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

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

For gnm version 0.8-5 , 2006-07-25

# Contents

# 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 some 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 continuous response variable in [0, 1] with the variance function $V(\mu) = \mu^2(1-\mu)^2$ as in Wedderburn (1974); 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 the main purpose of the package, 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. Agresti (2002), 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)
> 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 $i$th 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 Xie (1992). The data are 7 by 7 versions of social mobility tables from Erikson et al. (1982):

```
> 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 Erikson et al. (1982) or Xie (1992) — 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 Xie (1992) can now be fitted as follows:

```
> ### Fit the levels models given in Table 3 of Xie (1992)
> ## Null association between origin and destination
> nullModel <- gnm(Freq ~ country:origin + country:destination,
+                  family = poisson, data = erikson, verbose = FALSE)
>
> ## Interaction specified by levelMatrix, common to all countries
> commonTopo <- update(nullModel, ~ . +
+                      Topo(origin, destination, spec = levelMatrix),
+                      verbose = FALSE)
>
> ## Interaction specified by levelMatrix, different multiplier for
> ## each country
> multTopo <- update(nullModel, ~ . +
+                    Mult(Exp(-1 + country), -1 +
+                        Topo(origin, destination, spec = levelMatrix)),
+                    verbose = FALSE)
>
> ## Interaction specified by levelMatrix, different effects for
> ## each country
> separateTopo <- update(nullModel, ~ . +
+                        country:Topo(origin, destination, spec = levelMatrix),
+                        verbose = FALSE)
>
> 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 Wedderburn (1974) 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 McCullagh and Nelder (1989), which differs slightly from Wedderburn's own 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
 0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
            6             7             8             9            10
-1.776357e-15 -8.881784e-16 -8.881784e-16  0.000000e+00  0.000000e+00
           11            12            13            14            15
 0.000000e+00  0.000000e+00  0.000000e+00 -1.776357e-15 -8.881784e-16
           16
 0.000000e+00
```

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

# 3  Nonlinear Terms

The main purpose of the gnm package is to provide 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 (Goodman, 1979), 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.

The `Mult` function 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 (Goodman, 1979)

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

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 (Xie, 1992; Erikson and Goldthorpe, 1992)

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

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.

### 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.

---

[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.

### 3.2.2 *Dref*

The function *Dref* is a plug-in 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 $\gamma_l$ 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 $\gamma_l$, 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 $\delta_f$.

Factors defining the diagonal reference term are passed as unspecified arguments to *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))
```

The *Dref* function has one specified argument, *formula*, which is a symbolic description of the dependence of $\delta_f$ 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 $\delta_f$ or the $\xi_f$ and $\beta_f$, 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 through existing 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. If the plug-in function does not return a *start* component (see below), the *localDesignFunction* must also take the argument *ind*, which specifies the index of a column to be returned instead of the full 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 starting procedure for nonlinear parameters that will be used otherwise.

As an example of a *start* component, *Dref* returns

```
c(runif(nLocal) - 0.5, rep(0.5, nGlobal))
```

where *nLocal* is the number of weight parameters (parameters which are "local" to a specific factor) and *nGlobal* is the number of diagonal effects ("global" level effects across factors). The randomness in the starting values for the weight parameters ensures that arbitrariness of the final parameterization is emphasised.

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, ind = NULL, ...) {
    X <- 0
    vList <- lapply(designList, "%*%", coef)
    for (i in seq(designList)) {
        if (is.null(ind))
            X <- X + designList[[*]] * drop(do.call("pprod", vList[-i]))
        else
            X <- X + designList[[]][, ind] *
                drop(do.call("pprod", vList[-i]))}
    X}
```

Since the result of the predictor function is not needed here, the local design function does not have the specified argument `predictor`, but allows such an argument to be passed to the function by the use of the special argument '`...`'. Since `MultHomog` does not return a `start` component, the local design function can optionally return a single column of the local design matrix as specified by `ind`. This functionality is required by the default starting procedure for nonlinear parameters.

# 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 can be set using the arguments `tolerance`, `iterStart` and `iterMax`. Starting values for the parameter estimates can be set by `start` and parameters can be constrained to zero by specifying a `constrain` argument. Parameters of a stratification factor can be handled more efficiently by specifying the factor in an `eliminate` argument. These options are described in more detail below.

## 4.1 Basic control parameters

The arguments `iterStart` and `iterMax` control respectively the number of starting iterations (where applicable) and the number of main iterations used by the fitting algorithm. The progress of these iterations can be followed by setting either `verbose` or `trace` to `TRUE`. If `verbose` is `TRUE` and `trace` is `FALSE`, which is the default setting, progress is indicated by printing the character "." at the beginning of each iteration. If `trace` is `TRUE`, the deviance is printed at the beginning of each iteration (over-riding the printing of "." if necessary). Whenever `verbose` is `TRUE`, additional messages indicate each stage of the fitting process and diagnose any errors that cause that cause the algorithm to restart.

The fitting algorithm will terminate before the number of main iterations has reached `iterMax` if the convergence criteria have been met, with tolerance specified by `tolerance`. Convergence is judged by comparing the squared components of the score vector with corresponding elements of the diagonal of the Fisher information matrix. If, for all components of the score vector, the ratio is less than `tolerance^2`, or the corresponding diagonal element of the Fisher information matrix is less than 1e-20, the algorithm is deemed to have converged.

## 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 = "coefNames"` 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 $\theta_i$ 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 terms 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 linear, or if all nonlinear parameters are specified by *start* or a plug-in function.

## 4.3 Using *constrain*

By default, *gnm* only imposes identifiability constraints according to the general conventions used by *R* to handle linear aliasing. Therefore models that have any nonlinear terms will be usually be over-parameterized and *gnm* will return a random parameterization for unidentified coefficients.

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, verbose = FALSE)
```

Running the analysis again from a different seed

```
> set.seed(2)
> RChomog2 <- update(RChomog1)
```

gives a different representation of the same model:

```
> compareCoef <- cbind(coef(RChomog1), coef(RChomog2))
> colnames(compareCoef) <- c("RChomog1", "RChomog2")
> round(compareCoef, 4)
```

|  | RChomog1 | RChomog2 |
|---|---|---|
| (Intercept) | -0.1312 | 0.1844 |
| origin2 | 0.5367 | 0.5143 |
| origin3 | 1.6922 | 1.6083 |
| origin4 | 2.0597 | 1.9158 |
| origin5 | 0.8418 | 0.6961 |
| origin6 | 2.9463 | 2.7480 |
| origin7 | 1.6543 | 1.4132 |
| origin8 | 1.4128 | 1.1466 |
| destination2 | 0.9557 | 0.9333 |
| destination3 | 2.0366 | 1.9527 |
| destination4 | 2.3482 | 2.2042 |
| destination5 | 1.7412 | 1.5955 |
| destination6 | 3.2498 | 3.0514 |
| destination7 | 2.4059 | 2.1648 |
| destination8 | 1.9882 | 1.7220 |
| Diag(origin, destination)1 | 1.5267 | 1.5267 |
| Diag(origin, destination)2 | 0.4560 | 0.4560 |
| Diag(origin, destination)3 | -0.0160 | -0.0160 |
| Diag(origin, destination)4 | 0.3892 | 0.3892 |
| Diag(origin, destination)5 | 0.7385 | 0.7385 |
| Diag(origin, destination)6 | 0.1347 | 0.1347 |
| Diag(origin, destination)7 | 0.4576 | 0.4576 |
| Diag(origin, destination)8 | 0.3885 | 0.3885 |
| MultHomog(origin, destination).1 | -1.5864 | -1.4836 |
| MultHomog(origin, destination).2 | -1.3681 | -1.2653 |
| MultHomog(origin, destination).3 | -0.7699 | -0.6671 |
| MultHomog(origin, destination).4 | -0.1860 | -0.0832 |
| MultHomog(origin, destination).5 | -0.1689 | -0.0661 |
| MultHomog(origin, destination).6 | 0.3429 | 0.4457 |
| MultHomog(origin, destination).7 | 0.7590 | 0.8618 |
| MultHomog(origin, destination).8 | 1.0026 | 1.1054 |

Even though the linear terms are constrained, the parameter estimates for the main effects of *origin* and *destination* still change, because these terms are aliased with the higher order multiplicative interaction, which is unconstrained.

