Generalized nonlinear models in R: an overview of the ***gnm*** package

Heather Turner and David Firth[*]
University of Warwick, UK

For *gnm* version 0.7-1 , 2005-07-22

Contents

[*]This work was supported by the Economic and Social Research Council (UK) through a Professorial Fellowship.

1 Introduction

The *gnm* package provides facilities for fitting *generalized nonlinear models*, i.e., regression models in which the link-transformed mean is described as a sum of predictor terms, some of which may be non-linear in the unknown parameters. Linear and generalized linear models, as handled by the `lm` and `glm` functions in R, are included in the class of generalized nonlinear models, as the special case in which there is no nonlinear term.

This document gives an extended overview of the *gnm* package, with some examples of applications. The primary package documentation in the form of standard help pages, as viewed in R by, for example, `?gnm` or `help(gnm)`, is supplemented rather than replaced by the present document.

We begin below with a preliminary note (Section 2) on ways in which the *gnm* package extends R's facilities for specifying, fitting and working with generalized *linear* models. Then (Section 3 onwards) the facilities for nonlinear terms are introduced, explained and exemplified.

The *gnm* package is installed in the standard way for CRAN packages, for example by using `install.packages`. Once installed, the package is loaded into an R session by

```
> library(gnm)
```

2 Generalized Linear Models

2.1 Preamble

Central to the facilities provided by the *gnm* package is the model-fitting function `gnm`, which interprets a model formula and returns a model object. The user interface of `gnm` is patterned after `glm` (which is included in R's standard *stats* package), and indeed `gnm` can be viewed as a replacement for `glm` for specifying and fitting generalized linear models. In general there is no reason to prefer `gnm` to `glm` for fitting generalized linear models, except perhaps when the model involves a large number of incidental parameters which are treatable by `gnm`'s *eliminate* mechanism (see Section 4.4).

While the main purpose of the *gnm* package is to extend the class of models to include nonlinear terms, some of the new functions and methods can be used also with the familiar `lm` and `glm` model-fitting functions. These are: three new data-manipulation functions `Diag`, `Symm` and `Topo`, for setting up structured interactions between factors; a new *family* function, `wedderburn`, for modelling a response variable in $[0, 1]$ with the variance function $V(\mu) = \mu^2(1-\mu)^2$ as in **?**; and a new generic function `termPredictors` which extracts the contribution of each term to the predictor from a fitted model object. These functions are briefly introduced here, before we move on to nonlinear models in Section 3.

2.2 `Diag` and `Symm`

When dealing with *homologous* factors, that is, categorical variables whose levels are the same, statistical models often involve structured interaction terms which exploit the inherent symmetry. The functions `Diag` and `Symm` facilitate the specification of such structured interactions.

As a simple example of their use, consider the log-linear models of *quasi-independence*, *quasi-symmetry* and *symmetry* for a square contingency table. **?**, Section 10.4, gives data on migration between regions of the USA between 1980 and 1985:

```
> count <- c(11607, 100, 366, 124, 87, 13677, 515, 302, 172, 225,
+     17819, 270, 63, 176, 286, 10192)
> region <- c("NE", "MW", "S", "W")
> row <- gl(4, 4, labels = region)
> col <- gl(4, 1, length = 16, labels = region)
```

The comparison of models reported by Agresti can be achieved as follows:

```
> independence <- glm(count ~ row + col, family = poisson)
> quasi.indep <- glm(count ~ row + col + Diag(row, col), family = poisson)
> symmetry <- glm(count ~ Symm(row, col), family = poisson)
```

```
Loading required package: gtools
```

```
> quasi.symm <- glm(count ~ row + col + Symm(row, col), family = poisson)
> comparison1 <- anova(independence, quasi.indep, quasi.symm)
> print(comparison1, digits = 7)
```

```
Analysis of Deviance Table

Model 1: count ~ row + col
Model 2: count ~ row + col + Diag(row, col)
Model 3: count ~ row + col + Symm(row, col)
  Resid. Df Resid. Dev Df  Deviance
1         9  125923.29
2         5      69.51  4 125853.78
3         3       2.99  2     66.52
```

```
> comparison2 <- anova(symmetry, quasi.symm)
> print(comparison2)
```

```
Analysis of Deviance Table

Model 1: count ~ Symm(row, col)
Model 2: count ~ row + col + Symm(row, col)
  Resid. Df Resid. Dev Df Deviance
1         6    243.550
2         3      2.986  3  240.564
```

The `Diag` and `Symm` functions also generalize the notions of diagonal and symmetric interaction to cover situations involving more than two homologous factors.

2.3 Topo

More general structured interactions than those provided by `Diag` and `Symm` can be specified using the function `Topo`. (The name of this function is short for 'topological interaction', which is the nomenclature often used in sociology for factor interactions with structure derived from subject-matter theory.)

The `Topo` function operates on any number (k, say) of input factors, and requires an argument named *spec* which must be an array of dimension $L_1 \times \ldots \times L_k$, where L_i is the number of levels for the ith factor. The *spec* argument specifies the interaction level corresponding to every possible combination of the input factors, and the result is a new factor representing the specified interaction.

As an example, consider fitting the 'log-multiplicative layer effects' models described in **?**. The data are 7 by 7 versions of social mobility tables from **?**:

```
> data(erikson)
> erikson <- as.data.frame(erikson)
> lvl <- levels(erikson$origin)
> levels(erikson$origin) <- levels(erikson$destination) <- c(rep(paste(lvl[1:2],
+     collapse = " + "), 2), lvl[3], rep(paste(lvl[4:5], collapse = " + "),
+     2), lvl[6:9])
> erikson <- xtabs(Freq ~ origin + destination + country, data = erikson)
```

From sociological theory — for which see **?** or **?** — the log-linear interaction between origin and destination is assumed to have a particular structure:

```
> levelMatrix <- matrix(c(2, 3, 4, 6, 5, 6, 6,
+                         3, 3, 4, 6, 4, 5, 6,
+                         4, 4, 2, 5, 5, 5, 5,
+                         6, 6, 5, 1, 6, 5, 2,
+                         4, 4, 5, 6, 3, 4, 5,
+                         5, 4, 5, 5, 3, 3, 5,
+                         6, 6, 5, 3, 5, 4, 1), 7, 7, byrow = TRUE)
```

The models of table 3 of **?** can now be fitted as follows:

```
> ## Null association between origin and destination
> nullModel <- gnm(Freq ~ country:origin + country:destination,
+                  family = poisson, data = erikson)
Running main iterations.
Done
> ## Interaction specified by levelMatrix, common to all countries
> commonTopo <- update(nullModel, ~ . +
+                      Topo(origin, destination, spec = levelMatrix))
Running main iterations.
Done
> ## Interaction specified by levelMatrix, different multiplier for each country
> multTopo <- update(nullModel, ~ . + Mult(country, Topo(origin, destination,
+                                   spec = levelMatrix)))
Running start-up iterations..
Running main iterations.......
Done
> ## Interaction specified by levelMatrix, different effects for each country
> separateTopo <- update(nullModel, ~ . +
+                        country:Topo(origin, destination, spec = levelMatrix))
Running main iterations.
Done
>
> anova(nullModel, commonTopo, multTopo, separateTopo)
Analysis of Deviance Table

Model 1: Freq ~ country:origin + country:destination
Model 2: Freq ~ Topo(origin, destination, spec = levelMatrix) + country:origin +
    country:destination
Model 3: Freq ~ Mult(country, Topo(origin, destination, spec = levelMatrix)) +
    country:origin + country:destination
Model 4: Freq ~ country:origin + country:destination + country:Topo(origin,
    destination, spec = levelMatrix)
  Resid. Df Resid. Dev  Df Deviance
1       108     4860.0
2       103      244.3   5   4615.7
3       101      216.4   2     28.0
4        93      208.5   8      7.9
```

Here we have used `gnm` to fit all of these log-link models; the first, second and fourth are log-linear and could equally well have been fitted using `glm`.

2.4 The `wedderburn` family

In **?** it was suggested to represent the mean of a continuous response variable in $[0, 1]$ using a quasi-likelihood model with logit link and the variance function $\mu^2(1-\mu)^2$. This is not one of the variance functions made available as standard in R's `quasi` family. The `wedderburn` family provides it. As an example, Wedderburn's analysis of data on leaf blotch on barley can be reproduced as follows:

```
> data(barley)
> logitModel <- glm(y ~ site + variety, family = wedderburn, data = barley)
> fit <- fitted(logitModel)
> print(sum((barley$y - fit)^2/(fit * (1 - fit))^2))
```

```
[1] 71.17401
```

This agrees with the chi-squared value reported on page 331 of **?**, which differs slightly from Wedderburn's reported value.

2.5 `termPredictors`

The generic function `termPredictors` extracts a term-by-term decomposition of the predictor function in a linear, generalized linear or generalized nonlinear model.

As an illustrative example, we can decompose the linear predictor in the above quasi-symmetry model as follows:

```
> print(temp <- termPredictors(quasi.symm))
```

```
   (Intercept)       row      col Symm(row, col)
1   -0.2641848 0.0000000 0.000000     9.62354843
2   -0.2641848 0.0000000 4.918310    -0.09198126
3   -0.2641848 0.0000000 1.539852     4.63901793
4   -0.2641848 0.0000000 5.082641     0.00000000
5   -0.2641848 4.8693457 0.000000    -0.09198126
6   -0.2641848 4.8693457 4.918310     0.00000000
7   -0.2641848 4.8693457 1.539852     0.07295506
8   -0.2641848 4.8693457 5.082641    -3.94766844
9   -0.2641848 0.7465235 0.000000     4.63901793
10  -0.2641848 0.7465235 4.918310     0.07295506
11  -0.2641848 0.7465235 1.539852     7.76583039
12  -0.2641848 0.7465235 5.082641     0.00000000
13  -0.2641848 4.4109017 0.000000     0.00000000
14  -0.2641848 4.4109017 4.918310    -3.94766844
15  -0.2641848 4.4109017 1.539852     0.00000000
16  -0.2641848 4.4109017 5.082641     0.00000000
```

```
> rowSums(temp) - quasi.symm$linear.predictors
```

```
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Such a decomposition might be useful, for example, in assessing the relative contributions of different terms or groups of terms.

3 Nonlinear Terms

The *gnm* package provides a flexible framework for the specification and estimation of generalized models with nonlinear terms. Multiplicative interaction terms can be estimated using the in-built capability of the `gnm` function and are specified in the model formula using the symbolic function `Mult`. Other nonlinear terms can be estimated using plug-in functions for `gnm` and are specified using `Nonlin`.

There are two plug-in functions currently made available in the *gnm* package: `MultHomog` for fitting multiplicative interaction terms with homogeneous effects and `Dref` for fitting diagonal reference terms. Users of *gnm* can define their own custom plug-in functions to specify other types of nonlinear term.

3.1 Multiplicative Interaction Terms using `Mult`

Multiplicative interaction terms can be included in the formula argument to `gnm` by using the symbolic wrapper function `Mult`. Constituent multipliers[1] in the interaction are passed as unspecified arguments to `Mult` and are expressed by symbolic linear formulae. An intercept is automatically added to each constituent multiplier unless otherwise specified. For example, to fit the row-column association model

$$\log \mu_{rc} = \alpha_r + \beta_c + \gamma_r \delta_c,$$

also known as the Goodman RC model (**?**), the *formula* argument of `gnm` would be

