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

Heather Turner and David Firth
University of Warwick

2005-06-17

Contents

1 Introduction

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

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

We begin below with a preliminary note (Section 2) on 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: two new data-manipulation functions `Diag` and `Symm`, for setting up structured interactions between homologous factors; a new *family* function, `wedderburn`, for modelling a response variable in $[0, 1]$ with the variance function $V(\mu) = \mu^2(1-\mu)^2$ as in Wedderburn (1974); and a new generic function `termPredictors` which extracts the contribution of each term to the predictor from a fitted model object. These functions are briefly introduced here, before we move on to nonlinear models in Section 3.

2.2 `Diag` and `Symm`

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

As a simple example of their use, consider the log-linear models of *quasi-independence*, *quasi-symmetry* and *symmetry* for a square contingency table. Agresti (2002), page ??, 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 with more than two homologous factors.

2.3 The `wedderburn` family

In Wedderburn (1974) it was suggested to represent the mean of a continuous response variable in $[0, 1]$ using a quasi-likelihood model with logit link and the variance function $\mu^2(1-\mu)^2$. This is not one of the variance functions made available as standard in R's `quasi` family. The `wedderburn` family provides it. As an example, Wedderburn's analysis of data on leaf blotch on barley can be reproduced as follows:

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

[1] 71.17401
```

This agrees with the chi-squared value reported on page 331 of McCullagh and Nelder (1989), which differs slightly from Wedderburn's reported value.

2.4 `termPredictors`

What's a good example here? Just something simple to illustrate what it does.

I tried

```
> termPredictors(quasi.symm)
   (Intercept)       row      col Symm(row, col)
1   -0.2641848 0.0000000 0.000000             NA
2   -0.2641848 0.0000000 4.918310             NA
3   -0.2641848 0.0000000 1.539852             NA
4   -0.2641848 0.0000000 5.082641             NA
5   -0.2641848 4.8693457 0.000000             NA
6   -0.2641848 4.8693457 4.918310             NA
7   -0.2641848 4.8693457 1.539852             NA
8   -0.2641848 4.8693457 5.082641             NA
9   -0.2641848 0.7465235 0.000000             NA
10  -0.2641848 0.7465235 4.918310             NA
11  -0.2641848 0.7465235 1.539852             NA
12  -0.2641848 0.7465235 5.082641             NA
13  -0.2641848 4.4109017 0.000000             NA
14  -0.2641848 4.4109017 4.918310             NA
15  -0.2641848 4.4109017 1.539852             NA
16  -0.2641848 4.4109017 5.082641             NA
```

— is that a bug?

3 Nonlinear Terms

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

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

3.1 Multiplicative Interaction Terms using `Mult`

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

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

also known as the Goodman RC model (Goodman, 1979), the *formula* argument of `gnm` would be

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

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

`Mult` has one specified argument *multiplicity*, which is `1` by default. This argument determines the number of multiplicative components that are fitted. For example,

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

would give the RC(2) model (Goodman, 1979)

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

In some contexts, it may be desirable to constrain one or more of the multiplicative factors so that the factor is always nonnegative. This may be achieved by defining the factor as an exponential, as in the following 'uniform difference' model (Xie, 1992; Erikson and Goldthorpe, 1992)

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

Exponentiated factors can be specified in *gnm* models using the symbolic function `Exp`, for example the uniform difference model above would be specified by the formula

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

3.2 Other Nonlinear Terms using `Nonlin`

Nonlinear terms which can not be specified using `Mult` may be specified using `Nonlin`. This symbolic function indicates a term which requires a plug-in function to estimate the associated parameters. There are two arguments to `Nonlin`: a call to the relevant plug-in function and if necessary, a *data.frame* object containing any variables that are required by specified arguments of the plug-in function, which do not appear in any unspecified arguments of the plug-in function or elsewhere in the model formula.

For example, in the formula

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

the call to `Nonlin` is used to specify a term that requires the plug-in function `PlugInFunction`. As the factor C only appears in the specified arguments of the call to `PlugInFunction`, a data frame containing factor `C` has been passed as the *data* argument of `Nonlin`. Note that this would not be necessary if `C` could be found in an environment on the search path (given by `search()`).

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

3.2.1 `MultHomog`

The `MultHomog` function provides the tools required to fit multiplicative interaction terms in which the level effects are constrained to be equal across the factors. The arguments of RfunctionMultHomog are the factors in the interaction, which are assumed to be objects of class *factor*. Like a `Mult` term, the interaction can include any number of factors, but there is no multiplicity argument.

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

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

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

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

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

3.2.2 `Dref`

`Dref` is a plug-in function to fit diagonal reference terms involving two or more factors with a common set of levels. A diagonal reference term comprises an additive component for each factor. For a given data point, the component for the ith factor, say F, is

$$w_i \gamma_f$$

where w_i is the weight for factor i, γ_f is the "diagonal effect" for level f and f is the level of F for the given data point.

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

$$w_i = \frac{e^{\delta_i}}{\sum_r e^{\delta_r}}$$

and estimating the δ_i.

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

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

would be specified by the formula

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

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

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

specifies the following diagonal reference model

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

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

3.2.3 Custom Plug-in Functions

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

There are no constraints on the arguments that a plug-in function may have. However it should not be assumed that model variables exist in an environment on the search path, since `gnm` does not assume this. Rather, the function `getModelFrame` should be used to get the model frame used by `gnm`, which will have all the model variables and also attributes useful for `model.matrix` etc.

For example, the first few lines of the `MultHomog` function are