Standard errors are only meaningful for identified parameters and hence the output of *summary.gnm* will show clearly which coefficients are estimable:

```
> summary(RChomog2)

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


Deviance Residuals:
      Min         1Q      Median         3Q        Max
-1.659e+00  -4.297e-01  -4.463e-08   3.862e-01  1.721e+00

Coefficients:
                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                    0.18438         NA      NA       NA
origin2                        0.51428         NA      NA       NA
origin3                        1.60827         NA      NA       NA
origin4                        1.91578         NA      NA       NA
origin5                        0.69610         NA      NA       NA
origin6                        2.74796         NA      NA       NA
origin7                        1.41324         NA      NA       NA
origin8                        1.14664         NA      NA       NA
destination2                   0.93329         NA      NA       NA
destination3                   1.95269         NA      NA       NA
destination4                   2.20421         NA      NA       NA
destination5                   1.59552         NA      NA       NA
destination6                   3.05144         NA      NA       NA
destination7                   2.16483         NA      NA       NA
destination8                   1.72202         NA      NA       NA
Diag(origin, destination)1     1.52667    0.44658   3.419  0.00063 ***
Diag(origin, destination)2     0.45600    0.34595   1.318  0.18747
Diag(origin, destination)3    -0.01598    0.18098  -0.088  0.92965
Diag(origin, destination)4     0.38918    0.12748   3.053  0.00227 **
Diag(origin, destination)5     0.73852    0.23329   3.166  0.00155 **
Diag(origin, destination)6     0.13474    0.07934   1.698  0.08945 .
Diag(origin, destination)7     0.45764    0.15103   3.030  0.00245 **
Diag(origin, destination)8     0.38847    0.22172   1.752  0.07976 .
MultHomog(origin, destination).1 -1.48357       NA      NA       NA
MultHomog(origin, destination).2 -1.26528       NA      NA       NA
MultHomog(origin, destination).3 -0.66711       NA      NA       NA
MultHomog(origin, destination).4 -0.08323       NA      NA       NA
MultHomog(origin, destination).5 -0.06606       NA      NA       NA
MultHomog(origin, destination).6  0.44570       NA      NA       NA
MultHomog(origin, destination).7  0.86184       NA      NA       NA
MultHomog(origin, destination).8  1.10541       NA      NA       NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

Std. Error is NA where coefficient has been constrained or is unidentified

Residual deviance: 32.561 on 34 degrees of freedom
AIC: 414.9

Number of iterations: 7
```

Additional constraints may be specified through the `constrain` and `constrainTo` arguments of `gnm`. These arguments specify respectively parameters that are to be constrained in the fitting process and the values to which they should be constrained. Parameters may be specified by a regular expression to match against the parameter names, a numeric vector of indices, a character vector of names, or, if `constrain = "[?]"` they can be selected through a *Tk* dialog. The values to constrain to should be specified by a numeric vector; if `constrainTo` is missing, constrained parameters will be set to zero.

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)[pickCoef(RChomog1, "Mult")]
> set.seed(1)
> RChomogConstrained1 <- update(RChomog1, constrain = 31, start = c(rep(NA,
+     23), multCoef - multCoef[8]))
> set.seed(2)
> RChomogConstrained2 <- update(RChomogConstrained1)
> 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 use of `constrain` in conjunction with `constrainTo` is usually sufficient to make coefficients of interest identifiable . The functions `checkEstimable` or `getContrasts`, described in Section 5, may be used to check whether particular combinations of parameters are estimable.

## 4.4 Using `eliminate`

When a model contains the additive effect of a factor which has a large number of levels, the iterative algorithm by which maximum likelihood estimates are computed can usually be accelerated by use of the `eliminate` argument to `gnm`.

The factor to be `eliminate`-d should be specified by an expression, which is then interpreted as the first term in the model formula, replacing any intercept term. So, for example, in terms of the structure of the model,

```
gnm(mu ~ A + B + Mult(A, B), eliminate = strata1:strata2)
```

is equivalent to

```
gnm(mu ~ -1 + strata1:strata2 + A + B + Mult(A, B))
```

However, specifying a factor through `eliminate` has two advantages over the standard specification. First, the structure of the eliminated factor is exploited so that computational speed is improved — substantially so if the number of eliminated parameters is large. Second, unless otherwise specified through the `ofInterest` argument to `gnm`, the `ofInterest` component of the returned model object indexes the non-eliminated parameters. Thus eliminated parameters are excluded from printed model summaries and default selection by *gnm* methods. See Section 5.2 for further details on the use of the `ofInterest` component.

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 application see Section 6.6 below. Here we give an artificial illustration: 1000 randomly-generated trinomial responses, and a single predictor variable (whose effect on the data generation is null):

```
> set.seed(1)
> n <- 1000
> x <- rep(rnorm(n), rep(3, n))
> counts <- as.vector(rmultinom(n, 10, c(0.7, 0.1, 0.2)))
> rowID <- gl(n, 3, 3 * n)
> resp <- gl(3, 1, 3 * n)
```

The logistic model for dependence on `x` can be fitted as a Poisson log-linear model[2], using either `glm` or `gnm`:

---

[2]For this particular example, of course, it would be more economical to fit the model directly using `multinom` (from the recommended package nnet). But fitting as here via the 'Poisson trick' allows the model to be elaborated within the gnm framework using `Mult` or `Nonlin` terms.

```
> ## Timings on a Pentium M 1.6GHz, under Linux
> system.time(temp.glm <- glm(counts ~ rowID + resp + resp:x,
                               family = poisson))[1]

[1] 116.8

> system.time(temp.gnm <- gnm(counts ~ resp + resp:x, eliminate = rowID,
                              family = poisson, verbose = FALSE))[1]

[1] 22.0

> c(deviance(temp.glm), deviance(temp.gnm))

[1] 2462.556 2462.556
```

Here the use of *eliminate* causes the *gnm* calculations to run more quickly than *glm*. The speed advantage[3] increases with the number of eliminated parameters (here 1000). Since the default behaviour has not been over-ridden by an *ofInterest* argument, the eliminated parameters do not appear in printed model summaries:

```
> summary(temp.gnm)

Call:

gnm(formula = counts ~ resp + resp:x, eliminate = rowID, family = poisson,
    verbose = FALSE)


Deviance Residuals:
      Min        1Q     Median        3Q        Max
-2.852038  -0.786172  -0.004534   0.645278   2.755013

Coefficients of interest:
          Estimate Std. Error z value Pr(>|z|)
resp2   -1.9614483  0.0340074  -57.68   <2e-16
resp3   -1.2558460  0.0253589  -49.52   <2e-16
resp1:x  0.0001049         NA      NA       NA
resp2:x -0.0155083         NA      NA       NA
resp3:x  0.0078314         NA      NA       NA

(Dispersion parameter for poisson family taken to be 1)

Std. Error is NA where coefficient has been constrained or is unidentified

Residual deviance: 2462.6 on 1996 degrees of freedom
AIC: 12028

Number of iterations: 3
```

As usual, *gnm* has worked here with an over-parameterized representation of the model. The parameterization used by *glm* can be seen from

```
> coef(temp.glm)[-(1:1000)]

   resp2    resp3  resp1:x  resp2:x  resp3:x
-1.96145 -1.25585 -0.00773 -0.02334       NA
```

(we will not print the full summary of *temp.glm* here, since it gives details of all 1005 parameters!), which easily can be obtained, if required, by using *getContrasts*:

```
> getContrasts(temp.gnm, ofInterest(temp.gnm)[5:3])
```

---

[3] In fact *eliminate* is, in principle, capable of much bigger time savings than this: its implementation in the current version of gnm is really just a proof of concept, and it has not yet been optimized for speed.

```
                estimate      SE quasiSE quasiVar
resp3:x  0.00000 0.00000 0.02163 0.000468
resp2:x -0.02334 0.03761 0.03077 0.000947
resp1:x -0.00773 0.02452 0.01154 0.000133
```

The `eliminate` feature as implemented in gnm extends the earlier work of Hatzinger and Francis (2004) to a broader class of models and to over-parameterized model representations.

# 5 Methods and Accessor functions

## 5.1 Methods

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.

