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

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

For gnm version 0.9-1 , 2006-12-21

## Contents

**1 Introduction** **1**

**2 Generalized linear models** **1**
2.1 Preamble . . . . . 1
2.2 *Diag* and *Symm* . . . . . 2
2.3 *Topo* . . . . . 2
2.4 The *wedderburn* family . . . . . 4
2.5 *termPredictors* . . . . . 5
2.6 Const . . . . . 5

**3 Nonlinear terms** **6**
3.1 Basic mathematical functions of predictors . . . . . 6
3.2 *MultHomog* . . . . . 7
3.3 *Dref* . . . . . 7
3.4 *instances* . . . . . 7
3.5 Custom *nonlin* functions . . . . . 8
3.5.1 General description . . . . . 8
3.5.2 Example: a logistic function . . . . . 9
3.5.3 Example: *MultHomog* . . . . . 9
3.6 Using custom plug-ins to fit nonlinear terms . . . . . 10

**4 Controlling the fitting procedure** **10**
4.1 Basic control parameters . . . . . 10
4.2 Using *start* . . . . . 10
4.3 Using *constrain* . . . . . 11
4.4 Using *eliminate* . . . . . 13

**5 Methods and accessor functions** **15**
5.1 Methods . . . . . 15
5.2 *ofInterest* and *pickCoef* . . . . . 17
5.3 *checkEstimable* . . . . . 18
5.4 *getContrasts*, *se* . . . . . 19
5.5 *residSVD* . . . . . 19

**6 *gnm* or *nls*?** **20**

**7 Examples** **21**
7.1 Row-column association models . . . . . 21
7.1.1 RC(1) model . . . . . 21
7.1.2 RC(2) model . . . . . 23
7.1.3 Homogeneous effects . . . . . 24
7.2 Diagonal reference models . . . . . 25
7.3 Uniform difference (UNIDIFF) models . . . . . 32

*This work was supported by the Economic and Social Research Council (UK) through Professorial Fellowship RES-051-27-0055.

7.4 Generalized additive main effects and multiplicative interaction (GAMMI) models . . . 33
7.5 Biplot models . . . 34
7.6 Stereotype model for multinomial response . . . 37
7.7 Lee-Carter model for trends in age-specific mortality . . . 42
7.8 Exponential and sum-of-exponentials models for decay curves . . . 45
7.8.1 Example: single exponential decay term . . . 46
7.8.2 Example: sum of two exponentials . . . 47

**A User-level functions** **49**

# 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 **?**; 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. **?**, 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 **?**. The data are 7 by 7 versions of social mobility tables from **?**:

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

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

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

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

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

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

```
[1] 71.17401
```

This agrees with the chi-squared value reported on page 331 of **?**, which differs slightly from Wedderburn's 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             6
 0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00 -1.776357e-15
            7             8             9            10            11            12
-8.881784e-16 -8.881784e-16  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
           13            14            15            16
 0.000000e+00 -1.776357e-15 -8.881784e-16  0.000000e+00
```

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

## 2.6 Const

In Section 3 we shall see how nonlinear terms may be specified using functions of class *nonlin* that are analogous to basic mathematical functions. In order to specify certain nonlinear terms in this way, it is necessary to be able to specify a constant in the symbolic expression of a predictor. This is the function of `Const`, such that

```
Const(a)
```

is equivalent to

```
offset(rep(a, nObs))
```

where `nObs` is the number of observations.

## 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. The facility provided with *gnm* for the specification of nonlinear terms is designed to be compatible with the symbolic language used in *formula* objects. Primarily, nonlinear terms are specified in the model formula as calls to functions of the class *nonlin*. There are a number of *nonlin* functions included in the gnm package. Some of these specify simple mathematical functions of predictors: `Exp`, `Mult`, and `Inv`. Others specify more specialized nonlinear terms, in particular `MultHomog` specifies homogeneous multiplicative interactions and `Dref` specifies diagonal reference terms. Users may also define their own *nonlin* functions.

In previous versions of gnm, specialized nonlinear terms were implemented using plug-in functions and users could define custom plug-in functions as described in Section 3.6. Such functions still work in the current version of gnm, but as plug-in functions are less user-friendly than *nonlin* functions, support for plug-in functions is likely to be withdrawn in future versions.

### 3.1 Basic mathematical functions of predictors

Most of the *nonlin* functions included in gnm are basic mathematical functions of predictors:

**`Exp`:** the exponential of a predictor

**`Inv`:** the reciprocal of a predictor

**`Mult`:** the product of predictors

Predictors are specified by symbolic expressions that are interpreted as the right-hand side of a *formula* object, except that an intercept is **not** added by default.

The predictors may contain nonlinear terms, allowing more complex functions to be built up. For example, suppose we wanted to specify a logistic predictor with the same form as that used by `SSlogis` (a selfStart model for use with `nls` — see 6 for more on *gnm* vs. `nls`)

$$\frac{\text{Asym}}{1 + \exp((\text{xmid} - x)/\text{scal})}.$$

This expression could be simplified by re-parameterizing in terms of xmid/scal and 1/scal, however we shall continue with this form for illustration. We could express this predictor symbolically as follows

```
~ -1 + Mult(1, Inv(Const(1) + Exp(Mult(1 + offset(-x), Inv(1)))))
```

However, this is rather convoluted and it may be preferable to define a specialized *nonlin* function in such a case. Section 3.5 explains how users can define custom *nonlin* functions, with a function to specify logistic terms as an example.

One family of models usefully specified with the basic functions is the family of models with multiplicative interactions. For example, the row-column association model

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

also known as the Goodman RC model (**?**), would be specified as a log-link model (for response variable `resp`, say), with formula

```
resp ~ R + C + Mult(R, C)
```

where `R` and `C` are row and column factors respectively. In some contexts, it may be desirable to constrain one or more of the constituent multipliers[1] in a multiplicative interaction to be nonnegative . This may be achieved by specifying the multiplier as an exponential, as in the following 'uniform difference' model (**??**)

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

which would be represented by a formula of the form

```
resp ~ R:T + C:T + Mult(Exp(T), R:C)
```

[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 `MultHomog`

`MultHomog` is a *nonlin* function to specify multiplicative interaction terms 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 `resp`, say, the formula argument to `gnm` would be

```
resp ~ R + C + Diag(R, C) + MultHomog(R, C)
```

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

### 3.3 `Dref`

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

$$w_f \gamma_l$$

for an observation with level $l$ of factor $f$, where $w_f$ is the weight for factor $f$ and $\gamma_l$ 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` specifies the constraints on the weights by defining them as

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

where the $\delta_f$ are the parameters to be estimated.

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 a formula of the form

```
resp ~ -1 + Dref(R, C)
```

The `Dref` function has one specified argument, `delta`, which is a formula with no left-hand side, specifying the dependence (if any) of $\delta_f$ on covariates. For example, the formula

```
resp ~ -1 + x + Dref(R, C, delta = ~ 1 + x)
```

specifies the generalized diagonal reference model

$$\mu_{rci} = \beta x_i + \frac{e^{\xi_{01} + \xi_{11} x_i}}{e^{\xi_{01} + \xi_{11} x_i} + e^{\xi_{02} + \xi_{12} x_i}} \gamma_r + \frac{e^{\xi_{02} + \xi_{12} x_i}}{e^{\xi_{01} + \xi_{11} x_i} + e^{\xi_{02} + \xi_{12} x_i}} \gamma_c.$$

The default value of `delta` 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 $\xi_{.f}$, rather than the implied weights $w_f$.

### 3.4 `instances`

Multiple instances of a linear term will be aliased with each other, but this is not necessarily the case for nonlinear terms. Indeed, there are certain types of model where adding further instances of a nonlinear term is a natural way to extend the model. For example, Goodman's RC model, introduced in section 3.1

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

is naturally extended to the RC(2) model, with a two-component interaction

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

Currently all of the *nonlin* functions in gnm except Dref have an `inst` argument to allow the specification of multiple instances. So the RC(2) model could be specified as follows

```
resp ~ R + C + Mult(R, C, inst = 1) + Mult(R, C, inst = 2)
```

The convenience function `instances` allows multiple instances of a term to be specified at once

```
resp ~ R + C + instances(Mult(R, C), 2)
```

The formula is expanded by `gnm`, so that the instances are treated as separate terms. The `instances` function may be used with any function with an `inst` argument.

## 3.5 Custom *nonlin* functions

### 3.5.1 General description

Users may write their own *nonlin* functions to specify nonlinear terms which can not (easily) be specified using the *nonlin* functions in the gnm package. A function of class *nonlin* should return a list with the following components:

**`predictors`:** a list of symbolic expressions or formulae with no left hand side which represent (possibly nonlinear) predictors that form part of the term.

**`term`:** a function that takes the arguments `predLabels` and `varLabels`, which are labels generated by `gnm` for the specified predictors and variables (see below), and returns a deparsed mathematical expression of the nonlinear term. Only functions recognised by `deriv` should be used in the expression, e.g. `+` rather than `sum`.

Intercepts are added by default to predictors that are specified by formulae. If predictors are named, these names are used as a prefix for parameter labels or as the parameter label itself in the single-parameter case.

The list returned by the *nonlin* function should also include the following components as necessary:

**`variables`:** a list of expressions representing variables in the term (variables with a coefficient of 1).

**`common`:** a numeric index of `predictors` with duplicated indices identifying single factor predictors for which homologous effects are to be estimated.

and may include any of the following components:

**`call`:** a call to be used as a prefix for parameter labels.

**`match`:** (if `call` is non-`NULL`) a numeric index of `predictors` specifying which arguments of `call` the predictors match to — zero indicating no match. If `NULL`, predictors will be matched sequentially to the arguments of `call`.

**`start`:** an optional function which takes a named vector of parameters corresponding to the predictors and returns a vector of starting values for those parameters. This function is ignored if the term is nested within another nonlinear term.

Predictors which are matched to a specified argument of `call` should be given the same name as the argument. Matched predictors are labelled using "dot-style" labelling, e.g. the label for the intercept in the first constituent multiplier of the term `Mult(A, B)` would be `"Mult(. + A, 1 + B).(Intercept)"`. It is recommended that matches are specified wherever possible, to ensure parameter labels are well-defined.

The arguments of *nonlin* functions are as suited to the particular term, but will usually include symbolic representations of predictors in the term and/or the names of variables in the term. The function may also have an `inst` argument to allow specification of multiple instances (see 3.4).

### 3.5.2 Example: a logistic function

As an example, consider writing a *nonlin* function for the logistic term discussed in **??**:

$$\frac{\mathrm{Asym}}{1 + \exp((\mathrm{xmid} - x)/\mathrm{scal})}.$$

We can consider *Asym*, *xmid* and *scal* as the parameters of three separate predictors, each with a single intercept term. Thus our `predictors` component would be