```
mu ~ R + C + Mult(-1 + R, -1 + C)
```

where `R` and `C` are row and column factors respectively.

`Mult` has one specified argument *multiplicity*, which is `1` by default. This argument determines the number of times that the specified multiplicative structure appears in the model. For example,

```
mu ~ R + C + Mult(-1 + R, -1 + C, multiplicity = 2)
```

would give the RC(2) model (**?**)

$$\log \mu_{rc} = \alpha_r + \beta_c + \gamma_r \delta_c + \theta_r \phi_c.$$

In some contexts, it may be desirable to constrain one or more of the constituent multipliers so that it is always nonnegative. This may be achieved by specifying the multiplier as an exponential, as in the following 'uniform difference' model (**??**)

$$\log \mu_{rct} = \alpha_{rt} + \beta_{ct} + e^{\gamma_t} \delta_{rc}.$$

Exponentiated constituent multipliers are specified in *gnm* models using the symbolic function `Exp`; for example, the uniform difference model above would be specified by the formula

```
mu ~ R:T + C:T + Mult(Exp(-1 + T), R:C)
```

3.2 Other Nonlinear Terms using `Nonlin`

Nonlinear terms which can not be specified using `Mult` may be specified using `Nonlin`. This symbolic function indicates a term which requires a plug-in function to estimate the associated parameters. `Nonlin` takes a single argument, which is a call to the relevant plug-in function.

For example, in the formula

```
mu ~ x + A + B + Nonlin(PlugInFunction(A, B, arg1 = x, arg2 = C))
```

the call to `Nonlin` is used to specify a term that requires the plug-in function `PlugInFunction`.

The two plug-in functions already included in the *gnm* package are described below, followed by a guide to writing custom plug-in functions.

[1] A note on terminology: the rather cumbersome phrase 'constituent multiplier', or sometimes the abbreviation 'multiplier', will be used throughout this document in preference to the more elegant and standard mathematical term 'factor'. This will avoid possible confusion with the completely different meaning of the word 'factor' — that is, a categorical variable — in R.

3.2.1 `MultHomog`

The `MultHomog` function provides the tools required to fit multiplicative interaction terms with one component in which the constituent multipliers are the effects of two or more factors and the effects of these factors are constrained to be equal when the factor levels are equal. The arguments of `MultHomog` are the factors in the interaction, which are assumed to be objects of class *factor*.

As an example, consider the following association model with homogeneous row-column effects:

$$\log \mu_{rc} = \alpha_r + \beta_c + \theta_r I(r = c) + \gamma_r \gamma_c.$$

To fit this model, with response variable named `mu`, the formula argument to `gnm` would be

```
mu ~ R + C + Diag(R, C) + Nonlin(MultHomog(R, C))
```

If the factors passed to `MultHomog` do not have exactly the same levels, a common set of levels is obtained by taking the union of the levels of each factor, sorted into increasing order.

3.2.2 `Dref`

`Dref` is a plug-in function to fit diagonal reference terms involving two or more factors with a common set of levels. A diagonal reference term comprises an additive component for each factor. The component for factor f, is given by

$$w_f \gamma_l$$

for an observation with level l of factor f, where w_f is the weight for factor f and γ_l is the "diagonal effect" for level l.

The weights are constrained to be nonnegative and to sum to one so that a "diagonal effect", say γ_l, is the value of the diagonal reference term for data points with level l across the factors. `Dref` constrains the weights by defining them as

$$w_f = \frac{e^{\delta_f}}{\sum_i e^{\delta_i}}$$

and estimating the δ_f.

Factors in the interaction are passed to unspecified arguments of `Dref`. For example, the following diagonal reference model for a contingency table classified by the row factor `R` and the column factor `C`,

$$\mu_{rc} = \frac{e^{\delta_1}}{e^{\delta_1} + e^{\delta_2}} \gamma_r + \frac{e^{\delta_2}}{e^{\delta_1} + e^{\delta_2}} \gamma_c,$$

would be specified by the formula

```
mu ~ -1 + Nonlin(Dref(R, C))
```

`Dref` has one specified argument *formula*, which is a symbolic description of the dependence of δ_f on any covariates. For example, the formula

```
mu ~ -1 + x + Nonlin(Dref(R, C, formula = ~ 1 + x))
```

specifies the following diagonal reference model

$$\mu_{rc} = \beta_X x + \frac{e^{\xi_1 + \beta_1 x}}{e^{\xi_1 + \beta_1 x} + e^{\xi_2 + \beta_2 x}} \gamma_r + \frac{e^{\xi_2 + \beta_2 x}}{e^{\xi_1 + \beta_x} + e^{\xi_2 + \beta_2 x}} \gamma_c,$$

The default value of *formula* is `~1`, so that constant weights are estimated. The coefficients returned by `gnm` are those that are directly estimated, i.e. the δ_f or the ξ_f and β_f, rather than the implied weights w_f.

3.2.3 Custom Plug-in Functions

Custom plug-in functions may be written to enable `gnm` to fit nonlinear terms that can not be specified by `Mult` or the plug-in functions provided by the *gnm* package.

There are no constraints on the arguments that a plug-in function may take. However it is important that `Nonlin`, when given a call to the plug-in function, can determine the variables that are in the term, so that these variables can be added to the model frame. By default, expressions passed to unspecified arguments of the plug-in function are assumed to represent the variables in the term.

If the default action of `Nonlin` will not capture the required variables, a companion function must exist (in the environment of the plug-in function), which takes the same arguments as the plug-in function and returns deparsed expressions representing the necessary variables. The name of this function must be the name of the plug-in function suffixed with "Variables". For example, the (non-visible) companion function for `Dref` is defined as

```
DrefVariables <- function(..., formula = ~ 1) {
    as.character(c(match.call(expand.dots = FALSE)[[2]], formula[[2]]))
}
```

returning the expressions passed to unspecified arguments and the right-hand side of the formula passed to *formula*, as character strings. For instance

```
> gnm:::DrefVariables(A, B, formula = ~1 + C)

[1] "A"     "B"     "1 + C"
```

from which `Nonlin` will know that `A`, `B` and `C` need to be added to the model frame.

The call to the plug-in function is evaluated in the environment of the model frame and in the enclosing environment of the parent frame of the call to `gnm`. This should ensure that variables passed directly to the plug-in function can be found. However, to evaluate variables within the plug-in function, it may be necessary to access the model frame, which can be obtained using the function `getModelFrame`.

For example, the factors in a `Dref` term are passed directly to unspecified arguments, so the dummy variables for these factors can be found as follows

```
# get design matrices for Dref factors
designList <- lapply(list(...), class.ind)
```

But any covariates on which the weights depend are only represented symbolically in the *formula* argument, so the design matrix for these variables must be found in the context of the model frame

```
## get design matrix for local structure
gnmData <- getModelFrame()
local <- model.matrix(formula, data = gnmData)
```

The plug-in function should return a list with at least the following three components:

`labels` a character vector of labels for the parameters (to which `gnm` will prefix the call to the plug-in function).

`predictor` a function which takes a vector of parameter estimates and returns either a vector of fitted values or a matrix whose columns are additive components of the fitted values.

`localDesignFunction` a function which takes the specified arguments *coef* (a vector of parameter estimates) and *predictor* (the result of the predictor function), and returns the local design matrix.

and optionally one further component

`start` a vector of default starting values for the parameters. `NA` may be used to indicate parameters which may be treated as linear for the purpose of finding starting values, given the non-`NA` values. See Section 4.2 for details of how these starting values will be used if provided and the generic default values that will be used otherwise.

As an example of a `start` component, `Dref` simply returns

```
rep(0.5, length(labels))
```

where `labels` is the vector of parameter labels to be returned as the `labels` component, for instance

```
c("A", "B", "1", "2", "3", "4", "5", "6", "7")
```

The `MultHomog` function provides a simple example of a `predictor` component:

```
predictor <- function(coef) {
    do.call("pprod", lapply(designList, "%*%", coef))
}
```

which computes the product of the vectors found by multiplying the design matrix for each factor in the interaction (held in `designList`) by the homogeneous coefficients (in `coef`). This function takes advantage of *lexical scoping*: `designList` is an object defined in `MultHomog`, which `predictor` is able to find because `predictor` is also defined in `MultHomog` and hence `MultHomog` is the enclosing environment of `predictor`.

The `localDesignFunction` created by `MultHomog` is slightly more complicated:

```
localDesignFunction <- function(coef, ...) {
    productList <- designList
    for (i in seq(designList))
        productList[[i]] <- designList[[i]] *
            drop(do.call("pprod", lapply(designList[-i], "%*%", coef)))
    do.call("psum", productList)
}
```

This function only uses the argument *coef*, but since the local design function returned by a plug-in function must also accept the argument *predictor*, further arguments are allowed by the use of the special argument '`...`'.

4 Controlling the Fitting Procedure

The `gnm` function has a number of arguments which affect the way a model will be fitted. Basic control parameters and starting values can be set by *control* and *start* respectively. Parameters can be constrained to zero by specifying a *constrain* argument. Finally parameters of a stratification factor can be handled more efficiently by specifying the term in an *eliminate* argument. These options are described in more detail below.

4.1 Using *control* with `gnmControl`

The *control* argument provides a way to specify the tolerance level for convergence, the number of starting iterations and the maximum number of main iterations, as well as the option to trace the deviance throughout the fitting process. By default, the *control* argument is a call to `gnmControl` using any arguments passed on from `gnm`. The `gnmControl` function creates a list of the control parameters, including any at their default values. For example

```
gnm(mu ~ R + C + Mult(-1 + R, -1 + C), tolerance = 1e-6,
iterStart = 3)
```

is equivalent to

```
gnm(mu ~ R + C + Mult(-1 + R, -1 + C),
    control = gnmControl(tolerance = 1e-6, iterStart = 3))
```

which is the same as