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

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

| | |
|---|---|
| add1 | effects |
| alias | influence |
| dfbeta | kappa |
| dfbetas | predict |
| drop1 | proj |
| dummy.coef | |

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.

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 2.5 for an example of its use.

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

Similarly, `profile.gnm` and `confint.gnm` are only applicable to estimable parameters. The deviance function of a generalised nonlinear model can sometimes be far from quadratic and `profile.gnm` attempts to detect assymetry or asymptotic behaviour in order to return a sufficient profile for a given parameter. As an example, consider the following model, described later in Section 6.3:

```
data(yaish)
unidiff <- gnm(Freq ~ educ*orig + educ*dest +
               Mult(Exp(-1 + educ), -1 + orig:dest),
               constrain = "Mult.*educ1", family = poisson, data = yaish,
               subset = (dest != 7))
prof <- profile(unidiff, which = 61:65, trace = TRUE)
```

14

If the deviance is quadratic in a given parameter, the profile trace will be linear. We can plot the profile traces as follows:
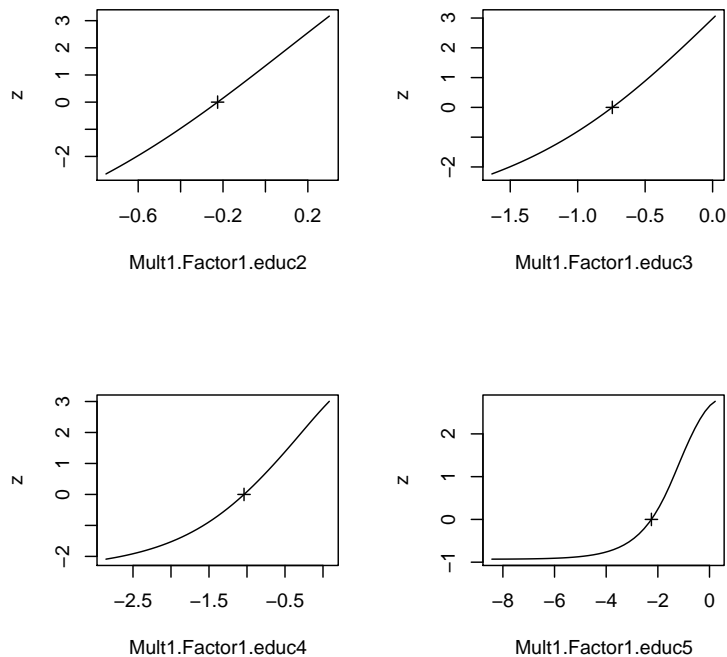


From these plots we can see that the deviance is approximately quadratic in `Mult1.Factor1.educ2`, assymetric in `Mult1.Factor1.educ3` and `Mult1.Factor1.educ4` and asymptotic in `Mult1.Factor1.educ5`. When the deviance is approximately quadratic in a given parameter, `profile.gnm` uses the same stepsize for profiling above and below the original estimate:

```
> diff(prof[[2]]$par.vals[, "Mult1.Factor1.educ2"])

 [1] 0.1053072 0.1053072 0.1053072 0.1053072 0.1053072 0.1053072 0.1053072
 [8] 0.1053072 0.1053072 0.1053072
```

When the deviance is assymmetric, `profile.gnm` uses different stepsizes to accommodate the skew:

```
> diff(prof[[4]]$par.vals[, "Mult1.Factor1.educ4"])

 [1] 0.2018393 0.2018393 0.2018393 0.2018393 0.2018393 0.2018393 0.2018393
 [8] 0.2018393 0.2018393 0.2243673 0.2243673 0.2243673 0.2243673 0.2243673
```

Finally, the presence of an asymptote is recorded in the `"asymptote"` attribute of the returned profile:

```
> attr(prof[[5]], "asymptote")

[1]  TRUE FALSE
```

This information is used by `confint.gnm` to return infinite limits for confidence intervals, as appropriate:

```
        confint(prof)
```

## 5.2  *ofInterest* and *pickCoef*

It is quite common for a statistical model to have a large number of parameters, but for only a subset of these parameters be of interest when it comes to interpreting the model. An example of this has been seen in Section 4.4, where a factor is required in the model in order to represent a structural aspect of the data, but the estimated factor effects have no substantive interpretation. Even for models in which all parameters correspond to variables of potential interest, the substantive focus may still be on a subset of parameters.

The *ofInterest* argument to *gnm* allows the user to specify a subset of the parameters which are of interest, so that *gnm* methods will focus on these parameters. In particular, printed model summaries will only show the parameters of interest, whilst methods for which a subset of parameters may be selected will by default select the parameters of interest, or where this may not be appropriate, provide a *Tk* dialog for selection from the parameters of interest. Parameters may be specified to the *ofInterest* argument by a regular expression to match against parameter names, by a numeric vector of indices, by a character vector of names, or, if *ofInterest = "[?]"* they can be selected through a *Tk* dialog.

The information regarding the parameters of interest is held in the *ofInterest* component of *gnm* objects, which is a named vector of numeric indices, or *NULL* if all parameters are of interest. This component may be accessed or replaced using *ofInterest* or *ofInterest<-* respectively.

The *pickCoef* function provides a simple way to obtain the indices of coefficients from any model object. It takes the model object as its first argument and has an optional *regexp* argument. If a regular expression is passed to *regexp*, the coefficients are selected by matching this regular expression against the coefficient names. Otherwise, coefficients may be selected via a *Tk* dialog.

So, returning to the example from the last section, if we had set *ofInterest* to index the education multipliers as follows

```
ofInterest(unidiff) <- pickCoef(unidiff, "Mult1.*educ")
```

then it would not have been necessary to specify the *which* argument of *profile* as these parameters would have been selected by default.


## 5.3  *checkEstimable*

The *checkEstimable* function can be used to check the estimability of a linear combination of parameters. For non-linear combinations the same function can be used to check estimability based on the (local) vector of partial derivatives. The *checkEstimable* function provides a numerical version of the sort of algebraic test described in Catchpole and Morgan (1997).

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)

Initialising
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 first level. The parameters to be estimated are then the differences between each effect and the effect of the first level. These differences can be represented by a contrast matrix as follows:

```
> coefs <- names(coef(doubleUnidiff))
> contrCoefs <- coefs[grep("Mult1.Factor1", coefs)]
> nContr <- length(contrCoefs)
> contrMatrix <- matrix(0, length(coefs), nContr, dimnames = list(coefs,
+     contrCoefs))
> contr <- contr.sum(contrCoefs)
> contr <- rbind(contr[nContr, ], contr[-nContr, ])
> contrMatrix[contrCoefs, 2:nContr] <- contr
> contrMatrix[contrCoefs, 2:nContr]
```

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

Then their estimability can be checked using `checkEstimable`

```
> checkEstimable(doubleUnidiff, contrMatrix)

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

which confirms that the effects for the other three levels are estimable when the parameter for the first 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)]
> nContr <- length(contrCoefs)
> contrMatrix <- matrix(0, length(coefs), length(contrCoefs), dimnames = list(coefs,
+     contrCoefs))
> contr <- contr.sum(contrCoefs)
> contrMatrix[contrCoefs, 2:nContr] <- rbind(contr[nContr, ], contr[-nContr,
+     ])
> checkEstimable(doubleUnidiff, contrMatrix)

Mult1.Factor2.religion1:vote1 Mult1.Factor2.religion2:vote1
                           NA                         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                         FALSE
```

## 5.4  `getContrasts, se`

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 first can be obtained as follows:

```
> myContrasts <- getContrasts(doubleUnidiff, pickCoef(doubleUnidiff,
+     "Mult1.Factor1"))
> myContrasts


                          estimate        SE    quasiSE     quasiVar
Mult1.Factor1.election1  0.0000000 0.0000000 0.09803075 0.009610029
Mult1.Factor1.election2 -0.0878181 0.1136832 0.05702819 0.003252214
Mult1.Factor1.election3 -0.2615200 0.1184134 0.06812239 0.004640660
Mult1.Factor1.election4 -0.3283459 0.1221302 0.07168290 0.005138439
```

Visualization of estimated contrasts using 'quasi standard errors' (Firth, 2003; Firth and de Menezes, 2004) is achieved by plotting the resulting object:

```
> plot(myContrasts, main = "Relative strength of religion-vote association, log scale",
+      xlab = "Election", levelNames = 1:4)
```

