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

Heather Turner and David Firth*
University of Warwick, UK

For gnm version 0.9-5 , 2008-06-05

Contents

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

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 ith factor. The *spec* argument specifies the interaction level corresponding to every possible combination of the input factors, and the result is a new factor representing the specified interaction.

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

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

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

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

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

```
> ### 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 -8.881784e-16  0.000000e+00  0.000000e+00 -8.881784e-16  0.000000e+00
            7             8             9            10            11            12
 0.000000e+00 -8.881784e-16  0.000000e+00  0.000000e+00  1.776357e-15  0.000000e+00
           13            14            15            16
 0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
```

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

3 Nonlinear terms

The main purpose of the gnm package is to provide a flexible framework for the specification and estimation of generalized models with nonlinear terms. 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.

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 section 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)))))
```

where *Const* is a convenience function to specify a constant in a *nonlin* term, equivalent to *offset(rep(1, nObs))* where *nObs* is the number of observations. 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)
```

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.

[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.

3.3 *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 γ_l is the "diagonal effect" for level l.

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

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

where the δ_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 δ_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 δ_f or the $\xi_{.f}$, rather than the implied weights w_f. However, these weights may be obtained from a fitted model using the *DrefWeights* function, which computes the corresponding standard errors using the delta method.

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 of arguments for the internal function `nonlinTerms`. The following arguments must be specified in all cases:

`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 following arguments of `nonlinTerms` must be specified whenever applicable to the nonlinear term:

`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.

The arguments below are optional:

`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 not be matched to the arguments of `call`.

`start`: a 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 3.1:

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

We can consider *Asym*, *xmid* and *scal* as the parameters of three separate predictors, each with a single intercept term. Thus we specify the `predictors` argument to `nonlinTerms` as

```
predictors = 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 specify the following `variables` argument

```
variables = 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).

Our *nonlin* function must also specify the `term` argument to `nonlinTerms`. This is a function that will paste together an expression for the term, given labels for the predictors and the variables:

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

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 taken from the labels of the `predictors` argument. Since we do not anticipate fitting models with multiple logistic terms, our *nonlin* function will not specify a `call` argument with which to prefix the parameter labels. We do however, have some idea of useful starting values, so we will specify the `start` argument as

```
start = 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()))
}
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 analogous to the use of `substitute` in the `Logistic` function: to obtain expressions for the factors as specified by the user.

The returned `common` argument 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, the `call` argument to `nonlinTerms` is specified, 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 the `match` argument is not specified here.

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 *lsMethod*, *ridge*, *tolerance*, *iterStart* and *iterMax*. Starting values for the parameter estimates can be set by *start* and parameters can be constrained via *constrain* and *constrainTo* arguments. 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 *lsMethod* argument specifies what numerical method is to be used to solve the (typically rank-deficient) least squares problem at the heart of the *gnm* fitting algorithm: the options are direct solution using a QR decomposition (*"qr"*), and matrix inversion via Cholesky decomposition (*"chol"*). In both cases, the design matrix is standardized and regularized (in the Levenberg-Marquardt sense) prior to solving; the *ridge* argument provides a degree of control over the regularization performed (smaller values may sometimes give faster convergence but can lead to numerical instability). If *lsMethod* is left unspecified, the default is *"qr"*, unless *eliminate* is used in which case the default *lsMethod* used is *"chol"*.

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 fitting 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*. For example, to get the number *10* for the value of *start* above, we could have done

```
gnm(mu ~ R + C + instances(Mult(R, C), 2), method = "coefNames")
```

from whose output it would be seen that there are 10 new coefficients in *model2*. When called with *method = "coefNames"*, *gnm* makes no attempt to fit the specified model; instead it returns just the names that the coefficients in the fitted model object would have.

The starting procedure used by *gnm* is as follows:

1. Begin with all parameters set to *NA*.
2. Replace *NA* values with any starting values set by *nonlin* functions.
3. Replace current values with any (non-*NA*) starting values specified by the *start* argument of *gnm*.

4. Set any values specified by the *constrain* argument to the values specified by the *constrainTo* argument (see Section 4.3).

5. Categorise remaining *NA* parameters as linear or nonlinear, treating non-*NA* parameters as fixed. Initialise the nonlinear parameters by generating values θ_i from the Uniform$(-0.1, 0.1)$ distribution and shifting these values away from zero as follows
$$\theta_i = \begin{cases} \theta_i - 0.1 & \text{if } \theta_i < 1 \\ \theta_i + 0.1 & \text{otherwise} \end{cases}$$

6. Compute the *glm* estimate of the linear parameters, offsetting the contribution to the predictor of any terms fully determined by steps 2 to 5.

7. Run starting iterations: update nonlinear parameters one at a time, jointly re-estimating linear parameters after each round of updates.

Note that no starting iterations (step 7) will be run if all parameters are linear, or if all nonlinear parameters are specified by *start*, *constrain* or a *nonlin* 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 typically be over-parameterized, and *gnm* will return a random parameterization for unidentified coefficients (determined by the randomly chosen starting values for the iterative algorithm, step 5 above).

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)                     0.1281  -0.0753
origin2                         0.5184   0.5329
origin3                         1.6237   1.6777
origin4                         1.9422   2.0349
origin5                         0.7228   0.8167
origin6                         2.7843   2.9121
origin7                         1.4574   1.6128
origin8                         1.1954   1.3669
destination2                    0.9374   0.9519
destination3                    1.9681   2.0221
destination4                    2.2306   2.3234
destination5                    1.6222   1.7161
destination6                    3.0878   3.2156
destination7                    2.2090   2.3644
destination8                    1.7708   1.9423
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   1.5024  -1.5686
MultHomog(origin, destination)2   1.2841  -1.3504
MultHomog(origin, destination)3   0.6860  -0.7522
MultHomog(origin, destination)4   0.1021  -0.1683
MultHomog(origin, destination)5   0.0849  -0.1511
MultHomog(origin, destination)6  -0.4268   0.3606
MultHomog(origin, destination)7  -0.8430   0.7768
MultHomog(origin, destination)8  -1.0866   1.0203
```

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
-1.6588  -0.4297   0.0000   0.3862   1.7208

Coefficients:
                                 Estimate Std. Error z value Pr(>|z|)
(Intercept)                      -0.07530         NA      NA       NA
origin2                           0.53285         NA      NA       NA
origin3                           1.67773         NA      NA       NA
origin4                           2.03492         NA      NA       NA
origin5                           0.81670         NA      NA       NA
origin6                           2.91210         NA      NA       NA
origin7                           1.61278         NA      NA       NA
origin8                           1.36691         NA      NA       NA
destination2                      0.95187         NA      NA       NA
destination3                      2.02215         NA      NA       NA
destination4                      2.32335         NA      NA       NA
destination5                      1.71612         NA      NA       NA
destination6                      3.21558         NA      NA       NA
destination7                      2.36438         NA      NA       NA
destination8                      1.94228         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 -1.56865         NA      NA       NA
MultHomog(origin, destination)2 -1.35035         NA      NA       NA
MultHomog(origin, destination)3 -0.75219         NA      NA       NA
MultHomog(origin, destination)4 -0.16831         NA      NA       NA
MultHomog(origin, destination)5 -0.15114         NA      NA       NA
MultHomog(origin, destination)6  0.36062         NA      NA       NA
```

```
MultHomog(origin, destination)7  0.77676         NA      NA       NA
MultHomog(origin, destination)8  1.02033         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] 121.007

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

[1] 19.985

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

[1] 2462.556 2462.556
```

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

```
> summary(temp.gnm)

Call:

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


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

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

(Dispersion parameter for poisson family taken to be 1)

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

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

Number of iterations: 3
```

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

[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.

