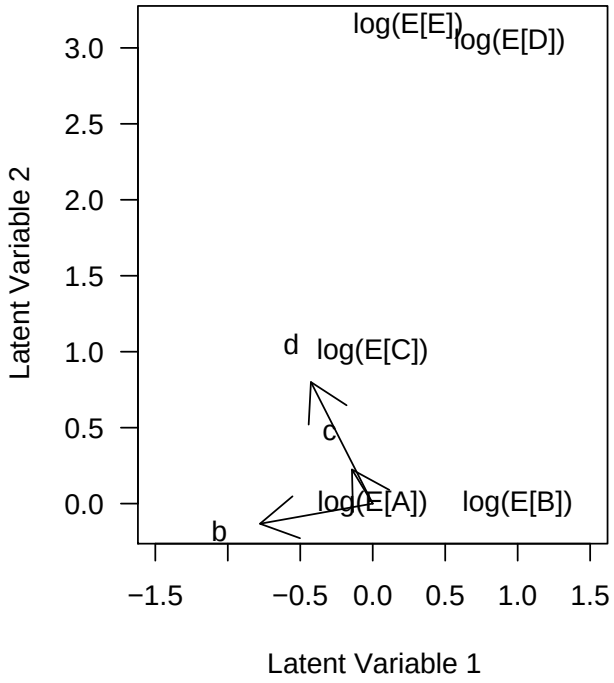

Figure 2: Biplot of a Goodman's RC(2) association model.

5.3 Some advice

Sometimes a `summary()` of a `"grc"` object fails because the estimated covariance matrix is not positive-definite. This is often due to numerical ill-conditioning: the position of the corner contraint for $\mathbf{I}_R$ in $\mathbf{A}$ causes the elements of $\widetilde{\mathbf{A}}$ to be very large or small. The easiest fix is to try choose different values for the `Index.corner` argument in `grc()`. For example, if

```
fit = grc(y, Rank=2, Index.corner=2:3)
sfit = summary(fit)
```

fails, try something like

```
fit = grc(y, Rank=2, Index.corner=c(3,4))
sfit = summary(fit)
```

5.4 A Goodman's RC Numerical Example

Here we fit another Goodman's RC association model, but to the data in Table 4 and explicitly setting up indicator variables etc. Yee and Hastie (2003) gives the details on how Goodman's RC model is a partial RR-VGLM, with the need for only some setting up of indicator variables and constraint matrices. For simplicity we fit a rank-1 model (a rank-2 model simply requires `Rank=2` below.) The VGAM family function `poissonff()` accepts a matrix response and has (11) as the default. The variable `y` contains the 4×5 matrix of counts.

```
> ei = function(i, n) diag(n)[, i, drop = FALSE]
> Col2 = Col3 = Col4 = Col5 = rep(1, nrow(y))
> Row2 = x22 = ei(2, nrow(y))
> Row3 = x23 = ei(3, nrow(y))
> Row4 = x24 = ei(4, nrow(y))
> cms = list("(Intercept)" = matrix(1, ncol(y), 1), Col2 = ei(2,
+     ncol(y)), Col3 = ei(3, ncol(y)), Col4 = ei(4, ncol(y)),
+     Col5 = diag(ncol(y)))
> cms$Row2 = cms$Row3 = cms$Row4 = matrix(1, ncol(y), 1)
> cms$x22 = cms$x23 = cms$x24 = diag(ncol(y))
> g1 = rrvglm(y ~ Row2 + Row3 + Row4 + Col2 + Col3 + Col4 +
+     Col5 + x22 + x23 + x24, Rank = 1, family = poissonff,
+     constraints = cms, Structural.zero = 1, Norrr = ~Row2 +
+         Row3 + Row4 + Row5 + Col2 + Col3 + Col4)
```

Table 4: Some undergraduate student enrolments at the University of Auckland in 1990. Each student is cross-classified by their colleges (Science and Engineering have been combined) and the socio-economic status (SES) of their fathers (1 =highest, 4 =lowest). Data: Dr Tony Morrison.

SES	Commerce	Arts	SciEng	Law	Medicine
1	446	895	496	170	184
2	937	1834	994	246	198
3	311	805	430	95	48
4	49	157	62	15	9

The omission of a variable called x21 causes $\mathbf{\Delta}$ to have a zero first row. Before looking at the estimates, let's check the model matrices.

```
> cbind(g1@x)
```

```
  (Intercept) Row2 Row3 Row4 Col2 Col3 Col4 Col5 x22 x23 x24
a           1    0    0    0    1    1    1    1   0   0   0
b           1    1    0    0    1    1    1    1   1   0   0
c           1    0    1    0    1    1    1    1   0   1   0
d           1    0    0    1    1    1    1    1   0   0   1
```

```
> cbind(model.matrix(g1))
```

```
    (Intercept) Row2 Row3 Row4 Col2 Col3 Col4     Col5      x22
a:1           1    0    0    0    0    0    0 0.000000 0.000000
a:2           1    0    0    0    1    0    0 1.000000 0.000000
a:3           1    0    0    0    0    1    0 3.131003 0.000000
a:4           1    0    0    0    0    0    1 8.897212 0.000000
a:5           1    0    0    0    0    0    0 9.580658 0.000000
b:1           1    1    0    0    0    0    0 0.000000 0.000000
b:2           1    1    0    0    1    0    0 1.000000 1.000000
b:3           1    1    0    0    0    1    0 3.131003 3.131003
b:4           1    1    0    0    0    0    1 8.897212 8.897212
b:5           1    1    0    0    0    0    0 9.580658 9.580658
c:1           1    0    1    0    0    0    0 0.000000 0.000000
c:2           1    0    1    0    1    0    0 1.000000 0.000000
c:3           1    0    1    0    0    1    0 3.131003 0.000000
c:4           1    0    1    0    0    0    1 8.897212 0.000000
c:5           1    0    1    0    0    0    0 9.580658 0.000000
d:1           1    0    0    1    0    0    0 0.000000 0.000000
d:2           1    0    0    1    1    0    0 1.000000 0.000000
d:3           1    0    0    1    0    1    0 3.131003 0.000000
d:4           1    0    0    1    0    0    1 8.897212 0.000000
d:5           1    0    0    1    0    0    0 9.580658 0.000000
         x23      x24
a:1 0.000000 0.000000
a:2 0.000000 0.000000
a:3 0.000000 0.000000
a:4 0.000000 0.000000
a:5 0.000000 0.000000
b:1 0.000000 0.000000
b:2 0.000000 0.000000
b:3 0.000000 0.000000
b:4 0.000000 0.000000
b:5 0.000000 0.000000
c:1 0.000000 0.000000
c:2 1.000000 0.000000
c:3 3.131003 0.000000
c:4 8.897212 0.000000
c:5 9.580658 0.000000
```

```
d:1 0.000000 0.000000
d:2 0.000000 1.000000
d:3 0.000000 3.131003
d:4 0.000000 8.897212
d:5 0.000000 9.580658
```

The matrix g1@x corresponds to an ordinary linear model, while model.matrix(g1) is a permutation of the VLM model matrix $\mathbf{X}_*$, with columns corresponding to $\boldsymbol{\beta}^* = (\boldsymbol{\beta}_1^{*T}, \ldots, \boldsymbol{\beta}_p^{*T})^T$. The model matrix has adjustments due to the constraint matrices.

Now the estimates are as follows.

```
> g1

Call:
rrvglm(formula = y ~ Row2 + Row3 + Row4 + Col2 + Col3 + Col4 +
    Col5 + x22 + x23 + x24, family = poissonff, constraints = cms,
    Rank = 1, Structural.zero = 1, Norrr = ~Row2 + Row3 + Row4 +
        Row5 + Col2 + Col3 + Col4)

Coefficients:

Residual Deviance: 1.969910
Log-likelihood: -32.70309

> round(coef(g1, matrix = TRUE), digits = 4)

            log(E[A]) log(E[B]) log(E[C]) log(E[D]) log(E[E])
(Intercept)    1.3872    1.3872    1.3872    1.3872    1.3872
Row2           0.2918    0.2918    0.2918    0.2918    0.2918
Row3           0.2389    0.2389    0.2389    0.2389    0.2389
Row4          -0.9473   -0.9473   -0.9473   -0.9473   -0.9473
Col2           0.0000    0.3672    0.0000    0.0000    0.0000
Col3           0.0000    0.0000    0.7084    0.0000    0.0000
Col4           0.0000    0.0000    0.0000    0.2600    0.0000
Col5           0.0000   -0.1359   -0.4255   -1.2091   -1.3020
x22            0.0000   -0.0522   -0.1635   -0.4645   -0.5002
x23            0.0000    0.0832    0.2606    0.7406    0.7975
x24            0.0000    0.2763    0.8650    2.4579    2.6467
```

The estimate of $\mathbf{A}$, which uses a corner constraint (7) in the second position and has a structural zero in the first position, is

```
> constraints(g1)[["x22"]]

         [,1]
[1,] 0.000000
[2,] 1.000000
[3,] 3.131003
[4,] 8.897212
[5,] 9.580658
```

and $\widehat{\mathbf{C}} = (0, 0.1393, 0.2740, 0.3144)^T$. From the output, it can be seen that $\hat{\mu} = 6.1651$, $\hat{\alpha}_2 = 0.6130$, $\hat{\gamma}_2 = 0.6128$ etc. These coefficients can be interpreted quite readily: adjusting for the row and column totals, as SES decreases, the number of arts students increases relative to commerce students. In contrast, the numbers of law and medical students decreases as SES decreases, relative to commerce students—this supports the widespread belief that medical students tend to come from high SES families.

The fitted values and a summary can be obtained by

```
> fitted(g1)

         A        B        C        D         E
a 4.003433 5.045548 5.312429 1.549676 1.0889144
b 5.360124 6.411754 6.040095 1.303917 0.8841101
c 5.084007 6.963610 8.755083 4.127346 3.0699538
d 1.552436 2.579088 4.892393 7.019061 5.9570217

> print(summary(g1))

Call:
rrvglm(formula = y ~ Row2 + Row3 + Row4 + Col2 + Col3 + Col4 +
    Col5 + x22 + x23 + x24, family = poissonff, constraints = cms,
    Rank = 1, Structural.zero = 1, Norrr = ~Row2 + Row3 + Row4 +
        Row5 + Col2 + Col3 + Col4)

Pearson Residuals:
   log(E[A]) log(E[B]) log(E[C])  log(E[D]) log(E[E])
a -0.0017155   0.42491 -0.569415  0.3617464 -0.085207
b  0.2763812  -0.55753  0.390577 -0.2661520  0.123252
c -0.4807609   0.39274  0.082773 -0.0626829 -0.039925
d  0.3592096  -0.36059  0.048650 -0.0071946  0.017609

Coefficients:
                Value Std. Error  t value
I(lv.mat):1  3.131002    9.63990  0.32480
I(lv.mat):2  8.897214   29.56469  0.30094
I(lv.mat):3  9.580653   32.03989  0.29902
(Intercept)  1.387152    0.38890  3.56687
Row2         0.291835    0.47707  0.61173
Row3         0.238948    0.46765  0.51096
Row4        -0.947327    0.79936 -1.18511
Col2         0.367250    0.44920  0.81756
Col3         0.708387    0.35165  2.01449
Col4         0.259987    0.39803  0.65319
Col5        -0.135896    0.45662 -0.29761
x22         -0.052208    0.25010 -0.20875
x23          0.083244    0.30787  0.27038
x24          0.276256    0.96495  0.28629

Number of linear predictors:  5
```

```
Names of linear predictors:
log(E[A]), log(E[B]), log(E[C]), log(E[D]), log(E[E])

(Default) Dispersion Parameter for poissonff family:   1

Residual Deviance: 1.96991 on 6 degrees of freedom

Log-likelihood: -32.70309 on 6 degrees of freedom

Number of Iterations: 8
```

Finally, the function `biplot()` returns quantities such the estimated $\mathbf{A}$ and $\mathbf{C}$, which can be multiplied to obtain $\widehat{\boldsymbol{\Delta}}$.

```
> i = Coef(g1)
> rbind(x21 = 0, i@C) %*% t(i@A)
```

```
     log(E[A])   log(E[B])  log(E[C])  log(E[D])  log(E[E])
x21          0  0.00000000  0.0000000  0.0000000  0.0000000
Col5         0 -0.13589577 -0.4254901 -1.2090935 -1.3019709
x22          0 -0.05220831 -0.1634644 -0.4645084 -0.5001899
x23          0  0.08324414  0.2606376  0.7406408  0.7975336
x24          0  0.27625619  0.8649590  2.4579100  2.6467160
```

Because of all the setting up that is required, it is no wonder that a function `grc()` has been written! As seen in Section 5.2, it operates on a general matrix of counts by automatically setting up all the indicator variables and constraint matrices, and allows for different parameterizations of the row and column scores, e.g., $\sum_i \alpha_i = \sum_j \gamma_j = 0$.