**Relative strength of religion−vote association, log scale**
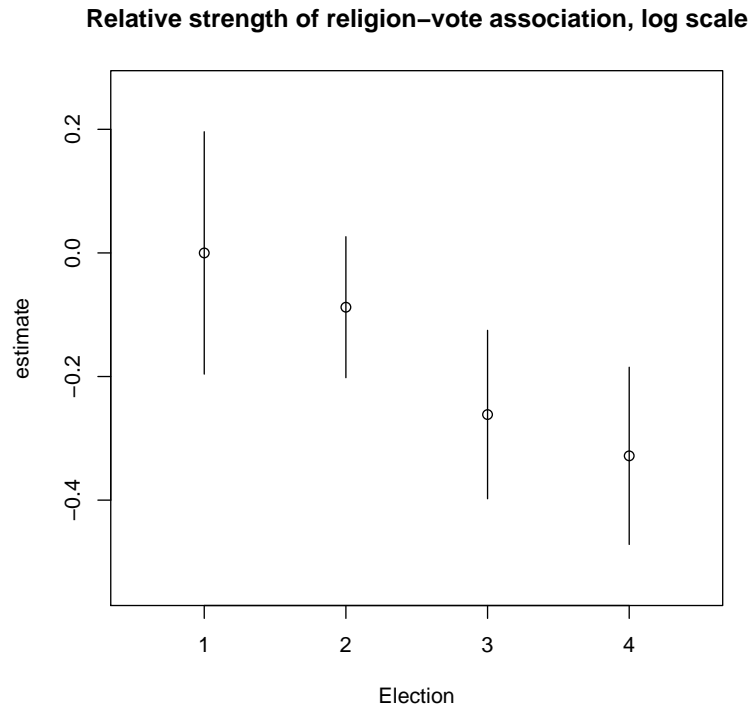


For more general linear combinations of parameters than contrasts, the lower-level `se` function (which is called internally by `getContrasts` and by the `summary` method) can be used directly. See `help(se)` for details.

## 5.5 `residSVD`

Sometimes it is useful to operate on the residuals of a model in order to create informative summaries of residual variation, or to obtain good starting values for additional parameters in a more elaborate model. The relevant arithmetical operations are weighted means of the so-called *working residuals*.

The `residSVD` function facilitates one particular residual analysis that is often useful when considering multiplicative interaction between factors as a model elaboration: in effect, `residSVD` provides a direct estimate of the parameters of such an interaction, by performing an appropriately weighted singular value decomposition on the working residuals.

As an illustration, consider the biplot model described in Section 6.5 below. We can proceed by fitting a smaller model, then use `residSVD` to obtain starting values for the parameters in the bilinear term:

```
> emptyModel <- gnm(y ~ -1, family = wedderburn, data = barley)
> biplotStart <- residSVD(emptyModel, barley$site, barley$variety,
+      d = 2)
> biplotModel <- gnm(y ~ -1 + Mult(-1 + site, -1 + variety, multiplicity = 2),
+      family = wedderburn, data = barley, start = biplotStart)

Running main iterations........................................................
...........................................................................
..
Done
```

In this instance, the use of purposive (as opposed to the default, random) starting values had little effect: the fairly large number of iterations needed in this example is caused by a rather flat (quasi-)likelihood surface near the maximum, not by poor starting values. In other situations, the use of `residSVD` may speed the calculations dramatically (see for example Section 6.4), or it may be crucial to success in locating the MLE (for example see `help(House2001)`, where the number of multiplicative parameters is in the hundreds).

The `residSVD` result in this instance provides a crude approximation to the MLE of the enlarged model, as can be seen in the following plot:

**Comparison of residSVD and MLE**
**for a 2–dimensional biplot model**
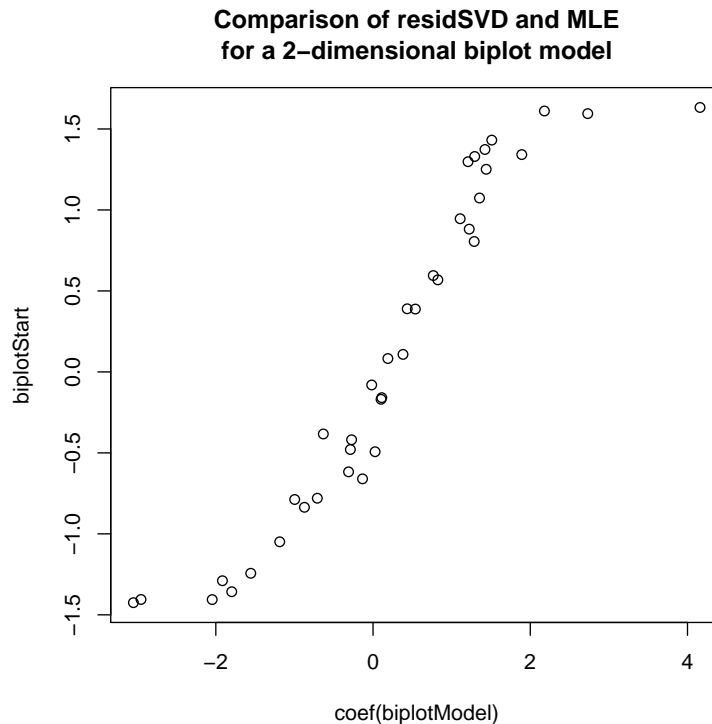


# 6   Examples

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 Goodman (1979). 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 $\mu_{rc}$ 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 Agresti (2002) 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)

Initialising
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
               3.831281                   -0.067413
                   SESC                        SESD
               0.109959                    0.404969
                   SESE                        SESF
               0.025257                   -0.200685
                MHSmild                  MHSmoderate
               0.712969                    0.204987
             MHSimpaired           Mult1.Factor1.SESA
               0.251749                    0.340495
     Mult1.Factor1.SESB           Mult1.Factor1.SESC
               0.343267                    0.114885
     Mult1.Factor1.SESD           Mult1.Factor1.SESE
              -0.006284                   -0.305574
     Mult1.Factor1.SESF           Mult1.Factor2.MHSwell
              -0.551460                    0.935600
   Mult1.Factor2.MHSmild    Mult1.Factor2.MHSmoderate
               0.094793                   -0.056941
Mult1.Factor2.MHSimpaired
              -0.755299

Deviance:              3.570562
Pearson chi-squared: 3.568088
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] 0.1664874


$mu
Mult1.Factor1.SESA Mult1.Factor1.SESB Mult1.Factor1.SESC Mult1.Factor1.SESD
       1.11233090          1.12143715          0.37107612         -0.02702946
Mult1.Factor1.SESE Mult1.Factor1.SESF
      -1.01036153         -1.81823284


$nu
    Mult1.Factor2.MHSwell     Mult1.Factor2.MHSmild Mult1.Factor2.MHSmoderate
               1.6775144                 0.1403989                -0.1369924
Mult1.Factor2.MHSimpaired
              -1.4136910
```

### 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)

Initialising
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
              3.85530                   -0.06444
                 SESC                       SESD
              0.11140                    0.38457
                 SESE                       SESF
              0.01081                   -0.18462
              MHSmild                MHSmoderate
              0.69860                    0.16975
           MHSimpaired          Mult1.Factor1.SESA
              0.22876                    0.95047
   Mult1.Factor1.SESB         Mult1.Factor1.SESC
              0.99597                    0.33932
   Mult1.Factor1.SESD         Mult1.Factor1.SESE
             -0.17292                   -0.91634
   Mult1.Factor1.SESF       Mult1.Factor2.MHSwell
             -1.39376                    0.35782
  Mult1.Factor2.MHSmild  Mult1.Factor2.MHSmoderate
              0.03795                   -0.02129
Mult1.Factor2.MHSimpaired         Mult2.Factor1.SESA
             -0.28029                   -0.17762
   Mult2.Factor1.SESB         Mult2.Factor1.SESC
```