```
gnm(mu ~ R + C + Mult(-1 + R, -1 + C),
    control = list(tolerance = 1e-6, iterStart = 3, iterMax = 500,
    trace = FALSE))
```

4.2 Using *start*

In some contexts, the default starting values may not be appropriate and the algorithm will fail to converge, or perhaps only converge after a large number of iterations. Alternative starting values may be passed on to `gnm` by specifying a *start* argument. This should be a numeric vector of length equal to the number of parameters (or possibly the non-eliminated parameters, see Section 4.4), however missing starting values (`NA`s) are allowed.

If there is no user-specified starting value for a parameter, the default value is used. This feature is particularly useful when adding terms to a model, since the estimates from the original model can be used as starting values, as in this example:

```
    model1 <- gnm(mu ~ R + C + Mult(-1 + R, -1 + C))
    model2 <- gnm(mu ~ R + C + Mult(-1 + R, -1 + C, multiplicity = 2),
                  start = c(coef(model1), rep(NA, 10)))
```

The `gnm` call can be made with `method = "coef"` to identify the parameters of a model prior to estimation, to assist with the specification of arguments such as *start*.

The starting procedure used by `gnm` is as follows

1. Generate starting values θ_i for all parameters $i = 1, \ldots, p$ from the Uniform(-0.1, 0.1) distribution. Shift these values away from zero as follows

$$\theta_i = \begin{cases} \theta_i - 0.1 & \text{if } \theta_i < 1 \\ \theta_i + 0.1 & \text{otherwise} \end{cases}$$

2. Replace generic starting values with default starting values set by plug-in functions, where applicable.

3. Replace default starting values with any starting values specified by the *start* argument of `gnm`.

4. Compute the `glm` estimate of any parameters that may be treated as linear (i.e. those in linear terms or those with a default starting value of `NA` set by a plug-in function), offsetting the contribution to the predictor of any parameters specified by *start* or a plug-in function.

5. Run starting iterations: update one at a time any remaining nonlinear parameters not specified by *start* or a plug-in function, updating *all* parameters that may be treated as linear after each round of updates.

Note that no starting iterations (step 5) will be run if all parameters are specified by *start* or a plug-in function.

4.3 Using *constrain*

By default, `gnm` only imposes identifiability constraints on any linear terms in the model to be fitted. For these terms, the constraints are determined in the same way as they would be in `glm`. Any nonlinear terms will usually be over-parameterized unless constraints are imposed by the defining plug-in function (as in the case of `Dref`, for example). For a model with nonlinear terms that are over-parameterized, `gnm` will return a random parameterisation.

To illustrate this point, consider the following application of `gnm`, discussed later in Section 6.1:

```
> data(occupationalStatus)
> set.seed(1)
> RChomog1 <- gnm(Freq ~ origin + destination + Diag(origin, destination) +
+     Nonlin(MultHomog(origin, destination)), family = poisson,
+     data = occupationalStatus)

Running start-up iterations..
Running main iterations.......
Done
```

Running the analysis again from a different seed

```
> set.seed(2)
> RChomog2 <- eval(RChomog1$call)

Running start-up iterations..
Running main iterations......
Done
```

gives a different representation of the same model:

```
> compareCoef <- cbind(coef(RChomog1), coef(RChomog2))
> colnames(compareCoef) <- c("RChomog1", "RChomog2")
> compareCoef

                                   RChomog1    RChomog2
(Intercept)                      0.01031358  0.10631042
origin2                          0.52684390  0.51997443
origin3                          1.65525382  1.62956305
origin4                          1.99636593  1.95230159
origin5                          0.77767542  0.73307058
origin6                          2.85898522  2.79827815
origin7                          1.54820728  1.47440621
origin8                          1.29563149  1.21416423
destination2                     0.94585703  0.93898798
destination3                     1.99966968  1.97397893
destination4                     2.28479944  2.24073545
destination5                     1.67709218  1.63248789
destination6                     3.16246317  3.10175638
destination7                     2.29980341  2.22600286
destination8                     1.87100856  1.78954180
Diag(origin, destination)1       1.52666556  1.52666846
Diag(origin, destination)2       0.45600920  0.45600795
Diag(origin, destination)3      -0.01597343 -0.01598066
Diag(origin, destination)4       0.38918303  0.38918427
Diag(origin, destination)5       0.73851492  0.73851696
Diag(origin, destination)6       0.13474284  0.13474352
Diag(origin, destination)7       0.45763249  0.45763821
Diag(origin, destination)8       0.38847753  0.38846397
MultHomog(origin, destination).1 -1.54111773 -1.50965033
MultHomog(origin, destination).2 -1.32282516 -1.29135537
MultHomog(origin, destination).3 -0.72465413 -0.69319228
MultHomog(origin, destination).4 -0.14077778 -0.10930985
MultHomog(origin, destination).5 -0.12361117 -0.09214108
MultHomog(origin, destination).6  0.38814928  0.41961438
MultHomog(origin, destination).7  0.80429340  0.83575531
MultHomog(origin, destination).8  1.04785874  1.07933252
```

Even though the linear terms are constrained, the parameter estimates for these terms still change, because these terms are aliased with the higher order multiplicative interaction, which is unconstrained.

Additional constraints may be specified through the *constrain* argument of `gnm`. This argument indicates parameters that are to be constrained to zero in the fitting process. Parameters can be indicated by a logical vector, a vector of indices or, if `constrain = "pick"` they can be selected through a *Tk* dialog.

In the case above, constraining one level of the homogeneous multiplicative factor is sufficient to make the parameters of the nonlinear term identifiable, and hence all parameters in the model identifiable. For example, setting the last level of the homogeneous multiplicative factor to zero,

```
> multCoef <- coef(RChomog1)[grep("Mult", names(coef(RChomog1)))]
> set.seed(1)
> RChomogConstrained1 <- update(RChomog1, constrain = 31, start = c(rep(NA,
+     23), multCoef - multCoef[8]))

Running main iterations.
Done

> set.seed(2)
> RChomogConstrained2 <- eval(RChomogConstrained1$call)

Running main iterations.
Done

> identical(coef(RChomogConstrained1), coef(RChomogConstrained2))

[1] TRUE
```

gives the same results regardless of the random seed set beforehand.

It is not usually so straightforward to constrain all the parameters in a generalized nonlinear model. However, the simple constraints imposed by *constrain* are often sufficient to make particular coefficients of interest identifiable. The functions `checkEstimable` or `getContrasts`, described in Section 5, may be used to check whether particular contrasts are estimable.

4.4 Using *eliminate*

Sometimes a model will include a “stratification” factor which identifies units for which a unit-specific intercept should be estimated. It is often the case that such factors have a large number of levels and though they are required in the model, are not of direct interest in themselves.

The *eliminate* argument of `gnm` can be used to specify a stratification factor in a model, so that the factor can be handled more efficiently. The factor should be specified as a formula with a single term, for example

```
gnm(mu ~ -1 + unitID + A + B + Mult(A, B), eliminate = ~ unitID)
```

The use of *eliminate* makes the specification of a stratification factor in the model formula redundant, so the above call is equivalent to

```
gnm(mu ~ A + B + Mult(A, B), eliminate = ~ unitID)
```

or even

```
gnm(mu ~ -1 + A + B + Mult(A, B), eliminate = ~ unitID)
```

Specifying a stratification factor through *eliminate* has two advantages. First, computational speed is improved — substantially so if the number of eliminated parameters is large. Second, the eliminated parameters are excluded from the returned vector of coefficients, so that summaries of the model focus on the coefficients of interest.

The *eliminate* feature is useful, for example, when multinomial-response models are fitted by using the well known equivalence between multinomial and (conditional) Poisson likelihoods. In such situations the sufficient statistic involves a potentially large number of fixed multinomial row totals, and the corresponding parameters are of no substantive interest. For an example see Section 6.6 below.

The *eliminate* feature as implemented in *gnm* extends the earlier work of **?** to a broader class of models and to over-parameterized model representations.

anova	hatvalues	residuals
case.names	influence	rstandard
coef	labels	summary
cooks.distance	logLik	variable.names
deviance	model.frame	vcov
extractAIC	model.matrix	weights
family	plot	
formula	print	

Figure 1: Generic functions in the *base*, *stats* and *graphics* packages that can be used to analyse *gnm* objects.

add1	dummy.coef
alias	effects
confint	kappa
dfbeta	predict
dfbetas	proj
drop1	

Figure 2: Generic functions in the *base*, *stats* and *graphics* packages for which methods have been written for *glm* or *lm* objects, but which are *not* implemented for *gnm* objects.

5 Methods and Accessor functions

The `gnm` function returns an object of class `c("gnm", "glm", "lm")`. There are several methods that have been written for objects of class *glm* or *lm* to facilitate inspection of fitted models. Out of the generic functions in the *base*, *stats* and *graphics* packages for which methods have been written for *glm* or *lm* objects, Figure 1 shows those that can be used to analyse *gnm* objects, whilst Figure 2 shows those that are not implemented for *gnm* objects.

In addition to the accessor functions shown in Figure 1, the *gnm* package provides a new generic function called `termPredictors` that has methods for objects of class *gnm*, *glm* and *lm*. This function returns the additive contribution of each term to the predictor. See Section 2 for an example of its use.

Most of the methods listed in Figure 1 can be used as they would be for *glm* or *lm* objects, however care must be taken with *vcov*, as the variance-covariance matrix will depend on the parameterisation of the model. In particular, standard errors calculated using the variance-covariance matrix will only be valid for parameters or contrasts that are estimable!

The `checkEstimable` function can be used to check the estimability of contrasts. Consider the following model, that is described later in Section 6.3:

```
> data(cautres)
> doubleUnidiff <- gnm(Freq ~ election:vote + election:class:religion +
+     Mult(Exp(election - 1), religion:vote - 1) + Mult(Exp(election -
+     1), class:vote - 1), family = poisson, data = cautres)

Running start-up iterations..
Running main iterations.......
Done
```

The effects of the first constituent multiplier in the first multiplicative interaction are identified when the estimate of one of these effects is constrained to zero, say for the effect of the last level. The parameters to be estimated are then the differences between each effect and the effect of the last level. These differences can be represented by a contrast matrix as follows:

```
> coefs <- names(coef(doubleUnidiff))
> contrCoefs <- coefs[grep("Mult1.Factor1", coefs)]
> contrMatrix <- matrix(0, length(coefs), length(contrCoefs), dimnames = list(coefs,
+     contrCoefs))
> contrMatrix[contrCoefs, 1:(ncol(contrMatrix) - 1)] <- contr.sum(contrCoefs)
> contrMatrix[contrCoefs, 1:(ncol(contrMatrix) - 1)]