For a class of models called generalized additive main effects and multiplicative ineraction effects models (GAMMI) see van Eeuwijk (1995).

5.5 Other Reduced-Rank Regression work

Recently, other work has extended reduced-rank regression (RRR) outside the Gaussian family. For example, Fiocco et al. (2005) extend RRR to survival models, and Heinen and Rengifo (2005) compare RR-VGLMs with RRR models based on distributions in the multivariate dispersion models framework. They call theirs RR-MDM.

6 Quadratic RR-VGLMs for CQO

Quadratic RR-VGLMs (QRR-VGLMs) are useful in ecology because they allow symmetric bell-shaped response curves/surfaces to be fitted to species' data. Such curves/surfaces are a tenet in community ecology. The result of a QRR-VGLM is a *constrained quadratic ordination* (CQO; formerly called *canonical Gaussian ordination* or CGO by Yee (2004)). Given species data $\mathbf{Y}$ and environmental data $\mathbf{X} = (\mathbf{X}_1 \ \mathbf{X}_2)$, one estimates optimal linear combinations of the environmental variables $\boldsymbol{x}_2$ and regresses the species data upon these latent variables using quadratic forms. Here, $\mathbf{X}_2$ are the 'real' environmental variables one want to use in the ordination, but we want to do it after adjusting for the explanatory variables $\mathbf{X}_1$—which is usually just an intercept term.

Quadratic (partial) RR-VGLMs extend (6) by adding on a quadratic form in $\boldsymbol{\nu}$. The result can be written

$$\boldsymbol{\eta} \ = \ \mathbf{B}_1^T \boldsymbol{x}_1 + \mathbf{A}\,\mathbf{C}^T \boldsymbol{x}_2 + \sum_{j=1}^{M} \boldsymbol{e}_j\, \boldsymbol{\nu}^T\, \mathbf{D}_j\, \boldsymbol{\nu} \ = \ \mathbf{B}_1^T \boldsymbol{x}_1 + \mathbf{A}\, \boldsymbol{\nu} + \begin{pmatrix} \boldsymbol{\nu}^T \mathbf{D}_1 \boldsymbol{\nu} \\ \vdots \\ \boldsymbol{\nu}^T \mathbf{D}_M \boldsymbol{\nu} \end{pmatrix}, \tag{12}$$

where $\mathbf{D}_j$ are $R \times R$ symmetric matrices. The quadratic forms allow for bell-shaped (Gaussian) response surfaces. Indeed, the jth linear predictor in (12) is bell-shaped in the latent variable space if and only if $\mathbf{D}_j$ is negative-definite. Note that, in general,

$$\eta_j \ = \ \left\{ \frac{1}{2} \boldsymbol{u}_j^T \mathbf{T}_j^{-1} \boldsymbol{u}_j + \boldsymbol{\beta}_j^T \boldsymbol{x}_1 \right\} - \frac{1}{2} \left(\boldsymbol{\nu} - \boldsymbol{u}_j \right)^T \mathbf{T}_j^{-1} \left(\boldsymbol{\nu} - \boldsymbol{u}_j \right), \qquad j = 1, \ldots, M, \tag{13}$$

where $\boldsymbol{u}_j = \mathbf{T}_j\, \boldsymbol{a}_j$ is the *optimum* of Species j. The function η_j is bell-shaped if and only if $\mathbf{T}_j = -\frac{1}{2} \mathbf{D}_j^{-1}$ is positive-definite. We call the $\mathbf{T}_j$ *tolerance matrices*, which are a measure of niche width. When the $\mathbf{T}_j$ are diagonal they contain the squared tolerances. An important assumption is the *equal-tolerances assumption*:

$$\mathbf{T}_1 \ = \ \mathbf{T}_2 \ = \ \cdots \ = \ \mathbf{T}_S, \tag{14}$$

and this is referred to many times below.

Here's a simple and important example of a QRR-VGLM: Poisson data with M species. The rank-1 model for Species j is the Poisson regression

$$\begin{aligned} \log \ \mu_j(\nu) \ &= \ \eta_j(\nu) \ = \ \beta_{(j)1} + \beta_{(j)2}\, \nu + \beta_{(j)3}\, \nu^2 \\ &= \ \alpha_j - \frac{1}{2} \left(\frac{\nu - u_j}{t_j} \right)^2, \quad j = 1, \ldots, M, \end{aligned} \tag{15}$$

where $\nu = \boldsymbol{c}^T \boldsymbol{x}_2$ and $\mu_j = E(Y_j)$ is the expected value for species j, i.e., mean abundance or counts. In the bottom representation of (15), u_j is often called Species j's *optimum*, and t_j (> 0) its *tolerance*, a measure of niche width. The quantity $\mu_j(u_j)$ is referred to as the *maximum* of Species j; it is the maximum expected abundance/count at its optimum environment.

If we have binary (e.g., presence/absence) data and if the $\log$ parameter link in (15) is replaced by a logit link then the result is known as a *Gaussian logit model* for each species—but because the latent variable ν is a common regressor of all species, the result is a *constrained* or canonical Gaussian logit ordination model.

Fitting QRR-VGLMs is more difficult than RR-VGLMs for a number of reasons. Firstly, the log-likelihood may contain local maxima so that a local solution may be obtained instead

of the global solution. Thus you need to fit the model several times with different starting values to increase the chances of obtaining *the* solution. Consequently, if the solution does not look right, try fitting the model several more times and/or adjusting some arguments. The argument `Bestof` specifies how many different initial values are to be used and should be assigned a reasonable integer value (e.g., at least 10) to help find the global solution. Secondly, the estimation is more prone to numerical difficulties. Thirdly, it can be many times more numerically intensive compared to RR-VGLMs. For this reason `trace=TRUE` is the default for QRR-VGLMs; if `trace=FALSE` many users would think their computer had locked up because it was taking so long!

Standard errors for $\widehat{\mathbf{A}}$, $\widehat{\mathbf{C}}$, $\widehat{\mathbf{B}}_1$ and $\widehat{\mathbf{C}}$ are presently too difficult to compute for QRR-VGLMs. However, some time in the near future it is hoped that they will be available.

6.1 Normalizations for QRR-VGLMs

For QRR-VGLMs, it is more convenient to abandon corner constraints. Instead, if `ITolerances=FALSE` and `EqualTolerances=TRUE` then

$$\widehat{\mathrm{Var}}(\hat{\boldsymbol{\nu}}_i) \;=\; \mathbf{I}_R \tag{16}$$

is the default during the fitting. This results in uncorrelated latent variables. Uncorrelated latent variables are a good idea because they can be loosely be thought of as unrelated to each other. With two ordination axes of uncorrelated latent variables, one can think of the second axis as being unrelated to the first axis, which represents the dominant gradient.

Equation (16) also means the latent variables have unit variances. With this, two advantages can be realized:

1. the tolerances of each species can be compared to unity, which is the amount of variability (measured by the standard deviation) in the data set with respect to each latent variable,

2. the algorithm is more numerically stable.

Furthermore, it is possible to rotate the solution so that at least one of the species has a diagonal tolerance matrix. For this (lucky) species, the interpretation is particularly nice: the effects of the latent variables are uncorrelated/independent on that species. The species chosen for this must be bell-shaped. The argument is `reference`, and applies to `Coef()` and `lvplot()` and other generic functions. See Section 6.3.1 for details.

When `ITolerances=TRUE`, (16) is relaxed to being just a diagonal matrix and $\mathbf{T}_s = \mathbf{I}_R$ for all species s.

6.2 Fitting QRR-VGLMs with VGAM

QRR-VGLMs are fitted using `cqo(...)` and the object returned has class `"qrrvglm"`. Currently, only `FastAlgorithm=TRUE` is supported—it uses a new algorithm described in Yee (2006a) and Yee (2005). In contrast, the slow algorithm described in Yee (2004) has been withdrawn.

Currently only two distributions (binomial and Poisson) are supported in VGAM family functions for QRR-VGLMs. They are `poissonff()` and `binomialff(mv=TRUE)`, including their "quasi" versions, `quasipoissonff()` and `quasibinomialff(mv=TRUE)`. The "`mv`" argument stands for "multivariate" and tells the binomial family function that the response is multivariate, i.e., comes from S species. This is necessary to retain upward compatability, e.g., `binomialff()` with a 2-column matrix response is interpreted as a matrix of sucesses and

failures rather than two species. If `mv=TRUE` is set then the response (matrix) must contain 0s and 1s only. The Poisson family functions handle multivariate responses automatically. In the case where one wishes to examine whether the data is overdispersed or underdispersed, the family functions `quasipoissonff()` and `quasibinomialff(mv=TRUE)` can be used.

Once a QRR-VGLM has been fitted, the useful generic functions listed in Table 5 can be applied to the fit.

Table 5: Methods functions for CQO and CAO objects in the VGAM package. Future modifications and additions are likely.

S function	Purpose
`Coef()`	$\widehat{\mathbf{A}}$, $\widehat{\mathbf{B}}_1$, $\widehat{\mathbf{C}}$, $\widehat{\mathbf{D}}$, $\widehat{\boldsymbol{u}}_s$, $\widehat{\mathbf{T}}_s$, $\widehat{\boldsymbol{\nu}}_i$, etc.
`ccoef()`	Canonical coefficients $\widehat{\mathbf{C}}$
`is.bell()`	Are the species' response curves/surfaces bell-shaped?
`lv()`	Latent variables $\widehat{\boldsymbol{\nu}}_i = \widehat{\mathbf{C}}^T \boldsymbol{x}_{2i}$ (site scores)
`Max()`	Maxima $E[Y_s\|\widehat{\boldsymbol{u}}_s] = g^{-1}(\widehat{\alpha}_s)$
`Opt()`	Optima $\widehat{\boldsymbol{u}}_s$ (species scores)
`Tol()`	Tolerances $\widehat{\mathbf{T}}_s$
`lvplot()`	Latent variable plot (ordination diagram; for $R = 1$ or 2)
`persp()`	Perspective plot (for $R = 1$ or 2)
`trplot()`	Trajectory plot (for $R = 1$ only)
`calibrate()`	Calibration: estimate $\boldsymbol{\nu}$ from $\boldsymbol{y}$
`predict()`	Prediction: estimate $\boldsymbol{y}$ from $\boldsymbol{x}$
`resid()`	Residuals (e.g., working, response, ...)
`summary()`	Summary of the object

6.2.1 Initial Values

Initial values require some comment. Appendix B of Yee (2005) proposed an efficient method to obtain an initial $\mathbf{C}$ based on an equal-tolerances Poisson model. It is the default because `Use.Init.Poisson.QO=TRUE`. The user can bypass this by assigning the `Cinit` argument a $p_2 \times R$ matrix. It is recommended that the elements of `Cinit` be close to zero because, like neural networks, large weights often lead to poor solutions. If `Use.Init.Poisson.QO=FALSE` and `Cinit` is not assigned a value then VGAM will choose some random normal variates. Users should use `set.seed()` with different seeds before fitting the same model and thus try to ensure the global solution is obtained.