21

```
                    -0.25156                        -0.16614
        Mult2.Factor1.SESD              Mult2.Factor1.SESE
                     0.28993                         0.22675
        Mult2.Factor1.SESF           Mult2.Factor2.MHSwell
                    -0.45554                         0.30776
      Mult2.Factor2.MHSmild   Mult2.Factor2.MHSmoderate
                     0.09804                        -0.25536
Mult2.Factor2.MHSimpaired
                     0.06769


Deviance:              0.5225353
Pearson chi-squared:  0.523331
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 Goodman (1979) is a contingency table classified by the occupational status of fathers (origin) and their sons (destination). Goodman (1979) 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)

Initialising
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
                     -0.67122                            0.57207
                      origin3                            origin4
                      1.82441                            2.28650
                      origin5                            origin6
                      1.07137                            3.25871
                      origin7                            origin8
                      2.03415                            1.83204
                 destination2                       destination3
                      0.99108                            2.16883
                 destination4                       destination5
                      2.57493                            1.97078
                 destination6                       destination7
                      3.56218                            2.78575
                 destination8         Diag(origin, destination)1
                      2.40741                            1.52667
    Diag(origin, destination)2         Diag(origin, destination)3
```

```
                           0.45600                                -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                                -1.74831
 MultHomog(origin, destination).2  MultHomog(origin, destination).3
                          -1.53001                                -0.93185
 MultHomog(origin, destination).4  MultHomog(origin, destination).5
                          -0.34797                                -0.33080
 MultHomog(origin, destination).6  MultHomog(origin, destination).7
                           0.18096                                 0.59710
 MultHomog(origin, destination).8
                           0.84068


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)

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

> anova(RChomog, RCheterog)

Analysis of Deviance Table

Model 1: Freq ~ origin + destination + Diag(origin, destination) + Nonlin(MultHomog(origin,
    destination))
Model 2: Freq ~ origin + destination + Diag(origin, destination) + Mult(origin,
    destination)
  Resid. Df Resid. Dev Df Deviance
1        34     32.561
2        28     29.149  6    3.412
```

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

## 6.2 Diagonal Reference Models

Diagonal reference models, proposed by Sobel (1981, 1985), 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 Clifford and Heath (1993). 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)
```

23

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 parameterization.

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

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

Initialising
Running main iterations........
Done

> classMobility

Call:

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


Coefficients:
                         (Intercept)          Dref(origin, destination).origin
                            -1.34325                                  -0.30736
Dref(origin, destination).destination          Dref(origin, destination).1
                            -0.05501                                  -0.83454
        Dref(origin, destination).2          Dref(origin, destination).3
                             0.21066                                  -0.61159
        Dref(origin, destination).4          Dref(origin, destination).5
                             0.76500                                   1.38370


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
                           0.4372469                             0.5627531
```

These results are slightly different from those reported by Clifford and Heath (1993). The reason for this is unclear: we are confident that the above results are correct for the data as given in Clifford and Heath (1993), 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.

Clifford and Heath (1993) 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

$$\mu_{od} = \begin{cases} \dfrac{e^{\delta_1}}{e^{\delta_1} + e^{\delta_2}} \gamma_o + \dfrac{e^{\delta_2}}{e^{\delta_1} + e^{\delta_2}} \gamma_d & \text{if } o = 1 \\[3ex] \dfrac{e^{\delta_3}}{e^{\delta_3} + e^{\delta_4}} \gamma_o + \dfrac{e^{\delta_4}}{e^{\delta_3} + e^{\delta_4}} \gamma_d & \text{if } d = 1 \\[3ex] \dfrac{e^{\delta_5}}{e^{\delta_5} + e^{\delta_6}} \gamma_o + \dfrac{e^{\delta_6}}{e^{\delta_5} + e^{\delta_6}} \gamma_d & \text{if } o \neq 1 \text{ and } d \neq 1 \end{cases}$$

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

24

```
> 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)

Initialising
Running main iterations..........
Done

> socialMobility

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


Coefficients:
                                                                   (Intercept)
                                                                      -1.31739
       Dref(origin, destination, formula = ~1 + downward + upward).origin.(Intercept)
                                                                      -0.39834
     Dref(origin, destination, formula = ~1 + downward + upward).origin.downwardTRUE
                                                                       0.37858
       Dref(origin, destination, formula = ~1 + downward + upward).origin.upwardTRUE
                                                                       0.06225
  Dref(origin, destination, formula = ~1 + downward + upward).destination.(Intercept)
                                                                      -0.01158
Dref(origin, destination, formula = ~1 + downward + upward).destination.downwardTRUE
                                                                      -0.43218
   Dref(origin, destination, formula = ~1 + downward + upward).destination.upwardTRUE
                                                                       0.12247
                      Dref(origin, destination, formula = ~1 + downward + upward).1
                                                                      -0.74021
                      Dref(origin, destination, formula = ~1 + downward + upward).2
                                                                       0.20469
                      Dref(origin, destination, formula = ~1 + downward + upward).3
                                                                      -0.67740
                      Dref(origin, destination, formula = ~1 + downward + upward).4
                                                                       0.74824
                      Dref(origin, destination, formula = ~1 + downward + upward).5
                                                                       1.37497

Deviance:            18.97407
Pearson chi-squared: 17.07493
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
                                                                         0.3900792
 Dref(origin, destination, formula = ~1 + downward + upward).destination.upwardTRUE
                                                                         0.6099208

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

```
        Dref(origin, destination, formula = ~1 + downward + upward).origin.downwardTRUE
                                                                          0.6044394
Dref(origin, destination, formula = ~1 + downward + upward).destination.downwardTRUE
                                                                          0.3955606

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

        Dref(origin, destination, formula = ~1 + downward + upward).origin.(Intercept)
                                                                          0.4044959
Dref(origin, destination, formula = ~1 + downward + upward).destination.(Intercept)
                                                                          0.5955041
```

Again, the results differ slightly from those reported by Clifford and Heath (1993), 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)

Initialising
Running main iterations.........
Done

> downwardMobility

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


Coefficients:
                                                                 (Intercept)
                                                                    -1.30747
       Dref(origin, destination, formula = ~1 + downward).origin.(Intercept)
                                                                    -0.02851
      Dref(origin, destination, formula = ~1 + downward).origin.downwardTRUE
                                                                     0.39013
  Dref(origin, destination, formula = ~1 + downward).destination.(Intercept)
                                                                     0.38028
 Dref(origin, destination, formula = ~1 + downward).destination.downwardTRUE
                                                                    -0.42061
                     Dref(origin, destination, formula = ~1 + downward).1
                                                                    -0.76240
                     Dref(origin, destination, formula = ~1 + downward).2
                                                                     0.20095
                     Dref(origin, destination, formula = ~1 + downward).3
                                                                    -0.68418
                     Dref(origin, destination, formula = ~1 + downward).4
                                                                     0.73440
                     Dref(origin, destination, formula = ~1 + downward).5
                                                                     1.36377

Deviance:            18.99389
Pearson chi-squared: 17.09981
Residual df:         18

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

        Dref(origin, destination, formula = ~1 + downward).origin.downwardTRUE
                                                                     0.5991571
Dref(origin, destination, formula = ~1 + downward).destination.downwardTRUE
                                                                     0.4008429
```

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

    Dref(origin, destination, formula = ~1 + downward).origin.(Intercept)
                                                                 0.3992031
Dref(origin, destination, formula = ~1 + downward).destination.(Intercept)
                                                                 0.6007969
```

### *Dref* example 2: Conformity to parental rules

Another application of diagonal reference models is given by van der Slik et al. (2002). The data from this paper are not publicly available[4], 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, van der Slik et al. (2002) 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                -0.33730
Dref(MOPLM, FOPLF).FOPLF     Dref(MOPLM, FOPLF).1    Dref(MOPLM, FOPLF).2
           -0.02507                 4.95123                 4.86328
   Dref(MOPLM, FOPLF).3     Dref(MOPLM, FOPLF).4    Dref(MOPLM, FOPLF).5
            4.86458                 4.72343                 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
```

The coefficients of the covariates are not aliased with the parameters of the diagonal reference term and thus the basic identifiability constraints that have been imposed are sufficient for these parameters to be identified. The diagonal effects do not need to be constrained as they represent contrasts with the off-diagonal cells. Therefore the only unidentified parameters in this model are the weight parameters. This is confirmed in the summary of the model:

```
> summary(A)
```

---

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

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


Deviance Residuals:
     Min       1Q   Median       3Q      Max
-3.63689  -0.50383  0.01714  0.56752  2.25140