                        Mult1.Factor1.election1 Mult1.Factor1.election2
Mult1.Factor1.election1                       1                       0
Mult1.Factor1.election2                       0                       1
Mult1.Factor1.election3                       0                       0
Mult1.Factor1.election4                      -1                      -1
                        Mult1.Factor1.election3
Mult1.Factor1.election1                       0
Mult1.Factor1.election2                       0
Mult1.Factor1.election3                       1
Mult1.Factor1.election4                      -1
```

Then their estimability can be checked using `checkEstimable`

```
> checkEstimable(doubleUnidiff, contrMatrix)

Mult1.Factor1.election1 Mult1.Factor1.election2 Mult1.Factor1.election3
                   TRUE                    TRUE                    TRUE
Mult1.Factor1.election4
                     NA
```

which confirms that the effects for the other three levels are estimable when the effect for the last level is set to zero.

However, applying the equivalent constraint to the second constituent multiplier in the interaction is not sufficient to make the parameters in that multiplier estimable:

```
> coefs <- names(coef(doubleUnidiff))
> contrCoefs <- coefs[grep("Mult1.Factor2", coefs)]
> contrMatrix <- matrix(0, length(coefs), length(contrCoefs), dimnames = list(coefs,
+     contrCoefs))
> contrMatrix[contrCoefs, 1:(ncol(contrMatrix) - 1)] <- contr.sum(contrCoefs)
> checkEstimable(doubleUnidiff, contrMatrix)

Mult1.Factor2.religion1:vote1 Mult1.Factor2.religion2:vote1
                        FALSE                         FALSE
Mult1.Factor2.religion3:vote1 Mult1.Factor2.religion4:vote1
                        FALSE                         FALSE
Mult1.Factor2.religion1:vote2 Mult1.Factor2.religion2:vote2
                        FALSE                         FALSE
Mult1.Factor2.religion3:vote2 Mult1.Factor2.religion4:vote2
                        FALSE                            NA
```

To investigate simple "sum to zero" contrasts such as those above, it is easiest to use the `getContrasts` function, which checks the estimability of the contrasts and returns the parameter estimates with their standard errors. Returning to the example of the first constituent multiplier in the first multiplicative interaction term, the differences between each election and the last can be obtained as follows:

```
> coefs.of.interest <- grep("Mult1.Factor1", names(coef(doubleUnidiff)))
> getContrasts(doubleUnidiff, coefs.of.interest)

Loading required package: qvcalc
[[1]]
[[1]]$summary
```

```
                             estimate         se   quasi.se
Mult1.Factor1.election1 0.32834637 0.12213023 0.09803072
Mult1.Factor1.election2 0.24052784 0.09116483 0.05702819
Mult1.Factor1.election3 0.06682575 0.09906919 0.06812240
Mult1.Factor1.election4 0.00000000 0.00000000 0.07168295

[[1]]$relative.errors
[1] "-0.6%" "0.8%"
```

Attempting to obtain the equivalent contrasts for the second (religion-vote association) multiplier produces the following result:

```
> coefs.of.interest <- grep("Mult1.Factor2", names(coef(doubleUnidiff)))
> getContrasts(doubleUnidiff, coefs.of.interest)

Mult1.Factor2.religion1:vote1 Mult1.Factor2.religion2:vote1
                        FALSE                         FALSE
Mult1.Factor2.religion3:vote1 Mult1.Factor2.religion4:vote1
                        FALSE                         FALSE
Mult1.Factor2.religion1:vote2 Mult1.Factor2.religion2:vote2
                        FALSE                         FALSE
Mult1.Factor2.religion3:vote2 Mult1.Factor2.religion4:vote2
                        FALSE                            NA
Note: not all of the specified contrasts in this set are estimable
[[1]]
[[1]]$summary
                              estimate se
Mult1.Factor2.religion4:vote2        0  0

[[1]]$relative.errors
NULL
```

6 Examples

This section provides some examples of the wide range of models that may be fitted using the *gnm* package. Sections 6.1, 6.2 and 6.3 consider various models for contingency tables; Section 6.4 considers AMMI and GAMMI models which are typically used in agricultural applications, and Section 6.6 considers the stereotype model, which is used to model an ordinal response.

6.1 Row-column Association Models

There are several models that have been proposed for modelling the relationship between the cell means of a contingency table and the cross-classifying factors. The following examples consider the row-column association models proposed by **?**. The examples shown use data from two-way contingency tables, but the *gnm* package can also be used to fit the equivalent models for higher order tables.

6.1.1 RC(1) model

The RC(1) model is a row and column association model with the interaction between row and column factors represented by one component of the multiplicative interaction. If the rows are indexed by r and the columns by c, then the log-multiplicative form of the RC(1) model for the cell means μ_{rc} is given by

$$\log \mu_{rc} = \alpha_r + \beta_c + \gamma_r \delta_c.$$

We shall fit this model to the `mentalHealth` data set taken from **?** page 381, which is a two-way contingency table classified by the child's mental impairment (MHS) and the parents' socioeconomic status (SES). Although both of these factors are ordered, we do not wish to use polynomial contrasts in the model, so we begin by setting the contrasts attribute of these factors to `"treatment"`:

```
> set.seed(1)
> data(mentalHealth)
> mentalHealth$MHS <- C(mentalHealth$MHS, treatment)
> mentalHealth$SES <- C(mentalHealth$SES, treatment)
```

The *gnm* model is then specified as follows, using the poisson family with a log link function:

```
> RC1model <- gnm(count ~ SES + MHS + Mult(-1 + SES, -1 + MHS),
+     family = poisson, data = mentalHealth)

Running start-up iterations..
Running main iterations.....
Done

> RC1model

Call:

gnm(formula = count ~ SES + MHS + Mult(-1 + SES, -1 + MHS), family = poisson,
    data = mentalHealth)


Coefficients:
              (Intercept)                          SESB
                 3.831001                     -0.067413
                     SESC                          SESD
                 0.109938                      0.404937
                     SESE                          SESF
                 0.025196                     -0.200766
                  MHSmild                   MHSmoderate
                 0.713248                      0.205317
              MHSimpaired            Mult1.Factor1.SESA
                 0.252311                      0.341189
       Mult1.Factor1.SESB            Mult1.Factor1.SESC
                 0.343966                      0.115341
       Mult1.Factor1.SESD            Mult1.Factor1.SESE
                -0.005967                     -0.305568
       Mult1.Factor1.SESF         Mult1.Factor2.MHSwell
                -0.551688                      0.934517
    Mult1.Factor2.MHSmild     Mult1.Factor2.MHSmoderate
                 0.094601                     -0.056957
Mult1.Factor2.MHSimpaired
                -0.754612

Deviance:            3.570562
Pearson chi-squared: 3.568094
Residual df:         8
```

The row scores (parameters 10 to 15) and the column scores (parameters 16 to 19) of the multiplicative interaction can be normalized as in Agresti's eqn (9.15):

```
> rowProbs <- with(mentalHealth, tapply(count, SES, sum)/sum(count))
> colProbs <- with(mentalHealth, tapply(count, MHS, sum)/sum(count))
> rowScores <- coef(RC1model)[10:15]
> colScores <- coef(RC1model)[16:19]
> rowScores <- rowScores - sum(rowScores * rowProbs)
> colScores <- colScores - sum(colScores * colProbs)
> beta1 <- sqrt(sum(rowScores^2 * rowProbs))
> beta2 <- sqrt(sum(colScores^2 * colProbs))
> assoc <- list(beta = beta1 * beta2, mu = rowScores/beta1, nu = colScores/beta2)
> assoc
```

```
$beta
[1] 0.1664870

$mu
Mult1.Factor1.SESA Mult1.Factor1.SESB Mult1.Factor1.SESC Mult1.Factor1.SESD
        1.11234361         1.12145891         0.37108476        -0.02706533
Mult1.Factor1.SESE Mult1.Factor1.SESF
       -1.01039041        -1.81818542

$nu
    Mult1.Factor2.MHSwell     Mult1.Factor2.MHSmild Mult1.Factor2.MHSmoderate
                1.6774975                 0.1404000                -0.1369601
Mult1.Factor2.MHSimpaired
               -1.4137095
```

6.1.2 RC(2) model

The RC(1) model can be extended to an RC(m) model with m components of the multiplicative interaction. For example, the RC(2) model is given by

$$\log \mu_{rc} = \alpha_r + \beta_c + \gamma_r\delta_c + \theta_r\phi_c.$$

Extra instances of the multiplicative interaction can be specified by the *multiplicity* argument of `Mult`, so the RC(2) model can be fitted to the `mentalHealth` data as follows

```
> RC2model <- gnm(count ~ SES + MHS + Mult(-1 + SES, -1 + MHS,
+     multiplicity = 2), family = poisson, data = mentalHealth)
```

```
Running start-up iterations..
Running main iterations.......
Done
```

```
> RC2model
```

```
Call:

gnm(formula = count ~ SES + MHS + Mult(-1 + SES, -1 + MHS, multiplicity = 2),
    family = poisson, data = mentalHealth)


Coefficients:
              (Intercept)                          SESB
                  3.85511                      -0.06447
                     SESC                          SESD
                  0.11139                       0.38471
```

```
                     SESE                        SESF
                  0.01090                    -0.18477
                  MHSmild                 MHSmoderate
                  0.69870                     0.17003
              MHSimpaired         Mult1.Factor1.SESA
                  0.22888                     0.94938
       Mult1.Factor1.SESB         Mult1.Factor1.SESC
                  0.99486                     0.33903
       Mult1.Factor1.SESD         Mult1.Factor1.SESE
                 -0.17301                    -0.91537
       Mult1.Factor1.SESF      Mult1.Factor2.MHSwell
                 -1.39141                     0.35835
    Mult1.Factor2.MHSmild  Mult1.Factor2.MHSmoderate
                  0.03799                    -0.02140
Mult1.Factor2.MHSimpaired         Mult2.Factor1.SESA
                 -0.28068                    -0.17737
       Mult2.Factor1.SESB         Mult2.Factor1.SESC
                 -0.25127                    -0.16575
       Mult2.Factor1.SESD         Mult2.Factor1.SESE
                  0.29054                     0.22753
       Mult2.Factor1.SESF      Mult2.Factor2.MHSwell
                 -0.45487                     0.30770
    Mult2.Factor2.MHSmild  Mult2.Factor2.MHSmoderate
                  0.09770                    -0.25568
Mult2.Factor2.MHSimpaired
                  0.06702

Deviance:            0.5225353
Pearson chi-squared: 0.5233306
Residual df:         3
```

6.1.3 Homogeneous effects

If the row and column factors have the same levels, or perhaps some levels in common, then the row-column interaction could be modelled by a multiplicative interaction with homogeneous effects, that is

$$\log \mu_{rc} = \alpha_r + \beta_c + \gamma_r \gamma_c.$$

For example, the `occupationalStatus` data set from **?** is a contingency table classified by the occupational status of fathers (origin) and their sons (destination). **?** fits a row-column association model with homogeneous effects to these data after deleting the cells on the main diagonal. Equivalently we can account for the diagonal effects by a separate `Diag` term:

```
> data(occupationalStatus)
> RChomog <- gnm(Freq ~ origin + destination + Diag(origin, destination) +
+     Nonlin(MultHomog(origin, destination)), family = poisson,
+     data = occupationalStatus)

Running start-up iterations..
Running main iterations.........
Done

> RChomog
```

```
Call:
gnm(formula = Freq ~ origin + destination + Diag(origin, destination) +
    Nonlin(MultHomog(origin, destination)), family = poisson,
    data = occupationalStatus)


Coefficients:
                     (Intercept)                              origin2
                        -0.12078                              0.53600
                         origin3                              origin4
                         1.68951                              2.05513
                         origin5                              origin6
                         0.83716                              2.93994
                         origin7                              origin8
                         1.64663                              1.40427
                    destination2                         destination3
                         0.95502                              2.03393
                    destination4                         destination5
                         2.34356                              1.73657
                    destination6                         destination7
                         3.24342                              2.39822
                    destination8           Diag(origin, destination)1
                         1.97965                              1.52667
      Diag(origin, destination)2           Diag(origin, destination)3
                         0.45601                             -0.01598
      Diag(origin, destination)4           Diag(origin, destination)5
                         0.38918                              0.73852
      Diag(origin, destination)6           Diag(origin, destination)7
                         0.13474                              0.45764
      Diag(origin, destination)8  MultHomog(origin, destination).1
                         0.38847                             -1.58308
MultHomog(origin, destination).2  MultHomog(origin, destination).3
                        -1.36478                             -0.76662
MultHomog(origin, destination).4  MultHomog(origin, destination).5
                        -0.18274                             -0.16557
MultHomog(origin, destination).6  MultHomog(origin, destination).7
                         0.34619                              0.76233
MultHomog(origin, destination).8
                         1.00590

Deviance:            32.56098
Pearson chi-squared: 31.20716
Residual df:         34
```

To determine whether it would be better to allow for heterogeneous effects on the association of the fathers' occupational status and the sons' occupational status, we can compare this model to the RC(1) model for these data:

```
> data(occupationalStatus)
> RCheterog <- gnm(Freq ~ origin + destination + Diag(origin, destination) +
+     Mult(origin, destination), family = poisson, data = occupationalStatus)
```

```
Running start-up iterations..
Running main iterations.........
Done
```

```
> RChomog$dev - RCheterog$dev

[1] 3.411823

> RChomog$df.residual - RCheterog$df.residual

[1] 6
```

In this case there is little gain in allowing heterogeneous effects.

6.2 Diagonal Reference Models

Diagonal reference models, proposed by **??**, are designed for contingency tables classified by factors with the same levels. The cell means are modelled as a function of the diagonal effects, i.e., the mean responses of the 'diagonal' cells in which the levels of the row and column factors are the same.

`Dref` example 1: Political consequences of social mobility

To illustrate the use of diagonal reference models we shall use the `voting` data from **?**. The data come from the 1987 British general election and are the percentage voting Labour in groups cross-classified by the class of the head of household (`destination`) and the class of their father (`origin`). In order to weight these percentages by the group size, we first back-transform them to the counts of those voting Labour and those not voting Labour:

```
> set.seed(1)
> data(voting)
> count <- with(voting, percentage/100 * total)
> yvar <- cbind(count, voting$total - count)
```

The grouped percentages may be modelled by a basic diagonal reference model, that is, a weighted sum of the diagonal effects for the corresponding origin and destination classes. This model may be expressed as

$$\mu_{od} = \frac{e^{\delta_1}}{e^{\delta_1} + e^{\delta_2}}\gamma_o + \frac{e^{\delta_2}}{e^{\delta_1} + e^{\delta_2}}\gamma_d.$$

See Section 3.2.2 for more detail on the parameterisation.

The basic diagonal reference model may be fitted using `gnm` as follows

```
> classMobility <- gnm(yvar ~ Nonlin(Dref(origin, destination)),
+     family = binomial, data = voting)

Running main iterations.......
Done

> classMobility

Call:

gnm(formula = yvar ~ Nonlin(Dref(origin, destination)), family = binomial,
    data = voting)


Coefficients:
                          (Intercept)       Dref(origin, destination).origin
                              -1.6055                                 0.3738
Dref(origin, destination).destination            Dref(origin, destination).1
                               0.6262                                -0.5723
```

```
           Dref(origin, destination).2           Dref(origin, destination).3
                                0.4729                               -0.3494
           Dref(origin, destination).4           Dref(origin, destination).5
                                1.0272                                1.6459

Deviance:            21.22093
Pearson chi-squared: 18.95311
Residual df:         19
```

and the origin and destination weights can be evaluated as below

```
> prop.table(exp(coef(classMobility)[2:3]))

     Dref(origin, destination).origin Dref(origin, destination).destination
                            0.4372474                             0.5627526
```

This model is slightly different from that reported by **?**. The reason for this is unclear: we are confident that the above results are correct for the data as given in **?**, but have not been able to confirm that the data as printed in the journal were exactly as used in Clifford and Heath's analysis.

? suggest that movements in and out of the salariat (class 1) should be treated differently from movements between the lower classes (classes 2 - 5), since the former has a greater effect on social status. Thus they propose the following model

$$\mu_{od} = \begin{cases} \dfrac{e^{\delta_1}}{e^{\delta_1}+e^{\delta_2}}\gamma_o + \dfrac{e^{\delta_2}}{e^{\delta_1}+e^{\delta_2}}\gamma_d & \text{if } i = 1 \\[2ex] \dfrac{e^{\delta_3}}{e^{\delta_3}+e^{\delta_4}}\gamma_o + \dfrac{e^{\delta_4}}{e^{\delta_3}+e^{\delta_4}}\gamma_d & \text{if } j = 1 \\[2ex] \dfrac{e^{\delta_5}}{e^{\delta_5}+e^{\delta_6}}\gamma_o + \dfrac{e^{\delta_6}}{e^{\delta_5}+e^{\delta_6}}\gamma_d & \text{if } i \neq 1 \text{ and } j \neq 1 \end{cases}$$

To fit this model we define factors indicating movement in (upward) and out (downward) of the salariat

```
> upward <- with(voting, origin != 1 & destination == 1)
> downward <- with(voting, origin == 1 & destination != 1)
```

Then the diagonal reference model with separate weights for socially mobile groups can be estimated as follows

```
> socialMobility <- gnm(yvar ~ Nonlin(Dref(origin, destination,
+     formula = ~1 + downward + upward)), family = binomial, data = voting)

Running main iterations.......
Done

> socialMobility

Call:
gnm(formula = yvar ~ Nonlin(Dref(origin, destination, formula = ~1 +
    downward + upward)), family = binomial, data = voting)


Coefficients:
                                                                        (Intercept)
                                                                            -1.5844
      Dref(origin, destination, formula = ~1 + downward + upward).origin.(Intercept)
                                                                             0.3066
```

```
     Dref(origin, destination, formula = ~1 + downward + upward).origin.downwardTRUE
                                                                          0.9054
       Dref(origin, destination, formula = ~1 + downward + upward).origin.upwardTRUE
                                                                          0.4699
 Dref(origin, destination, formula = ~1 + downward + upward).destination.(Intercept)
                                                                          0.6934
Dref(origin, destination, formula = ~1 + downward + upward).destination.downwardTRUE
                                                                          0.0946
  Dref(origin, destination, formula = ~1 + downward + upward).destination.upwardTRUE
                                                                          0.5301
                       Dref(origin, destination, formula = ~1 + downward + upward).1
                                                                         -0.4732
                       Dref(origin, destination, formula = ~1 + downward + upward).2
                                                                          0.4717
                       Dref(origin, destination, formula = ~1 + downward + upward).3
                                                                         -0.4104
                       Dref(origin, destination, formula = ~1 + downward + upward).4
                                                                          1.0153
                       Dref(origin, destination, formula = ~1 + downward + upward).5
                                                                          1.6420

Deviance:            18.97407
Pearson chi-squared: 17.07495
Residual df:         17
```

The weights for those moving into the salariat, those moving out of the salariat and those in any other group, can be evaluated as below

```
> prop.table(exp(coef(socialMobility)[c(4, 7)] + coef(socialMobility)[c(2,
+     5)]))

     Dref(origin, destination, formula = ~1 + downward + upward).origin.upwardTRUE
                                                                     0.3900752
Dref(origin, destination, formula = ~1 + downward + upward).destination.upwardTRUE
                                                                     0.6099248

> prop.table(exp(coef(socialMobility)[c(3, 6)] + coef(socialMobility)[c(2,
+     5)]))

     Dref(origin, destination, formula = ~1 + downward + upward).origin.downwardTRUE
                                                                       0.6044571
Dref(origin, destination, formula = ~1 + downward + upward).destination.downwardTRUE
                                                                       0.3955429

> prop.table(exp(coef(socialMobility)[c(2, 5)]))

     Dref(origin, destination, formula = ~1 + downward + upward).origin.(Intercept)
                                                                      0.4045022
Dref(origin, destination, formula = ~1 + downward + upward).destination.(Intercept)
                                                                      0.5954978
```

Again, the results differ slightly from those reported by **?**, but the essence of the results is the same: the origin weight is much larger for the downwardly mobile groups than for the other groups. The weights for the upwardly mobile groups are very similar to the base level weights, so the model may be simplified by only fitting separate weights for the downwardly mobile groups:

```
> downwardMobility <- gnm(yvar ~ Nonlin(Dref(origin, destination,
+     formula = ~1 + downward)), family = binomial, data = voting)

Running main iterations.......
Done

> downwardMobility

Call:
gnm(formula = yvar ~ Nonlin(Dref(origin, destination, formula = ~1 +
    downward)), family = binomial, data = voting)


Coefficients:
                                                                   (Intercept)
                                                                      -1.58578
      Dref(origin, destination, formula = ~1 + downward).origin.(Intercept)
                                                                       0.29561
     Dref(origin, destination, formula = ~1 + downward).origin.downwardTRUE
                                                                       0.90538
 Dref(origin, destination, formula = ~1 + downward).destination.(Intercept)
                                                                       0.70439
Dref(origin, destination, formula = ~1 + downward).destination.downwardTRUE
                                                                       0.09462
                       Dref(origin, destination, formula = ~1 + downward).1
                                                                      -0.48409
                       Dref(origin, destination, formula = ~1 + downward).2
                                                                       0.47924
                       Dref(origin, destination, formula = ~1 + downward).3
                                                                      -0.40588
                       Dref(origin, destination, formula = ~1 + downward).4
                                                                       1.01270
                       Dref(origin, destination, formula = ~1 + downward).5
                                                                       1.64207