```
list(Asym = 1, xmid = 1, scal = 1)
```

The term also depends on the variable $x$, which would need to be specified by the user. Suppose this is specified to our *nonlin* function through an argument named `x`. Then our *nonlin* function would return the following `variables` component

```
list(substitute(x))
```

We need to use `substitute` here to list the variable specified by the user rather than the variable named `"x"` (if it exists).

Now we need to specify a function that will paste together an expression for the term, given labels for the predictors and the variables:

```
function(predLabels, varLabels) {
  paste(predLabels[1], "/(1 + exp((", predLabels[2], "-",
  varLabels[1], ")/", predLabels[3], "))")
}
```

This would be the `term` component returned by our *nonlin* function.

We now have all the necessary ingredients of a *nonlin* function to specify the logistic term. Since the parameterization does not depend on user-specified values, it does not make sense to use call-matched labelling in this case. The labels for our parameters will be specified by the labels of the `predictors` component. Since we do not anticipate fitting models with multiple logistic terms, our *nonlin* will not return a `call` component with which to prefix the parameter labels. We do however, have some idea of useful starting values, so we will specify the `start` component as

```
function(theta){
    theta[3] <- 1
    theta
}
```

which sets the initial scale parameter to one.

Putting all these ingredients together we have

```
Logistic <- function(x){
    list(predictors = list(Asym = 1, xmid = 1, scal = 1),
         variables = list(substitute(x)),
         term = function(predLabels, varLabels) {
             paste(predLabels[1], "/(1 + exp((", predLabels[2], "-",
             varLabels[1], ")/", predLabels[3], "))")
         },
         start = function(theta){
             theta[3] <- 1
             theta
         })
}
class(Logistic) <- "nonlin"
```

### 3.5.3 Example: `MultHomog`

The `MultHomog` function included in the gnm package provides a further example of a *nonlin* function, showing how to specify a term with quite different features from the preceding example. The definition is

```
MultHomog <- function(..., inst = NULL){
    dots <- match.call(expand.dots = FALSE)[["..."]]
    list(predictors = dots,
         common = rep(1, length(dots)),
         term = function(predLabels, ...) {
             paste("(", paste(predLabels, collapse = ")*("), ")", sep = "")},
         call = as.expression(match.call()),
         match = rep(0, length(dots)))
}
class(MultHomog) <- "nonlin"
```

Firstly, the interaction may be based on any number of factors, hence the use of the special "`...`" argument. The use of `match.call` is equivalent to the use of `substitute` in the `Logistic` function: to obtain expressions for the factors as specified by the user.

The `common` component specifies that homogeneous effects are to be estimated across all the specified factors. The term only depends on these factors, but the `term` function allows for the empty `varLabels` vector that will be passed to it, by having a "`...`" argument.

Since the user may wish to specify multiple instances, a `call` component is returned, so that parameters in different instances of the term will have unique labels (due to the `inst` argument in the call). However as the expressions passed to "`...`" may only represent single factors, rather than general predictors, it is not necessary to use call-matched labelling, so a vector of zeros is returned as the `match` component.

### 3.6 Using custom plug-ins to fit nonlinear terms

Prior to the introduction of *nonlin* functions, nonlinear terms that could not be specified and estimated using the in-built capability of `gnm` had to be fitted using plug-in functions. The plug-in functions previously distributed with gnm have now been implemented as *nonlin* functions, however user-specified plug-in functions may still be used with the current version of gnm as documented in `?Nonlin`. Nevertheless, support for plug-in functions is likely to be withdrawn in future versions, in favour of the simpler *nonlin* functions.

## 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(R, C))
model2 <- gnm(mu ~ R + C + instances(Mult(R, C), 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 *nonlin* functions or 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 fully terms specified by steps 2 and 3.

5. Run starting iterations: update one at a time any nonlinear parameters not specified by steps 2 and 3, 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 7.1:

```
> data(occupationalStatus)
> set.seed(1)
> RChomog1 <- gnm(Freq ~ origin + destination + Diag(origin, destination) +
+     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)                       0.0017   0.1090
origin2                           0.5275   0.5198
origin3                           1.6575   1.6288
```

```
origin4                          2.0003   1.9510
origin5                          0.7817   0.7318
origin6                          2.8644   2.7965
origin7                          1.5548   1.4723
origin8                          1.3029   1.2118
destination2                     0.9465   0.9388
destination3                     2.0020   1.9732
destination4                     2.2887   2.2395
destination5                     1.6811   1.6312
destination6                     3.1679   3.1000
destination7                     2.3064   2.2239
destination8                     1.8783   1.7872
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  -1.5439   1.5087
MultHomog(origin, destination)2  -1.3256   1.2905
MultHomog(origin, destination)3  -0.7275   0.6923
MultHomog(origin, destination)4  -0.1436   0.1084
MultHomog(origin, destination)5  -0.1264   0.0912
MultHomog(origin, destination)6   0.3853  -0.4205
MultHomog(origin, destination)7   0.8015  -0.8367
MultHomog(origin, destination)8   1.0451  -1.0802
```

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) +
    MultHomog(origin, destination), family = poisson, data = occupationalStatus,
    verbose = FALSE)


Deviance Residuals:
       Min          1Q      Median          3Q         Max
-1.659e+00  -4.297e-01  -1.825e-08   3.862e-01   1.721e+00

Coefficients:
                                Estimate Std. Error z value Pr(>|z|)
(Intercept)                      0.10903         NA      NA       NA
origin2                          0.51978         NA      NA       NA
origin3                          1.62883         NA      NA       NA
origin4                          1.95104         NA      NA       NA
origin5                          0.73179         NA      NA       NA
origin6                          2.79654         NA      NA       NA
origin7                          1.47229         NA      NA       NA
origin8                          1.21183         NA      NA       NA
destination2                     0.93879         NA      NA       NA
destination3                     1.97324         NA      NA       NA
destination4                     2.23947         NA      NA       NA
destination5                     1.63121         NA      NA       NA
destination6                     3.10002         NA      NA       NA
destination7                     2.22389         NA      NA       NA
```

```
destination8                     1.78721         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  1.50875         NA      NA        NA
MultHomog(origin, destination)2  1.29045         NA      NA        NA
MultHomog(origin, destination)3  0.69229         NA      NA        NA
MultHomog(origin, destination)4  0.10841         NA      NA        NA
MultHomog(origin, destination)5  0.09124         NA      NA        NA
MultHomog(origin, destination)6 -0.42052         NA      NA        NA
MultHomog(origin, destination)7 -0.83666         NA      NA        NA
MultHomog(origin, destination)8 -1.08023         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: 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 7.6 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`:

```
> ## 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|)
```

[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 other *nonlin* terms.

[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.

```
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])

        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 **?** 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.

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.

| | |
|---|---|
| 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.

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 generalized 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 7.3:

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

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

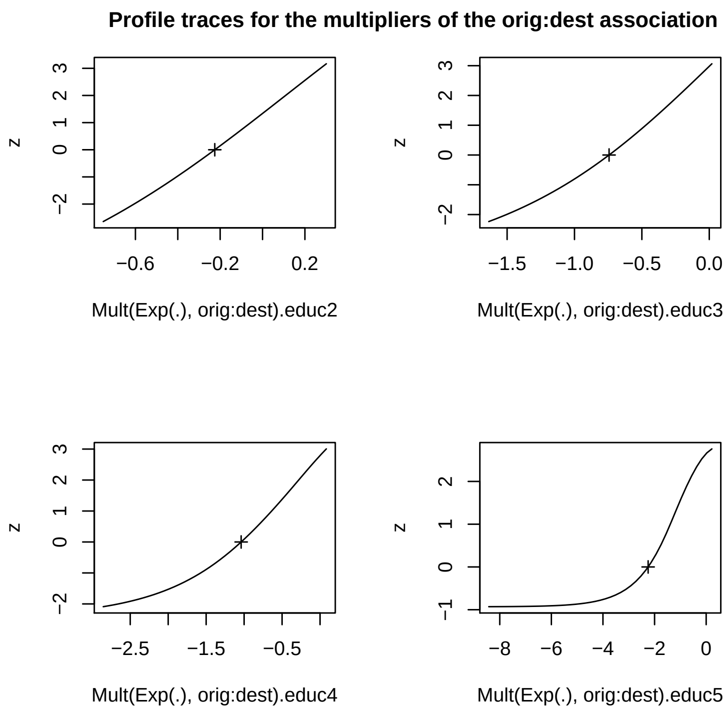


Figure 3: Profile traces for the multipliers of the orig:dest association

From these plots we can see that the deviance is approximately quadratic in `Mult(Exp(.), orig:dest).educ2`, assymetric in `Mult(Exp(.), orig:dest).educ3` and `Mult(Exp(.), orig:dest).educ4` and asymptotic in `Mult(Exp(.), orig:dest).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[, "Mult(Exp(.), orig:dest).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[, "Mult(Exp(.), orig:dest).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, "[.]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 **?**.