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)
AGEM                      0.06364    0.07375   0.863  0.38859
MRMM                     -0.32425    0.07766  -4.175 3.44e-05
FRMF                     -0.25324    0.07681  -3.297  0.00104
MWORK                    -0.06430    0.07431  -0.865  0.38727
MFCM                     -0.06043    0.07123  -0.848  0.39663
Dref(MOPLM, FOPLF).MOPLM -0.33730         NA      NA       NA
Dref(MOPLM, FOPLF).FOPLF -0.02507         NA      NA       NA
Dref(MOPLM, FOPLF).1      4.95123    0.16639  29.757  < 2e-16
Dref(MOPLM, FOPLF).2      4.86328    0.10436  46.601  < 2e-16
Dref(MOPLM, FOPLF).3      4.86458    0.12855  37.842  < 2e-16
Dref(MOPLM, FOPLF).4      4.72343    0.13523  34.928  < 2e-16
Dref(MOPLM, FOPLF).5      4.43516    0.19315  22.963  < 2e-16
Dref(MOPLM, FOPLF).6      4.18873    0.17142  24.435  < 2e-16
Dref(MOPLM, FOPLF).7      4.43379    0.16903  26.231  < 2e-16


(Dispersion parameter for gaussian family taken to be 0.7384355)

Std. Error is NA where coefficient has been constrained or is unidentified

Residual deviance: 425.34 on 576 degrees of freedom
AIC: 1507.8

Number of iterations: 10
```

The over-parameterization of the weights is immaterial, since the weights have been constrained to sum to one as described earlier, so the weights themselves are estimable. The weights may be evaluated as follows:

```
> prop.table(exp(coef(A)[6:7]))

Dref(MOPLM, FOPLF).MOPLM Dref(MOPLM, FOPLF).FOPLF
             0.4225701                0.5774299
```

giving the values reported by van der Slik et al. (2002). All the other coefficients of model A are the same as those reported by van der Slik et al. (2002) except the coefficients of the mother's gender role (MRMM) and the father's gender role (FRMF). van der Slik et al. (2002) 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 the diagonal effects depend on the estimates of these coefficients. 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                     -0.08270
Dref(MOPLM, FOPLF).FOPLF     Dref(MOPLM, FOPLF).1       Dref(MOPLM, FOPLF).2
            0.22955                   4.37372                      4.28579
   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 van der Slik et al. (2002), but the diagonal effects have been adjusted appropriately.

van der Slik et al. (2002) 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 van der Slik et al. (2002),

```
> 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)
                                                   0.79413
     Dref(MOPLM, FOPLF, formula = ~1 + MFCM).MOPLM.MFCM
                                                  -2.51751
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.(Intercept)
                                                  -0.27618
     Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.MFCM
                                                   2.03673
          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)
                                        0.7446574
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.(Intercept)
                                        0.2553426


> prop.table(exp(coef(F)[c(7,9)] + coef(F)[c(6,8)]))

Dref(MOPLM, FOPLF, formula = ~1 + MFCM).MOPLM.MFCM
                                        0.02977308
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.MFCM
                                        0.97022692
```

giving the same weights as in Table 4 of van der Slik et al. (2002).

## 6.3   Uniform Difference (UNIDIFF) Models

Uniform difference models (Xie, 1992; Erikson and Goldthorpe, 1992) 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 $\gamma_{ij}$ 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 (Yaish, 1998, 2004), 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. We omit the sub-table which corresponds to level 7 of *dest*, because its information content is negligible:

```
> set.seed(1)
> data(yaish)
> unidiff <- gnm(Freq ~ educ * orig + educ * dest + Mult(Exp(-1 +
+     educ), -1 + orig:dest), ofInterest = "Mult1.Factor1", family = poisson,
+     data = yaish, subset = (dest != 7))

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

```
> coef(unidiff)

Coefficients of interest:
Mult1.Factor1.educ1 Mult1.Factor1.educ2 Mult1.Factor1.educ3 Mult1.Factor1.educ4
         -0.8242673          -1.0496391          -1.5676909          -1.8632059
Mult1.Factor1.educ5
         -3.0737699
```

The *ofInterest* component has been set to index the multipliers of the association between the social class of father and son. We can contrast each multiplier to that of the lowest education level and obtain the standard errors for these parameters as follows:

```
> getContrasts(unidiff, ofInterest(unidiff))

                       estimate        SE    quasiSE    quasiVar
Mult1.Factor1.educ1   0.0000000 0.0000000 0.09757438 0.00952076
Mult1.Factor1.educ2  -0.2253718 0.1611874 0.12885847 0.01660450
Mult1.Factor1.educ3  -0.7434236 0.2335083 0.21182123 0.04486823
Mult1.Factor1.educ4  -1.0389386 0.3434256 0.32609379 0.10633716
Mult1.Factor1.educ5  -2.2495026 0.9453763 0.93560641 0.87535936
```

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 Cautres et al. (1998), 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)

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

> getContrasts(doubleUnidiff, rev(pickCoef(doubleUnidiff, "Mult1.Factor1")))

                          estimate         SE    quasiSE    quasiVar
Mult1.Factor1.election4 0.00000000 0.00000000 0.07168290 0.005138439
Mult1.Factor1.election3 0.06682585 0.09906916 0.06812239 0.004640660
Mult1.Factor1.election2 0.24052778 0.09116479 0.05702819 0.003252214
Mult1.Factor1.election1 0.32834588 0.12213023 0.09803075 0.009610029

> getContrasts(doubleUnidiff, rev(pickCoef(doubleUnidiff, "Mult2.Factor1")))

                           estimate        SE    quasiSE    quasiVar
Mult2.Factor1.election4  0.00000000 0.0000000 0.10934798 0.011956980
Mult2.Factor1.election3  0.08754435 0.1446833 0.09475938 0.008979340
Mult2.Factor1.election2  0.31990726 0.1320022 0.07395886 0.005469913
Mult2.Factor1.election1 -0.36183013 0.2534754 0.22854400 0.052232362
```

## 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 $\mu_{rc}$ 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, $\alpha_r$ and $\beta_c$ are the row and column main effects, $\gamma_{kr}$ and $\delta_{kc}$ are the row and column scores for multiplicative component $k$ and $\sigma_k$ 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 Vargas et al. (2001), 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 Vargas et al. (2001):

```
> 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. We will proceed by first fitting the main effects model, then using *residSVD* (see Section 5.5) for the parameters of the multiplicative term:

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

Linear predictor - using glm.fit

> svdStart <- residSVD(mainEffects, year, treatment, 3)
> bilinear1 <- update(mainEffects, . ~ . + Mult(year - 1, treatment -
+     1), start = c(coef(mainEffects), svdStart[, 1]))

Running main iterations
Done
```

We can compare the AMMI-1 model to the main effects model,

```
> anova(mainEffects, bilinear1)

Analysis of Deviance Table

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

giving the same results as in Table 1 of Vargas et al. (2001) (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 (Wedderburn, 1974; Gabriel, 1998). 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)

Initialising
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,]  4.1948212 -0.39186806
 [2,]  2.7642419 -0.33951298
 [3,]  1.4250456 -0.04654256
 [4,]  1.8463067  0.33365981
 [5,]  1.2704091  0.15776743
 [6,]  1.1562916  0.40048225
 [7,]  1.0172048  0.72727990
 [8,]  0.6451366  1.46162702
 [9,] -0.1470898  2.13234195