```
MultHomog <- function(...){
  labelList <- as.character((match.call(expand.dots = FALSE))[[2]])
  gnmData <- getModelFrame()
  designList <- lapply(gnmData[, labelList], class.ind)
...
```

The names of the factors in the interaction are assigned to `labelList`, and the model frame used by `gnm` is assigned to `gnmData`. The factors can then be accessed by name from `gnmData`, as in the call to `lapply`.

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

`start` (optional) either a vector of default starting values for the parameters or a function which takes the number of parameters and returns a vector of default starting values. See Section 4.2 for details of how these values will be used if provided and the generic default values that will be used otherwise.

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

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

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

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

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

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

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

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

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

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

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

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

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

4 Controlling the Fitting Procedure

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

4.1 Using *control* with `gnmControl`

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

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

is equivalent to

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

which is the same as

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

4.2 Using *start*

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

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

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

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

The starting procedure used by `gnm` is as follows

1. Generate starting values θ_i for all parameters $i = 1, \ldots, p$ from the Uniform(-0.1, 0.1) distribution. Shift these values away from zero as follows
$$\theta_i = \begin{cases} \theta_i - 0.1 & \text{if } \theta_i < 1 \\ \theta_i + 0.1 & \text{otherwise} \end{cases}$$
2. Replace generic starting values with any starting values specified by plug-in functions.
3. Replace default starting values with any starting values specified by the *start* argument of `gnm`.
4. Compute the `glm` estimate of any parameters in linear terms that were not specified by *start*, offsetting the contribution to the predictor of any parameters specified by *start* or a plug-in function.
5. Run starting iterations: update any parameters in nonlinear terms that were not specified by *start* or a plug-in function one at a time, updating *all* linear terms after each round of nonlinear updates.

Note that no starting iterations (step 5) will be run if all parameters are specified by the *start* argument of `gnm`.

4.3 Using *constrain*

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

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

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

Running the analysis again from a different seed

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

gives a different representation of the same model:

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

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

```
MultHomog(origin, destination).4 -0.14077778 -0.10930985
MultHomog(origin, destination).5 -0.12361117 -0.09214108
MultHomog(origin, destination).6  0.38814928  0.41961438
MultHomog(origin, destination).7  0.80429340  0.83575531
MultHomog(origin, destination).8  1.04785874  1.07933252
```

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

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

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