Consider the following model, that is described later in Section 7.3:

```
> data(cautres)
> doubleUnidiff <- gnm(Freq ~ election:vote + election:class:religion +
+     Mult(Exp(election), religion:vote) + Mult(Exp(election), class:vote),
+     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(", religion:vote", 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]

                                  Mult(Exp(.), religion:vote).election2
Mult(Exp(.), religion:vote).election1                                    -1
Mult(Exp(.), religion:vote).election2                                     1
Mult(Exp(.), religion:vote).election3                                     0
Mult(Exp(.), religion:vote).election4                                     0
                                  Mult(Exp(.), religion:vote).election3
Mult(Exp(.), religion:vote).election1                                    -1
Mult(Exp(.), religion:vote).election2                                     0
Mult(Exp(.), religion:vote).election3                                     1
Mult(Exp(.), religion:vote).election4                                     0
                                  Mult(Exp(.), religion:vote).election4
Mult(Exp(.), religion:vote).election1                                    -1
Mult(Exp(.), religion:vote).election2                                     0
Mult(Exp(.), religion:vote).election3                                     0
Mult(Exp(.), religion:vote).election4                                     1
```

Then their estimability can be checked using `checkEstimable`

```
> checkEstimable(doubleUnidiff, contrMatrix)

Mult(Exp(.), religion:vote).election1 Mult(Exp(.), religion:vote).election2
                                   NA                                  TRUE
Mult(Exp(.), religion:vote).election3 Mult(Exp(.), religion:vote).election4
                                 TRUE                                  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("[.]religion", 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)

Mult(Exp(election), .).religion1:vote1 Mult(Exp(election), .).religion2:vote1
                                    NA                                  FALSE
Mult(Exp(election), .).religion3:vote1 Mult(Exp(election), .).religion4:vote1
                                 FALSE                                  FALSE
Mult(Exp(election), .).religion1:vote2 Mult(Exp(election), .).religion2:vote2
                                 FALSE                                  FALSE
Mult(Exp(election), .).religion3:vote2 Mult(Exp(election), .).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, ", religion:vote"))
> myContrasts

                                        estimate        SE    quasiSE    quasiVar
Mult(Exp(.), religion:vote).election1  0.0000000 0.0000000 0.09803075 0.009610029
Mult(Exp(.), religion:vote).election2 -0.0878181 0.1136832 0.05702819 0.003252214
Mult(Exp(.), religion:vote).election3 -0.2615200 0.1184134 0.06812239 0.004640660
Mult(Exp(.), religion:vote).election4 -0.3283459 0.1221302 0.07168290 0.005138439
```

Visualization of estimated contrasts using 'quasi standard errors' (**??**) is achieved by plotting the resulting object:

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

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 7.5 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 + instances(Mult(site, variety), 2), family = wedderburn,
+     data = barley, start = biplotStart)

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

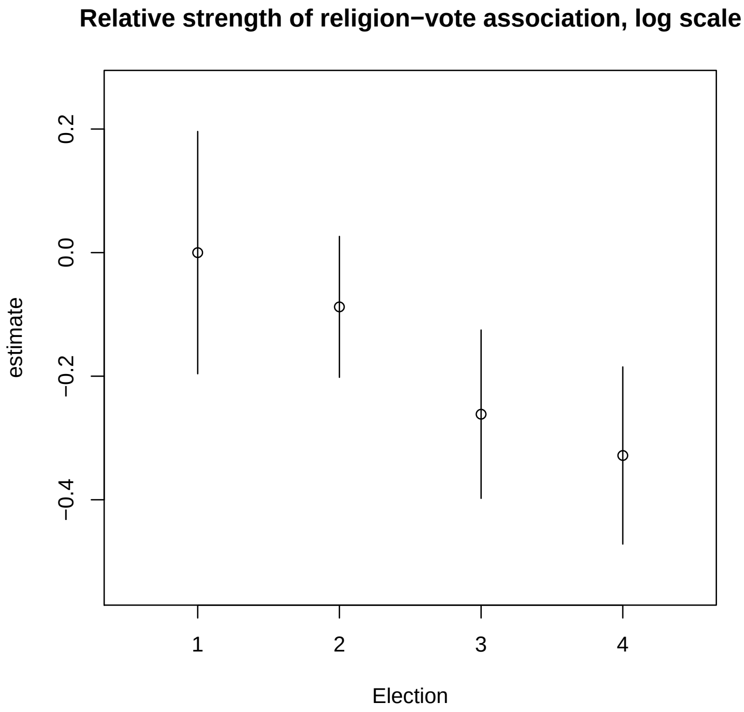


Figure 4: Relative strength of religion-vote association, log scale

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 7.4), 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 5:

# 6 gnm or nls?

The `nls` function in the stats package may be used to fit a nonlinear model via least-squares estimation. Statistically speaking, `gnm` is to `nls` as `glm` is to `lm`, in that a nonlinear least-squares model is equivalent to a generalized nonlinear model with `family = gaussian`. Whilst `nls` assumes that the responses are distributed either with constant variance or with fixed relative variances, `gnm` allows for responses distributed with variances that are a specified (via the `family` argument) function of the mean.

In terms of the interface, however, `gnm` and `nls` are quite different. Models are specified to `nls` in terms of a mathematical formula, which is convenient for models that have a small number of parameters. For models that have a large number of parameters, or can not easily be represented by a mathematical formula, the symbolic model specification used by `gnm` may be more convenient. This would usually be the case for models involving factors, which would need to be represented by dummy variables in a `nls` formula.

When working with artificial data, `gnm` has the minor advantage that it does not fail when a model is an exact fit to the data (see `help(nls)`). Therefore it is not necessary with `gnm` to add noise to artificial data, which can be useful when testing methods.

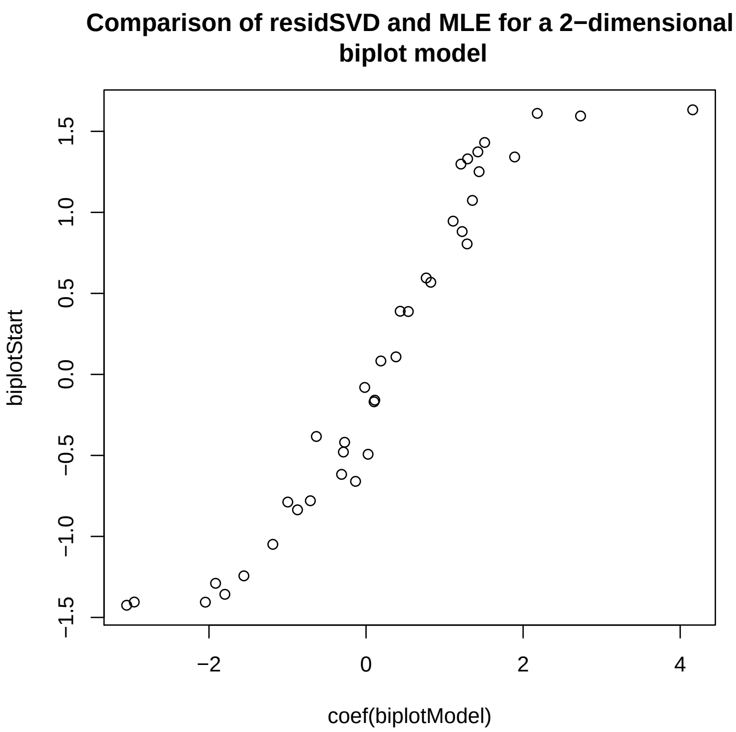


Figure 5: Comparison of residSVD and the MLE for a 2-dimensional biplot model

# 7 Examples

This section provides some examples of the wide range of models that may be fitted using the gnm package. Sections 7.1, 7.2 and 7.3 consider various models for contingency tables; Section 7.4 considers AMMI and GAMMI models which are typically used in agricultural applications, and Section 7.6 considers the stereotype model, which is used to model an ordinal response.

## 7.1 Row-column association models

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

### 7.1.1 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 **?** 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(SES, MHS), family = poisson,
+     data = mentalHealth)

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

> RC1model

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


Coefficients:
            (Intercept)                    SESB                    SESC
                3.83543                -0.06739                 0.10800
                   SESD                    SESE                    SESF
                0.40196                 0.01966                -0.20842
               MHSmild             MHSmoderate             MHSimpaired
                0.71188                 0.20370                 0.24956
      Mult(., MHS).SESA       Mult(., MHS).SESB       Mult(., MHS).SESC
                0.95853                 0.96636                 0.32099
      Mult(., MHS).SESD       Mult(., MHS).SESE       Mult(., MHS).SESF
               -0.02141                -0.86716                -1.56200
   Mult(SES, .).MHSwell     Mult(SES, .).MHSmild Mult(SES, .).MHSmoderate
                0.32802                 0.03048                -0.02322
Mult(SES, .).MHSimpaired
               -0.27035

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
Mult(., MHS).SESA Mult(., MHS).SESB Mult(., MHS).SESC Mult(., MHS).SESD Mult(., MHS).SESE
       1.11233085        1.12143706        0.37107608       -0.02702931       -1.01036141
Mult(., MHS).SESF
      -1.81823304
```

```
$nu
    Mult(SES, .).MHSwell     Mult(SES, .).MHSmild Mult(SES, .).MHSmoderate
              1.6775144                0.1403989               -0.1369926
Mult(SES, .).MHSimpaired
             -1.4136909
```

### 7.1.2 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 + instances(Mult(SES, MHS), 2), family = poisson,
+     data = mentalHealth)

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

> RC2model

Call:
gnm(formula = count ~ SES + MHS + instances(Mult(SES, MHS), 2),
    family = poisson, data = mentalHealth)


Coefficients:
                       (Intercept)                                 SESB
                         3.8603353                           -0.0556106
                              SESC                                 SESD
                         0.1097869                            0.3284008
                              SESE                                 SESF
                        -0.0381411                           -0.1524531
                           MHSmild                          MHSmoderate
                         0.7102766                            0.1892520
                       MHSimpaired         Mult(., MHS, inst = 1).SESA
                         0.2488493                           -0.0175314
       Mult(., MHS, inst = 1).SESB         Mult(., MHS, inst = 1).SESC
                        -0.2146166                           -0.2679378
       Mult(., MHS, inst = 1).SESD         Mult(., MHS, inst = 1).SESE
                         0.8399440                            0.3076632
       Mult(., MHS, inst = 1).SESF      Mult(SES, ., inst = 1).MHSwell
                        -1.9282818                            0.6285745
    Mult(SES, ., inst = 1).MHSmild  Mult(SES, ., inst = 1).MHSmoderate
                         0.0791084                           -0.0465425
Mult(SES, ., inst = 1).MHSimpaired         Mult(., MHS, inst = 2).SESA
                        -0.4539996                           -0.3575749
       Mult(., MHS, inst = 2).SESB         Mult(., MHS, inst = 2).SESC
                        -0.4908029                           -0.2886945
       Mult(., MHS, inst = 2).SESD         Mult(., MHS, inst = 2).SESE
                         0.5515182                            0.5001104
       Mult(., MHS, inst = 2).SESF      Mult(SES, ., inst = 2).MHSwell
                        -0.6576153                           -0.8149901
    Mult(SES, ., inst = 2).MHSmild  Mult(SES, ., inst = 2).MHSmoderate
                        -0.0092990                            0.0006713
Mult(SES, ., inst = 2).MHSimpaired
                         0.8711873
```

```
Deviance:            0.5225353
Pearson chi-squared: 0.523331
Residual df:         3
```

### 7.1.3 Homogeneous effects

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

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

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

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


Coefficients:
                     (Intercept)                          origin2
                        -0.71372                          0.57472
                         origin3                          origin4
                         1.83431                          2.30346
                         origin5                          origin6
                         1.08854                          3.28208
                         origin7                          origin8
                         2.06256                          1.86340
                    destination2                     destination3
                         0.99373                          2.17872
                    destination4                     destination5
                         2.59190                          1.98795
                    destination6                     destination7
                         3.58556                          2.81416
                    destination8       Diag(origin, destination)1
                         2.43878                          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                         -1.76042
 MultHomog(origin, destination)2  MultHomog(origin, destination)3
                        -1.54213                         -0.94396
 MultHomog(origin, destination)4  MultHomog(origin, destination)5
                        -0.36008                         -0.34291
 MultHomog(origin, destination)6  MultHomog(origin, destination)7
```

```
                         0.16885                          0.58499
MultHomog(origin, destination)8
                         0.82856

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) + 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.