The solution of a rank-R QRR-VGLM is not nested within a lower rank model. An idea is therefore to use as initial values, $\mathbf{C}_R^0 = (\widehat{\mathbf{C}}_{R-1}, \boldsymbol{\varepsilon})$ where $\boldsymbol{\varepsilon} \sim N_{p_2}(\mathbf{0}, \sigma^2 \mathbf{I}_{p_2})$, $\sigma \approx 0$. Another idea is to fit only R species first, and then fit them all using the estimate of $\mathbf{C}$ as initial values.

6.2.2 Some Tricks and Advice

CQO models are computationally expensive and prone to numerical difficulties. To help with these problems, Yee (2004) suggest several ideas such as

1. omitting some species (e.g., rare ones, those with small tolerances)

2. setting `EqualTolerances=FALSE` and `ITolerances=FALSE`,

3. omitting some environmental variables from the analysis,

4. and choosing good `Cinit` if `Use.Init.Poisson.QO()` results in poor initial values.

Other ideas are:

1. If `ITolerances=TRUE` then careful choice of values for `isdlv` is important. Note that setting `ITolerances=TRUE` is currently the fastest of all algorithm settings.

2. Initially work with a simple random sample of the sites to cut down on the computation.

3. It is a good idea to fit both an equal-tolerances and unequal-tolerances model and compare them. When $R = 2$, the ordination diagram of the equal-tolerances model is more easily interpreted because elliptical contours are required for the unequal-tolerances model—otherwise it would be susceptible to misinterpretation. See Section 6.3.2 for more details.

6.2.3 Timings

To give some idea about the speed of CQO under different options, Table 6 gives the timings of some models fitted to simulated data (see Appendix A of Yee (2006a)). Here, $\boldsymbol{x}_1 = 1$. The timings are optimistic in that the data and model coincide and there are no outliers etc. In practice, the timings on real data would be higher.

For fixed R, S and p_2, it can be seen that the fastest is `ITolerances=TRUE`, followed by `EqualTolerances=FALSE` (the default), followed by `EqualTolerances=TRUE` (times in parentheses ()). The RR-MLM is one method for 'fitting' the equal-tolerances Poisson model, and seems more expensive than `EqualTolerances=FALSE` models. However, RR-MLMs only give an approximation to $\mathbf{C}$ and do not provide an estimate of the common tolerance matrix.

Table 6: Speed tests with the fast CQO algorithm. Some of the combinations compare with Yee (2004)—in brackets [ ]. All times are in seconds, and used `Use.Init.Poisson.QO=TRUE` to obtain initial values. The well-conditioned simulated Poisson data were fitted using R 2.2.0 on a 2.4GHz Pentium 4 machine running Linux. The average times of 10 fits are given. "Unequal" and "Equal" refer to the species' tolerances; "Equal" were fitted using `ITolerances=TRUE`. Values in parentheses () are times given in Appendix A of Yee (2006a)—they have `EqualTolerances=TRUE`. Values in brackets [ ] are times using the slow Yee (2004) algorithm.

R	n	S	p_2	Time for new CQO algorithm		Time for RR-MLM
				Unequal	Equal	
1	500	10	10	9 [117]	3 (19) [89]	6
		20	10	11	4 (132)	21
		100	10	35	20	
	1000	10	10	8	3 (53)	9
		20	10	30	8 (253)	39
		100	10	125	46	
2	500	10	10	10	3 (67)	6
		20	10	25	12 (599)	23
		100	10	143	36	
	1000	10	10	25	8 (435)	20
		20	10	44	20 (2279)	68
		100	10	223	62	

6.2.4 Arguments ITolerances and EqualTolerances

For QRR-VGLMs, an important argument in `qrrvglm.control()` is `EqualTolerances`, which can be assigned `TRUE` or `FALSE`. If `TRUE`, an equal-tolerances model $\mathbf{T}_1 = \cdots = \mathbf{T}_S$ is fitted. Another related argument is `ITolerances` which can have the same effect. However, their differences should be understood. Choosing between an equal-tolerances and unequal-tolerances model is a tradeoff between interpretability and quality of fit. In real life, it is unrealistic assumption. But then it can be argued that bell-shaped curves/surfaces are an unrealistic assumption too. Certainly for $R = 2$, equal-tolerances makes interpretation much easier because elliptical contours are not required. With $R = 1$ one doesn't need an equal-tolerances assumption so much because you can gauge how large the tolerances are by applying a function such as `persp()`.

So how are the arguments `ITolerances` and `EqualTolerances` related? And how are algorithms for fitting the models affected by these? The answers are given in Table 7. The argument `EqualTolerances` refers to whether $\mathbf{T}_s = \mathbf{T}$ for all $s = 1, \ldots, S$, for some order-R matrix $\mathbf{T}$. Note that $\mathbf{T}$ may or may not be positive-definite; ideally it is. In contrast, the argument `ITolerances` is more directed at specifying the algorithm, and if `TRUE`, offsets (GLM jargon) of the form $-\frac{1}{2}\nu_{ir}^2$ are used in the algorithm because $\mathbf{T}_s = \mathbf{I}_R$ by definition. Note that setting `ITolerances=TRUE` *forces* bell-shaped curves/surfaces on the data regardless of whether this is appropriate or not. Having `ITolerances=TRUE` implies `EqualTolerances=TRUE` but not vice versa.

Computationally, any offset values which are large will cause numerical problems. Therefore it is highly recommended that all numerical variables (i.e., all but factors) in $\boldsymbol{x}_2$ be standardized to mean 0 and unit variance. This is because we want the site scores be centered at 0, and this is ensured if the $\boldsymbol{x}_2$ have mean $\mathbf{0}$:

$$E[\boldsymbol{\nu}] \;=\; E[\mathbf{C}^T\boldsymbol{x}_2] \;=\; \mathbf{C}^T E[\boldsymbol{x}_2] \;=\; \mathbf{C}^T\mathbf{0} \;=\; \mathbf{0}.$$

Standardizing variables can be achieved with `scale()`, hence something like

```
cqo(cbind(spp1,spp2,spp3,spp4) ~ scale(temperature) + scale(rainfall) +
    scale(lognitrogen), fam = poissonff, data=mydata, ITolerances=TRUE)
```

is a good idea for count data.

In practice, setting `ITolerances=TRUE` is the recommended way of fitting an equal-tolerance model because they are computed very efficiently. Each species can be fitted separately and the number of parameters is low, for example, with $R = 1$ there are 2 parameters per species, and for $R = 2$ there are 3 parameters per species (In general, there are $R + 1$ parameters for a rank R problem). This contrasts with `EqualTolerances=FALSE` where there are 3 and 6 parameters respectively for $R = 1$ and 2. Hence, fitting a rank-2 equal-tolerances model is usually faster than a rank-1 unequal-tolerances model. See Table 6 for some timings.

However, setting `ITolerances=TRUE` may fail on some data sets because of numerical problems, so the next best thing to do is to leave `ITolerances=FALSE` alone and only set `EqualTolerances=TRUE`. This will result in a different algorithm being used, which will usually be be a lot slower but there is less risk of numerical problems.

6.2.5 The isdlv argument

The `isdlv` argument specifies the initial standard deviation of the latent variable values $\boldsymbol{\nu}_i$ (site scores). It is used only if `ITolerances=TRUE` so that all species tolerances are unity, and the initial sites scores are scaled to a hopefully reasonable spread relative to these response curves.

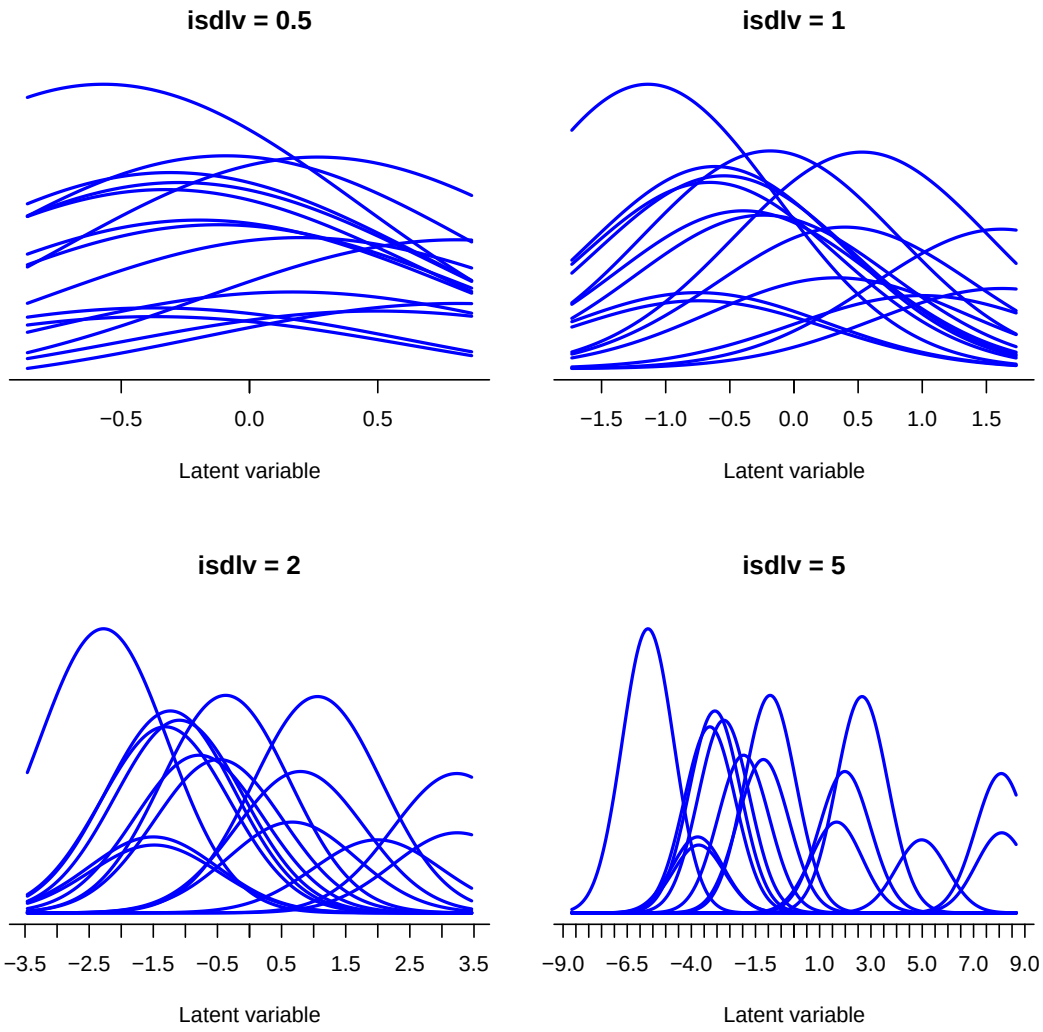

Figure 3: The effect of the argument `isdlv` on the site scores. All response curves have unit tolerances (because $\mathbf{T}_s = \mathbf{I}_R$), and the optima are located the same relative distance from each other. The site scores are uniformly distributed over the latent variable space and have been scaled to have a standard deviation `isdlv`. The tick marks are at the same values.

Table 7: The relationship between the arguments `EqualTolerances` and `ITolerances`. It is assumed that the constraint matrices of all $\boldsymbol{x}_1$ variables is $\mathbf{I}_S$. "Separately" means each species can be fitted separately, otherwise "Jointly" means fitting one big model involving all the species. The index s $(= 1, \ldots, S)$ indexes species. The significance of the three cases is discussed in Section 6.3.1.

	`EqualTolerances=TRUE`	`EqualTolerances=FALSE`
`ITolerances=TRUE`	Separately. $\mathbf{T}_s = \mathbf{I}_R$. Case 1	Error message.
`ITolerances=FALSE`	Jointly. $\mathbf{T}_s = \mathbf{T}$ but may not be positive-definite. Case 2	Separately. $\mathbf{T}_s$ unequal and may not be positive-definite. Case 3

The effect of several values of `isdlv` is illustrated in Figure 3. It can be seen that as `isdlv` increases, the range of the sites scores increases relative to species' tolerances. That is, there is more environmental range in the data as `isdlv` increases.