```
> multCoef <- coef(RChomog1)[grep("Mult", names(coef(RChomog1)))]
> set.seed(1)
> RChomogConstrained1 <- update(RChomog1, constrain = 31,
+     start = c(rep(NA, 23), multCoef - multCoef[8]))
> set.seed(2)
> RChomogConstrained2 <- eval(RChomogConstrained1$call)
> identical(coef(RChomogConstrained1), coef(RChomogConstrained2))

[1] TRUE
```

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

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

4.4 Using *eliminate*

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

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

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

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

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

or even

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

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

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

5 Methods and Accessor functions

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

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

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

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

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 `termPredictors` that has methods for objects of class *gnm*, *glm* and *lm*. This function returns the additive contribution of each term to the predictor. See Section 2 for an example of its use.

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

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

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

The effects of the first factor of the first multiplicative term are estimable when the estimate of one of these effects is constrained to zero, say the effect of the last level. The parameters to be estimated are then the differences between each effect and the effect of the last level. These differences can be represented by a contrast matrix as follows

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

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

and then their estimability can be checked using `checkEstimable`

```
> checkEstimable(doubleUnidiff, contrMatrix)

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

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

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

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

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

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

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

[[1]]
                           estimate         se
Mult1.Factor1.election1 0.32834637 0.12213023
Mult1.Factor1.election2 0.24052784 0.09116483
Mult1.Factor1.election3 0.06682575 0.09906919
Mult1.Factor1.election4 0.00000000 0.00000000
```

Attempting to obtain the equivalent contrasts for the second factor produces the following result

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

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

6 Examples

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

6.1 Row-column Association Models

There are several models that have been proposed for modelling the relationship between the cell means of a contingency table and the cross-classifying factors. The following examples consider the row-column association models proposed by Goodman (1979). The examples shown use data from two-way contingency tables, but the *gnm* package can also be used to fit the equivalent models for higher order tables.

6.1.1 RC(1) model

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

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

We shall fit this model to the `mentalHealth` data set taken from Agresti (2002) page ??, which is a two-way contingency table classified by the child's mental impairment (MHS) and the parents' socioeconomic status (SES). Although both of these factors are ordered, we do not wish to use polynomial contrasts in the model, so we begin by setting the contrasts attribute of these factors to "treatment"

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

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

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

Call:

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


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

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

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

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

$beta
[1] 0.1664870

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

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

6.1.2 RC(2) model

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

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

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

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

Call:

gnm(formula = count ~ SES + MHS + Mult(-1 + SES, -1 + MHS, multiplicity = 2),
```

```
    family = poisson, data = mentalHealth)


Coefficients:
              (Intercept)                      SESB
                  3.85511                  -0.06447
                     SESC                      SESD
                  0.11139                   0.38471
                     SESE                      SESF
                  0.01090                  -0.18477
                  MHSmild             MHSmoderate
                  0.69870                   0.17003
             MHSimpaired       Mult1.Factor1.SESA
                  0.22888                   0.94938
      Mult1.Factor1.SESB       Mult1.Factor1.SESC
                  0.99486                   0.33903
      Mult1.Factor1.SESD       Mult1.Factor1.SESE
                 -0.17301                  -0.91537
      Mult1.Factor1.SESF    Mult1.Factor2.MHSwell
                 -1.39141                   0.35835
   Mult1.Factor2.MHSmild Mult1.Factor2.MHSmoderate
                  0.03799                  -0.02140
Mult1.Factor2.MHSimpaired       Mult2.Factor1.SESA
                 -0.28068                  -0.17737
      Mult2.Factor1.SESB       Mult2.Factor1.SESC
                 -0.25127                  -0.16575
      Mult2.Factor1.SESD       Mult2.Factor1.SESE
                  0.29054                   0.22753
      Mult2.Factor1.SESF    Mult2.Factor2.MHSwell
                 -0.45487                   0.30770
   Mult2.Factor2.MHSmild Mult2.Factor2.MHSmoderate
                  0.09770                  -0.25568
Mult2.Factor2.MHSimpaired
                  0.06702

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

6.1.3 Homogeneous effects

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

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

For example, the `occupationalStatus` data set from Goodman (1979) is a contingency table classified by the occupational status of fathers (origin) and their sons (destination). Goodman (1979) fits a row-column association model with homogeneous effects to these data after deleting the cells on the main diagonal. Equivalently we can account for the diagonal effects by a separate `Diag` term:

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