Deviance:            18.99389
Pearson chi-squared: 17.09983
Residual df:         18

> prop.table(exp(coef(downwardMobility)[c(3, 5)] + coef(downwardMobility)[c(2,
+     4)]))

     Dref(origin, destination, formula = ~1 + downward).origin.downwardTRUE
                                                                     0.5991644
Dref(origin, destination, formula = ~1 + downward).destination.downwardTRUE
                                                                     0.4008356

> prop.table(exp(coef(downwardMobility)[c(2, 4)]))

     Dref(origin, destination, formula = ~1 + downward).origin.(Intercept)
                                                                    0.3992041
Dref(origin, destination, formula = ~1 + downward).destination.(Intercept)
                                                                    0.6007959
```

`Dref` example 2: Conformity to parental rules

Another application of diagonal reference models is given by **?**. The data from this paper are not publicly available[2], but we shall show how the models presented in the paper may be estimated using `gnm`.

The data relate to the value parents place on their children conforming to their rules. There are two response variables: the mother's conformity score (MCFM) and the father's conformity score (FCFF). The data are cross-classified by two factors describing the education level of the mother (MOPLM) and the father (FOPLF), and there are six further covariates (AGEM, MRMM, FRMF, MWORK, MFCM and FFCF).

In their baseline model for the mother's conformity score, **?** include five of the six covariates (leaving out the father's family conflict score, FCFF) and a diagonal reference term with constant weights based on the two education factors. This model may be expressed as

$$\mu_{rc} = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \frac{e^{\delta_1}}{e^{\delta_1} + e^{\delta_2}}\gamma_r + \frac{e^{\delta_2}}{e^{\delta_1} + e^{\delta_2}}\gamma_c.$$

The baseline model can be fitted as follows:

```
> set.seed(1)
> A <- gnm(MCFM ~ -1 + AGEM + MRMM + FRMF + MWORK + MFCM +
+          Nonlin(Dref(MOPLM, FOPLF)), family = gaussian, data = conformity,
+          verbose = FALSE)
> A

Call:
gnm(formula = MCFM ~ -1 + AGEM + MRMM + FRMF + MWORK + MFCM +
    Nonlin(Dref(MOPLM, FOPLF)), family = gaussian, data = conformity,
    verbose = FALSE)


Coefficients:
                     AGEM                       MRMM                       FRMF
                  0.06364                   -0.32425                   -0.25324
                    MWORK                       MFCM  Dref(MOPLM, FOPLF).MOPLM
                 -0.06430                   -0.06043                    0.34389
Dref(MOPLM, FOPLF).FOPLF      Dref(MOPLM, FOPLF).1      Dref(MOPLM, FOPLF).2
                  0.65611                    4.95123                    4.86328
    Dref(MOPLM, FOPLF).3      Dref(MOPLM, FOPLF).4      Dref(MOPLM, FOPLF).5
                  4.86458                    4.72342                    4.43516
    Dref(MOPLM, FOPLF).6      Dref(MOPLM, FOPLF).7
                  4.18873                    4.43379

Deviance:            425.3389
Pearson chi-squared: 425.3389
Residual df:         576
```

Due to the constraints imposed on the weights in the diagonal reference term, the coefficients of model `A` are the unique solutions. Therefore these estimates should correspond to those reported in Table 4 of **?**. The weights in the diagonal reference term can be evaluated as follows:

```
> prop.table(exp(coef(A)[6:7]))
Dref(MOPLM, FOPLF).MOPLM Dref(MOPLM, FOPLF).FOPLF
               0.4225734                0.5774266
```

[2] We thank Frans van der Slik for his kindness in sending us the data.

giving the values reported by **?**. All the other coefficients of model A are the same as those reported by **?** except the coefficients of the mother's gender role (MRMM) and the father's gender role (FRMF). **?** reversed the signs of the coefficients of these factors since they were coded in the direction of liberal values, unlike the other covariates. However, simply reversing the signs of these coefficients does not give the same model, since the estimates of these coefficients are not independent of the estimates of the diagonal effects. For consistent interpretation of the covariate coefficients, it is better to recode the gender role factors as follows:

```
> MRMM2 <- as.numeric(!conformity$MRMM)
> FRMF2 <- as.numeric(!conformity$FRMF)
> A <- gnm(MCFM ~ -1 + AGEM + MRMM2 + FRMF2 + MWORK + MFCM +
+          Nonlin(Dref(MOPLM, FOPLF)), family = gaussian, data = conformity,
+          verbose = FALSE)
> A

Call:
gnm(formula = MCFM ~ -1 + AGEM + MRMM2 + FRMF2 + MWORK + MFCM +
    Nonlin(Dref(MOPLM, FOPLF)), family = gaussian, data = conformity,
    verbose = FALSE)


Coefficients:
                    AGEM                     MRMM2                     FRMF2
                 0.06364                   0.32425                   0.25324
                   MWORK                      MFCM  Dref(MOPLM, FOPLF).MOPLM
                -0.06430                  -0.06043                   0.34389
Dref(MOPLM, FOPLF).FOPLF      Dref(MOPLM, FOPLF).1      Dref(MOPLM, FOPLF).2
                 0.65611                   4.37373                   4.28578
    Dref(MOPLM, FOPLF).3      Dref(MOPLM, FOPLF).4      Dref(MOPLM, FOPLF).5
                 4.28708                   4.14593                   3.85766
    Dref(MOPLM, FOPLF).6      Dref(MOPLM, FOPLF).7
                 3.61123                   3.85629

Deviance:            425.3389
Pearson chi-squared: 425.3389
Residual df:         576
```

The coefficients of the covariates are now as reported by **?**, but the diagonal effects have been adjusted appropriately.

? compare the baseline model for the mother's conformity score to several other models in which the weights in the diagonal reference term are dependent on one of the covariates. One particular model they consider incorporates an interaction of the weights with the mother's conflict score as follows:

$$\mu_{rc} = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \beta_4 x_4 + \beta_5 x_5 + \frac{e^{\xi_1 + \beta_1 x}}{e^{\xi_1 + \beta_1 x} + e^{\xi_2 + \beta_2 x}} \gamma_r + \frac{e^{\xi_2 + \beta_2 x}}{e^{\xi_1 + \beta_1 x} + e^{\xi_2 + \beta_2 x}} \gamma_c.$$

This model can be fitted as below, using the original coding for the gender role factors for ease of comparison to the results reported by **?**,

```
> F <- gnm(MCFM ~ -1 + AGEM + MRMM + FRMF + MWORK + MFCM +
+          Nonlin(Dref(MOPLM, FOPLF, formula = ~ 1 + MFCM)), family = gaussian,
+          data = conformity, verbose = FALSE)
> F

Call:
gnm(formula = MCFM ~ -1 + AGEM + MRMM + FRMF + MWORK + MFCM +
    Nonlin(Dref(MOPLM, FOPLF, formula = ~1 + MFCM)), family = gaussian,
    data = conformity, verbose = FALSE)


Coefficients:
                                                     AGEM
                                                  0.05818
                                                     MRMM
                                                 -0.32701
                                                     FRMF
                                                 -0.25772
                                                    MWORK
                                                 -0.07847
                                                     MFCM
                                                 -0.01694
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).MOPLM.(Intercept)
                                                  1.03516
       Dref(MOPLM, FOPLF, formula = ~1 + MFCM).MOPLM.MFCM
                                                 -1.77703
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.(Intercept)
                                                 -0.03516
       Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.MFCM
                                                  2.77703
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).1
                                                  4.82477
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).2
                                                  4.88066
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).3
                                                  4.83969
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).4
                                                  4.74849
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).5
                                                  4.42019
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).6
                                                  4.17956
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).7
                                                  4.40819

Deviance:            420.9022
Pearson chi-squared: 420.9022
Residual df:         575
```

In this case there are two sets of weights, one for when the mother's conflict score is less than average (coded as zero) and one for when the score is greater than average (coded as one). These can be evaluated as follows:

```
> prop.table(exp(coef(F))[c(6,8)])
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).MOPLM.(Intercept)
                                                0.7446585
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.(Intercept)
                                                0.2553415
> prop.table(exp(coef(F)[c(7,9)] + coef(F)[c(6,8)]))
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).MOPLM.MFCM
                                        0.02977851
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.MFCM
```

```
                                    0.97022149
```

giving the same weights as in Table 4 of **?**.

6.3 Uniform Difference (UNIDIFF) Models

Uniform difference models (**??**) use a simplified three-way interaction to provide an interpretable model of contingency tables classified by three or more variables. For example, the uniform difference model for a three-way contingency table, also known as the UNIDIFF model, is given by

$$\mu_{ijk} = \alpha_{ik} + \beta_{jk} + \exp(\delta_k)\gamma_{ij}.$$

The γ_{ij} represent a pattern of association that varies in strength over the dimension indexed by k, and $\exp(\delta_k)$ represents the relative strength of that association at level k.

This model can be applied to the `yaish` data set (**??**), which is a contingency table cross-classified by father's social class (`orig`), son's social class (`dest`) and son's education level (`educ`). In this case, we can consider the importance of the association between the social class of father and son across the education levels:

```
> set.seed(1)
> data(yaish)
> unidiff <- gnm(Freq ~ educ:orig + educ:dest + Mult(Exp(-1 + educ),
+     orig:dest), family = poisson, data = yaish)

Running start-up iterations..
Running main iterations.................................
Done

> coefs.of.interest <- grep("Mult1.Factor1", names(coef(unidiff)))
> coef(unidiff)[coefs.of.interest]

Mult1.Factor1.educ1 Mult1.Factor1.educ2 Mult1.Factor1.educ3 Mult1.Factor1.educ4
         1.08253343          0.86578203          0.35129837          0.05240976
Mult1.Factor1.educ5
        -1.15532838
```

The `coefs.of.interest` are the multipliers of the association between the social class of father and son. We can contrast each multiplier to that of the highest education level and obtain the standard errors for these parameters as follows:

```
> getContrasts(unidiff, coefs.of.interest)

[[1]]
[[1]]$summary
                    estimate        se   quasi.se
Mult1.Factor1.educ1 2.237862 0.9411152 0.09743542
Mult1.Factor1.educ2 2.021110 0.9435045 0.12813630
Mult1.Factor1.educ3 1.506627 0.9535672 0.20930038
Mult1.Factor1.educ4 1.207738 0.9780882 0.32517402
Mult1.Factor1.educ5 0.000000 0.0000000 0.93130651