## 7.2 Diagonal reference models

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

### `Dref` example 1: Political consequences of social mobility

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

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

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

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

See Section 3.3 for more detail on the parameterization.

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

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

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

> classMobility

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


Coefficients:
                   (Intercept)  Dref(origin, destination)delta1
                      -1.34325                         -0.30736
Dref(origin, destination)delta2   Dref(., .).origin|destination1
                      -0.05501                         -0.83454
 Dref(., .).origin|destination2   Dref(., .).origin|destination3
                       0.21066                         -0.61159
 Dref(., .).origin|destination4   Dref(., .).origin|destination5
                       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)delta1 Dref(origin, destination)delta2
                      0.4372469                       0.5627531
```

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

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

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

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

```
> 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 ~ Dref(origin, destination, delta = ~1 +
+     downward + upward), family = binomial, data = voting)

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

> socialMobility
```

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


Coefficients:
                                                                         (Intercept)
                                                                            -1.32619
           Dref(upward, origin, delta = ~ . + 1 + downward).delta1(Intercept)
                                                                             0.30662
Dref(upward, origin, delta = ~ destination + . + downward).delta1downwardTRUE
                                                                             0.90538
         Dref(upward, origin, delta = ~ destination + 1 + .).delta1upwardTRUE
                                                                             0.46989
           Dref(upward, origin, delta = ~ . + 1 + downward).delta2(Intercept)
                                                                             0.69338
Dref(upward, origin, delta = ~ destination + . + downward).delta2downwardTRUE
                                                                             0.09462
         Dref(upward, origin, delta = ~ destination + 1 + .).delta2upwardTRUE
                                                                             0.53011
         Dref(., ., delta = ~ destination + 1 + downward).origin|destination1
                                                                            -0.73140
         Dref(., ., delta = ~ destination + 1 + downward).origin|destination2
                                                                             0.21349
         Dref(., ., delta = ~ destination + 1 + downward).origin|destination3
                                                                            -0.66859
         Dref(., ., delta = ~ destination + 1 + downward).origin|destination4
                                                                             0.75704
         Dref(., ., delta = ~ destination + 1 + downward).origin|destination5
                                                                             1.38378

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(upward, origin, delta = ~ destination + 1 + .).delta1upwardTRUE
                                                           0.3900791
Dref(upward, origin, delta = ~ destination + 1 + .).delta2upwardTRUE
                                                           0.6099209

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

Dref(upward, origin, delta = ~ destination + . + downward).delta1downwardTRUE
                                                                    0.6044395
Dref(upward, origin, delta = ~ destination + . + downward).delta2downwardTRUE
                                                                    0.3955605

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

Dref(upward, origin, delta = ~ . + 1 + downward).delta1(Intercept)
                                                          0.404496
Dref(upward, origin, delta = ~ . + 1 + downward).delta2(Intercept)
                                                          0.595504
```

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

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

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

> downwardMobility

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


Coefficients:
                                                           (Intercept)
                                                              -1.32728
Dref(origin, destination, delta = ~ . + downward).delta1(Intercept)
                                                               0.29561
      Dref(origin, destination, delta = ~ 1 + .).delta1downwardTRUE
                                                               0.90537
Dref(origin, destination, delta = ~ . + downward).delta2(Intercept)
                                                               0.70439
      Dref(origin, destination, delta = ~ 1 + .).delta2downwardTRUE
                                                               0.09463
             Dref(., ., delta = ~ 1 + downward).origin|destination1
                                                              -0.74258
             Dref(., ., delta = ~ 1 + downward).origin|destination2
                                                               0.22076
             Dref(., ., delta = ~ 1 + downward).origin|destination3
                                                              -0.66437
             Dref(., ., delta = ~ 1 + downward).origin|destination4
                                                               0.75421
             Dref(., ., delta = ~ 1 + downward).origin|destination5
                                                               1.38358

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, delta = ~ 1 + .).delta1downwardTRUE
                                                     0.5991571
Dref(origin, destination, delta = ~ 1 + .).delta2downwardTRUE
                                                     0.4008429

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

Dref(origin, destination, delta = ~ . + downward).delta1(Intercept)
                                                             0.3992031
Dref(origin, destination, delta = ~ . + downward).delta2(Intercept)
                                                             0.6007969
```

### `Dref` example 2: conformity to parental rules

Another application of diagonal reference models is given by **?**. The data from this paper are not publicly available[4], but we shall show how the models presented in the paper may be estimated using `gnm`.

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

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

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

$$\mu_{rci} = \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \beta_4 x_{4i} + \beta_5 x_{5i} + \frac{e^{\delta_1}}{e^{\delta_1} + e^{\delta_2}}\gamma_r + \frac{e^{\delta_2}}{e^{\delta_1} + e^{\delta_2}}\gamma_c.$$

The baseline model can be fitted as follows:

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

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


Coefficients:
                   AGEM                       MRMM                       FRMF
                0.06363                   -0.32425                   -0.25324
                  MWORK                       MFCM  Dref(MOPLM, FOPLF)delta1
               -0.06430                   -0.06043                   -0.33731
Dref(MOPLM, FOPLF)delta2   Dref(., .).MOPLM|FOPLF1   Dref(., .).MOPLM|FOPLF2
               -0.02505                    4.95121                    4.86329
 Dref(., .).MOPLM|FOPLF3   Dref(., .).MOPLM|FOPLF4   Dref(., .).MOPLM|FOPLF5
                4.86458                    4.72343                    4.43516
 Dref(., .).MOPLM|FOPLF6   Dref(., .).MOPLM|FOPLF7
                4.18873                    4.43378

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)

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


Deviance Residuals:
     Min        1Q    Median        3Q       Max
-3.63688  -0.50383   0.01714   0.56753   2.25139

Coefficients:
                         Estimate Std. Error t value Pr(>|t|)
AGEM                      0.06363    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)delta1 -0.33731         NA      NA       NA
Dref(MOPLM, FOPLF)delta2 -0.02505         NA      NA       NA
Dref(., .).MOPLM|FOPLF1   4.95121    0.16639  29.757  < 2e-16 ***
Dref(., .).MOPLM|FOPLF2   4.86329    0.10436  46.602  < 2e-16 ***
Dref(., .).MOPLM|FOPLF3   4.86458    0.12855  37.842  < 2e-16 ***
Dref(., .).MOPLM|FOPLF4   4.72343    0.13523  34.929  < 2e-16 ***
Dref(., .).MOPLM|FOPLF5   4.43516    0.19314  22.963  < 2e-16 ***
Dref(., .).MOPLM|FOPLF6   4.18873    0.17142  24.435  < 2e-16 ***
Dref(., .).MOPLM|FOPLF7   4.43378    0.16903  26.231  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(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: 15
```

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)delta1 Dref(MOPLM, FOPLF)delta2
               0.4225638                0.5774362
```

giving the values reported by **?**. All the other coefficients of model A are the same as those reported by **?** except the coefficients of the mother's gender role (MRMM) and the father's gender role (FRMF). **?** reversed the signs of the coefficients of these factors since they were coded in the direction of liberal values, unlike the other covariates. However, simply reversing the signs of these coefficients does not give the same model, since the estimates of 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 +
+           Dref(MOPLM, FOPLF), family = gaussian, data = conformity,
+           verbose = FALSE)
> A

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


Coefficients:
                    AGEM                     MRMM2                      FRMF2
                 0.06363                   0.32425                    0.25324
                   MWORK                      MFCM  Dref(MOPLM, FOPLF)delta1
                -0.06430                  -0.06043                    0.08440
Dref(MOPLM, FOPLF)delta2   Dref(., .).MOPLM|FOPLF1   Dref(., .).MOPLM|FOPLF2
                 0.39666                   4.37371                    4.28579
 Dref(., .).MOPLM|FOPLF3   Dref(., .).MOPLM|FOPLF4   Dref(., .).MOPLM|FOPLF5
                 4.28708                   4.14593                    3.85767
 Dref(., .).MOPLM|FOPLF6   Dref(., .).MOPLM|FOPLF7
```

```
                3.61123                     3.85629

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

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

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

$$\mu_{rci} = \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \beta_4 x_{4i} + \beta_5 x_{5i} + \frac{e^{\xi_{01}+\xi_{11}x_{5i}}}{e^{\xi_{01}+\xi_{11}x_{5i}} + e^{\xi_{02}+\xi_{12}x_{5i}}}\gamma_r + \frac{e^{\xi_{02}+\xi_{12}x_{5i}}}{e^{\xi_{01}+\xi_{11}x_{5i}} + e^{\xi_{02}+\xi_{12}x_{5i}}}\gamma_c.$$

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

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

Call:
gnm(formula = MCFM ~ -1 + AGEM + MRMM + FRMF + MWORK + MFCM +
    Dref(MOPLM, FOPLF, delta = ~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, delta = ~ . + MFCM).delta1(Intercept)
                                                      1.03515
          Dref(MOPLM, FOPLF, delta = ~ 1 + .).delta1MFCM
                                                     -1.77756
Dref(MOPLM, FOPLF, delta = ~ . + MFCM).delta2(Intercept)
                                                     -0.03515
          Dref(MOPLM, FOPLF, delta = ~ 1 + .).delta2MFCM
                                                      2.77756
             Dref(., ., delta = ~ 1 + MFCM).MOPLM|FOPLF1
                                                      4.82476
             Dref(., ., delta = ~ 1 + MFCM).MOPLM|FOPLF2
                                                      4.88066
             Dref(., ., delta = ~ 1 + MFCM).MOPLM|FOPLF3
                                                      4.83969
             Dref(., ., delta = ~ 1 + MFCM).MOPLM|FOPLF4
                                                      4.74850
             Dref(., ., delta = ~ 1 + MFCM).MOPLM|FOPLF5
                                                      4.42020
             Dref(., ., delta = ~ 1 + MFCM).MOPLM|FOPLF6
                                                      4.17957
             Dref(., ., delta = ~ 1 + MFCM).MOPLM|FOPLF7
                                                      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, delta = ~ . + MFCM).delta1(Intercept)
                                               0.7446523
Dref(MOPLM, FOPLF, delta = ~ . + MFCM).delta2(Intercept)
                                               0.2553477

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

Dref(MOPLM, FOPLF, delta = ~ 1 + .).delta1MFCM
                                    0.02974698
Dref(MOPLM, FOPLF, delta = ~ 1 + .).delta2MFCM
                                    0.97025302
```