In practice, the `isdlv` values should be set to lie between 0.5 to 10 (approximately). The argument should actually be of length R and is recycled to this length if necessary. Each successive value should be less than the previous one, e.g., `c(2,1)` might be appropriate for a rank-2 problem. This is because the first ordination axis should have the greatest spread of site scores. Each successive ordination axis will have less explanatory power compared to previous axis, hence a decreasing `isdlv` sequence. If convergence failure occurs, try varying this argument somewhat, e.g., `isdlv=c(5,2)` or `isdlv=c(0.8,0.6)`, because good initial values are usually needed for CQO. Big data sets with a lot of species collected over a wide range of environments should warrant larger values of `isdlv`.

A related argument is `MUXfactor`. If any offset value are greater than `MUXfactor * isdlv[r]` in absolute value then the rth ordination axis is scaled so that the standard deviation of the site scores is `MUXfactor * isdlv[r]`. This is why it is a good idea for the site scores to be centred at 0—and this can be achieved if all the variables in $\boldsymbol{x}_2$ are centred at 0. The reason for `MUXfactor` is that `optim()` may perform a linesearch at a value of $\mathbf{C}$ that gives a very large spread of site scores. If values are too large then numerical difficulties will occur. Usually a value of `MUXfactor` between 3 or 4 should be ok. If not, decrease the value slightly.

6.3 After Fitting a QRR-VGLM

This section looks at what can be done after a QRR-VGLM is fitted. These include numerical summaries and plotting. There are several types of plots available and these are described below. However, it is first necessary to discuss the three cases of CQO fits because a clear understanding is necessary when using the CQO generic/methods functions.

6.3.1 The Three Cases

Table 7 defines the three cases of CQO fits. These are important because generic/methods functions such as `Coef()`, `Tol()`, `lvplot()` etc. (see Table 5) treat the three cases differently by default. In this section we look at each case separately with respect to the two arguments `varlvI=FALSE` and `reference=NULL` which the generic/methods functions have as defaults.

The `varlvI` argument specifies whether

$$\widehat{\mathrm{Var}}(\widehat{\boldsymbol{\nu}}_i) \;=\; \mathbf{I}_R \tag{17}$$

or not. If `TRUE` then the site scores are uncorrelated and have a standard deviation along each ordination axis equal to unity. With this option, species' tolerances can be compared with the amount of variability of the data set. If `varlvI=FALSE` then the site scores are uncorrelated but each ordination axis has a different standard deviation of sites scores. The ordination axes are sorted so that the standard deviation of sites scores never goes up. In other word, $\widehat{\mathrm{Var}}(\widehat{\boldsymbol{\nu}}_i)$ is always a diagonal matrix, and the elements along the diagonal are either all 1s or a decreasing sequence.

The argument `reference` specifies which species is to be chosen as the *reference species*. The reference species, which only makes a difference in Case 3, could be chosen as the dominant species or some species that all other species can be compared with. The default value of `NULL` means the software searches from the first to the last species until one with a positive-definite tolerance matrix is found. Alternatively, the user can circumvent this searching procedure by specifying the reference species, e.g., by giving its column number. This reference species needs to have a positive-definite tolerance matrix, which is only assured in Case 1.

For all three cases, the general algorithm is:

1. find a reference species if necessary, then
2. if possible, transform it so that its tolerance matrix is $\mathbf{I}_R$, then
3. transform the site scores to be uncorrelated (i.e., $\widehat{\mathrm{Var}}(\widehat{\boldsymbol{\nu}}_i)$ is diagonal), and
4. if `varlvI=TRUE` then scale the ordination axes so that the standard deviation of the site scores are unity (i.e., (17)).

Let's look at what the general algorithm does in the three individual cases.

Case 1 This is the nicest of all cases. All tolerance matrices are equal and positive-definite because they are all $\mathbf{I}_R$. The general algorithm gives a unique solution, and the first ordination axis has the greatest spread of the sites scores (as measured by the standard deviation), followed by the second ordination axis, etc. That is, (17) does not hold—the matrix is diagonal only. The general algorithm results in an ordination diagram where distances have their intuitive meanings. If the second ordination axis has a small standard deviation of sites scores relative to the unit tolerance and the first axis' standard deviation of sites scores then this suggests that a rank-1 ordination should suffice.

Case 2 This case is equivalent to Case 1 if the (common) estimated tolerance matrix is positive-definite. Ideally this is so. If not, then the general algorithm will only return uncorrelated ordination axes. If `varlvI=TRUE` then (17) will hold.

Case 3 This case is the most arbitrary. Each species has its own tolerance matrix which may or may not be positive-definite. The reference species ends up with a $\mathbf{I}_R$ tolerance matrix, and the sites scores are uncorrelated but have a different standard deviation along each ordination axis. Choosing the reference species does make a difference for $R = 2$: the ordination diagram is rotated so that the elliptical contours of the reference species has semi-major and semi-minor axes parallel to the ordination axes. Other species will generally have semi-major and semi-minor axes not parallel to the ordination axes.

A species whose tolerance matrix is not positive-definite will not have an optimum or maximum.

6.3.2 Latent variable plots

"Latent variable plots" are another name for "ordination diagram" and are implemented by the generic function `lvplot()`. For a rank-2 equal-tolerances model, the default output of `lvplot()` will give a 'natural' latent variable plot (aka ordination diagram) in the sense that distances between points will be subject to their intuitive interpretation (the closer they are, the more similar). This is because the contours of the ellipses are scaled so that they are circular. Consequently, $\widehat{\mathrm{Var}}(\widehat{\boldsymbol{\nu}}_i)$ will be diagonal. In order for the latent variable plot to look accurate, the sides of the graph must be scaled so that the circular contours do actually appear circular. On a computer screen, this is easy since it simply entails resizing the graphics window using the mouse. If `EqualTolerances=TRUE` then these latent variable plots are computed by rotating the species so that their tolerance matrices are diagonal, and then the canonical axes are stretched/shrunken so that the estimated $\mathbf{T}_j$ are now $\mathbf{I}_R$.

One of the beauties about CQO is that a lot of information is available from the fit. For example, consider (15). If `ITolerances=TRUE` in a Poisson CQO then the estimated mean abundance one tolerance unit away from the optimum is about 40 percent less than the species' maximum. This comes about by (24) (see Exercises) and

```
> exp(-0.5 * (1:3)^2)

[1] 0.60653066 0.13533528 0.01110900
```

Yee (2004) gives several more examples.

If `EqualTolerances=FALSE` is used for a rank-2 model, then it is necessary to interpret the latent variable plot with reference to the elliptical contours. You have to compute Mahalanobis distances in your head to correctly interpret distances!

6.3.3 Perspective Plots

For rank-2 models with $\boldsymbol{x}_1 = 1$, the response surface of any subset of the species can be plotted using the generic function `persp()`.

For rank-1 models with $\boldsymbol{x}_1 = 1$, `persp()` will produce a plot similar to `lvplot()` but with the fitted curves smoothed out. The choice between `lvplot()` and `persp()` then depends on the purpose of plotting them; `lvplot()` is more 'closer' to the data set while `persp()` can be wrongly interpreted as the "truth".

6.3.4 Trajectory plots

These are suitable for rank-1 models and plot the fitted values of pairwise combinations of species. If S is large then it is wise to select only a few species to plot. A log scale on both axes is often more effective. See Wartenberg et al. (1987) for an example.

6.4 Negative Binomial and Gamma Data

Currently `cqo()` can handle the `negbinomial()` and `gamma2()` family functions. However these 2-parameter distributions can be more difficult to fit due to inherent numerical problems. There are one or two tricks described below to help alleviate these problems.

The negative binomial distribution is actually more relistic in practice than the Poisson distribution for the reason that some people maintain overdispersion is more often the norm rather than the exception (pp.124–125 of McCullagh and Nelder (1989)).

Fitting CQO negative binomial CQO models can be done with a trick or two. Firstly, having `ITolerances=TRUE` is *not* recommended because it is prone to numerical difficulties. Instead, set `ITolerances=FALSE` and leave `EqualTolerances=TRUE`. This fits the same model as with `ITolerances=TRUE` but using a more stable and slower algorithm.

The second trick involves transformation. With data that is negative binomial the trick is to take the square root and fit a Poisson ordination to it. The justification is the formula

$$\mathrm{Var}(g(Y)) \;\approx\; [g'(\mu)]^2 \,\mathrm{Var}(g(Y)) \tag{18}$$

by a Taylor series expansion. Then

$$\mathrm{Var}\left(Y^{\frac{1}{2}}\right) \;\approx\; c_0 + c_1\mu$$

where $c_0 = \frac{1}{4}$ and $c_1 = \frac{1}{4k}$. This is at least proportional to μ. Or better,

$$\mathrm{Var}\left(2\sqrt{k}Y^{\frac{1}{2}}\right) \;\approx\; k + \mu$$

which is close to μ if $k \approx 0$. See Section 6.9 for an example.

If the species data has a 2-parameter gamma distribution then following the argument behind (18) one has $\mathrm{Var}\left(\log(Y)\right) \;\approx$ a constant. Thus it is possible to fit the model with normal errors, i.e., Gaussian errors. The way to simulate this is something like

```
> n = 200
> p = 5
> S = 3
> mydata = rcqo(n, p, S, fam = "gamma2", ESOpt = TRUE, Log = TRUE,
+     seed = 123)
> myform = attr(mydata, "formula")
> fit5 = cqo(myform, fam = gaussianff, ITol = TRUE, dat = mydata,
+     trace = FALSE)
```

6.5 A Summary

Fitting CQO models well require a substantial amount of experience and an understanding of the mathematics behind the models. It is best to gain experience with simulated data first because the "truth" is known.

Because there is quite a lot involved with CQO modelling, here is a brief summary of things to watch out for. Of course, the online help needs to be read carefully too.

1. An equal-tolerances model can be fitted in two ways: setting `ITolerances=TRUE` or `EqualTolerances=TRUE`. Try `ITolerances=TRUE` first because it is much faster—however it may fail to converge unless good values of `isdlv` and `MUXfactor` are chosen. Setting `ITolerances=TRUE` forces bell-shaped curves/surfaces on to all species ($T_s = \mathbf{I}_R$) regardless of whether this is appropriate. If `ITolerances=TRUE` fails after many attempts, then try `EqualTolerances=TRUE` because it is more numerically stable—however, $\widehat{\mathbf{T}}_s$ may not be positive-definite.

2. If `ITolerances=TRUE` then $\mathbf{T}_s = \mathbf{I}_R$ for all s, and $\widehat{\mathrm{Var}}(\widehat{\boldsymbol{\nu}}_i)$ is only constrained to be diagonal. Hence the latent variables are uncorrelated and the tolerances are all unity. Also, the diagonal elements of $\widehat{\mathrm{Var}}(\widehat{\boldsymbol{\nu}}_i)$ will be a decreasing sequence, therefore the first ordination axis contains the greatest spread followed by the second ordination axis, etc. This means the ordination axes are sorted by their dominance or importance.

3. For an `ITolerances=TRUE` model, it pays to standardize all the $\boldsymbol{x}_2$ variables apart from factors (grouping variables). This is to avoid numerical problems that can arise from the fast algorithm. The function `scale()` can be used on each variable in the formula, or can be used on the $\mathbf{X}_2$ matrix as a whole.

4. A RR-MLM (see Section 2.5) is an alternative method for fitting an equal-tolerances Poisson CQO. Although $\mathbf{T}_s$ is not available from this, the constrained coefficients $\widehat{\mathbf{C}}$ should be ok. These can be fed into `cqo()` as initial values—the argument is `Cinit`. Further details can be found in Appendix C of Yee (2006a).

5. CAO (see Section 8) can help fit better CQO models. It is a good idea to fit a CAO first to see what the response curves look like. Species whose $\widehat{f}_s$ are very different from a quadratic may be omitted from the CQO analysis or transformations on the $\boldsymbol{x}_2$ variables sought to improve the appropriateness of CQO. For example, if the first variable in $\boldsymbol{x}_2$ is nitrogen concentration, it would probably be much better to use the logarithm of nitrogen concentration instead (and then standardize it).