```
> 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
```

See Section 5.4 for further details of the *getContrasts* function.

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

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

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

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

In addition to the accessor functions shown in Figure 1, the gnm package provides a new generic function called *termPredictors* that has methods for objects of class *gnm*, *glm* and *lm*. This function returns the additive contribution of each term to the predictor. See Section 2.5 for an example of its use.

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

Similarly, *profile.gnm* and *confint.gnm* are only applicable to estimable parameters. The deviance function of a 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:

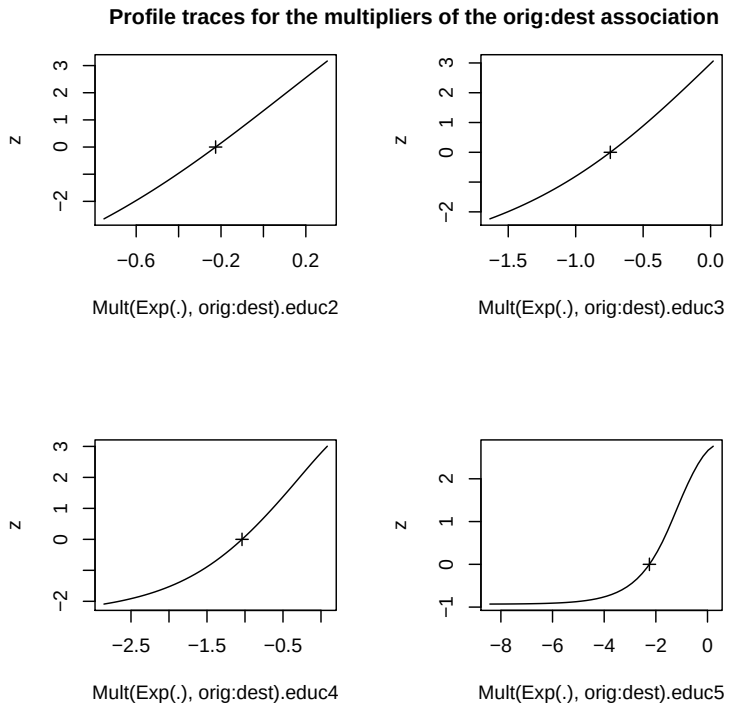

Figure 3: Profile traces for the multipliers of the orig:dest association

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 0.2243673 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, level = 0.95)

                                    2.5 %     97.5 %
Mult(Exp(.), orig:dest).educ1          NA         NA
Mult(Exp(.), orig:dest).educ2 -0.5978901  0.1022447
Mult(Exp(.), orig:dest).educ3 -1.4836854 -0.2362378
Mult(Exp(.), orig:dest).educ4 -2.5792398 -0.2953420
Mult(Exp(.), orig:dest).educ5       -Inf -0.7006889
```

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, which 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 parameter for one of the levels — say for the first level — is constrained to zero. The parameters to be estimated are then the differences between each other level and the first. 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 possibly scaled 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)
```

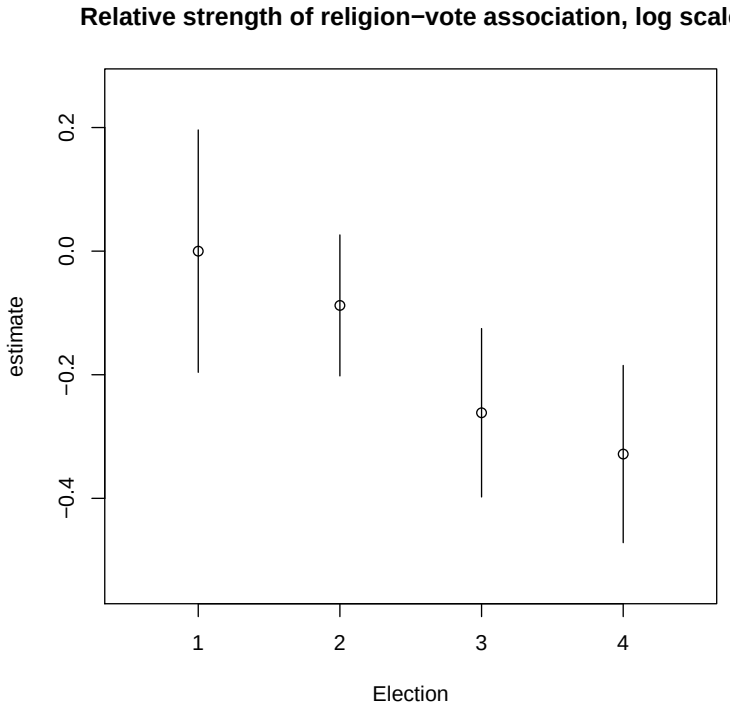

Figure 4: Relative strength of religion-vote association, log scale

By default, *getContrasts* uses the first parameter of the specified set as the reference level; alternatives may be set via the *ref* argument.

In the above example, the simple contrasts are estimable without scaling. In certain other applications, for example row-column association models (see Section 7.1), the contrasts are identified only after fixing their scale. A more general family of *scaled* contrasts for a set of parameters $\gamma_r, r = 1, \ldots, R$ is given by

$$\gamma_r^* = \frac{\gamma_r - \overline{\gamma}_w}{\sqrt{\sum_r v_r(\gamma_r - \overline{\gamma}_u)^2}}$$

where $\overline{\gamma}_w = \sum w_r \gamma_r$ is the reference level against which the contrasts are taken, $\overline{\gamma}_u = \sum u_r \gamma_r$ is a possibly different weighted mean of the parameters to be used as reference level for a set of "scaling contrasts", and v_r is a further set of weights. Thus, for example, the choice

$$w_r = \begin{cases} 1 & (r = 1) \\ 0 & (\text{otherwise}) \end{cases}, \qquad u_r = v_r = 1/R$$

specifies contrasts with the first level, with the coefficients scaled to have variance 1. This general type of scaling can be obtained by specifying the form of $\overline{\gamma}_u$ and v_r via the `scaleRef` and `scaleWeights` arguments of `getContrasts`.

As an example, consider the following model, described in Section 7.1:

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

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

The effects of the constituent multipliers of the multiplicative interaction are identified when both their scale and location are constrained. A simple way to achieve this is to set the first parameter to zero and the last parameter to one:

```
> RC1model2 <- gnm(count ~ SES + MHS + Mult(1, SES, MHS), constrain = "[.]SES[AF]",
+     constrainTo = c(0, 1), ofInterest = "[.]SES", family = poisson,
+     data = mentalHealth)

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

> summary(RC1model2)

Call:
gnm(formula = count ~ SES + MHS + Mult(1, SES, MHS), ofInterest = "[.]SES",
    constrain = "[.]SES[AF]", constrainTo = c(0, 1), family = poisson,
    data = mentalHealth)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-0.87231  -0.30983   0.01026   0.29898   0.87866