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

## 7.3 Uniform difference (UNIDIFF) models

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

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

The $\gamma_{ij}$ 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 (**??**), 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(educ), orig:dest),
+     ofInterest = "[.]educ", family = poisson, data = yaish, subset = (dest !=
+         7))

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

> coef(unidiff)

Coefficients of interest:
Mult(Exp(.), orig:dest).educ1 Mult(Exp(.), orig:dest).educ2 Mult(Exp(.), orig:dest).educ3
                   -0.5513258                    -0.7766976                    -1.2947494
Mult(Exp(.), orig:dest).educ4 Mult(Exp(.), orig:dest).educ5
                   -1.5902644                    -2.8008285
```

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
Mult(Exp(.), orig:dest).educ1  0.0000000 0.0000000 0.09757438 0.00952076
Mult(Exp(.), orig:dest).educ2 -0.2253718 0.1611874 0.12885847 0.01660450
Mult(Exp(.), orig:dest).educ3 -0.7434236 0.2335083 0.21182123 0.04486823
Mult(Exp(.), orig:dest).educ4 -1.0389386 0.3434256 0.32609380 0.10633716
Mult(Exp(.), orig:dest).educ5 -2.2495026 0.9453764 0.93560643 0.87535940
```

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

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

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

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

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

> getContrasts(doubleUnidiff, rev(pickCoef(doubleUnidiff, ", religion:vote")))

                                            estimate         SE    quasiSE    quasiVar
Mult(Exp(.), religion:vote).election4 0.00000000 0.00000000 0.07168290 0.005138439
Mult(Exp(.), religion:vote).election3 0.06682585 0.09906916 0.06812239 0.004640660
Mult(Exp(.), religion:vote).election2 0.24052778 0.09116479 0.05702819 0.003252214
Mult(Exp(.), religion:vote).election1 0.32834589 0.12213023 0.09803075 0.009610029

> getContrasts(doubleUnidiff, rev(pickCoef(doubleUnidiff, "[.]religion")))

Mult(Exp(election), .).religion4:vote2 Mult(Exp(election), .).religion3:vote2
                                    NA                                  FALSE
Mult(Exp(election), .).religion2:vote2 Mult(Exp(election), .).religion1:vote2
                                 FALSE                                  FALSE
Mult(Exp(election), .).religion4:vote1 Mult(Exp(election), .).religion3:vote1
                                 FALSE                                  FALSE
Mult(Exp(election), .).religion2:vote1 Mult(Exp(election), .).religion1:vote1
                                 FALSE                                  FALSE
Note: not all of the specified contrasts in this set are estimable
                                       Estimate Std. Error
Mult(Exp(election), .).religion4:vote2        0          0
```

## 7.4 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 7.1 are examples of GAMMI models, with a log link and poisson variance. Here we illustrate the use of an AMMI model, which is a GAMMI model with an identity link and a constant variance.

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

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

Now we can fit the AMMI-1 model, to the scaled yields using the combined treatment factor and the year factor from the `wheat` dataset. 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, treatment), 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, treatment)
  Resid. Df Resid. Dev  Df Deviance
1       207     279515
2       176     128383  31   151133
```

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

## 7.5 Biplot models

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

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

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

```
> data(barley)
> set.seed(1)
> biplotModel <- gnm(y ~ -1 + instances(Mult(site, variety), 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,]  4.1948225 -0.39186722
 [2,]  2.7642411 -0.33951388
 [3,]  1.4250454 -0.04654266
 [4,]  1.8463067  0.33365989
 [5,]  1.2704088  0.15776722
 [6,]  1.1562913  0.40048199
 [7,]  1.0172048  0.72727987
 [8,]  0.6451366  1.46162701
 [9,] -0.1470898  2.13234201

> B

           [,1]       [,2]
 [1,] -2.0673648 -0.9742045
 [2,] -3.0599797 -0.5068301
 [3,] -2.9598031 -0.3319063
 [4,] -1.8086247 -0.4975848
 [5,] -1.5579477 -0.0844451
 [6,] -1.8939995  1.0846055
 [7,] -1.1790432  0.4068701
 [8,] -0.8490092  1.1467135
 [9,] -0.9704664  1.2655820
[10,] -0.6036789  1.3965588
```

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

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

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

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

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

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

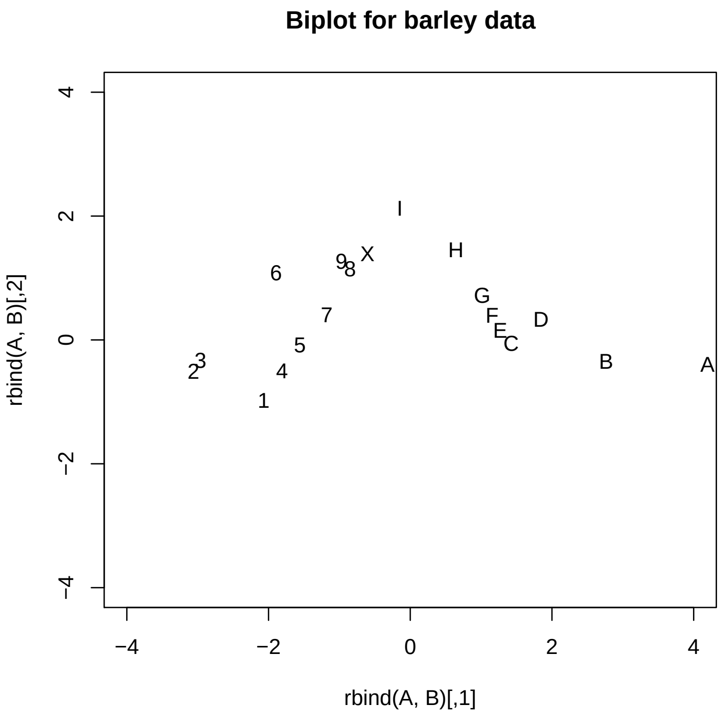


Figure 6: Biplot for barley data

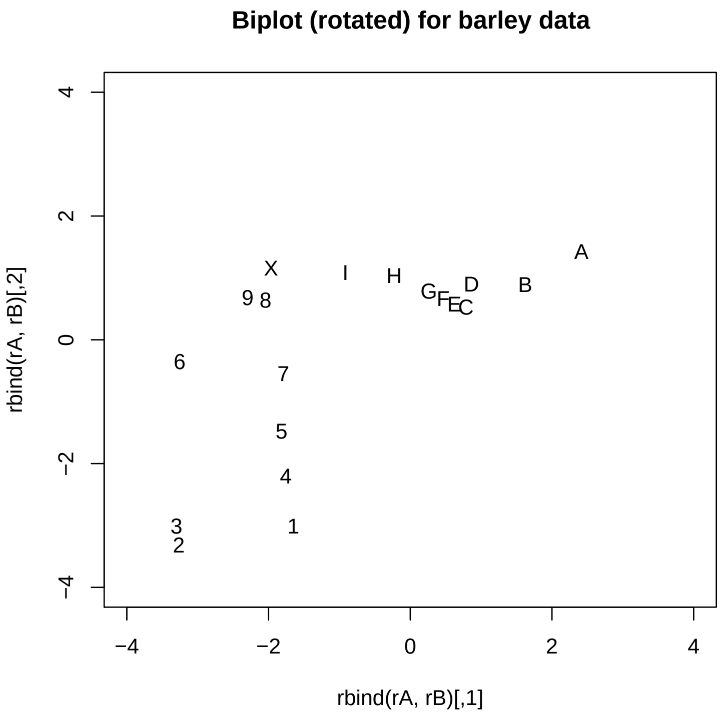


Figure 7: Rotated biplot for barley data

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:

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

**?** 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]  9.513774 15.000000
```

## 7.6 Stereotype model for multinomial response

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

$$\log \mu_{ic} = \beta_{0c} + \gamma_c \sum_r \boldsymbol{\beta}_r \boldsymbol{x}_{ir}.$$

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 **?** 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, x1 + x2 + x3), eliminate = id,
+     family = "poisson", data = backPainLong)

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

> oneDimensional

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


Coefficients of interest:
                                   painsame
                                    16.1578
                     painslight.improvement
                                    15.6848
                   painmoderate.improvement
                                    12.4556
                     painmarked.improvement
                                    19.9140
                        paincomplete.relief
                                    21.6653
              Mult(., x1 + x2 + x3).painworse
                                     0.1595
               Mult(., x1 + x2 + x3).painsame
                                    -1.4973
  Mult(., x1 + x2 + x3).painslight.improvement
                                    -1.4080
Mult(., x1 + x2 + x3).painmoderate.improvement
                                    -1.0164
  Mult(., x1 + x2 + x3).painmarked.improvement
                                    -1.9001
     Mult(., x1 + x2 + x3).paincomplete.relief
                                    -2.2396
                   Mult(pain, . + x2 + x3).x1
                                     2.2390
                   Mult(pain, x1 + . + x3).x2
                                     1.2842
                   Mult(pain, x1 + x2 + .).x3
                                     1.1307

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     painmoderate.improvement
                  36.3326                      35.9518                      32.8344
     painmarked.improvement          paincomplete.relief                  painworse:x1
                  40.0350                      42.4830                      10.2481
              painsame:x1    painslight.improvement:x1  painmoderate.improvement:x1
                  -3.4248                      -3.0952                      -2.8318
  painmarked.improvement:x1       paincomplete.relief:x1                  painworse:x2
                  -4.6550                      -5.1669                       0.3331
              painsame:x2    painslight.improvement:x2  painmoderate.improvement:x2
                  -2.3409                      -2.2183                      -1.3389
  painmarked.improvement:x2       paincomplete.relief:x2                  painworse:x3
                  -2.5107                      -2.9419                      -2.9783
              painsame:x3    painslight.improvement:x3  painmoderate.improvement:x3
                  -4.1338                      -4.2704                      -3.7246
  painmarked.improvement:x3       paincomplete.relief:x3
                  -4.6699                      -5.9190

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 **?** 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, x1 + x2 + x3), 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         NA      NA       NA
painmoderate.improvement                        12.4556         NA      NA       NA
painmarked.improvement                          19.9140         NA      NA       NA
paincomplete.relief                             21.6653         NA      NA       NA
Mult(., x1 + x2 + x3).painworse                  0.1595         NA      NA       NA
Mult(., x1 + x2 + x3).painsame                  -1.4973         NA      NA       NA
Mult(., x1 + x2 + x3).painslight.improvement    -1.4080         NA      NA       NA
Mult(., x1 + x2 + x3).painmoderate.improvement  -1.0164         NA      NA       NA
Mult(., x1 + x2 + x3).painmarked.improvement    -1.9001         NA      NA       NA
Mult(., x1 + x2 + x3).paincomplete.relief       -2.2396         NA      NA       NA
Mult(pain, . + x2 + x3).x1                       2.2390         NA      NA       NA
Mult(pain, x1 + . + x3).x2                       1.2842         NA      NA       NA
Mult(pain, x1 + x2 + .).x3                       1.1307         NA      NA       NA

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

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

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

> summary(oneDimensional)

Call:
gnm(formula = count ~ pain + Mult(pain, offset(x1) + x2 + x3),
    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
painsame                                                16.1578     6.5741   2.458
painslight.improvement                                  15.6848     6.5274   2.403
painmoderate.improvement                                12.4555     6.4312   1.937
painmarked.improvement                                  19.9140     6.4975   3.065
paincomplete.relief                                     21.6653     6.5571   3.304
Mult(., x2 + x3 + offset(x1)).painworse                  1.3211         NA      NA
Mult(., x2 + x3 + offset(x1)).painsame                  -2.3886         NA      NA
Mult(., x2 + x3 + offset(x1)).painslight.improvement    -2.1886         NA      NA
```

```
Mult(., x2 + x3 + offset(x1)).painmoderate.improvement  -1.3118         NA      NA
Mult(., x2 + x3 + offset(x1)).painmarked.improvement    -3.2905         NA      NA
Mult(., x2 + x3 + offset(x1)).paincomplete.relief       -4.0508         NA      NA
Mult(pain, . + x3 + offset(x1)).x2                       0.5736     0.2178   2.633
Mult(pain, x2 + . + offset(x1)).x3                       0.5050     0.2431   2.077
                                                        Pr(>|z|)
painsame                                                0.013980
painslight.improvement                                  0.016265
painmoderate.improvement                                0.052777
painmarked.improvement                                  0.002178
paincomplete.relief                                     0.000953
Mult(., x2 + x3 + offset(x1)).painworse                       NA
Mult(., x2 + x3 + offset(x1)).painsame                        NA
Mult(., x2 + x3 + offset(x1)).painslight.improvement          NA
Mult(., x2 + x3 + offset(x1)).painmoderate.improvement        NA
Mult(., x2 + x3 + offset(x1)).painmarked.improvement          NA
Mult(., x2 + x3 + offset(x1)).paincomplete.relief             NA
Mult(pain, . + x3 + offset(x1)).x2                      0.008451
Mult(pain, x2 + . + offset(x1)).x3                      0.037807

(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: 13
```

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"))

               Mult(., x2 + x3 + offset(x1)).painworse
                                                    NA
                Mult(., x2 + x3 + offset(x1)).painsame
                                                  TRUE
  Mult(., x2 + x3 + offset(x1)).painslight.improvement
                                                  TRUE
Mult(., x2 + x3 + offset(x1)).painmoderate.improvement
                                                  TRUE
  Mult(., x2 + x3 + offset(x1)).painmarked.improvement
                                                  TRUE
     Mult(., x2 + x3 + offset(x1)).paincomplete.relief
                                                  TRUE
                    Mult(pain, . + x3 + offset(x1)).x2
                                                 FALSE
                    Mult(pain, x2 + . + offset(x1)).x3
                                                 FALSE
Note: not all of the specified contrasts in this set are estimable
                                                        estimate       SE   quasiSE
Mult(., x2 + x3 + offset(x1)).painworse                 0.000000 0.000000 1.7797286
Mult(., x2 + x3 + offset(x1)).painsame                 -3.709724 1.825561 0.4281331
Mult(., x2 + x3 + offset(x1)).painslight.improvement   -3.509685 1.791725 0.4024680
Mult(., x2 + x3 + offset(x1)).painmoderate.improvement -2.632931 1.669250 0.5518545
Mult(., x2 + x3 + offset(x1)).painmarked.improvement   -4.611584 1.895233 0.3133219
Mult(., x2 + x3 + offset(x1)).paincomplete.relief      -5.371842 1.999651 0.4919552
                                                        quasiVar
Mult(., x2 + x3 + offset(x1)).painworse                3.1674338
Mult(., x2 + x3 + offset(x1)).painsame                 0.1832980
Mult(., x2 + x3 + offset(x1)).painslight.improvement   0.1619805
```

```
Mult(., x2 + x3 + offset(x1)).painmoderate.improvement 0.3045433
Mult(., x2 + x3 + offset(x1)).painmarked.improvement   0.0981706
Mult(., x2 + x3 + offset(x1)).paincomplete.relief      0.2420199
```

giving the required estimates.

## 7.7 Lee-Carter model for trends in age-specific mortality

In the study and projection of population mortality rates, the model proposed by **?** forms the basis of many if not most current analyses. Here we consider the quasi-Poisson version of the model (**????**), in which the death count $D_{ay}$ for individuals of age $a$ in year $y$ has mean $\mu_{ay}$ and variance $\phi\mu_{ay}$ (where $\phi$ is 1 for Poisson-distributed counts, and is respectively greater than or less than 1 in cases of over-dispersion or under-dispersion). In the Lee-Carter model, the expected counts follow the log-bilinear form

$$\log(\mu_{ay}/e_{ay}) = \alpha_a + \beta_a \gamma_y,$$

where $e_{ay}$ is the 'exposure' (number of lives at risk). This is a generalized nonlinear model with a single multiplicative term.

The use of gnm to fit this model is straightforward. We will illustrate by using data from the Human Mortality Database[5] (HMD, at `http://www.mortality.org`) on male deaths in Canada between 1921 and 2003. The data are not made available as part of gnm because of license restrictions; but they are readily available via the web simply by registering with the HMD. We assume that the data for Canadian males (both deaths and exposure-to-risk) have been downloaded from the HMD and organised into a data frame named `Canada` in R, with columns `Year` (a factor, with levels `1921` to `2003`), `Age` (a factor, with levels `20` to `99`), `mDeaths` and `mExposure` (both quantitative). The Lee-Carter model may then be specified as

```
LCmodel.male <- gnm(mDeaths ~ Age + Mult(Exp(Age), Year),
                    offset = log(mExposure), family = "quasipoisson",
                    data = Canada)
```

Here we have acknowledged the fact that the model only really makes sense if all of the $\beta_a$ parameters, which represent the 'sensitivity' of age group $a$ to a change in the level of general mortality (e.g., **?**), have the same sign. Without loss of generality we assume $\beta_a > 0$ for all $a$, and we impose this constraint by using `Exp(Age)` instead of just `Age` in the multiplicative term. Convergence is to a fitted model with residual deviance 32422.68 on 6400 degrees of freedom — representing clear evidence of substantial overdispersion relative to the Poisson distribution. In order to explore the lack of fit a little further, we plot the distribution of Pearson residuals in Figure 8:

```
par(mfrow = c(2,2))
age <- as.numeric(as.character(Canada$Age))
with(Canada,{
    res <- residuals(LCmodel.male, type = "pearson")
    plot(Age, res, xlab="Age", ylab="Pearson residual",
         main = "(a) Residuals by age")
    plot(Year, res, xlab="Year", ylab="Pearson residual",
         main = "(b) Residuals by year")
    plot(Year[(age>24) & (age<36)], res[(age>24) & (age<36)],
         xlab = "Year", ylab = "Pearson residual",
         main = "(c) Age group 25-35")
    plot(Year[(age>49) & (age<66)], res[(age>49) & (age<66)],
         xlab = "Year", ylab = "Pearson residual",
         main = "(d) Age group 50-65")
})
```

Panel (a) of Figure 8 indicates that the overdispersion is not evenly spread through the data, but is largely concentrated in two age groups, roughly ages 25–35 and 50–65. Panels (c) and (d) focus on the residuals in each of these two age groups: there is a clear (and roughly cancelling) dependence on `Year`, indicating that the assumed bilinear interaction between `Age` and `Year` does not hold for the full range of ages and years considered here.

A somewhat more satisfactory Lee-Carter model fit is obtained if only a subset of the data is used, namely only those males aged 45 or over:

[5] Thanks to Iain Currie for helpful advice relating to this section

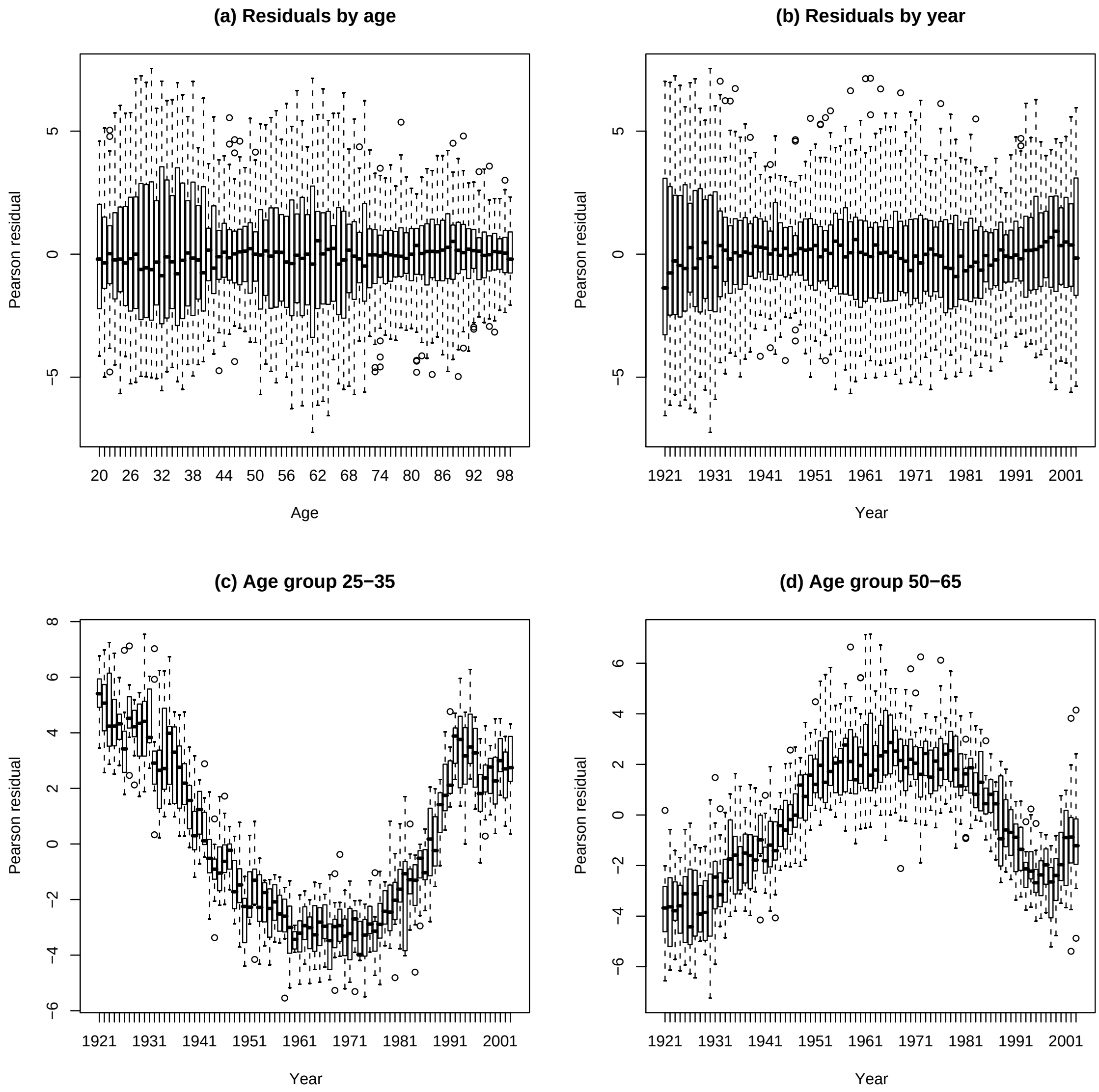


Figure 8: Canada, males: plots of residuals from the Lee-Carter model of mortality