6. It's a good idea to fit a rank-1 model first, followed by a rank-2 model. The rank-1 model is to be preferred if it looks reasonable and the rank-2 model is very difficult to fit and there are many non-positive-definite tolerance matrices etc.

7. Numerical problems often are indicative that the model and data do not agree. Trying to fit bell-shaped curves/surfaces to data that isn't is like banging your head against a brick wall. It pays to check your data first to see whether a CQO model is reasonable in the first case, e.g., by applying a CAO model.

6.6 CQO Examples

Here are some basic examples illustrating the use of VGAM for fitting CQO models.

6.7 Example 1

Here's a simple rank-1 example in R involving simulated data. Note that the species' tolerances are unequal.

```
> set.seed(123)
> n = 100
> mysim = data.frame(x2 = rnorm(n), x3 = rnorm(n), x4 = rnorm(n))
> mysim = transform(mysim, lv1 = 0 + x3 - 2 * x4)
> mysim = transform(mysim, lambda1 = exp(3 - 0.5 * (lv1 -
+     0)^2), lambda2 = exp(2 - 0.5 * (lv1 - 1)^2), lambda3 = exp(2 -
+     0.5 * ((lv1 - (-4))/2)^2))
> mysim = transform(mysim, y1 = rpois(n, lambda1), y2 = rpois(n,
+     lambda2), y3 = rpois(n, lambda3))
> set.seed(111)
```

```
> p1 = cqo(cbind(y1, y2, y3) ~ x2 + x3 + x4, poissonff, ITol = FALSE,
+     data = mysim, trace = FALSE)
> c1 = Coef(p1)
```

Then

```
> deviance(p1)

[1] 240.5971

> is.bell(c1)

  y1   y2   y3
TRUE TRUE TRUE

> c1

C matrix (constrained/canonical coefficients)
            lv
x2  0.03332381
x3  1.06296508
x4 -2.07083501

B1 and A matrices
            (Intercept)          A
log(E[y1])  3.011572405 -0.0470535
log(E[y2])  1.547553364  0.8977449
log(E[y3]) -0.005169979 -1.0058999

Optima and maxima
      Optimum   Maximum
y1 -0.0470535 20.341831
y2  0.9381768  7.161182
y3 -3.9782596  7.357318

Tolerance
         lv
y1 1.000000
y2 1.022271
y3 1.988700

Standard deviation of the latent variables (site scores)
      lv
2.194405
```

Not surprising, the model fits the data well, e.g.,

- $\hat{u}_1$ is sandwiched between $\hat{u}_2$ and $\hat{u}_3$, and is about four times closer to $\hat{u}_2$ than $\hat{u}_3$,
- the estimated tolerances look correct etc., i.e., $\hat{t}_3$ is approximately twice the other two.
- all species have bell-shaped curves, and Species 1 is the most abundant at its optimum (Nb. $e^3 \approx 20$)

```
> sj = ncol(p1@y)
> lvplot(p1, plot = TRUE, type = "f", lcol = 1:sj, llwd = 3,
+     llty = 1, y = TRUE, pch = 1:sj, pcol = 1:sj, las = 1,
+     main = "Simulated data")
> abline(v = Opt(p1), lty = 2, col = 1:sj, lwd = 2)
> abline(h = Max(p1), lty = 2, col = 1:sj, lwd = 2)
```

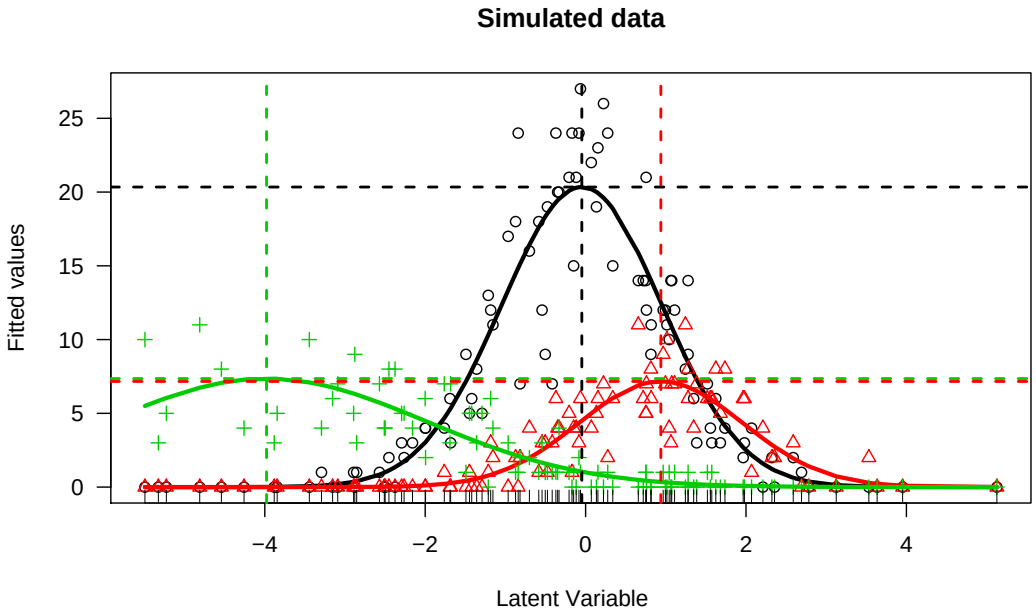

Figure 4: Latent variable plot for simulated Poisson data. The horizontal and vertical lines denote the estimated optima and maxima respectively.

- the first coefficient of $\widehat{\mathbf{C}}$ is very small compared to the others.

Now a latent variable plot can be obtained—see Figure 4.

Finally,

```
> print(summary(p1, disper = 0))

Call:
cqo(formula = cbind(y1, y2, y3) ~ x2 + x3 + x4, family = poissonff,
    data = mysim, ITol = FALSE, trace = FALSE)

C matrix (constrained/canonical coefficients)
            lv
x2  0.03332381
x3  1.06296508
x4 -2.07083501

B1 and A matrices
            (Intercept)          A
log(E[y1])  3.011572405 -0.0470535
log(E[y2])  1.547553364  0.8977449
```

```
log(E[y3]) -0.005169979 -1.0058999

Optima and maxima
      Optimum   Maximum
y1 -0.0470535 20.341831
y2  0.9381768  7.161182
y3 -3.9782596  7.357318

Tolerance
         lv
y1 1.000000
y2 1.022271
y3 1.988700

Standard deviation of the latent variables (site scores)
      lv
2.194405

Dispersion parameter:   1
```

so that $\hat{\phi} = 0.81 \approx 1$. Note the two warnings: we emphasize again here that these standard errors are biased downward because they are based on the assumption that $\widehat{\mathrm{C}}$ is known.

6.8 Example 2

Rather than manually generating a data set as in Section 6.7, it is more convenient to use the function `rcqo()`. In particular, data from a species packing model can be generated. Here is an example.

```
> n = 200
> p = 5
> S = 5
> mydata = rcqo(n, p, S, fam = "binomial", hiabundance = 5,
+     seed = 123, EqualTol = TRUE, ESOpt = TRUE, EqualMax = TRUE)
> myform = attr(mydata, "formula")
> b1 = cqo(myform, fam = binomialff(mv = TRUE), data = mydata,
+     trace = FALSE)
```

The `hiabundance` argument has been given a low value so that the maximum of each species' probability of presence is not too close to 1. Then

```
> sort(b1@misc$deviance.Bestof)

 [1] 533.7351 533.7351 533.7352 533.7357 533.7357 533.7360 533.7370
 [8] 533.7370 533.7389 533.7389
```

gives a history of all the iterations. Now a latent variable plot can be obtained—see Figure 5.

Let's compare the constrained coefficients of the fitted model with the 'truth':

```
> cbind(truth = attr(mydata, "ccoefficients"), fitted = ccoef(b1))
```

```
> lvplot(b1, y = TRUE, lcol = 1:S, pch = 1:S, pcol = 1:S,
+     las = 1)
```

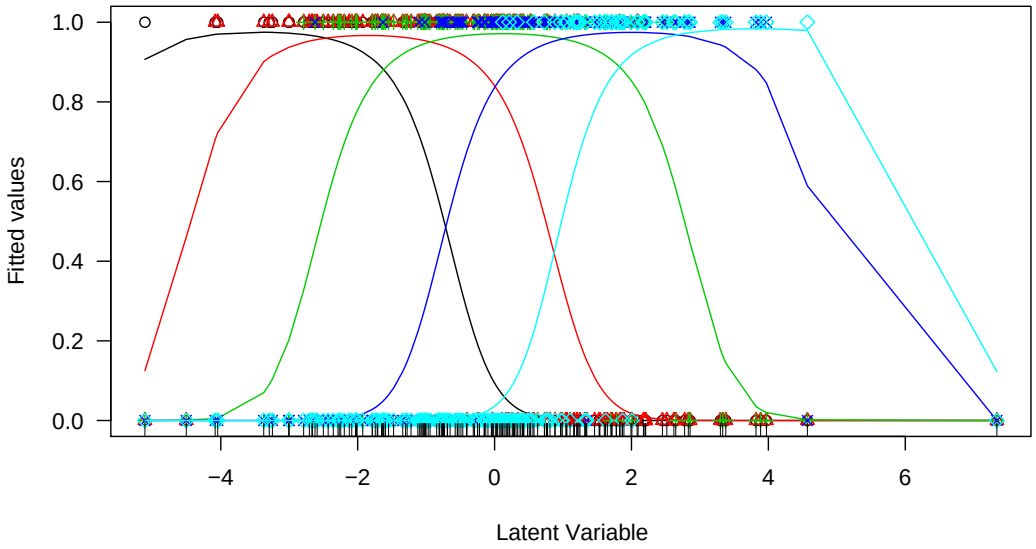

Figure 5: Latent variable plot for simulated binomial data.

```
           lv         lv
x2  0.3241401  0.3684269
x3 -0.5986456 -0.7075643
x4  0.7780473  1.0235454
x5  1.0489203  1.1602155
```

It can be seen that they are similar. Similarly,

```
> cbind(truth = attr(mydata, "optima"), fitted = t(Opt(b1)))
```

```
          lv         lv
y1 -2.846050 -3.4432761
y2 -1.423025 -1.8481410
y3  0.000000  0.1224865
y4  1.423025  2.0075783
y5  2.846050  3.8493058
```

Once again they are similar. All species have unit tolerances:

```
> Coef(b1)
```

```
C matrix (constrained/canonical coefficients)
           lv
x2  0.3684269
x3 -0.7075643
x4  1.0235454
```

```
x5  1.1602155

B1 and A matrices
             (Intercept)          A
logit(E[y1])   -2.265935 -3.4432761
logit(E[y2])    1.661390 -1.8481410
logit(E[y3])    3.511222  0.1224865
logit(E[y4])    1.628489  2.0075783
logit(E[y5])   -3.288104  3.8493058

Optima and maxima
      Optimum   Maximum
y1 -3.4432761 0.9749653
y2 -1.8481410 0.9667281
y3  0.1224865 0.9712158
y4  2.0075783 0.9745106
y5  3.8493058 0.9840226

Tolerance
   lv
y1  1
y2  1
y3  1
y4  1
y5  1

Standard deviation of the latent variables (site scores)
      lv
1.780794

> attr(mydata, "tolerances")

   lv
y1  1
y2  1
y3  1
y4  1
y5  1
```

Lastly, each species has equal maxima (on the Poisson counts scale), therefore the output from

```
> Max(b1)

       y1        y2        y3        y4        y5
0.9749653 0.9667281 0.9712158 0.9745106 0.9840226
```

should also give values that are equal. With binary data this is a little awkward because the values are so close to 1 anyway.