[[1]]$relative.errors
[1] "-0.9%" "1.4%"
```

Four-way contingency tables may sometimes be described by a "double UNIDIFF" model

$$\mu_{ijkl} = \alpha_{il} + \beta_{jkl} + \exp(\delta_l)\gamma_{ij} + \exp(\phi_l)\theta_{ik},$$

where the strengths of two, two-way associations with a common variable are estimated across the levels of the fourth variable. The `cautres` data set, from **?**, can be used to illustrate the application of the double UNIDIFF model. This data set is classified by the variables vote, class, religion and election. Using a double UNIDIFF model, we can see how the association between class and vote, and the association between religion and vote, differ between the most recent election and the other elections:

```
> set.seed(1)
> data(cautres)
> doubleUnidiff <- gnm(Freq ~ election:vote + election:class:religion +
+     Mult(Exp(-1 + election), religion:vote) + Mult(Exp(-1 + election),
+     class:vote), family = poisson, data = cautres)

Running start-up iterations..
Running main iterations........
Done

> getContrasts(doubleUnidiff, grep("Mult1.Factor1", names(coef(doubleUnidiff))))

[[1]]
[[1]]$summary
                          estimate         se   quasi.se
Mult1.Factor1.election1 0.32834578 0.12213024 0.09803077
Mult1.Factor1.election2 0.24052783 0.09116478 0.05702818
Mult1.Factor1.election3 0.06682591 0.09906915 0.06812239
Mult1.Factor1.election4 0.00000000 0.00000000 0.07168289

[[1]]$relative.errors
[1] "-0.6%" "0.8%"

> getContrasts(doubleUnidiff, grep("Mult2.Factor1", names(coef(doubleUnidiff))))

[[1]]
[[1]]$summary
                           estimate        se   quasi.se
Mult2.Factor1.election1 -0.36182804 0.2534753 0.22854386
Mult2.Factor1.election2  0.31990886 0.1320022 0.07395883
Mult2.Factor1.election3  0.08754582 0.1446833 0.09475936
Mult2.Factor1.election4  0.00000000 0.0000000 0.10934808

[[1]]$relative.errors
[1] "0%" "0%"
```

6.4 Generalized Additive Main Effects and Multiplicative Interaction (GAMMI) Models

Generalized additive main effects and multiplicative interaction models, or GAMMI models, were motivated by two-way contingency tables and comprise the row and column main effects plus one or more components of the multiplicative interaction. The singular value corresponding to each multiplicative component is often factored out, as a measure of the strength of association between the row and column scores, indicating the importance of the component, or axis.

For cell means μ_{rc} a GAMMI-K model has the form

$$g(\mu_{rc}) = \alpha_r + \beta_c + \sum_{k=1}^{K} \sigma_k \gamma_{kr} \delta_{kc},$$

in which g is a link function, α_r and β_c are the row and column main effects, γ_{kr} and δ_{kc} are the row and column scores for multiplicative component k and σ_k is the singular value for component k. The number of multiplicative components, K, is less than or equal to the rank of the matrix of residuals from the main effects.

The row-column association models discussed in Section 6.1 are examples of GAMMI models, with a log link and poisson variance. Here we illustrate the use of an AMMI model, which is a GAMMI model with an identity link and a constant variance.

We shall use the `wheat` data set taken from **?**, which gives wheat yields measured over ten years. First we scale these yields and create a new treatment factor, so that we can reproduce the analysis of **?**:

```
> set.seed(1)
> data(wheat)
> yield.scaled <- wheat$yield * sqrt(3/1000)
> treatment <- interaction(wheat$tillage, wheat$summerCrop, wheat$manure,
+     wheat$N, sep = "")
```

Now we can fit the AMMI-1 model, to the scaled yields using the combined treatment factor and the year factor from the `wheat` dataset:

```
> bilinear1 <- gnm(yield.scaled ~ year + treatment + Mult(year,
+     treatment), family = gaussian, data = wheat)
```

```
Running start-up iterations..
Running main iterations...................
Done
```

and compare the AMMI-1 model to the main effects model

```
> mainEffects <- glm(yield.scaled ~ year + treatment, family = gaussian,
+     data = wheat)
> anova(mainEffects, bilinear1)
```

```
Analysis of Deviance Table

Model 1: yield.scaled ~ year + treatment
Model 2: yield.scaled ~ year + treatment + Mult(year, treatment)
  Resid. Df Resid. Dev  Df Deviance
1       207     279515
2       176     128383  31   151133
```

giving the same results as in Table 1 of **?** (up to error caused by rounding).

6.5 Biplot Models

Biplots are used to display two-dimensional data transformed into a space spanned by linearly independent vectors, such as the principal components or singular vectors. The plot represents the levels of the two classifying factors by their scores on the two axes which show the most information about the data, for example the first two principal components.

A rank-n model is a model based on the first n components of the decomposition. In the case of a singular value decomposition, this is equivalent to a model with n components of the multiplicative interaction.

To illustrate the use of biplot models, we shall use the `barley` data set which describes the incidence of leaf blotch over ten varieties of barley grown at nine sites (**??**). The biplot model is fitted as follows:

```
> data(barley)
> set.seed(1)
> biplotModel <- gnm(y ~ -1 + Mult(site, variety, multiplicity = 2),
+     family = wedderburn, data = barley)
```

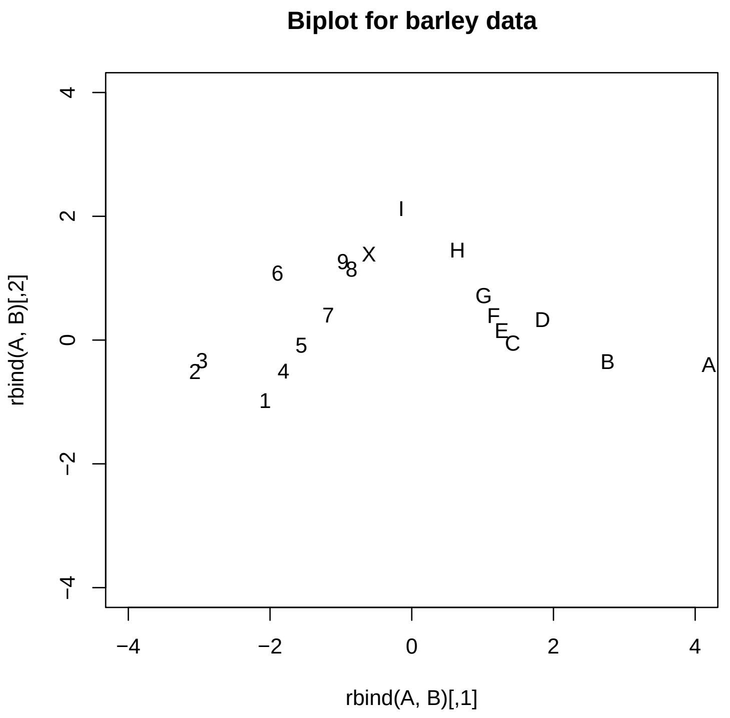
```
Running start-up iterations..
Running main iterations.....................................................
.........................
Done
```

using the `wedderburn` family function introduced in Section 2. Matrices of the row and column scores for the first two singular vectors can then be obtained by:

```
> barleySVD <- svd(matrix(biplotModel$predictors, 10, 9))
> A <- sweep(barleySVD$v, 2, sqrt(barleySVD$d), "*")[, 1:2]
> B <- sweep(barleySVD$u, 2, sqrt(barleySVD$d), "*")[, 1:2]
> A

            [,1]        [,2]
 [1,]  4.1945581 -0.39203762
 [2,]  2.7643876 -0.33933197
 [3,]  1.4250932 -0.04652144
 [4,]  1.8463184  0.33364399
 [5,]  1.2704687  0.15780901
 [6,]  1.1563616  0.40053626
 [7,]  1.0171974  0.72728762
 [8,]  0.6451498  1.46162874
 [9,] -0.1471004  2.13232959

> B

            [,1]       [,2]
 [1,] -2.0675116 -0.9742098
 [2,] -3.0597870 -0.5068344
 [3,] -2.9595994 -0.3318903
 [4,] -1.8087092 -0.4976057
 [5,] -1.5580232 -0.0844504
 [6,] -1.8940658  1.0845658
 [7,] -1.1790575  0.4068721
 [8,] -0.8490158  1.1467214
 [9,] -0.9704780  1.2655639
[10,] -0.6036867  1.3965960
```

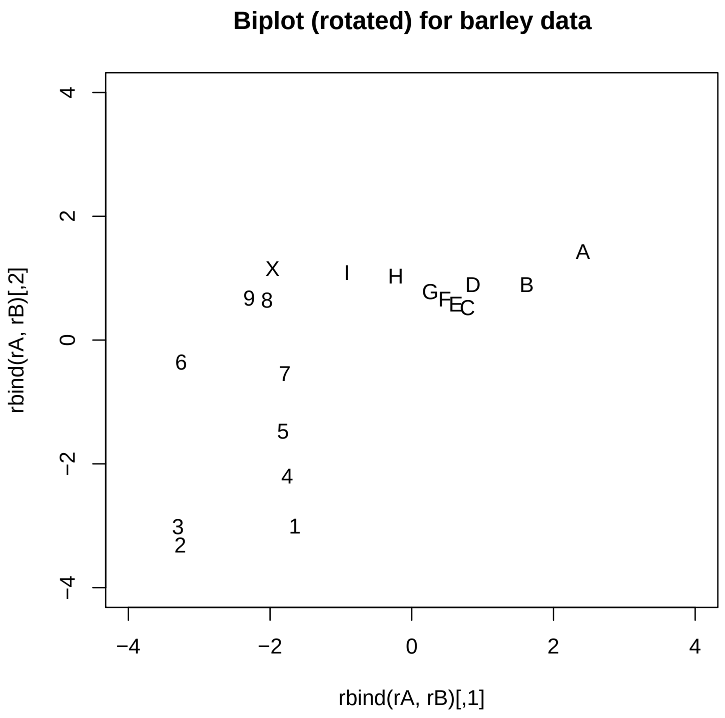
These matrices are essentially the same as in **?**. From these the biplot can be produced, for sites $A \dots I$ and varieties $1 \dots 9, X$:

```
> plot(rbind(A, B), pch = c(levels(barley$site), levels(barley$variety)),
+     xlim = c(-4, 4), ylim = c(-4, 4), main = "Biplot for barley data")
```



The product of the matrices A and B is unaffected by rotation or reciprocal scaling along either axis, so we can rotate the data so that the points for the sites are roughly parallel to the horizontal axis and the points for the varieties are roughly parallel to the vertical axis. In addition, we can scale the data so that points for the sites are about the line one unit about the horizontal axis, roughly