Coefficients of interest:
                     Estimate Std. Error z value Pr(>|z|)
Mult(1, ., MHS).SESA       NA         NA      NA       NA
Mult(1, ., MHS).SESB -0.003107   0.181567  -0.017    0.986
Mult(1, ., MHS).SESC  0.252939   0.158922   1.592    0.111
Mult(1, ., MHS).SESD  0.388785   0.144164   2.697    0.007
Mult(1, ., MHS).SESE  0.724329   0.172325   4.203 2.63e-05
Mult(1, ., MHS).SESF       NA         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: 3.5706 on 8 degrees of freedom
AIC: 179.74

Number of iterations: 10
```

Note that a constant multiplier must be incorporated into the interaction term, i.e., the multiplicative term *Mult(SES, MHS)* becomes *Mult(1, SES, MHS)*, in order to maintain equivalence with the original model specification. The constraints specified for *RC1model2* result in the estimation of scaled contrasts with level *A* of *SES*, in which the scaling fixes the magnitude of the contrast between level *F* and level *A* to be equal to 1. The equivalent use of *getContrasts*, together with the *unconstrained* fit (*RC1model*), in this case is as follows:

```
> getContrasts(RC1model, pickCoef(RC1model, "[.]SES"), ref = "first",
+     scaleRef = "first", scaleWeights = c(rep(0, 5), 1))

                      Estimate Std. Error
Mult(., MHS).SESA  0.000000000  0.0000000
Mult(., MHS).SESB  0.003107335  0.1815673
Mult(., MHS).SESC -0.252939313  0.1589219
Mult(., MHS).SESD -0.388785381  0.1441637
Mult(., MHS).SESE -0.724329033  0.1723248
Mult(., MHS).SESF -1.000000000  0.0000000
```

Quasi-variances and standard errors are not returned here as they can not (currently) be computed for scaled contrasts. When the scaling uses the same reference level as the contrasts, equal scale weights produce "spherical" contrasts, whilst unequal weights produce "elliptical" contrasts. Further examples are given in Sections 7.1 and 7.4.

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
```

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 *(g)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*. A *nls* model assumes that the responses are distributed either with constant variance or with fixed relative variances (specified via the *weights* argument). The *gnls* function in the nlme package extends

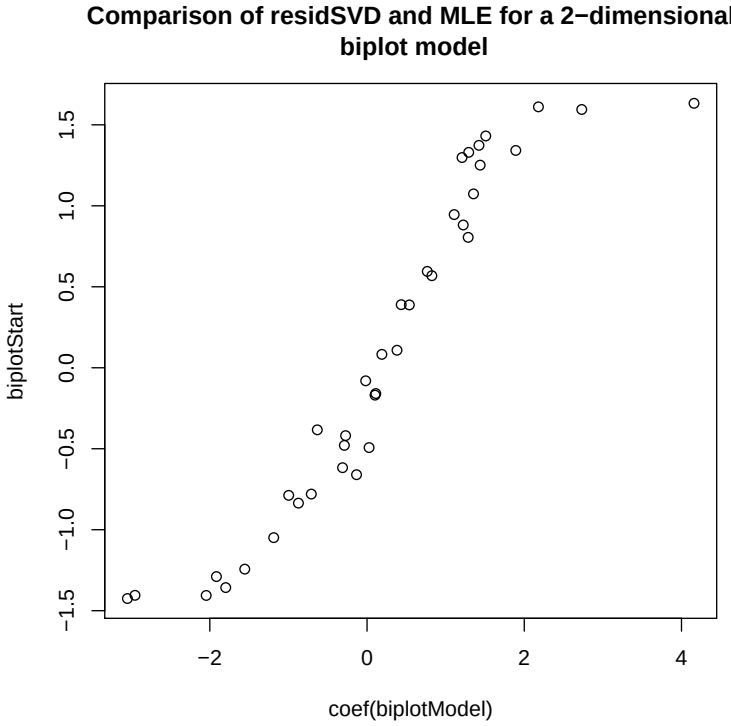

Figure 5: Comparison of residSVD and the MLE for a 2-dimensional biplot model

`nls` to allow correlated responses. On the other hand, `gnm` allows for responses distributed with variances that are a specified (via the `family` argument) function of the mean; as with `nls`, no correlation is allowed.

The `gnm` function also differs from `nls`/`gnls` in terms of the interface. Models are specified to `nls` and `gnls` in terms of a mathematical formula or a *selfStart* function based on such a 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.

7 Examples

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 μ_{rc} is given by

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

We shall fit this model to the *mentalHealth* data set 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
                 3.84143                   -0.06741                   0.10999
                    SESD                      SESE                      SESF
                 0.40502                    0.02535                  -0.20055
                 MHSmild               MHSmoderate               MHSimpaired
                 0.70380                    0.19416                   0.23331
       Mult(., MHS).SESA         Mult(., MHS).SESB         Mult(., MHS).SESC
                -0.41864                   -0.42216                  -0.13207
       Mult(., MHS).SESD         Mult(., MHS).SESE         Mult(., MHS).SESF
                 0.02183                    0.40198                   0.71429
    Mult(SES, .).MHSwell      Mult(SES, .).MHSmild  Mult(SES, .).MHSmoderate
                -0.73671                   -0.07475                   0.04471
Mult(SES, .).MHSimpaired
                 0.59453

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
      -1.11233093       -1.12143720       -0.37107614        0.02702955        1.01036159
Mult(., MHS).SESF
       1.81823273

$nu
    Mult(SES, .).MHSwell     Mult(SES, .).MHSmild Mult(SES, .).MHSmoderate
              -1.6775143               -0.1403989                0.1369924
Mult(SES, .).MHSimpaired
               1.4136910
```

Alternatively, the elliptical contrasts *mu* and *nu* can be obtained using *getContrasts*, with the advantage that the standard errors for the contrasts will also be computed:

```
> mu <- getContrasts(RC1model, pickCoef(RC1model, "[.]SES"), ref = rowProbs,
+     scaleWeights = rowProbs)
> nu <- getContrasts(RC1model, pickCoef(RC1model, "[.]MHS"), ref = colProbs,
+     scaleWeights = colProbs)
> mu

                     Estimate Std. Error
Mult(., MHS).SESA -1.11136061  0.2992108
Mult(., MHS).SESB -1.12045893  0.3142156
Mult(., MHS).SESC -0.37075244  0.3191514
Mult(., MHS).SESD  0.02700597  0.2732755
Mult(., MHS).SESE  1.00948022  0.3146991
Mult(., MHS).SESF  1.81664662  0.2809531