> B

           [,1]        [,2]
 [1,] -2.0673655 -0.97420449
 [2,] -3.0599788 -0.50683009
 [3,] -2.9598021 -0.33190618
 [4,] -1.8086251 -0.49758487
 [5,] -1.5579480 -0.08444513
 [6,] -1.8939998  1.08460534
 [7,] -1.1790432  0.40687015
 [8,] -0.8490092  1.14671353
 [9,] -0.9704664  1.26558193
[10,] -0.6036790  1.39655898
```
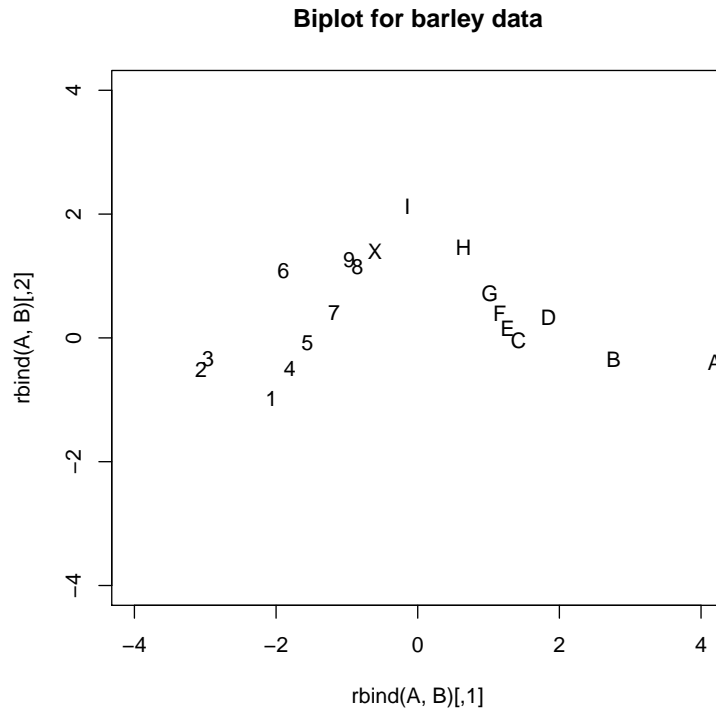
These matrices are essentially the same as in Gabriel (1998). From these the biplot can be produced, for sites *A* . . . *I* and varieties 1 . . . 9, *X*:

```
> 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")
```

**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")
```

**Biplot (rotated) for barley data**
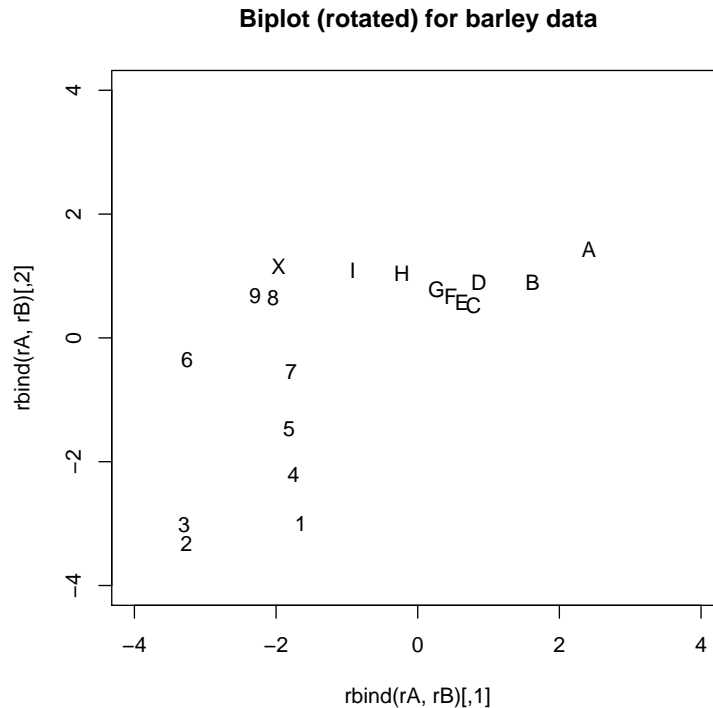


In the original biplot, the co-ordinates for the sites and varieties were given by the rows of A and B respectively, i.e

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

The rotated and scaled biplot suggests the simpler model

$$\alpha_i^T = (\gamma_i, 1)$$
$$\beta_j^T = (\delta_j, \tau_j)$$

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

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

Gabriel (1998) 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)

Initialising
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]  9.513782 15.000000
```

35

## 6.6 Stereotype Model

The stereotype model was proposed by Anderson (1984) 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:

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

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. The gnm package includes the utility function *expandCategorical* to facilitate the required data processing.

For example, the *backPain* data set from Anderson (1984) 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

> backPainLong <- expandCategorical(backPain, "pain")
> backPainLong[1:12, ]

    x1 x2 x3                 pain count id
1    1  1  1               worse     0  1
1.1  1  1  1                same     1  1
1.2  1  1  1   slight.improvement     0  1
1.3  1  1  1 moderate.improvement     0  1
1.4  1  1  1   marked.improvement     0  1
1.5  1  1  1      complete.relief     0  1
2    1  1  1               worse     0  2
2.1  1  1  1                same     0  2
2.2  1  1  1   slight.improvement     0  2
2.3  1  1  1 moderate.improvement     0  2
2.4  1  1  1   marked.improvement     1  2
2.5  1  1  1      complete.relief     0  2
```

We can now fit the stereotype model to these data:

```
> oneDimensional <- gnm(count ~ pain + Mult(pain - 1, x1 + x2 +
+     x3 - 1), eliminate = id, family = "poisson", data = backPainLong)

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

> oneDimensional

Call:
gnm(formula = count ~ pain + Mult(pain - 1, x1 + x2 + x3 - 1),
    eliminate = id, family = "poisson", data = backPainLong)


Coefficients of interest:
                  painsame           painslight.improvement
                   16.1578                          15.6848
      painmoderate.improvement         painmarked.improvement
                   12.4556                          19.9140
```

```
                    paincomplete.relief                        Mult1.Factor1.painworse
                             21.6653                                         -0.3351
                  Mult1.Factor1.painsame           Mult1.Factor1.painslight.improvement
                              1.9154                                          1.7941
  Mult1.Factor1.painmoderate.improvement           Mult1.Factor1.painmarked.improvement
                              1.2622                                          2.4625
          Mult1.Factor1.paincomplete.relief                           Mult1.Factor2.x1
                              2.9238                                         -1.6484
                        Mult1.Factor2.x2                            Mult1.Factor2.x3
                             -0.9455                                         -0.8324
```

```
Deviance:              303.1003
Pearson chi-squared: 433.3727
Residual df:           493
```

specifying the `id` factor through `eliminate` so that the 101 `id` effects are estimated more efficiently and are excluded from printed model summaries by default. 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(count ~ pain + pain:(x1 + x2 + x3), eliminate = id,
+       family = "poisson", data = backPainLong)

Initialising
Running main iterations..........
Done

> threeDimensional

Call:
gnm(formula = count ~ pain + pain:(x1 + x2 + x3), eliminate = id,
    family = "poisson", data = backPainLong)


Coefficients of interest:
                    painsame            painslight.improvement
                     36.0994                           35.7186
      painmoderate.improvement          painmarked.improvement
                     32.6011                           39.8017
            paincomplete.relief                     painworse:x1
                     42.2498                           10.6708
                   painsame:x1          painslight.improvement:x1
                     -2.8855                           -2.5559
   painmoderate.improvement:x1       painmarked.improvement:x1
                     -2.2924                           -4.1156
           paincomplete.relief:x1                  painworse:x2
                     -4.6275                            0.4189
                   painsame:x2          painslight.improvement:x2
                     -2.2551                           -2.1325
   painmoderate.improvement:x2       painmarked.improvement:x2
                     -1.2531                           -2.4249
           paincomplete.relief:x2                  painworse:x3
                     -2.8561                           -1.7476
                   painsame:x3          painslight.improvement:x3
                     -2.9031                           -3.0397
   painmoderate.improvement:x3       painmarked.improvement:x3
                     -2.4938                           -3.4391
           paincomplete.relief:x3
                     -4.6883


Deviance:              299.0152
Pearson chi-squared: 443.0043
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 Anderson (1984) 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.