6.9 Example 3

The following shows data that is negative binomial, but a Poisson CQO model is fitted upon a square root transformation.

```
> n = 200
> p = 5
> S = 3
> mydata = rcqo(n, p, S, fam = "negbin", ESOpt = TRUE, sqrt = TRUE,
+     seed = 123)
```

The raw data can be seen in Figure 6. It can be clearly seen that there is more 'noise' compared to a Poisson distribution.

```
> matplot(attr(mydata, "lv"), (mydata[, -(1:(p - 1))])^2,
+     col = 1:S)
```

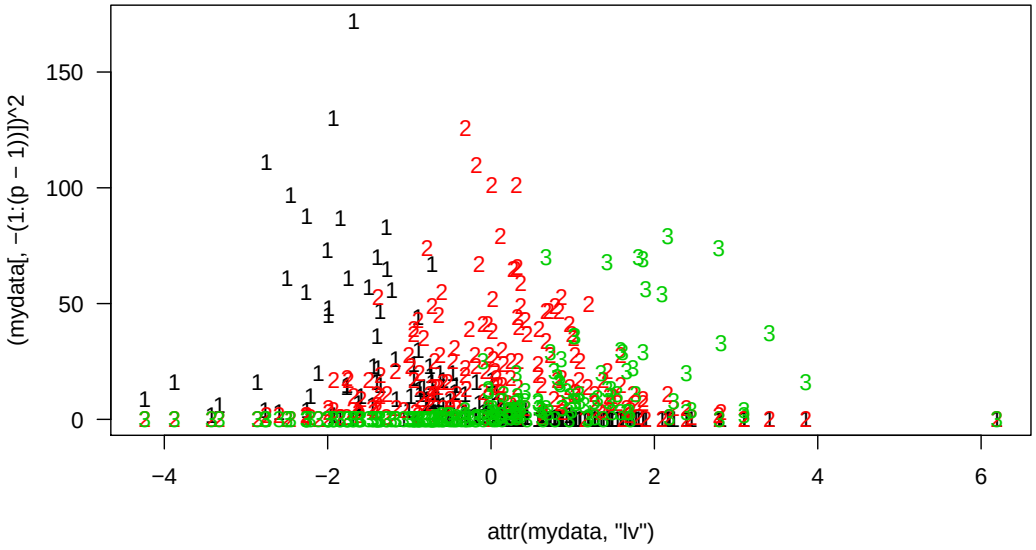

Figure 6: Raw counts from a simulated negative binomial data set.

Now we can fit a Poisson CQO to the transformed data as follows.

```
> myform = attr(mydata, "formula")
> fit4 = cqo(myform, fam = poissonff, ITol = TRUE, dat = mydata,
+     trace = FALSE)
```

Then a latent variable plot is given in Figure 7.

Ideally, the optima in the figure should be at $\sqrt{100} = 10$. This data set shows an underestimation. In terms of the constrained coefficients, one can compare the fitted model with the 'truth' by

```
> cbind(truth = attr(mydata, "ccoefficients"), fitted = ccoef(fit4))
```

```
> lvplot(fit4, lcol = 1:S, y = TRUE, pcol = 1:S)
```

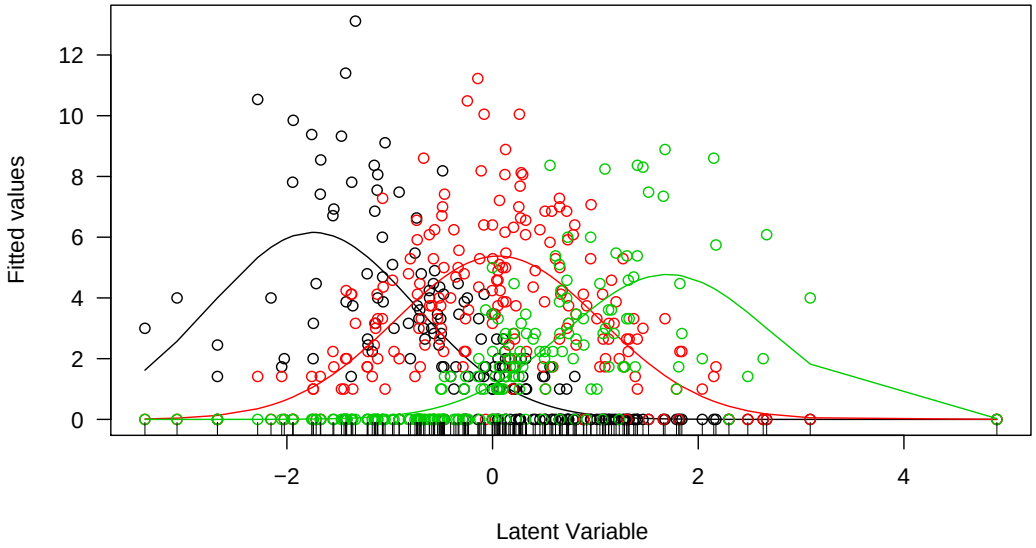

Figure 7: Latent variable plot for simulated negative binomial data fitted with a Poisson CQO model.

```
           lv         lv
x2  0.3241401  0.3253477
x3 -0.5986456 -0.4844184
x4  0.7780473  0.6198870
x5  1.0489203  0.7790064
```

There is a rough match.

6.10 Calibration

Yee (2005) describes maximum likelihood calibration whereby the values of the latent variables at a site can be estimated, given the species data there. The generic function `calibrate()` is available, and methods functions for CQO and CAO objects have been written for these.

6.11 Miscellaneous Notes

1. If $\boldsymbol{x}_1 \neq 1$ then the maximum of a species is undefined as it depends on values of variables in $\boldsymbol{x}_1$. Output from `Coef()` etc. will give a `NA`.

2. The constraint matrices of the $\boldsymbol{x}_2$ variables are considered to be the identity matrix.

3. If the "truth" is rank-R then fitting a higher rank model will give numerical problems because the site scores will lie in a R-dimensional space. For example, if $R = 1$ but you fit a rank-2 model, the site scores will lie effectively on a line. This is provided you manage to fit the model in the first place. The tolerance and/or canonical coefficients for the second ordination axis would then be difficult to estimate.

4. Practical experience with `ITolerances=TRUE` shows that `optim()` may give different results due to its (faulty?) convergence criterion. That is, it may terminate prematurely near the solution. This can be partly explained by the use of finite-difference approximations to the derivatives: the argument `Hstep` may not have a particularly good value. If premature termination becomes a problem, trying applying `scale()` to the numerical variables in $\boldsymbol{x}_2$ and/or setting `EqualTolerances=TRUE` (it results in a more accurate answer usually, however, it can be much slower).

7 Unconstrained Quadratic Ordination (UQO)

Yee (2006a) discusses a new nomenclature that is partly given in Table 1. All three "constrained" methods have an "unconstrained" counterpart whereby the site scores $\boldsymbol{\nu}_i$ are not constrained to be linear combinations of $\boldsymbol{x}_2$. For example, UQO is CQO but where the site scores are largely free parameters.

Solving for the optimal site scores $\widehat{\boldsymbol{\nu}}_i$ by maximum likelihood estimation presents a very difficult optimization problem, and very good initial values are required using the algorithm currently implemented in `uqo()`. Unconstrained quadratic ordination is detailed in Yee (2005).

8 Constrained Additive Ordination (CAO)

Constrained additive ordination (CAO) can be performed by fitting a *Reduced-Rank VGAM* (RR-VGAM). This is a nonparametric extension of QRR-VGLMs, and can loosely be thought of as a GAM fitted to a very small number of latent variables $\boldsymbol{\nu}$. The function `cao()` currently implements CAO, but is currently restricted to rank $R = 1$ and Poisson and binary responses with known dispersion parameters. This means the user needs to use `poissonff()` and `binomialff()` families. Also, $\boldsymbol{x}_1 = 1$ is also needed.

In more detail, suppose $R = 1$ and we have presence/absence data. Then

```
cao(cbind(spp1,spp2,spp3,...,sppS) ~ ..., family = binomialff(mv=TRUE))
```

will fit the model

$$\text{logit}\, \mu_{is} \;=\; \log \frac{\mu_{is}}{1-\mu_{is}} \;=\; f_s(\nu_i), \quad\quad s = 1, \ldots, S, \tag{19}$$

where f_s are arbitrary smooth functions estimated by a smoothing spline. The variables `spp1`,...,`sppS` should have values 1 and 0 for presence and absence respectively. In (19), the intercept term normally associated with $\boldsymbol{x}_1$ has been absorbed into the function f_s. With Poisson abundance/count data, (19) becomes

$$\log\, \mu_{is} \;=\; f_s(\nu_i), \quad\quad s = 1, \ldots, S, \tag{20}$$

and the call would be

```
cao(cbind(spp1,spp2,spp3,...,sppS) ~ ..., family = poissonff)
```

Constrained additive ordination models were proposed by Yee (2006a).

Practitioners who know how to fit GAMs should find `cao()` easy to use. Books and articles on GAM modelling abound; some are Hastie and Tibshirani (1990), Yee and Mitchell (1991), Chambers and Hastie (1993), Hastie et al. (2001), Venables and Ripley (2002), Oksanen and Minchin (2002), Ruppert et al. (2003).

8.1 Controlling Function Flexibility

Importantly, the `df1.nl` argument controls how smooth the functions f_s are. A *nonlinear degrees of freedom* value of 0 means that function is linear, so it is a constrained linear ordination (CLO) model for that species. As the nonlinear degrees of freedom increases the smooth can become more wiggly. A value between 0 and 3 is recommended, and possibly up to 4. However, a common mistake is to allow too much flexibility to the curves, especially when the data set is not too large. As the nonlinear degrees of freedom increases, the optimization problem becomes more difficult because of the increased number of local solutions. A value of about 1.5 to 2 gives the approximate flexibility of a quadratic, hence `df1.nl=2` may give approximately the same results as a CQO. The `df1.nl` argument allows different species to have different nonlinear degrees of freedom, e.g., `df1.nl=c(3, spp8=3.5, spp4=2.5)` means Species 8 and 4 have 3.5 and 2.5 nonlinear degrees of freedom respectively, and all other species have 3 nonlinear degrees of freedom. Assigning a `df1.nl` value that is too low for a species may result in a lack-of-convergence problem (`errcode == 3`) in the local scoring algorithm. The remedy, of course, is to assign a slightly larger value.

8.2 CAO Example

Yee (2006a) gives two examples of CAO modelling involving Poisson and binomial species data. Here is a brief CAO analysis of the simulated data given in Section 6.7. After running the code for the data generation, we can type

```
> set.seed(123)
> a1 = cao(cbind(y1, y2, y3) ~ x2 + x3 + x4, fam = poissonff,
+     data = mysim, trace = FALSE, Bestof = 4, df1.nl = 3)
> a1@misc$deviance

[1] 236.9799 236.9799 236.9799 236.9799

> print(Coef(a1))

C matrix (constrained/canonical coefficients)
          lv
x2  0.017022
x3  0.481185
x4 -0.945096

Optima and maxima
    Optimum Maximum
y1 0.006756  20.792
y2 0.462582   7.076
y3       NA      NA

Nonlinear degrees of freedom
   df1.nl
y1  3.002
y2  3.002
y3  2.998
```

for a start. Actually, "`trace=FALSE`" is only used to stop voluminous printout here, but for the user it is best to use `trace=TRUE`. For these data all the fits give essentially the same deviance. This is not usually the case because we have got 'perfect' data here. With real data, the fitted models usually are very different and fitting many models is needed to get a good fit, e.g., use `Bestof=20`.

A latent variable plot is given in Figure 8. Note that all three curves should be bell-shaped, but one isn't because there is insufficient data at the LHS side. That is, it isn't because of statistical error because there are few site scores on the LHS so there would be much uncertainty there. Standard error bands (giving ± 2 pointwise standard errors, say) are usual for GAM plots, but here they are currently unavailable because the theory hasn't been worked out for them yet (the x-axis are latent variable estimates therefore contain uncertainty in them).

Note also that the latent variable axes of Figure 4 and Figure 8 do not match because the CQO is scaled using a unit tolerance for Species 1 whereas the CAO has unit variance for the site scores.