```
Call:

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


Coefficients:
                     (Intercept)                          origin2
                        -0.11931                          0.53590
                         origin3                          origin4
                         1.68913                          2.05448
                         origin5                          origin6
                         0.83650                          2.93904
                         origin7                          origin8
                         1.64554                          1.40307
                    destination2                     destination3
                         0.95492                          2.03355
                    destination4                     destination5
                         2.34291                          1.73591
                    destination6                     destination7
                         3.24252                          2.39713
                    destination8       Diag(origin, destination)1
                         1.97844                          1.52667
      Diag(origin, destination)2       Diag(origin, destination)3
                         0.45601                         -0.01598
      Diag(origin, destination)4       Diag(origin, destination)5
                         0.38918                          0.73852
      Diag(origin, destination)6       Diag(origin, destination)7
                         0.13474                          0.45764
      Diag(origin, destination)8 MultHomog(origin, destination).1
                         0.38847                         -1.58261
MultHomog(origin, destination).2 MultHomog(origin, destination).3
                        -1.36432                         -0.76615
MultHomog(origin, destination).4 MultHomog(origin, destination).5
                        -0.18227                         -0.16511
MultHomog(origin, destination).6 MultHomog(origin, destination).7
                         0.34665                          0.76279
MultHomog(origin, destination).8
                         1.00637

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)
> RChomog$dev - RCheterog$dev
```

```
[1] 3.411823
```

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

```
[1] 6
```

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

6.2 Diagonal Reference Models

Diagonal reference models, proposed by (Sobel, 1981, 1985), are designed for contingency tables classified by factors with the same levels. The cell means are modelled as a function of the diagonal effects, i.e., the mean responses of the 'diagonal' cells in which the levels of the row and column factors are the same.

We shall use the `conformity` data set discussed in Van der Slik et al. (2002) to illustrate the use of diagonal reference models. These data relate to the value parents place on their children conforming to their rules. There are two response variables: the mother's conformity score and the father's conformity score. The data are cross-classified by two factors describing the education level of the mother and the father, and there are six further covariates.

In their baseline model for the mother's conformity score, Van der Slik et al. (2002) include five of the six covariates (leaving out the father's family conflict score) and a diagonal reference term with constant weights based on the two education factors. This model may be expressed as

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

See Section 3.2.2 for more detail on the choice of parameterisation.

The baseline model can be fitted as follows

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

```
Call:

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



Coefficients:
                    AGEM                     MRMM
                 0.06364                  -0.32425
```

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

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

Due to the constraints imposed on the weights in the diagonal reference term, the coefficients of model `A` are the unique solutions. Therefore these estimates should correspond to those reported in Table 4 of Van der Slik et al. (2002). The weights in the diagonal reference term can be evaluated as follows:

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

Dref(MOPLM, FOPLF).MOPLM Dref(MOPLM, FOPLF).FOPLF
               0.4225734                0.5774266
```

giving the values reported by Van der Slik et al. (2002). All the other coefficients of model A are the same as those reported by Van der Slik et al. (2002) except the coefficients of the mother's gender role (MRMM) and the father's gender role (FRMF). Van der Slik et al. (2002) reversed the signs of the coefficients of these factors since they were coded in the direction of liberal values, unlike the other covariates. However, simply reversing the signs of these coefficents does not give the same model, since the estimates of these coefficients are not independent of the estimates of the diagonal effects. For consistent interpretation of the covariate coefficients, it is better to recode the gender role factors as follows