```
LCmodel.maleOver45 <- gnm(mDeaths ~ Age + Mult(Exp(Age), Year),
                          offset = log(mExposure), family = "quasipoisson",
                          data = Canada[age>44,])
```

The residual deviance now is 12595.44 on 4375 degrees of freedom: still substantially overdispersed, but less severely so than before. Again we plot the distributions of Pearson residuals (Figure 9). Still clear departures from the assumed

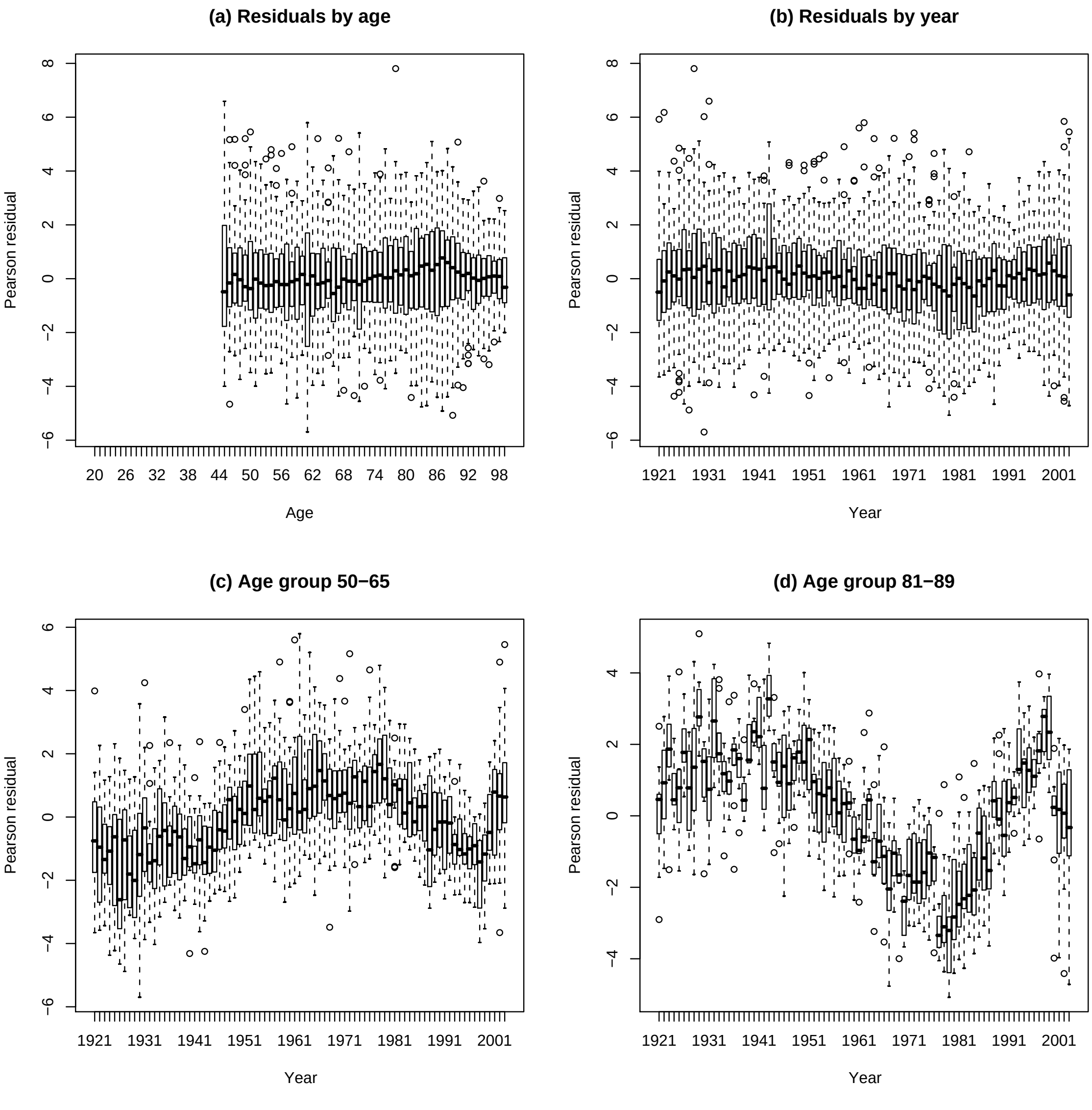


Figure 9: Canada, males over 45: plots of residuals from the Lee-Carter model of mortality

bilinear structure are evident, especially for age group 81–89; but they are less pronounced than in the previous model fit.

The main purpose here is only to illustrate how straightforward it is to work with the Lee-Carter model using *gnm*, but we will take this example a little further by examining the estimated $\beta_a$ parameters from the last fitted model. We can use `getContrasts` to compute quasi standard errors for the logarithms of $\hat{\beta}_a$ — the logarithms being the result of having used `Exp(Age)` in the model specification — and use these in a plot of the coefficients:

```
AgeContrasts <- getContrasts(LCmodel.maleOver45, 56:100) ## ages 45 to 89 only
```

The plot shows that sensitivity to the general level of mortality is highest at younger ages, as expected. An *unexpected* feature is the clear outlying positions occupied by the estimates for ages 51, 61, 71 and 81: for each of those ages,

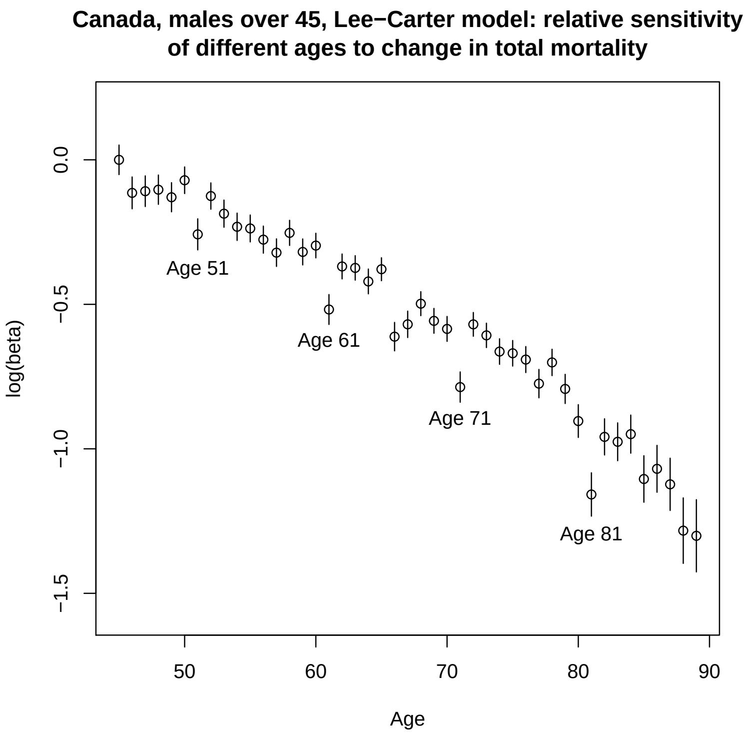


Figure 10: Canada, males over 45, Lee-Carter model: relative sensitivity of different ages to change in total mortality.

the estimated $\beta_a$ coefficient is substantially less than it is for the neighbouring age groups (and the error bars indicate clearly that the deviations are larger than could plausibly be due to chance variation). This is a curious finding. A partial explanation comes from a look back at the raw death-count data. In the years between 1921 and 1940, the death counts for ages 31, 41, 51, 61, 71 and 81 all stand out as being very substantially lower than those of neighbouring ages (Figure 11: the ages concerned are highlighted in solid red). The same does *not* hold for later years: after about 1940, the '1' ages fall in with the general pattern. We do not know the reason for this, but it does explain our finding above regarding the $\beta_a$ coefficients: whilst all age groups have benefited from the general trend of reduced mortality, the '1' age groups appear to have benefited least because their starting point (in the 1920s and 1930s) was lower than would have been indicated by the general pattern — hence $\hat{\beta}_a$ is smaller for ages $a = 31$, $a = 41, \ldots, a = 81$.

## 7.8 Exponential and sum-of-exponentials models for decay curves

A class of nonlinear functions which arise in various application contexts — a notable one being pharmacokinetic studies – involves one or more *exponential decay* terms. For example, a simple decay model with additive error is

$$y = \alpha + \exp(\beta + \gamma x) + e \tag{1}$$

(with $\gamma < 0$), while a more complex ('sum of exponentials') model might involve two decay terms:

$$y = \alpha + \exp(\beta_1 + \gamma_1 x) + \exp(\beta_2 + \gamma_2 x) + e. \tag{2}$$

Estimation and inference with such models are typically not straightforward, partly on account of multiple local maxima in the likelihood (e.g., **?**, Ch.3). We illustrate the difficulties here, with a couple of artificial examples. These examples will make clear the value of making repeated calls to `gnm`, in order to use different, randomly-generated parameterizations and starting values and thus improve the chances of locating both the global maximum and all local maxima of the likelihood.

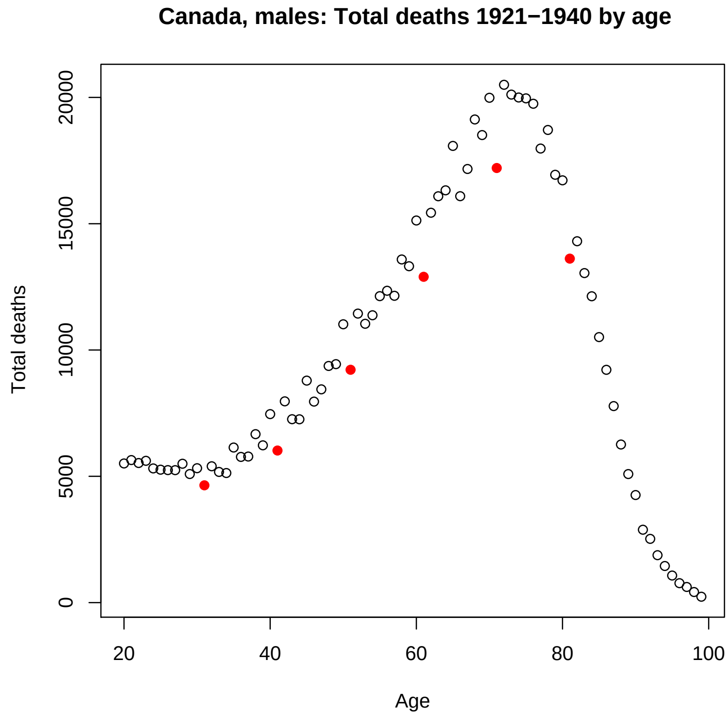


Figure 11: Canada, males: Deaths 1921 to 1940 by age

#### 7.8.1 Example: single exponential decay term

Let us first construct some data from model (1). For our illustrative purposes here, we will use *noise-free* data, i.e., we fix the variance of $e$ to be zero; for the other parameters we will use $\alpha = 0, \beta = 0, \gamma = -0.1$.