> nu

                         Estimate Std. Error
Mult(SES, .).MHSwell    -1.6737832  0.1904282
Mult(SES, .).MHSmild    -0.1400866  0.2001792
Mult(SES, .).MHSmoderate  0.1366877  0.2794787
Mult(SES, .).MHSimpaired  1.4105467  0.1741818
```

Since the value of *beta* is dependent upon the particular scaling used for the contrasts, it is typically not of interest to conduct inference on this parameter directly. The standard error for *beta* could be obtained, if desired, via the delta method.

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
                           3.81539                            -0.06452
                              SESC                                SESD
                           0.11327                             0.38762
                              SESE                                SESF
                           0.01619                            -0.17718
                           MHSmild                          MHSmoderate
                           0.72796                             0.22209
                       MHSimpaired         Mult(., MHS, inst = 1).SESA
                           0.27738                            -0.19609
      Mult(., MHS, inst = 1).SESB         Mult(., MHS, inst = 1).SESC
                          -0.23247                            -0.10207
      Mult(., MHS, inst = 1).SESD         Mult(., MHS, inst = 1).SESE
                           0.15618                             0.23954
      Mult(., MHS, inst = 1).SESF      Mult(SES, ., inst = 1).MHSwell
                           0.03515                            -1.00815
   Mult(SES, ., inst = 1).MHSmild  Mult(SES, ., inst = 1).MHSmoderate
                          -0.04298                            -0.21716
Mult(SES, ., inst = 1).MHSimpaired         Mult(., MHS, inst = 2).SESA
                           1.11729                             0.39218
      Mult(., MHS, inst = 2).SESB         Mult(., MHS, inst = 2).SESC
                           0.25985                             0.01665
      Mult(., MHS, inst = 2).SESD         Mult(., MHS, inst = 2).SESE
                           0.68097                             0.05502
      Mult(., MHS, inst = 2).SESF      Mult(SES, ., inst = 2).MHSwell
                          -1.75425                             0.32550
   Mult(SES, ., inst = 2).MHSmild  Mult(SES, ., inst = 2).MHSmoderate
                           0.05297                            -0.07626
Mult(SES, ., inst = 2).MHSimpaired
                          -0.17352

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
                        -1.55466                          0.62373
                         origin3                          origin4
                         2.01762                          2.61788
                         origin5                          origin6
                         1.40681                          3.71525
                         origin7                          origin8
                         2.58917                          2.44470
                    destination2                     destination3
                         1.04274                          2.36204
                    destination4                     destination5
                         2.90631                          2.30623
                    destination6                     destination7
                         4.01873                          3.34077
                    destination8       Diag(origin, destination)1
                         3.02008                          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                         -1.98495
MultHomog(origin, destination)2  MultHomog(origin, destination)3
                        -1.76665                         -1.16849
MultHomog(origin, destination)4  MultHomog(origin, destination)5
                        -0.58461                         -0.56744
MultHomog(origin, destination)6  MultHomog(origin, destination)7
                        -0.05568                          0.36046
MultHomog(origin, destination)8
                         0.60403

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

```
> DrefWeights(classMobility)

$origin
    weight         se
0.43724694 0.03996404

$destination
    weight         se
0.56275306 0.03996404
```

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

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

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)
                                                                        -1.3211
Dref(origin, destination, delta = ~ . + downward + upward).delta1(Intercept)
                                                                         0.2753
      Dref(origin, destination, delta = ~ 1 + . + upward).delta1downwardTRUE
                                                                         0.2122
      Dref(origin, destination, delta = ~ 1 + downward + .).delta1upwardTRUE
                                                                         0.1474
Dref(origin, destination, delta = ~ . + downward + upward).delta2(Intercept)
                                                                         0.6620
      Dref(origin, destination, delta = ~ 1 + . + upward).delta2downwardTRUE
                                                                        -0.5986
      Dref(origin, destination, delta = ~ 1 + downward + .).delta2upwardTRUE
                                                                         0.2076
             Dref(., ., delta = ~ 1 + downward + upward).origin|destination1
                                                                        -0.7365
```

```
            Dref(., ., delta = ~ 1 + downward + upward).origin|destination2
                                                                      0.2084
            Dref(., ., delta = ~ 1 + downward + upward).origin|destination3
                                                                     -0.6737
            Dref(., ., delta = ~ 1 + downward + upward).origin|destination4
                                                                      0.7519
            Dref(., ., delta = ~ 1 + downward + upward).origin|destination5
                                                                      1.3787

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

```
> DrefWeights(socialMobility)

$origin
  downward upward    weight         se
1    FALSE  FALSE 0.4044959 0.05918141
2     TRUE  FALSE 0.6044393 0.12371032
3    FALSE   TRUE 0.3900792 0.08134359

$destination
  downward upward    weight         se
1    FALSE  FALSE 0.5955041 0.05918141
2     TRUE  FALSE 0.3955607 0.12371032
3    FALSE   TRUE 0.6099208 0.08134359
```

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 group than for the other groups. The weights for the upwardly mobile group are very similar to the base level weights, so the model may be simplified by only fitting separate weights for the downwardly mobile group:

```
> 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)
                                                           -1.31336
Dref(origin, destination, delta = ~ . + downward).delta1(Intercept)
                                                           -0.04679
      Dref(origin, destination, delta = ~ 1 + .).delta1downwardTRUE
                                                            0.58421
Dref(origin, destination, delta = ~ . + downward).delta2(Intercept)
                                                            0.36199
      Dref(origin, destination, delta = ~ 1 + .).delta2downwardTRUE
                                                           -0.22653
            Dref(., ., delta = ~ 1 + downward).origin|destination1
                                                           -0.75650
            Dref(., ., delta = ~ 1 + downward).origin|destination2
```

```
                                                                0.20684
            Dref(., ., delta = ~ 1 + downward).origin|destination3
                                                               -0.67829
            Dref(., ., delta = ~ 1 + downward).origin|destination4
                                                                0.74029
            Dref(., ., delta = ~ 1 + downward).origin|destination5
                                                                1.36966

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

> DrefWeights(downwardMobility)

$origin
  downward    weight         se
1    FALSE 0.3992031 0.04750643
2     TRUE 0.5991569 0.11951340

$destination
  downward    weight         se
1    FALSE 0.6007969 0.04750643
2     TRUE 0.4008431 0.11951340
```

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*.

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

```
                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 weights have been constrained to sum to one as described in Section 3.3, so the weights themselves may be estimated as follows:

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

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

However, in order to estimate corresponding standard errors, the parameters of one of the weights must be constrained. If no such constraints were applied when the model was fitted, `DrefWeights` will refit the model constraining the parameters of the first weight to zero:

```
> DrefWeights(A)

Refitting with parameters of first Dref weight constrained to zero
$MOPLM
   weight        se
0.4225636 0.1439829

$FOPLF
   weight        se
0.5774364 0.1439829
```

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:

```
> DrefWeights(F)

Refitting with parameters of first Dref weight constrained to zero
$MOPLM
  MFCM     weight        se
1    1 0.02974675 0.2277711
2    0 0.74465224 0.2006916

$FOPLF
  MFCM    weight        se
1    1 0.9702532 0.2277711
2    0 0.2553478 0.2006916
```

giving the same weights as in Table 4 of **?**, though we obtain a lower standard error in the case where MFCM is equal to one.