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

Call:

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



Coefficients:
                    AGEM                     MRMM2
                 0.06364                  0.32425
                   FRMF2                     MWORK
```

```
                 0.25324                   -0.06430
                    MFCM  Dref(MOPLM, FOPLF).MOPLM
                -0.06043                    0.34389
Dref(MOPLM, FOPLF).FOPLF      Dref(MOPLM, FOPLF).1
                 0.65611                    4.37373
    Dref(MOPLM, FOPLF).2      Dref(MOPLM, FOPLF).3
                 4.28578                    4.28708
    Dref(MOPLM, FOPLF).4      Dref(MOPLM, FOPLF).5
                 4.14593                    3.85766
    Dref(MOPLM, FOPLF).6      Dref(MOPLM, FOPLF).7
                 3.61123                    3.85629

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

The coefficients of the covariates are now as reported by Van der Slik et al. (2002), but the diagonal effects have been adjusted appropriately.

Van der Slik et al. (2002) compare the baseline model for the mother's conformity score to several other models in which the weights in the diagonal reference term are dependent on one of the covariates. One particular model they consider incorporates an interaction of the weights with the mother's conflict score as follows

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

This model can be fitted as below, using the original coding for the gender role factors for ease of comparison to the results reported by Van der Slik et al. (2002),

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

Call:

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


Coefficients:
                                                    AGEM
                                                 0.05818
                                                    MRMM
                                                -0.32701
                                                    FRMF
                                                -0.25772
                                                   MWORK
                                                -0.07847
                                                    MFCM
                                                -0.01694
```

```
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).MOPLM.(Intercept)
                                                  1.03516
       Dref(MOPLM, FOPLF, formula = ~1 + MFCM).MOPLM.MFCM
                                                 -1.77703
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.(Intercept)
                                                 -0.03516
       Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.MFCM
                                                  2.77703
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).1
                                                  4.82477
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).2
                                                  4.88066
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).3
                                                  4.83969
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).4
                                                  4.74849
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).5
                                                  4.42019
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).6
                                                  4.17956
                Dref(MOPLM, FOPLF, formula = ~1 + MFCM).7
                                                  4.40819

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

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

```
> prop.table(exp(coef(F))[c(6, 8)])

Dref(MOPLM, FOPLF, formula = ~1 + MFCM).MOPLM.(Intercept)
                                                0.7446585
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.(Intercept)
                                                0.2553415

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

Dref(MOPLM, FOPLF, formula = ~1 + MFCM).MOPLM.MFCM
                                          0.02977851
Dref(MOPLM, FOPLF, formula = ~1 + MFCM).FOPLF.MFCM
                                          0.97022149
```

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

6.3 Uniform Difference (UNIDIFF) Models

Uniform difference models (Xie, 1992; Erikson and Goldthorpe, 1992) use a simplified three-way interaction to provide an interpretable model of contingency tables classified by three or more variables. For example, the uniform difference model for a three-way contingency table, also known as the UNIDIFF model, is given by

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

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

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

```
> set.seed(1)
> data(yaish)
> unidiff <- gnm(Freq ~ educ:orig + educ:dest + Mult(Exp(-1 +
+     educ), orig:dest), family = poisson, data = yaish)
> coefs.of.interest <- grep("Mult1.Factor1", names(coef(unidiff)))
> coef(unidiff)[coefs.of.interest]

Mult1.Factor1.educ1 Mult1.Factor1.educ2 Mult1.Factor1.educ3
         1.08253469          0.86578329          0.35129963
Mult1.Factor1.educ4 Mult1.Factor1.educ5
         0.05241102         -1.15532711
```