As an exercise, the reader can rerun this code with 1000 sites instead and show that all three curves are bell-shaped as expected.

```
> lvplot(a1, lcol = 1:3)
```

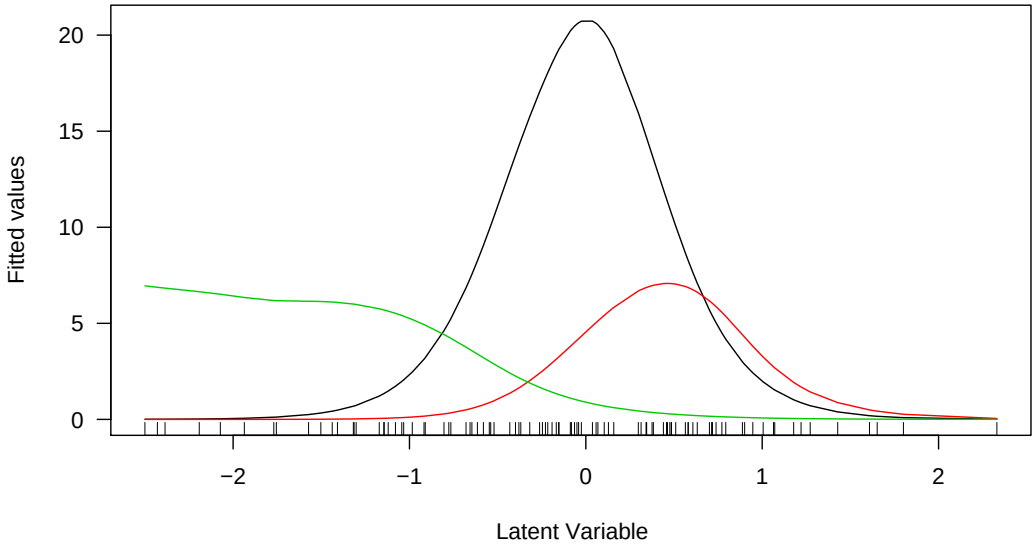

Figure 8: Latent variable plot of `a1`. This should be similar to Figure 4.

9 Ordinal Ordination

Ordinal responses obtained from an underlying Poisson or negative binomial distribution are a common form of data. For example, in the biological sciences, it is common to measure species data as an ordinal response taking levels $\{1, 2, \ldots, L\}$, say, where $L \geq 2$. For example, 1 means absence, 2 means low abundance, 3 means medium abundance and 4 means high abundance. A specific example of this is the Braun-Blanquet scale in plant ecology with $L = 7$ levels. The need to measure the response as an ordinal factor often arises because it is too difficult to count individuals, e.g., ants, diatoms, plankton are very small and/or moving. Instead, it is more convenient to categorize the abundance into a factor Y^*, say, with L levels. We have $Y^* = l$ if $K_{l-1} < Y \leq K_l$ for integer *cut points* or thresholds K_l ($l = 1, 2, \ldots, L$) where $K_0 = -\infty$ and $K_L = \infty$. Here, Y^* the categorical response.

Yee (2006b) has developed a new methodology which allows ordination with ordinal species response. The main ideas are as follows. We assume the underlying count distribution (for a random variable Y) is either Poisson or negative binomial with

$$\log \mu \;\; = \;\; \eta \tag{21}$$

where $\mu = E(Y)$ and η is a linear or additive predictor in covariates $\boldsymbol{x}$ (Hastie and Tibshirani, 1990). This is the natural and ideal situation. However, we are stuck with Y^* instead of Y.

With ordinal responses Y^*, a very popular approach is to model the cumulative probabilities $\gamma_l = \mathrm{pr}[Y^* \leq l]$. For example, the proportional odds model has

$$g^*(\mathrm{pr}[Y^* > l]) \;\; = \;\; \eta_l^*, \qquad l = 1, 2, \ldots, L-1, \tag{22}$$

where g^* is the logit link and η_l^* is a linear or additive predictor in $\boldsymbol{x}$ (see Yee and Wild (1996)).

Yee (2006b) has shown how it is possible to fit (21) by actually fitting (22). The key is to use new link functions, called the "Poisson-ordinal" link function (or POLF) or the "negative binomial ordinal" link function (or NBOLF). The disadvantage with these is that the cut points must be known, something which is not possible for all studies. However, we can approximate the negative binomial distribution by a 2-parameter gamma distribution (giving a GOLF) and it is possible to obtain an approximate solution with the GOLF without knowing the cut points.

Note that, with Poisson counts and $K_1 = 0$, it is recommended that a complementary log-log link be used. That is, with presence/absence data coming from an underlying Poisson distribution, a constrained ordination should use the complementary log-log link instead of the logit link which is more popular and is the default for `cumulative()`.

Useful functions in the VGAM package for ordinal ordination are listed in Table 8.

Table 8: A summary of VGAM functions associated with ordinal ordination.

Function	Purpose
`golf()`	gamma-ordinal link function, or GOLF.
`polf()`	Poisson-ordinal link function, or POLF.
`nbolf()`	negative binomial-ordinal link function, or NBOLF.
`nbolf2()`	Alternative NBOLF (based on a Camp-Paulson approximation).
`cumulative()`	VGAM family function for ordinal regression.
`Cut()`	Converts counts to an ordinal response
`optim()`	General-purpose optimization function.

Here are some notes about the current VGAM implementation of ordinal ordination.

1. The code is not finished! It is planned that invocations such as

   ```
   cqo(ymat ~ x2 + x3 + x4 + x5,
       fam=cumulative(link="polf", reverse=TRUE, mv=TRUE), data=d)
   ```

   be done in FORTRAN—this is simple and clean for the typical user. Currently one needs to do it all in R, e.g., in the example below there is the need for a function (called `newfun()`) to set things up and this is messy and complicated for the typical user. Hopefully the need for such a function will be banished soon with the FORTRAN implementation.

2. The input/response to `cumulative(mv=TRUE)` is a matrix of ordinal responses taking integer values $\{1, 2, \ldots, L\}$ for *each* column. That is, each response must have some of each of L levels of the factor. This hopefully will be relaxed in the future. The output from `Cut()` is specifically designed to be input for `cumulative(mv=TRUE)`.

3. The link functions in Table 8 should be operated on the *reverse* cumulative probabilites, i.e., $\gamma_l^c = 1 - \gamma_l$ to maintain a suitable direction for η_l^*. This is so that the tolerance matrices are positive-definite for bell-shaped curves/surfaces. That is why `reverse=TRUE` is recommended.

4. When using `nbolf()` and `polf()` the cut points are entered using the `earg` argument. The component name of this list is `"cutpoint"`. Since the reverse cumulative probabilites are used, the `"cutpoint"` vector should be an increasing sequence. For example,

   ```
   > prob = seq(0.1, 0.9, by = 0.2)
   > polf(cbind(prob, prob), earg = list(cutpoint = c(5, 20)))
   ```

```
         prob     prob
[1,] 1.156667 2.736333
[2,] 1.541734 2.920352
[3,] 1.770706 3.038552
[4,] 1.976136 3.150155
[5,] 2.239844 3.301042
```

Here, the first argument of `polf()` must have two columns because there are two cut points. The function `nbolf()` requires k and the component name is simply `"k"`.

5. When using `golf()` the cut points are optional. However, λ isn't, and it is entered using the `earg` argument with component name `"lambda"`.

6. For `Cut()` the argument `breaks` matches that of the S function `cut()`. It is recommended that the first and last values of `breaks` should be `-Inf` and `Inf` respectively.

9.1 Example

Here's an ordinal ordination example from the hunting spiders data set. The following code is specific to the `hspider` data frame, however you could adapt it for your own data set. We fit a rank-1 equal tolerances CQO negative binomial model with $k = 1$ for each species.

```
> data(hspider)
> hspider[, 1:6] = scale(hspider[, 1:6])
> newfun = function(bic, mylink = "nbolf", lambda = 1, kay = 1,
+     dev = TRUE, omitcutpt = FALSE, trace = FALSE, mybreaks = c(-Inf,
+         5, 20, Inf), altbreaks = c(-Inf, 1, 10, Inf), altspp = NULL,
+     spp. = c(7, 8, 9, 10, 11, 12, 14, 15, 16, 17, 18)) {
+     xmat = as.matrix(hspider[, 1:6])
+     lv1 = c(xmat %*% cbind(bic))
+     lv12 = -0.5 * lv1 * lv1
+     matMYbreaks = matrix(mybreaks, length(mybreaks), 18)
+     if (is.Numeric(altspp))
+         matMYbreaks[, altspp] = altbreaks
+     LLength = nrow(matMYbreaks) - 1
+     allmyb = c(matMYbreaks[2:LLength, spp.])
+     ycut = Cut(hspider[, spp.], breaks = mybreaks)
+     counter = 1
+     for (ii in spp.) {
+         ycut[, counter] = Cut(hspider[, ii], breaks = matMYbreaks[,
+             ii])
+         counter = counter + 1
+     }
+     if (mylink == "polf")
+         earg = list(cutpoint = allmyb)
+     if (mylink == "nbolf")
+         earg = list(cutpoint = allmyb, k = kay)
+     if (mylink == "golf") {
+         if (omitcutpt) {
+             earg = list(lambda = lambda)
```

```
+         }
+         else {
+             earg = list(cutpoint = allmyb, lambda = lambda)
+         }
+     }
+     M = LLength - 1
+     NOS = length(spp.)
+     mylist = list("(Intercept)" = if (mylink == "golf") {
+         if (length(earg$cutpoint)) kronecker(diag(NOS),
+             matrix(1, M, 1)) else kronecker(diag(NOS),
+             diag(M))
+     } else kronecker(diag(NOS), matrix(1, M, 1)), lv1 = kronecker(diag(NOS),
+         matrix(1, M, 1)), lv12 = kronecker(matrix(1, NOS,
+         1), matrix(1, M, 1)))
+     fit = vglm(ycut ~ lv1 + lv12, constraint = mylist,
+         cumulative(link = mylink, rev = TRUE, mv = TRUE,
+             par = TRUE, earg = earg, intercept.apply = TRUE),
+         trace = trace)
+     if (dev)
+         deviance(fit)
+     else fit
+ }
```

The function `newfun()` above uses two sets of cutpoints over all species. For example, rare species can be handled by the argument `altbreaks` having lower values for the cut points. The function `newfun()` fits the CQO given the values of $\boldsymbol{c}$ in the argument `bic`. The list `mylist` contains the constraint matrices; understanding the technical details requires knowledge about VGLMs and QRR-VGLMs in general.

Now we will get initial values for the constrained coefficients $\boldsymbol{c}$ by fitting an ordinary CQO to the counts.

```
> ymat = as.matrix(hspider[, -(1:6)])
> p1 = cqo(ymat ~ WaterCon + BareSand + FallTwig + CoveMoss +
+     CoveHerb + ReflLux, fam = poissonff, trace = FALSE,
+     data = hspider, Crow1positive = FALSE, ITol = TRUE,
+     Bestof = 4)
> sort(p1@misc$deviance.Bestof)

[1] 1585.133 1719.542 2472.063 2472.070

> if (deviance(p1) > 1586) stop("suboptimal fit obtained")
> round(c(ccoef(p1)), dig = 2)

[1] -0.36  0.55 -0.92  0.32 -0.31  0.70
```

Now we can finally solve for $\boldsymbol{c}$ with ordinal ordination! The following code requires a substantial amount of number crunching and can take a minute or two to run.

```
> set.seed(123)
> solveit = optim(par = c(ccoef(p1)), method = "L-BFGS-B",
+     fn = newfun, altspp = c(9, 10, 11), altbreaks = c(-Inf,
```

```
+         0, 10, Inf), mylink = "nbolf", kay = 1, trace = FALSE,
+     lower = -Inf, upper = +Inf)
> solveit$value

[1] 232.1201

> solveit$par

[1] -0.1692321  0.7802340 -0.6223791  0.5538531 -0.1043394  0.9682084

> if (solveit$convergence != 0) stop("did not converge")
```

Now we scale the solution so that the species have unit tolerances.

```
> fit <- newfun(solveit$par, dev = FALSE, altspp = c(9, 10,
+     11), altbreaks = c(-Inf, 0, 10, Inf), mylink = "nbolf",
+     kay = 1, trace = FALSE)
> ssoln = solveit$par * sqrt(coef(fit)[length(coef(fit))])
> fit <- newfun(ssoln, dev = FALSE, altspp = c(9, 10, 11),
+     altbreaks = c(-Inf, 0, 10, Inf), mylink = "nbolf",
+     kay = 1, trace = FALSE)
> deviance(fit)

[1] 232.1201

> coef(fit)["lv12"]

lv12
   1

> round(ssoln, dig = 2)

[1] -0.19  0.89 -0.71  0.63 -0.12  1.11
```

(We checked the unit tolerances by seeing that the coefficient for the `"lv12"` is 1). Here, `ssoln` is the scaled solution. Looking at the results, we can see that the constrained coefficients from this ordinal ordination have the same sign as that of the raw count data. This is reassuring; it shows there is some stability in the ordination. Fitting various ordinal CQO and CAO models (and with Poisson, negative binomial and gamma links) shows that the signs of the constrained coefficients do not seem to change much—see the Exercises.