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

This model can be applied to the *yaish* data set (**??**), which is a contingency table cross-classified by father's social class (*orig*), son's social class (*dest*) and son's education level (*educ*). In this case, we can consider the importance of the association between the social class of father and son across the education levels. 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
                   -0.2364828                    -0.4618546                    -0.9799063
Mult(Exp(.), orig:dest).educ4 Mult(Exp(.), orig:dest).educ5
                   -1.2754212                    -2.4859851
```

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 -0.7434235 0.2335083 0.21182122 0.04486823
Mult(Exp(.), orig:dest).educ4 -1.0389385 0.3434256 0.32609377 0.10633714
Mult(Exp(.), orig:dest).educ5 -2.2495024 0.9453762 0.93560622 0.87535900
```

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, ", class:vote")))

                                   estimate        SE    quasiSE    quasiVar
Mult(Exp(.), class:vote).election4 0.0000000 0.0000000 0.10934796 0.011956977
Mult(Exp(.), class:vote).election3 0.6817374 0.1446833 0.09475939 0.008979341
Mult(Exp(.), class:vote).election2 0.4493745 0.1320022 0.07395886 0.005469914
Mult(Exp(.), class:vote).election1 0.3618304 0.2534754 0.22854401 0.052232364

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

                                      estimate         SE    quasiSE    quasiVar
Mult(Exp(.), religion:vote).election4  0.0000000 0.00000000 0.07168290 0.005138439
Mult(Exp(.), religion:vote).election3 -0.0878181 0.09906916 0.06812239 0.004640660
Mult(Exp(.), religion:vote).election2 -0.2615200 0.09116479 0.05702819 0.003252214
Mult(Exp(.), religion:vote).election1 -0.3283459 0.12213023 0.09803075 0.009610029
```

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 μ_{rc} a GAMMI-K model has the form

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

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

The row-column association models discussed in Section 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, test = "F")

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      F    Pr(>F)
1       207     279515
2       176     128383  31   151133 6.6835 < 2.2e-16
```

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

Thus the significance of the multiplicative interaction can be tested without applying constraints to this term. If the multiplicative interaction is significant, we may wish to apply constraints to obtain estimates of the row and column scores. We illustrate this using the *barleyHeights* data, which records the average height for 15 genotypes of barley over 9 years.

For this small dataset the AMMI-1 model is easily estimated with the default settings:

```
> set.seed(1)
> data(barleyHeights)
> barleyModel <- gnm(height ~ year + genotype + Mult(year, genotype),
+     data = barleyHeights)

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

To obtain the parameterization of Equation 1 in which σ_k is the singular value for component k, the row and column scores must be constrained so that the scores sum to zero and the squared scores sum to one. These contrasts can be obtained using *getContrasts*:

```
> gamma <- getContrasts(barleyModel, pickCoef(barleyModel, "[.]y"), ref = "mean",
+     scaleWeights = "unit")
> delta <- getContrasts(barleyModel, pickCoef(barleyModel, "[.]g"), ref = "mean",
+     scaleWeights = "unit")
> gamma

                              Estimate Std. Error
Mult(., genotype).year1974 -0.22662083 0.05621029
Mult(., genotype).year1975 -0.51350519 0.04856519
Mult(., genotype).year1976 -0.43898738 0.05124785
Mult(., genotype).year1977  0.20046457 0.05658391
Mult(., genotype).year1978  0.53320299 0.04775769
Mult(., genotype).year1979 -0.14181352 0.05724921
Mult(., genotype).year1980  0.33249439 0.05418746
Mult(., genotype).year1981  0.17553769 0.05689550
Mult(., genotype).year1982  0.07922728 0.05770322

> delta

                             Estimate Std. Error
Mult(year, .).genotype1  -0.079143299 0.05913864
Mult(year, .).genotype10  0.335345230 0.05564859
Mult(year, .).genotype11  0.375909514 0.05466190
Mult(year, .).genotype12 -0.285239059 0.05669279
Mult(year, .).genotype13  0.168316670 0.05843057
Mult(year, .).genotype14 -0.293052674 0.05654225
Mult(year, .).genotype15 -0.460366628 0.05216774
Mult(year, .).genotype2  -0.007543292 0.05933628
Mult(year, .).genotype3  -0.169718828 0.05841527
```

```
Mult(year, .).genotype4   0.047168474 0.05926732
Mult(year, .).genotype5  -0.181497848 0.05828153
Mult(year, .).genotype6  -0.129417612 0.05880326
Mult(year, .).genotype7  -0.013581892 0.05933222
Mult(year, .).genotype8   0.422242964 0.05337051
Mult(year, .).genotype9   0.270578281 0.05696326
```

Confidence intervals based on the assumption of asymptotic normality can be computed as follows:

```
> gamma[[2]][, 1] + (gamma[[2]][, 2]) %o% c(-1.96, 1.96)

             [,1]        [,2]
 [1,] -0.33679300 -0.11644866
 [2,] -0.60869296 -0.41831741
 [3,] -0.53943316 -0.33854160
 [4,]  0.08956010  0.31136904
 [5,]  0.43959792  0.62680805
 [6,] -0.25402198 -0.02960507
 [7,]  0.22628697  0.43870182
 [8,]  0.06402252  0.28705286
 [9,] -0.03387103  0.19232559

> delta[[2]][, 1] + (delta[[2]][, 2]) %o% c(-1.96, 1.96)

             [,1]        [,2]
 [1,] -0.19505504  0.03676844
 [2,]  0.22627399  0.44441647
 [3,]  0.26877219  0.48304683
 [4,] -0.39635693 -0.17412119
 [5,]  0.05379275  0.28284059
 [6,] -0.40387548 -0.18222986
 [7,] -0.56261539 -0.35811786
 [8,] -0.12384240  0.10875582
 [9,] -0.28421275 -0.05522490
[10,] -0.06899548  0.16333242
[11,] -0.29572965 -0.06726605
[12,] -0.24467200 -0.01416322
[13,] -0.12987305  0.10270927
[14,]  0.31763676  0.52684916
[15,]  0.15893028  0.38222628
```

which broadly agree with Table 8 of Chadouef and Denis (1991), allowing for the change in sign.

On the basis of such confidence intervals we can investigate simplifications of the model such as combining levels of the factors or fitting an additive model to a subset of the data.

The singular value σ_k may be obtained as follows

```
> height <- matrix(scale(barleyHeights$height, scale = FALSE), 15, 9)
> R <- height - outer(rowMeans(height), colMeans(height), "+")
> svd(R)$d[1]

[1] 43.49599
```

This parameter is of little interest in itself, given that the significance of the term as a whole can be tested using ANOVA.

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,]  4.1948224 -0.39186730
 [2,]  2.7642412 -0.33951379
 [3,]  1.4250454 -0.04654265
 [4,]  1.8463067  0.33365988
 [5,]  1.2704088  0.15776724
 [6,]  1.1562913  0.40048201
 [7,]  1.0172048  0.72727987
 [8,]  0.6451366  1.46162701
 [9,] -0.1470898  2.13234201

> B

            [,1]        [,2]
 [1,] -2.0673648 -0.97420446
 [2,] -3.0599796 -0.50683007
 [3,] -2.9598030 -0.33190625
 [4,] -1.8086247 -0.49758478
 [5,] -1.5579477 -0.08444511
 [6,] -1.8939995  1.08460552
 [7,] -1.1790432  0.40687014
 [8,] -0.8490092  1.14671349
 [9,] -0.9704664  1.26558201
[10,] -0.6036789  1.39655882
```

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

```
> 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")
```