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

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

[[1]]
                     estimate        se
Mult1.Factor1.educ1 2.237862 0.9411152
Mult1.Factor1.educ2 2.021110 0.9435045
Mult1.Factor1.educ3 1.506627 0.9535672
Mult1.Factor1.educ4 1.207738 0.9780882
Mult1.Factor1.educ5 0.000000 0.0000000
```

Four-way contingency tables may be described by the "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 can be used to illustrate the application of the double UNIDIFF model. This data set is classified by the variables vote, class, religion and election. Using a double UNIDIFF model, we can see how the association between class and vote, and the association between religion and vote, differ between the most recent election and the other elections:

```
> set.seed(1)
> data(cautres)
> doubleUnidiff <- gnm(Freq ~ election:vote + election:class:religion +
+     Mult(Exp(-1 + election), religion:vote) + Mult(Exp(-1 +
+     election), class:vote), family = poisson, data = cautres)
> getContrasts(doubleUnidiff, grep("Mult1.Factor1", names(coef(doubleUnidiff))))

[[1]]
                          estimate         se
Mult1.Factor1.election1 0.32834578 0.12213024
```

```
Mult1.Factor1.election2 0.24052783 0.09116478
Mult1.Factor1.election3 0.06682591 0.09906915
Mult1.Factor1.election4 0.00000000 0.00000000
```

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

```
[[1]]
                            estimate        se
Mult2.Factor1.election1 -0.36182804 0.2534753
Mult2.Factor1.election2  0.31990886 0.1320022
Mult2.Factor1.election3  0.08754582 0.1446833
Mult2.Factor1.election4  0.00000000 0.0000000
```

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

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

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

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

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

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

We shall use the `wheat` data set taken from Vargas et al. (2001), which gives wheat yields measured over ten years. First we shall scale these yields and create a new treatment factor, so that we can reproduce the analysis of Vargas et al. (2001)

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

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

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

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

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

```
Analysis of Deviance Table

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

giving the same results as in Table 1 of Vargas et al. (2001) (up to error caused by rounding).

6.5 Biplot Models

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

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

To illustrate the use of biplot models, we shall use the `barley` data set which describes the incidence of leaf blotch over ten varieties of barley grown at nine sites (Wedderburn, 1974; Gabriel, 1998). The biplot model is fitted as follows

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

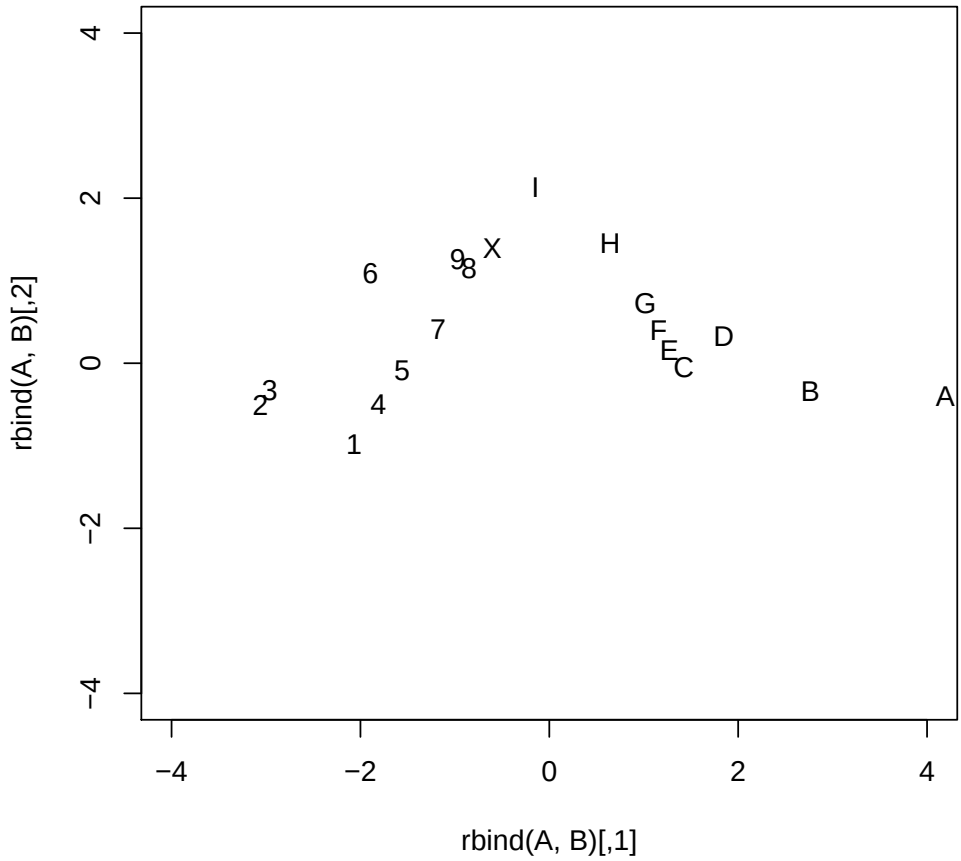
using the `wedderburn` family function introduced in Section 2. Matrices of the row and column scores for the first two singular vectors can then be obtained as below