To obtain estimates of the category-specific multipliers in the stereotype model, we need to constrain both the location and the scale of these parameters. The latter constraint can be imposed by fixing the slope of one of the covariates in the second multiplier to *1*, which may be achieved by specifying the covariate as an offset:

```
> summary(oneDimensional)

Call:
gnm(formula = count ~ pain + Mult(pain - 1, x1 + x2 + x3 - 1),
    eliminate = id, family = "poisson", data = backPainLong)


Deviance Residuals:
    Min      1Q  Median      3Q     Max
-0.9708 -0.6506 -0.4438 -0.1448  2.1385

Coefficients of interest:
                                        Estimate Std. Error z value Pr(>|z|)
painsame                                 16.1578         NA      NA       NA
painslight.improvement                   15.6848     6.5274   2.403  0.01626 *
painmoderate.improvement                 12.4556         NA      NA       NA
painmarked.improvement                   19.9140     6.4976   3.065  0.00218 **
paincomplete.relief                      21.6653         NA      NA       NA
Mult1.Factor1.painworse                  -0.3351         NA      NA       NA
Mult1.Factor1.painsame                    1.9154         NA      NA       NA
Mult1.Factor1.painslight.improvement      1.7941         NA      NA       NA
Mult1.Factor1.painmoderate.improvement    1.2622         NA      NA       NA
Mult1.Factor1.painmarked.improvement      2.4625         NA      NA       NA
Mult1.Factor1.paincomplete.relief         2.9238         NA      NA       NA
Mult1.Factor2.x1                         -1.6484         NA      NA       NA
Mult1.Factor2.x2                         -0.9455         NA      NA       NA
Mult1.Factor2.x3                         -0.8324         NA      NA       NA
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

Std. Error is NA where coefficient has been constrained or is unidentified

Residual deviance: 303.1 on 493 degrees of freedom
AIC: 731.1

Number of iterations: 16
```

```
> oneDimensional <- gnm(count ~ pain + Mult(pain - 1, offset(x1) +
+     x2 + x3 - 1), eliminate = id, family = "poisson", data = backPainLong)

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

> summary(oneDimensional)

Call:
gnm(formula = count ~ pain + Mult(pain - 1, offset(x1) + x2 +
    x3 - 1), eliminate = id, family = "poisson", data = backPainLong)


Deviance Residuals:
    Min      1Q  Median      3Q     Max
-0.9708 -0.6506 -0.4438 -0.1448  2.1385

Coefficients of interest:
                                      Estimate Std. Error z value Pr(>|z|)
painsame                               16.1578     6.5741   2.458 0.013980 *
painslight.improvement                 15.6848     6.5274   2.403 0.016265 *
painmoderate.improvement               12.4555     6.4312   1.937 0.052777 .
painmarked.improvement                 19.9140     6.4975   3.065 0.002178 **
paincomplete.relief                    21.6653     6.5571   3.304 0.000953 ***
Mult1.Factor1.painworse                 1.3694         NA      NA       NA
Mult1.Factor1.painsame                 -2.3404         NA      NA       NA
Mult1.Factor1.painslight.improvement   -2.1403         NA      NA       NA
Mult1.Factor1.painmoderate.improvement -1.2636         NA      NA       NA
Mult1.Factor1.painmarked.improvement   -3.2422         NA      NA       NA
Mult1.Factor1.paincomplete.relief      -4.0025         NA      NA       NA
Mult1.Factor2.x2                        0.5736     0.2178   2.633 0.008451 **
Mult1.Factor2.x3                        0.5050     0.2431   2.077 0.037807 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

Std. Error is NA where coefficient has been constrained or is unidentified

Residual deviance: 303.1 on 493 degrees of freedom
AIC: 731.1

Number of iterations: 14
```

The location of the category-specific multipliers can constrained by setting one of the parameters to zero, either through the *constrain* argument of *gnm* or with *getContrasts*:

```
> getContrasts(oneDimensional, pickCoef(oneDimensional, "Mult.*pain"))

                                         estimate        SE   quasiSE   quasiVar
Mult1.Factor1.painworse                  0.000000  0.000000 1.7797287 3.1674342
Mult1.Factor1.painsame                  -3.709725  1.825561 0.4281331 0.1832980
Mult1.Factor1.painslight.improvement    -3.509685  1.791725 0.4024680 0.1619805
Mult1.Factor1.painmoderate.improvement  -2.632931  1.669250 0.5518544 0.3045433
Mult1.Factor1.painmarked.improvement    -4.611584  1.895233 0.3133219 0.0981706
Mult1.Factor1.paincomplete.relief       -5.371842  1.999651 0.4919552 0.2420199
```

giving the required estimates.

# 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 |

| **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** | |
| --- | --- |
| *confint.gnm* | compute confidence intervals of *gnm* parameters based on the profiled deviance |
| *confint.profile.gnm* | compute confidence intervals of parameters from a *profile.gnm* object |
| *profile.gnm* | profile deviance for parameters in a *gnm* model |
| *plot.profile.gnm* | plot profile traces from a *profile.gnm* object |
| *summary.gnm* | summarize *gnm* fits |
| *residSVD* | multiplicative approximation of model residuals |
| *exitInfo* | print numerical details of last iteration when *gnm* has not converged |
| *ofInterest* | extract the *ofInterest* component of a *gnm* object |
| *ofInterest<-* | replace the *ofInterest* component of a *gnm* object |
| *parameters* | get model parameters from a *gnm* object, including parameters that were constrained |
| *pickCoef* | get indices of model parameters |
| *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** | |
| --- | --- |
| *asGnm* | coerce an object of class *lm* or *glm* to class *gnm* |
| *expandCategorical* | expand a data frame by re-expressing categorical data as counts |
| *getModelFrame* | get the model frame in use by *gnm* |
| *MPinv* | Moore-Penrose pseudoinverse of a real-valued matrix |
| *qrSolve* | Minimum-length solution of a linear system |

# References

Agresti, A. (2002). *Categorical Data Analysis* (2nd ed.). New York: Wiley.

Anderson, J. A. (1984). Regression and ordered categorical variables. *J. R. Statist. Soc. B 46*(1), 1–30.

Catchpole, E. and B. Morgan (1997). Detecting parameter redundancy. *Biometrika 84*, 187–196.

Cautres, B., A. F. Heath, and D. Firth (1998). Class, religion and vote in Britain and France. *La Lettre de la Maison Française 8*.

Clifford, P. and A. F. Heath (1993). The political consequences of social mobility. *J. Roy. Stat. Soc. A 156*(1), 51–61.

Erikson, R. and J. H. Goldthorpe (1992). *The Constant Flux*. Oxford: Clarendon Press.

Erikson, R., J. H. Goldthorpe, and L. Portocarero (1982). Social fluidity in industrial nations: England, France and Sweden. *British Journal of Sociology 33*, 1–34.

Firth, D. (2003). Overcoming the reference category problem in the presentation of statistical models. *Sociological Methodology 33*, 1–18.

Firth, D. and R. X. de Menezes (2004). Quasi-variances. *Biometrika 91*, 65–80.

Gabriel, K. R. (1998). Generalised bilinear regression. *Biometrika 85*, 689–700.

Goodman, L. A. (1979). Simple models for the analysis of association in cross-classifications having ordered categories. *J. Amer. Statist. Assoc. 74*, 537–552.

Hatzinger, R. and B. J. Francis (2004). Fitting paired comparison models in R. Technical Report 3, Department of Statistics and Mathematics, Wirtschaftsuniversität Wien.

McCullagh, P. and J. A. Nelder (1989). *Generalized Linear Models (Second Edition)*. Chapman & Hall Ltd.

Sobel, M. E. (1981). Diagonal mobility models: A substantively motivated class of designs for the analysis of mobility effects. *Amer. Soc. Rev. 46*, 893–906.

Sobel, M. E. (1985). Social mobility and fertility revisited: Some new models for the analysis of the mobility effects hypothesis. *Amer. Soc. Rev. 50*, 699–712.

van der Slik, F. W. P., N. D. de Graaf, and J. R. M. Gerris (2002, 4). Conformity to parental rules: Asymmetric influences of father's and mother's levels of education. *Europ. Soc. Rev. 18*, 489–502.

Vargas, M., J. Crossa, F. van Eeuwijk, K. D. Sayre, and M. P. Reynolds (2001). Interpreting treatment by environment interaction in agronomy trials. *Agronomy Journal 93*, 949–960.

Wedderburn, R. W. M. (1974). Quasi-likelihood functions, generalized linear models, and the Gauss-Newton method. *Biometrika 61*, 439–447.

Xie, Y. (1992). The log-multiplicative layer effect model for comparing mobility tables. *American Sociological Review 57*, 380–395.

Yaish, M. (1998). *Opportunities, Little Change. Class Mobility in Israeli Society, 1974–1991*. Ph. D. thesis, Nuffield College, University of Oxford.

Yaish, M. (2004). *Class Mobility Trends in Israeli Society, 1974-1991*. Lewiston: Edwin Mellen Press.