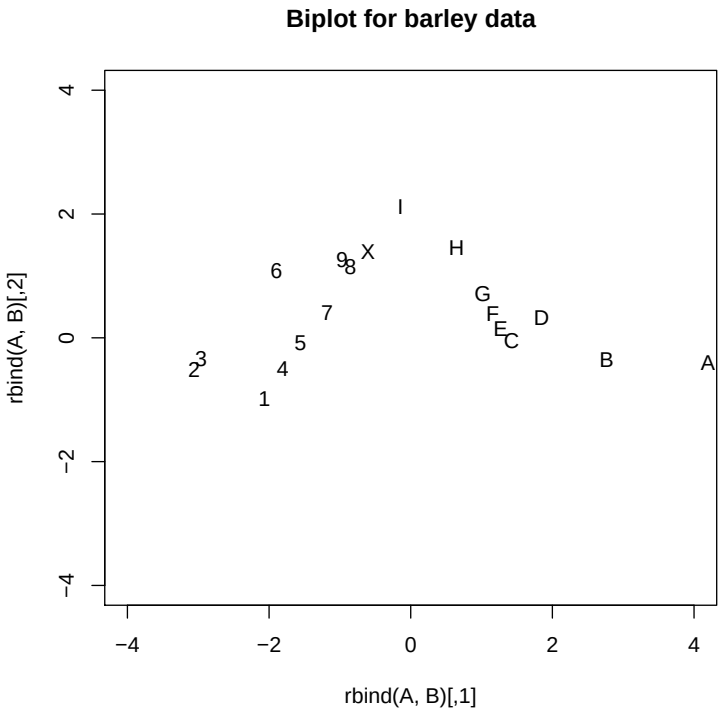

Figure 6: Biplot for barley data

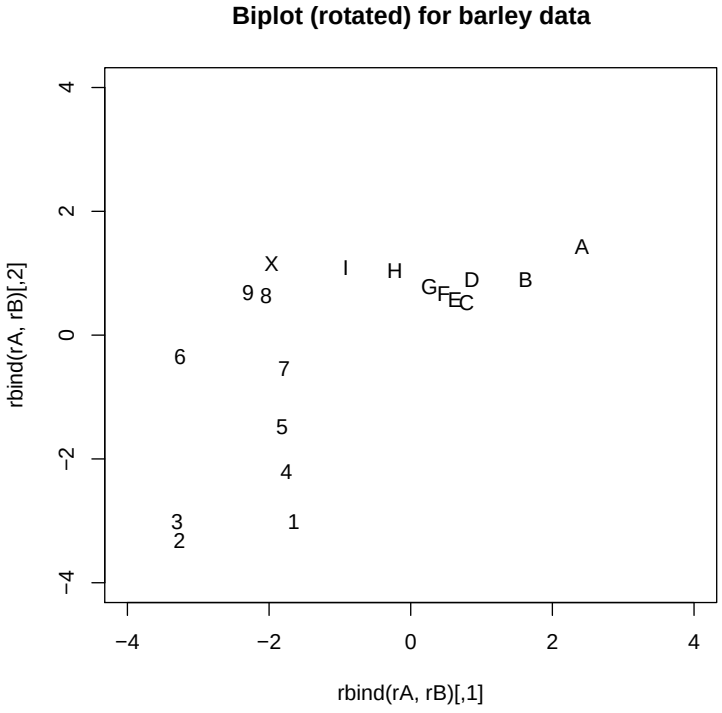

Figure 7: Rotated biplot for barley data

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

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

The rotated and scaled biplot suggests the simpler model

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

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

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

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

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

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
                                        0.3950
                  Mult(., x1 + x2 + x3).painsame
                                       -3.0297
  Mult(., x1 + x2 + x3).painslight.improvement
                                       -2.8450
Mult(., x1 + x2 + x3).painmoderate.improvement
                                       -2.0356
  Mult(., x1 + x2 + x3).painmarked.improvement
                                       -3.8622
     Mult(., x1 + x2 + x3).paincomplete.relief
                                       -4.5641
                     Mult(pain, . + x2 + x3).x1
                                        1.0832
                     Mult(pain, x1 + . + x3).x2
                                        0.6213
                     Mult(pain, x1 + x2 + .).x3
                                        0.5470

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     6.5742   2.458 0.013980
painslight.improvement                         15.6848     6.5274   2.403 0.016265
painmoderate.improvement                       12.4556     6.4312   1.937 0.052777
painmarked.improvement                         19.9140     6.4976   3.065 0.002178
paincomplete.relief                            21.6653     6.5571   3.304 0.000953
Mult(., x1 + x2 + x3).painworse                 0.3950         NA      NA       NA
Mult(., x1 + x2 + x3).painsame                 -3.0297         NA      NA       NA
Mult(., x1 + x2 + x3).painslight.improvement   -2.8450         NA      NA       NA
Mult(., x1 + x2 + x3).painmoderate.improvement -2.0356         NA      NA       NA
Mult(., x1 + x2 + x3).painmarked.improvement   -3.8622         NA      NA       NA
Mult(., x1 + x2 + x3).paincomplete.relief      -4.5641         NA      NA       NA
Mult(pain, . + x2 + x3).x1                      1.0832         NA      NA       NA
Mult(pain, x1 + . + x3).x2                      0.6213         NA      NA       NA
Mult(pain, x1 + x2 + .).x3                      0.5470         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: 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                           12.4556     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             1.3471         NA      NA
```

```
Mult(., x2 + x3 + offset(x1)).painsame                  -2.3626         NA      NA
Mult(., x2 + x3 + offset(x1)).painslight.improvement    -2.1626         NA      NA
Mult(., x2 + x3 + offset(x1)).painmoderate.improvement  -1.2858         NA      NA
Mult(., x2 + x3 + offset(x1)).painmarked.improvement    -3.2645         NA      NA
Mult(., x2 + x3 + offset(x1)).paincomplete.relief       -4.0247         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"))

Note: the following contrasts are unestimable:
[1] "Mult(pain, . + x3 + offset(x1)).x2" "Mult(pain, x2 + . + offset(x1)).x3"
                                                        estimate       SE   quasiSE
Mult(., x2 + x3 + offset(x1)).painworse                 0.000000 0.000000 1.7797297
Mult(., x2 + x3 + offset(x1)).painsame                 -3.709726 1.825562 0.4281332
Mult(., x2 + x3 + offset(x1)).painslight.improvement   -3.509687 1.791726 0.4024681
Mult(., x2 + x3 + offset(x1)).painmoderate.improvement -2.632933 1.669251 0.5518545
Mult(., x2 + x3 + offset(x1)).painmarked.improvement   -4.611586 1.895234 0.3133219
Mult(., x2 + x3 + offset(x1)).paincomplete.relief      -5.371844 1.999652 0.4919551
                                                         quasiVar
Mult(., x2 + x3 + offset(x1)).painworse                3.16743768
Mult(., x2 + x3 + offset(x1)).painsame                 0.18329802
Mult(., x2 + x3 + offset(x1)).painslight.improvement   0.16198057
Mult(., x2 + x3 + offset(x1)).painmoderate.improvement 0.30454334
Mult(., x2 + x3 + offset(x1)).painmarked.improvement   0.09817061
Mult(., x2 + x3 + offset(x1)).paincomplete.relief      0.24201985
```

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 μ_{ay} and variance $\phi\mu_{ay}$ (where ϕ 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 *0* 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 β_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:

```
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 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 β_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:

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

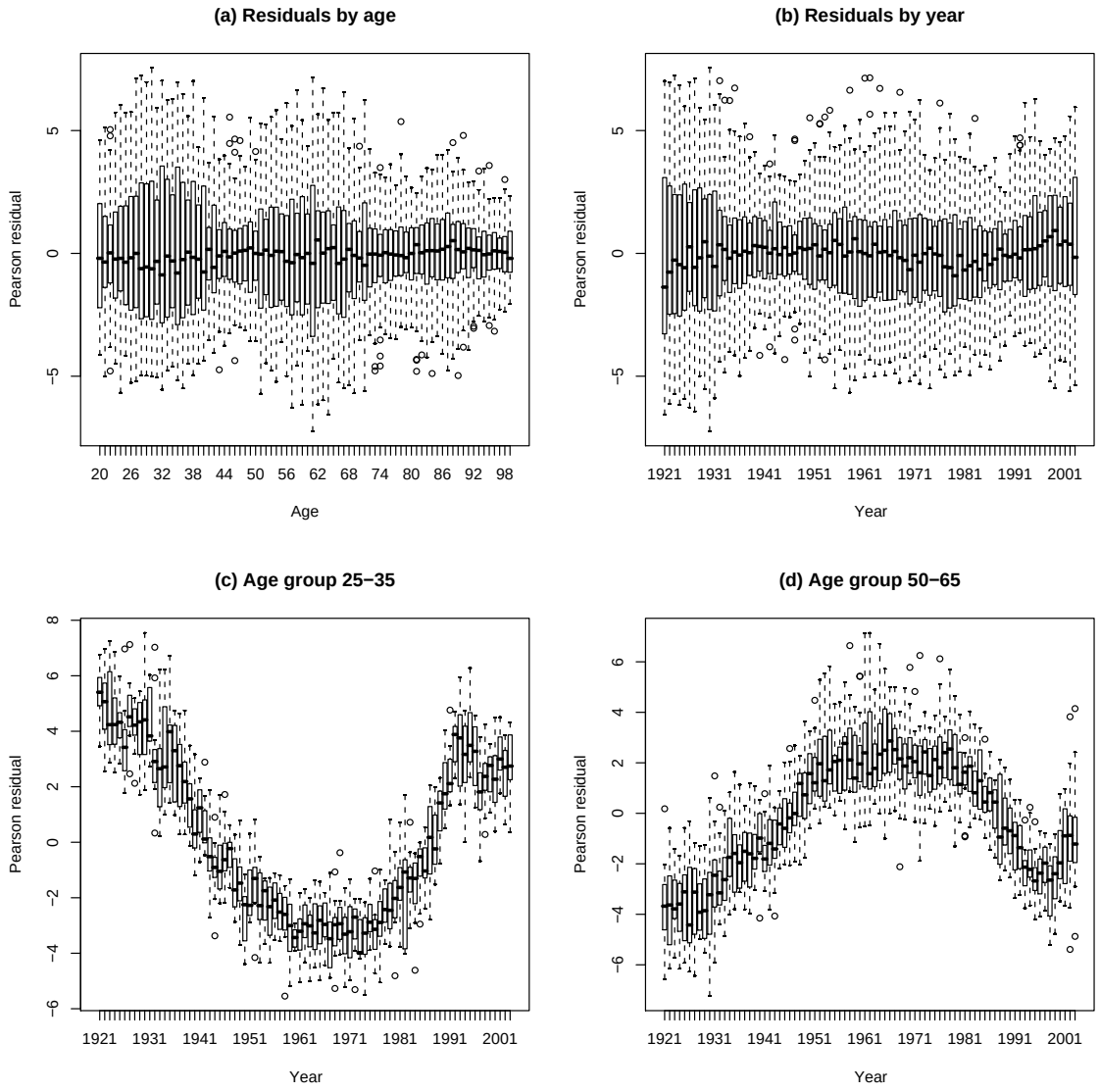

Figure 8: Canada, males: plots of residuals from the Lee-Carter model of mortality

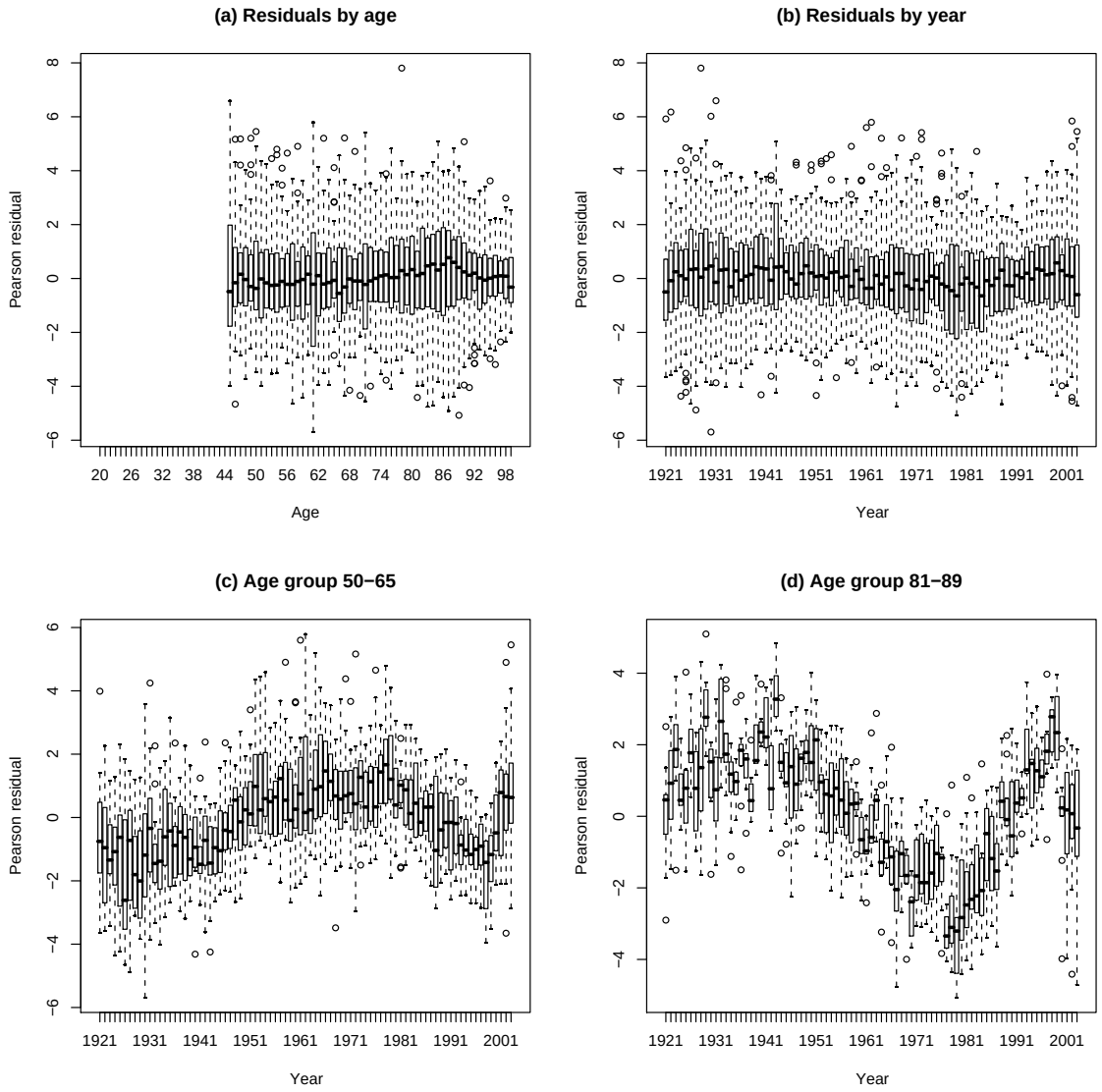

Figure 9: Canada, males over 45: plots of residuals from the Lee-Carter model of mortality