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

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

> B

            [,1]       [,2]
 [1,] -2.0675116 -0.9742098
```

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

These matrices are essentially the same as in Gabriel (1998). From these the biplot can be produced:

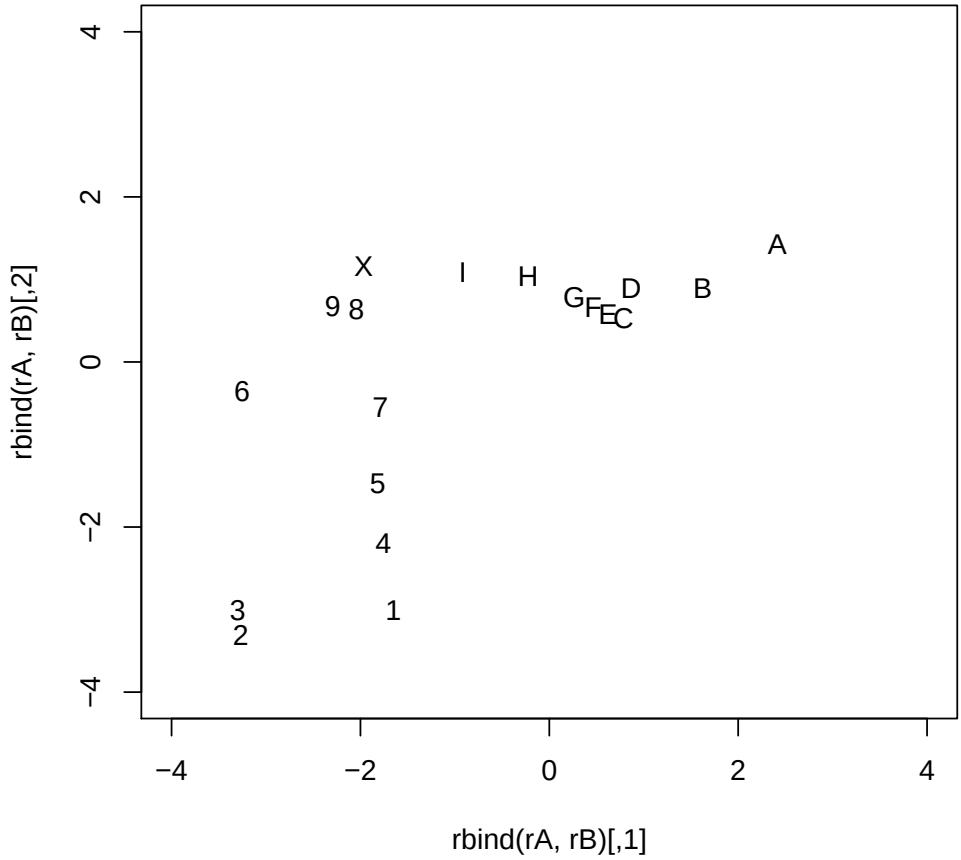
```
> plot(rbind(A, B), pch = c(levels(barley$site), levels(barley$variety)),
+     xlim = c(-4, 4), ylim = c(-4, 4))
```



for sites $A \ldots I$ and varieties $1 \ldots 9, X$. 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 parellel 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))
```