```
> a <- pi/5
> rotation <- matrix(c(cos(a), sin(a), -sin(a), cos(a)), 2, 2,
+     byrow = TRUE)
> rA <- (2 * A/3) %*% rotation
> rB <- (3 * B/2) %*% rotation
> plot(rbind(rA, rB), pch = c(levels(barley$site), levels(barley$variety)),
+     xlim = c(-4, 4), ylim = c(-4, 4), main = "Biplot (rotated) for barley data")
```



In the original biplot, the co-ordinates for the sites and varieties were given by the rows of A and B respectively, i.e

$$\begin{aligned} \alpha_i^T &= \sqrt(d)(u_{1i}, u_{2i}) \\ \beta_j^T &= \sqrt(d)(v_{1j}, v_{2j}) \end{aligned}$$

The rotated and scaled biplot suggests the simpler model

$$\begin{aligned} \alpha_i^T &= (\gamma_i, 1) \\ \beta_j^T &= (\delta_j, \tau_j) \end{aligned}$$

which implies the following model for the logits of the leaf blotch incidence:

$$\alpha_i^T \beta_j = \gamma_i \delta_j + \tau_j.$$

? describes this as a double additive model, which we can fit as follows:

```
> variety.binary <- factor(match(barley$variety, c(2, 3, 6), nomatch = 0) >
+     0, labels = c("rest", "2,3,6"))
> doubleAdditive <- gnm(y ~ variety + Mult(site, variety.binary),
+     family = wedderburn, data = barley)
```

```
Running start-up iterations..
Running main iterations.....................
Done
```

Comparing the chi-squared statistics, we see that the double additive model is an adequate model for the leaf blotch incidence:

```
> biplotModChiSq <- sum(residuals(biplotModel, type = "pearson")^2)
> doubleAddChiSq <- sum(residuals(doubleAdditive, type = "pearson")^2)
> c(doubleAddChiSq - biplotModChiSq, doubleAdditive$df.residual -
+     biplotModel$df.residual)
```

```
[1]  9.515599 15.000000
```

6.6 Stereotype Model

The stereotype model was proposed by **?** for ordered categorical data. It is a linear logistic model, in which there is assumed to be a common relationship between the response and the covariates in the model, but the scale of this association varies between categories and there is an additional category main effect or category-specific intercept:

$$\log \mu_{ic} = \beta_{0c} + \gamma_c \sum_r \beta_{rc} x_{ir}.$$

This model can be estimated by re-expressing the categorical data as counts and using a *gnm* model with a log link and poisson variance function.

For example, the `backPain` data set from **?** describes the progress of patients with back pain. The data set consists of an ordered factor quantifying the progress of each patient, and three prognostic variables. These data can be re-expressed as follows:

```
> set.seed(1)
> data(backPain)
> backPain[1:2, ]
```

```
  x1 x2 x3               pain
1  1  1  1               same
2  1  1  1 marked.improvement
```

```
> library(nnet)
> .incidence <- class.ind(backPain$pain)
> .counts <- as.vector(t(.incidence))
> .rowID <- factor(t(row(.incidence)))
> backPain <- backPain[.rowID, ]
> backPain$pain <- C(factor(rep(levels(backPain$pain), nrow(.incidence)),
+     levels = levels(backPain$pain), ordered = TRUE), treatment)
> cbind(.rowID[1:12], .counts[1:12], backPain[1:12, 4:1])
```

```
    .rowID[1:12] .counts[1:12]                pain x3 x2 x1
1              1             0               worse  1  1  1
1.1            1             1                same  1  1  1
1.2            1             0   slight.improvement  1  1  1
1.3            1             0 moderate.improvement  1  1  1
1.4            1             0   marked.improvement  1  1  1
1.5            1             0      complete.relief  1  1  1
2              2             0               worse  1  1  1
2.1            2             0                same  1  1  1
2.2            2             0   slight.improvement  1  1  1
2.3            2             0 moderate.improvement  1  1  1
2.4            2             1   marked.improvement  1  1  1
2.5            2             0      complete.relief  1  1  1
```

We can now fit the stereotype model to these data:

```
> oneDimensional <- gnm(.counts ~ pain + Mult(pain - 1, x1 + x2 +
+     x3 - 1), eliminate = ~.rowID, family = "poisson", data = backPain,
+     iterStart = 3)
```

```
Running start-up iterations...
Running main iterations..........
Done
```

```
> oneDimensional

Call:
gnm(formula = .counts ~ pain + Mult(pain - 1, x1 + x2 + x3 -
    1), eliminate = ~.rowID, family = "poisson", data = backPain,
    iterStart = 3)


Coefficients:
                              painsame                  painslight.improvement
                             16.154749                               15.682003
              painmoderate.improvement                  painmarked.improvement
                             12.452459                               19.911107
                  paincomplete.relief                 Mult1.Factor1.painworse
                             21.663012                               -2.499234
               Mult1.Factor1.painsame    Mult1.Factor1.painslight.improvement
                              0.137516                               -0.004642
Mult1.Factor1.painmoderate.improvement    Mult1.Factor1.painmarked.improvement
                             -0.627987                                0.778672
     Mult1.Factor1.paincomplete.relief                        Mult1.Factor2.x1
                              1.319279                               -1.406570
                      Mult1.Factor2.x2                        Mult1.Factor2.x3
                             -0.806779                               -0.710535

Deviance:            303.1003
Pearson chi-squared: 433.3762
Residual df:         493
```

using *eliminate* to handle the `.rowID` so that these structural parameters do not appear in the returned coefficients. This model is one dimensional since it involves only one function of $\mathbf{x} = (x1, x2, x3)$. We can compare this model to one with category-specific coefficents of the x variables, as may be used for a qualitative categorical response:

```
> threeDimensional <- gnm(.counts ~ pain + pain:(x1 + x2 + x3),
+     eliminate = ~.rowID, family = "poisson", data = backPain)

Running main iterations.
Done

> threeDimensional

Call:
gnm(formula = .counts ~ pain + pain:(x1 + x2 + x3), eliminate = ~.rowID,
    family = "poisson", data = backPain)


Coefficients:
                painsame       painslight.improvement
                 39.3495                      38.9688
   painmoderate.improvement       painmarked.improvement
                 35.8513                      43.0519
       paincomplete.relief                  painworse:x1
                 45.4999                      16.9234
             painsame:x1    painslight.improvement:x1
                  1.7421                       2.0717
```

```
painmoderate.improvement:x1    painmarked.improvement:x1
                     2.3351                       0.5119
     paincomplete.relief:x1                 painworse:x2
                     0.0000                       3.2750
                painsame:x2    painslight.improvement:x2
                     0.6009                       0.7236
painmoderate.improvement:x2    painmarked.improvement:x2
                     1.6029                       0.4311
     paincomplete.relief:x2                 painworse:x3
                     0.0000                       2.9407
                painsame:x3    painslight.improvement:x3
                     1.7852                       1.6486
painmoderate.improvement:x3    painmarked.improvement:x3
                     2.1944                       1.2491
     paincomplete.relief:x3
                     0.0000

Deviance:            299.0152
Pearson chi-squared: 443.0044
Residual df:         485
```

This model has the maximum dimensionality of three (as determined by the number of covariates). To obtain the log-likelihoods as reported in **?** we need to adjust for the extra parameters introduced to formulate the models as Poisson models. We write a simple function to do this and compare the log-likelihoods of the one dimensional model and the three dimensional model:

```
> logLikMultinom <- function(model) {
+     object <- get(model)
+     if (inherits(object, "gnm")) {
+         l <- logLik(object) + object$eliminate
+         c(nParameters = attr(l, "df") - object$eliminate, logLikelihood = l)
+     }
+     else c(nParameters = object$edf, logLikelihood = -deviance(object)/2)
+ }
> t(sapply(c("oneDimensional", "threeDimensional"), logLikMultinom))

                 nParameters logLikelihood
oneDimensional            12     -151.5501
threeDimensional          20     -149.5076
```

which show that the `oneDimensional` model is adequate.

A User-level Functions

We list here, for easy reference, all of the user-level functions in the *gnm* package. For full documentation see the package help pages.

Model Fitting	
`gnm`	fit generalized nonlinear models
`gnmControl`	set control parameters for fitting *gnm* models
Model Specification	
`Diag`	create factor differentiating diagonal elements
`Symm`	create symmetric interaction of factors
`Topo`	create ‘topological’ interaction factors
`Mult`	specify a multiplicative interaction in a `gnm` formula
`Exp`	specify an exponentiated constituent multiplier in a `Mult` term
`Nonlin`	specify a special nonlinear term in a `gnm` formula
`Dref`	gnm plug-in function to fit diagonal reference terms
`MultHomog`	gnm plug-in function to fit multiplicative interactions with homogeneous effects
`wedderburn`	specify the Wedderburn quasi-likelihood family
Methods and Accessor Functions	
summary.gnm	summarize *gnm* fits
`getContrasts`	estimate contrasts and their standard errors for parameters in a gnm model
`checkEstimable`	check whether one or more parameter combinations in a *gnm* model is identified
`se`	get standard errors of linear parameter combinations in *gnm* models
`termPredictors`	(*generic*) extract term contributions to predictor
Auxiliary Functions	
`getModelFrame`	get the model frame in use by `gnm`
`MPinv`	Moore-Penrose pseudoinverse of a real-valued matrix

B Key Changes since Last Release

The new features, improvements and changes in behaviour since the last release are given below. For bug fixes since the last release and the changes made in previous releases, see the CHANGES file in the package directory.

```
Changes in gnm 0.7-1
=======================

New Features
------------

    o   Topo() introduced for creating topological interaction factors.

    o   anova() implemented for objects of class c("gnm", "glm").


Improvements
------------

    o   Diagnostic messages given by gnm() have been improved.

    o   Step-halving introduced in main iterations of gnm() to ensure deviance
        is reduced at every iteration.

    o   getContrasts() now (additionally) reports quasi standard errors, when
        available.

    o   Calls to gnm() plug-in functions are now evaluated in the environment
        of the model frame and the enclosing environment of the parent frame
        of the call to gnm(). This means that variables can be found in a
        more standard fashion.


Changes in Behaviour
--------------------

    o   The 'data' argument of Nonlin() is defunct: Nonlin() now identifies
        variables to be added to the model frame as those passed to unspecified
        arguments of the plug-in function or those identified by a companion
        function to the plug-in, which is of a specified format.

    o   The (optional) 'start' object returned by a plug-in function can no
        longer be a function, only a vector. However it may now include NA
        values, to indicate parameters which may be treated as linear for the
        purpose of finding starting values, given the non-NA values.
```

References