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

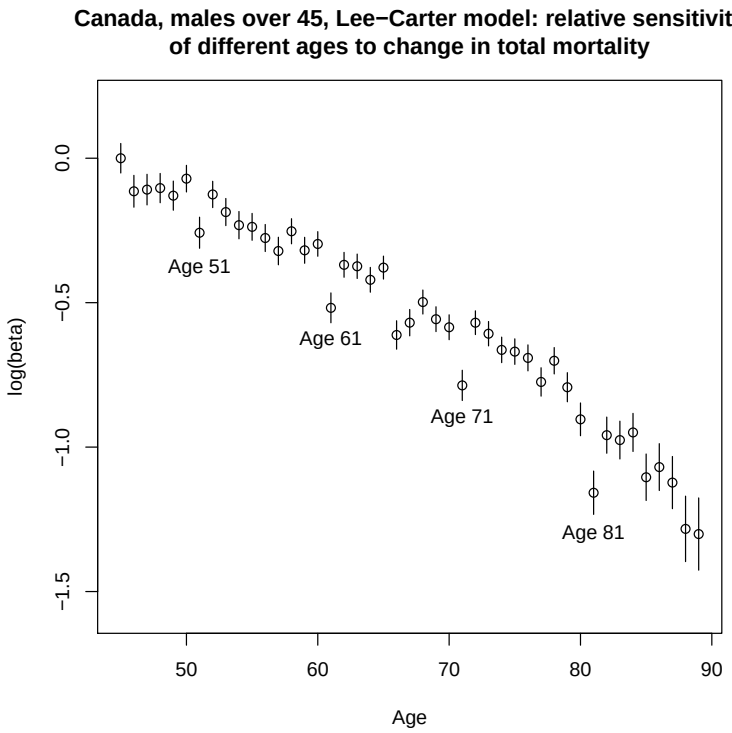

Figure 10: Canada, males over 45, Lee-Carter model: relative sensitivity of different ages to change in total mortality.

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, the estimated β_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 β_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 $a = 31$, $a = 41$,..., $a = 81$.

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

$$y = \alpha + \exp(\beta + \gamma x) + e \tag{2}$$

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

$$y = \alpha + \exp(\beta_1 + \gamma_1 x) + \exp(\beta_2 + \gamma_2 x) + e. \tag{3}$$

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.

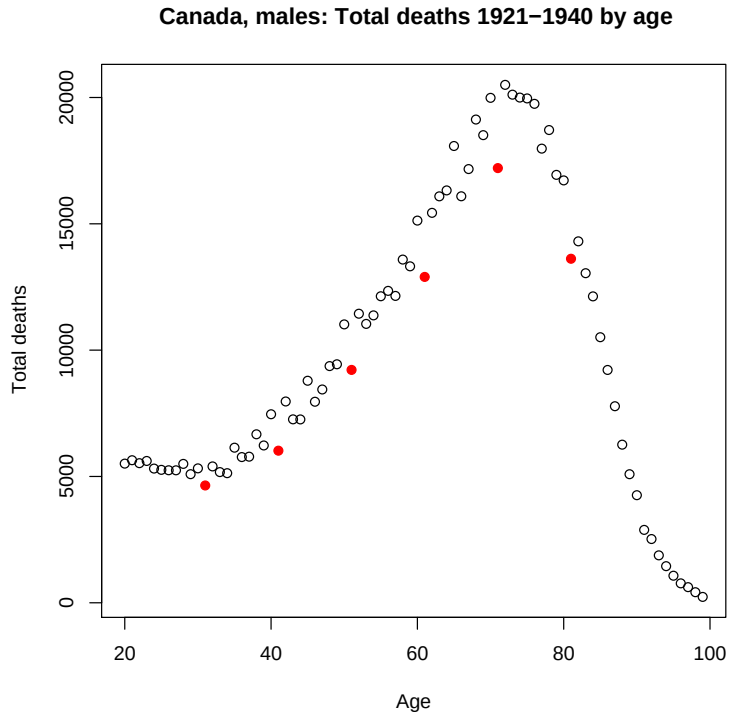

Figure 11: Canada, males: Deaths 1921 to 1940 by age

7.8.1 Example: single exponential decay term

Let us first construct some data from model (2). 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
      45       55
```

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 (α, β, γ). 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              -1.424e+03               1.377e+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 (3):

```
> 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))
> round(unlist(sapply(saved.fits, deviance)), 4)

 [1]  0.1589  0.1589  0.0000  0.0000  0.1589 41.6439  0.1589  0.0000 41.6439  0.0000
[11]  0.1589  0.1589  0.0000 41.6439  0.1589  0.1589 41.6439  0.1589  0.1589  0.1589
[21]  0.1589  0.0000  0.1589  0.1589  0.1589  0.1589  0.1589  0.0000  0.0000  0.0000
[31]  0.1589 41.6439  0.1589  0.0000  0.1589  0.1589  0.1589  0.1589  0.1589 41.6439
[41]  0.0000
```

In this instance, only 37 of the 100 calls to *gnm* have successfully located a local maximum of the likelihood: in the remaining 63 cases the starting values generated were such that numerical problems resulted, and the fitting algorithm was abandoned (giving a *NULL* result). Among the 37 'successful' fits, it is evident that there are three distinct solutions (with respective residual sums of squares equal to 0.1589, 41.64, and essentially zero — the last of these, the exact fit to the data, having been found 12 times out of the above 37). 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 41.64), 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

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

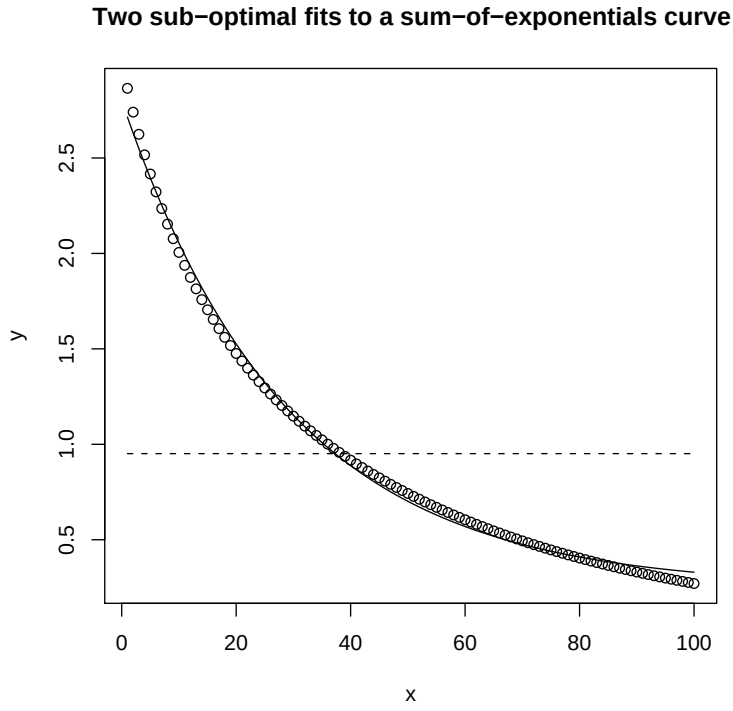

Figure 12: Two sub-optimal fits to a sum-of-exponentials curve

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 γ_1 and γ_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)
                       1.844e-12                         -3.640e-12
          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:            1.520979e-24
Pearson chi-squared: 1.520979e-24
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 *nonlin* function predictor
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
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
predict.gnm	predict from a *gnm* model
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
Dref	estimate weights and corresponding standard errors for a diagonal reference term in a *gnm* model
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