It is expected that ordinal ordination should work reasonably well, especially for large simulated data sets. Given the current software limitation that each species must have the same number of levels, how many levels should be used? From a practical point of view, I currently recommend $L = 3$ or 4. Decreasing L means there is less sensitivity to outliers in the response. Increasing L means less information loss, but with L too large it becomes more difficult to find cut points that work well in data sets with both rare and abundant species.

Just to finish the example, note that

```
> fit@y
```

	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
1	0	0	1	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0
2	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	1	0	1	0
3	0	1	0	0	1	0	0	1	0	0	1	0	1	0	0	0	1	0	0	0	1	0	1	0	0	0	1	0	0	1	1	0	0
4	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	0	1	0	1	0	0	0	1	0	0	1	0	0	1	0	0	1
5	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	1	0	1	0
6	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0	1
7	1	0	0	0	1	0	1	0	0	0	0	1	1	0	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0	1	0	1	0
8	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0
9	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
10	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
11	0	1	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0
12	0	1	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0	0	1	0	0	1	0	1	0	0
13	1	0	0	0	0	1	0	1	0	0	1	0	1	0	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1
14	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	1	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	1
15	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0
16	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0
17	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0
18	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0
19	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0
20	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	1	1	0	0
21	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0
22	0	1	0	1	0	0	0	0	1	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
23	0	1	0	1	0	0	0	0	1	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0
24	0	1	0	1	0	0	0	0	1	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
25	0	1	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	0	1	0	0	1	0	1	0	0	0	1	0	0	1	0
26	1	0	0	1	0	0	0	1	0	1	0	0	0	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
27	0	0	1	1	0	0	0	0	1	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0	1	0	0
28	0	1	0	1	0	0	0	0	1	1	0	0	0	1	0	1	0	0	0	1	0	1	0	0	1	0	0	1	0	0	1	0	0

Here, the first three columns correspond to the first species, etc. and the column names are l for $l = 1, \ldots, L$. A 1 indicates the presence of that level, otherwise a 0, so `fit@y` is a matrix of indicator variables really. Each column must contain at least one 1 and one 0.

10 Not Yet Been Implemented

The following is a list of things not yet been implemented in VGAM.

1. Along the lines of Fu (2003) and Fu (1998), one could apply a Lasso type operator to the latent variables. This would mean that some of the coeffients would be zero, and possibly the log-likelihood function would be unimodal rather than multimodal. This needs investigation.

Software Changes

[12 Jan 2004] `biplot.qrrvglm()` merged with `lvplot.qrrvglm()`; the two functions are equivalent.

[27 Sep 2005] `FastAlgorithm=FALSE` is not allowed.

[17 Oct 2005] An undocumented faster algorithm is implemented when `ITolerances=TRUE`.

[7 Aug 2006] `ITolerances=TRUE` is now the default.

Exercises

1. More general than (6) is

$$\boldsymbol{\eta} \;=\; \mathbf{B}_1^T \boldsymbol{x}_1 + \sum_{k=1}^{K} \mathbf{A}_k \mathbf{C}_k^T \boldsymbol{x}_{k+1},$$

   where K is any specified positive integer, $\mathbf{A}_k$ is $M \times R_k$ and unknown, and $\boldsymbol{x}_k$ is a p_k-vector with $p_1 + \cdots + p_{K+1} = p$. This is currently not yet implemented in VGAM. Discuss the computational details behind estimating such a model.

2. In the example of Section 5.2, corner constraints were used. Repeat the analysis with summation constraints, $\sum_i \alpha_i = \sum_j \gamma_j = 0$, and reconcile the estimated regression coefficients.

3. Search the literature for a model that has many parameters for which a reduced-rank fit would be a good idea. Write a VGAM family function for the model and run it under `rrvglm()` on a suitable data set. Does a biplot add to your understanding?

4. Suppose $\mathbf{W}_i,\ i = 1, \ldots, n$, are general $M \times M$ positive-definite matrices and $\mathbf{Z}$ is a general $n \times M$ matrix. Suppose we wish to approximate $\mathbf{Z}$ by a specified lower rank matrix $\widehat{\mathbf{Z}} = \mathbf{A}\mathbf{C}^T$ where $\mathbf{A}$ is $n \times R$ and $\mathbf{C}$ is $M \times R$. One can do this by minimizing

$$\| \, \boldsymbol{\Omega} * \left(\mathbf{Z} - \widehat{\mathbf{Z}} \right) \, \|^2 \;=\; \sum_{j=1}^{n} \sum_{k=1}^{M} (\boldsymbol{\Omega})_{jk} \left(z_{jk} - \boldsymbol{a}_j^T \boldsymbol{c}_k \right)^2, \tag{23}$$

   where $\boldsymbol{\Omega}$ is an associated weight matrix and $*$ is the Hadamard (element-by-element) product. Show that this may be solved by an alternating algorithm. Hint: look at the underlying minimization problem behind VGAM. [Gabriel and Zamir (1979)]

5. In the car example, show that a MLM treating $\widehat{\mathbf{A}}$ as known and fixed produces standard errors that are smaller than that obtained from the RR-MLM. Of course, your fitted coefficients should be identical of the RR-MLM.

6. Suppose one runs

   ```
   rrvglm(ymatrix ~ bs(x, 4), family=multinomial, Rank=1, Corner=TRUE)
   ```

   Show that this is a multinomial logit model with

   $$\eta_j = \beta_{(j)0} + a_j f(x), \quad a_1 = 1, \quad j = 1, \ldots, M,$$

   where f is a smooth function of x estimated by a B-spline. If "`multinomial`" is replaced by "`cumulative`" the curves won't intersect if a_j are all the same sign and f is monotonic.

7. Consider a rank-1 Poisson CQO model with $\boldsymbol{x}_1 = 1$. Let t_j be the tolerance of species j. Show that

   $$\frac{\mu_j(u_j \pm k\, t_j)}{\mu_j(u_j)} \;=\; \exp\left\{-\frac{1}{2}k^2\right\} \tag{24}$$

   where $k > 0$. This says that if you are k tolerance units away from the species' optimum then the ratio of the mean abundance there relative to the species' maximum declines exponentially in $-\frac{1}{2}k^2$. Show that the RHS of (24) is also the corresponding expression for the rank-2 equal-tolerances Poisson CQO model with $\mathbf{T}_j = \mathbf{I}_2$ when k is the Euclidean distance away from the species' optimum. Is this true for general rank-R?

8. Consider the CQO model when `ITolerances=TRUE`. Discuss the uniqueness of the parameters for this model.

9. Extend the example of Section 6.7 by generating data from a rank-2 model, and then fit a rank-2 QRR-VGLM. Obtain a latent variable plot, and check that it gives sensible output. Also check that the output from `summary()` and `Coef()` is 'right'.

10. In the example of Section 9.1, $k = 1$ was used for all species. Suppose all species have a common k value but $k \neq 1$. Perform the ordinal ordination for various values of k and so maximize a profile likelihood. What is the optimal k to one decimal place accuracy?

11. Using `newfun()` for a start, fit various ordinal ordinations to the hunting spiders data. Vary the link function and the cut points. Do the results appear to change much?

References

Ahn, S., Reinsel, G. C., 1988. Nested reduced-rank autoregressive models for multiple time series. Journal of the American Statistical Association 83, 849–856.

Anderson, J. A., 1984. Regression and ordered categorical variables (with discussion). Journal of the Royal Statistical Society, Series B, Methodological 46, 1–30.

Anderson, T. W., 1951. Estimating linear restrictions on regression coefficients for multivariate normal distributions. The Annals of Mathematical Statistics 22, 327–351.

Chambers, J. M., Hastie, T. J. (Eds.), 1993. Statistical Models in S. Chapman & Hall, New York.

Fiocco, M., Putter, H., van Houwelingen, J. C., 2005. Reduced rank proportional hazards model for competing risks. Biostatistics 6, 465–478.

Fu, W. J., 1998. Penalized regressions: The bridge versus the lasso. Journal of Computational and Graphical Statistics 7, 397–416.

Fu, W. J., 2003. Penalized estimating equations. Biometrics 59, 126–132.

Gabriel, K. R., Zamir, S., 1979. Lower rank approximation of matrices by least squares with any choice of weights (corr: V22 p136). Technometrics 21, 489–498.

Goodman, L. A., 1981. Association models and canonical correlation in the analysis of cross-classifications having ordered categories. Journal of the American Statistical Association 76, 320–334.

Greenland, S., 1994. Alternative models for ordinal logistic regression. Statistics in Medicine 13, 1665–1677.

Hastie, T. J., Tibshirani, R. J., 1990. Generalized Additive Models. Chapman & Hall, London.

Hastie, T. J., Tibshirani, R. J., Friedman, J. H., 2001. Elements of Statistical Learning: Data Mining, Inference and Prediction. Springer-Verlag, New York.

Heinen, A., Rengifo, E., 2005. Multivariate reduced rank regression in non-Gaussian contexts, using copulas. Preprint submitted to Journal of Multivariate Analysis, 9 July 2004 .

McCullagh, P., 1980. Regression models for ordinal data. Journal of the Royal Statistical Society, Series B, Methodological 42, 109–142.

McCullagh, P., Nelder, J. A., 1989. Generalized Linear Models, 2nd Edition. Chapman & Hall, London.

Oksanen, J., Minchin, P. R., 2002. Continuum theory revisited: what shape are species responses along ecological gradients? Ecological Modelling 157, 119–129.

Reinsel, G. C., Velu, R. P., 1998. Multivariate Reduced-Rank Regression: Theory and Applications. Springer-Verlag, New York.

Ruppert, D., Wand, M. P., Carroll, R. J., 2003. Semiparametric Regression. Cambridge University Press, Cambridge.

van Eeuwijk, F. A., 1995. Multiplicative interaction in generalized linear models. Biometrics 51, 1017–1032.

Venables, W. N., Ripley, B. D., 2002. Modern Applied Statistics With S, 4th Edition. Springer-Verlag, New York.

Wartenberg, D., Ferson, S., Rohlf, F. J., 1987. Putting things in order: a critique of detrended correspondence analysis. The American Naturalist 129, 434–448.

Yee, T. W., 2004. A new technique for maximum-likelihood canonical Gaussian ordination. Ecological Monographs 74, 685–701.

Yee, T. W., 2005. On constrained and unconstrained quadratic and additive ordination. Manuscript in preparation .

Yee, T. W., 2006a. Constrained additive ordination. Ecology 87, 203–213.

Yee, T. W., 2006b. Ordinal ordination with normalizing link functions for count data. Submitted for publication .

Yee, T. W., Hastie, T. J., 2003. Reduced-rank vector generalized linear models. Statistical Modelling 3, 15–41.

Yee, T. W., Mitchell, N. D., 1991. Generalized additive models in plant ecology. Journal of Vegetation Science 2, 587–602.

Yee, T. W., Wild, C. J., 1996. Vector generalized additive models. Journal of the Royal Statistical Society, Series B, Methodological 58, 481–493.