```
> x <- 1:100
> y <- exp(-x/10)
> set.seed(1)
> saved.fits <- list()
> for (i in 1:100) saved.fits[[i]] <- gnm(y ~ Exp(1 + x), verbose = FALSE)
> table(zapsmall(sapply(saved.fits, deviance)))

        0 3.612654
       52       48
```

The `saved.fits` object thus contains the results of 100 calls to `gnm`, each using a different, randomly-generated starting value for the vector of parameters $(\alpha, \beta, \gamma)$. Out of 100 fits, 52 reproduce the data exactly, to machine accuracy. The remaining 48 fits are all identical to one another, but they are far from globally optimal, with residual sum of squares 3.61: they result from divergence of $\hat{\gamma}$ to $+\infty$, and correspondingly of $\hat{\beta}$ to $-\infty$, such that the fitted 'curve' is in fact just a constant, with level equal to $\bar{y} = 0.09508$. For example, the second of the 100 fits is of this kind:

```
> saved.fits[[2]]

Call:
gnm(formula = y ~ Exp(1 + x), verbose = FALSE)


Coefficients:
          (Intercept)  Exp(. + x).(Intercept)            Exp(1 + .).x
            9.508e-02              -1.493e+04               5.464e+01
```

```
Deviance:            3.612654
Pearson chi-squared: 3.612654
Residual df:         99
```

The use of repeated calls to *gnm*, as here, allows the local and global maxima to be easily distinguished.

### 7.8.2 Example: sum of two exponentials

We can conduct a similar exercise based on the more complex model (2):

```
> x <- 1:100
> y <- exp(-x/10) + 2 * exp(-x/50)
> set.seed(1)
> saved.fits <- list()
> for (i in 1:100) saved.fits[[i]] <- suppressWarnings(gnm(y ~ Exp(1 +
+     x, inst = 1) + Exp(1 + x, inst = 2), verbose = FALSE))
> unlist(sapply(saved.fits, deviance))

 [1] 2.051091e-13 4.164390e+01 1.589496e-01 4.164390e+01 1.589496e-01 4.164390e+01
 [7] 1.589496e-01 2.832244e-13 4.164390e+01 1.589496e-01 4.164390e+01 1.589496e-01
[13] 7.631865e-14 6.684421e-20
```

In this instance, only 14 of the 100 calls to *gnm* have successfully located a local maximum of the likelihood: in the remaining 86 cases the starting values generated were such that numerical problems resulted, and the fitting algorithm was abandoned (giving a *NULL* result). Among the 14 'successful' fits, it is evident that there are three distinct solutions (with respective residual sums of squares equal to 0.1589, 41.6, and essentially zero — the last of these, the exact fit to the data, having been found 4 times out of the above 13). The two non-optimal local maxima here correspond to the best fit with a single exponential (which has residual sum of squares 0.1589) and to the fit with no dependence at all on $x$ (residual sum of squares 41.6), as we can see by comparing with:

```
> singleExp <- gnm(y ~ Exp(1 + x), start = c(NA, NA, -0.1), verbose = FALSE)
> singleExp

Call:
gnm(formula = y ~ Exp(1 + x), start = c(NA, NA, -0.1), verbose = FALSE)


Coefficients:
          (Intercept)  Exp(. + x).(Intercept)            Exp(1 + .).x
              0.25007                 0.93664                -0.03465

Deviance:            0.1589496
Pearson chi-squared: 0.1589496
Residual df:         97

> meanOnly <- gnm(y ~ 1, verbose = FALSE)
> meanOnly

Call:
gnm(formula = y ~ 1, verbose = FALSE)


Coefficients:
(Intercept)
     0.9511

Deviance:            41.6439
Pearson chi-squared: 41.6439
Residual df:         99
```

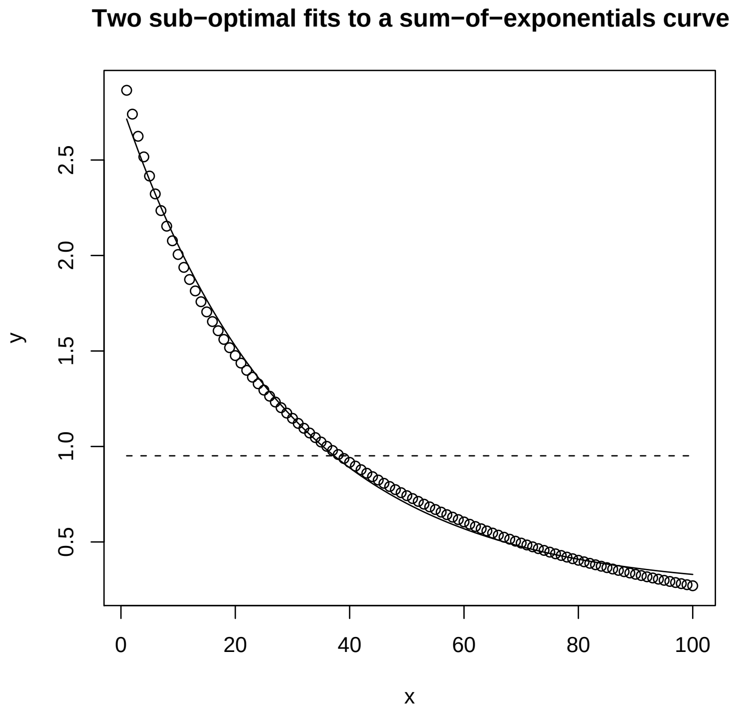


Figure 12: Two sub-optimal fits to a sum-of-exponentials curve

```
> plot(x, y, main = "Two sub-optimal fits to a sum-of-exponentials curve")
> lines(x, fitted(singleExp))
> lines(x, fitted(meanOnly), lty = "dashed")
```

In this example, it is clear that even a small amount of noise in the data would make it practically impossible to distinguish between competing models containing one and two exponential-decay terms.

In summary: the default `gnm` setting of randomly-chosen starting values is useful for identifying multiple local maxima in the likelihood; and reasonably good starting values are needed if the global maximum is to be found. In the present example, knowing that $\gamma_1$ and $\gamma_2$ should both be small and negative, we might perhaps have tried

```
> gnm(y ~ instances(Exp(1 + x), 2), start = c(NA, NA, -0.1, NA, -0.1),
+     verbose = FALSE)

Call:
gnm(formula = y ~ instances(Exp(1 + x), 2), start = c(NA, NA,
    -0.1, NA, -0.1), verbose = FALSE)


Coefficients:
                    (Intercept)  Exp(. + x, inst = 1).(Intercept)
                      5.972e-08                        -1.316e-08
          Exp(1 + ., inst = 1).x  Exp(. + x, inst = 2).(Intercept)
                     -1.000e-01                         6.931e-01
          Exp(1 + ., inst = 2).x
                     -2.000e-02

Deviance:            9.991902e-14
Pearson chi-squared: 9.991902e-14
Residual df:         95
```

which reliably yields the (globally optimal) perfect fit to the data.

# 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 |
| *Const* | specify a constant in a *gnm* model formula |
| *Dref* | specify a diagonal reference term in a *gnm* model formula |
| *Mult* | specify a product of predictors in a *gnm* formula |
| *MultHomog* | specify a multiplicative interaction with homogeneous effects in a *gnm* formula |
| *Exp* | specify the exponential of a predictor in a *gnm* formula |
| *Inv* | specify the reciprocal of a predictor in a *gnm* formula |
| *Nonlin* | specify a special nonlinear term in a *gnm* formula (using external plug-in function) |
| *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 |