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

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

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

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

Gabriel (1998) describes this as a double additive model, which we can fit as follows

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

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

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

```
[1]  9.515599 15.000000
```

6.6 Stereotype Model

The stereotype model was proposed by Anderson (1984) for ordered categorical data. It is a linear logistic model, in which there is assumed to be a common relationship between the response and the covariates in the model, but the scale of this association varies between categories and there is an additional category main effect or category-specific intercept:

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

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

For example, the `backPain` data set from Anderson (1984) describes the progress of patients with back pain. The data set consists of an ordered factor quantifying the progress of each patient, and three prognostic variables. These data can be re-expressed as follows

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

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

```
> library(nnet)
> .incidence <- class.ind(backPain$pain)
> .counts <- as.vector(t(.incidence))
> .rowID <- factor(t(row(.incidence)))
```

```
> backPain <- backPain[.rowID, ]
> backPain$pain <- C(factor(rep(levels(backPain$pain),
+     nrow(.incidence)), levels = levels(backPain$pain),
+     ordered = TRUE), treatment)
> cbind(.rowID[1:12], .counts[1:12], backPain[1:12, 4:1])
```

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

We can now fit the stereotype model to these data

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

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


Coefficients:
                              painsame
                               16.1574
                painslight.improvement
                               15.6844
             painmoderate.improvement
                               12.4558
                painmarked.improvement
                               19.9140
                   paincomplete.relief
                               21.6652
              Mult1.Factor1.painworse
                               -4.0617
               Mult1.Factor1.painsame
                                0.3283
Mult1.Factor1.painslight.improvement
                                0.0916
```

```
Mult1.Factor1.painmoderate.improvement
                              -0.9458
  Mult1.Factor1.painmarked.improvement
                               1.3958
     Mult1.Factor1.paincomplete.relief
                               2.2954
                     Mult1.Factor2.x1
                              -0.8450
                     Mult1.Factor2.x2
                              -0.4847
                     Mult1.Factor2.x3
                              -0.4267

Deviance:            303.1003
Pearson chi-squared: 433.3728
Residual df:         493
```

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

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

Call:

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


Coefficients:
                   painsame       painslight.improvement
                    39.3495                      38.9688
   painmoderate.improvement       painmarked.improvement
                    35.8513                      43.0519
        paincomplete.relief                  painworse:x1
                    45.4999                      16.9234
                painsame:x1    painslight.improvement:x1
                     1.7421                       2.0717
painmoderate.improvement:x1    painmarked.improvement:x1
                     2.3351                       0.5119
     paincomplete.relief:x1                  painworse:x2
                     0.0000                       3.2750
                painsame:x2    painslight.improvement:x2
                     0.6009                       0.7236
painmoderate.improvement:x2    painmarked.improvement:x2
                     1.6029                       0.4311
     paincomplete.relief:x2                  painworse:x3
```

```
                    0.0000                         2.9407
               painsame:x3    painslight.improvement:x3
                    1.7852                         1.6486
painmoderate.improvement:x3    painmarked.improvement:x3
                    2.1944                         1.2491
     paincomplete.relief:x3
                    0.0000

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

this model has the maximum dimensionality of three (as determined by the number of covariates). To obtain the log-likelihoods as reported in Anderson (1984) we need to adjust for the extra parameters introduced to formulate the models as poisson models. We write a simple function to do this and compare the log-likelihoods of the one dimensional model and the three dimensional model:

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

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

which show that the `oneDimensional` model is adequate.

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.

References

A Agresti. *Categorical Data Analysis*. New York: Wiley, 2nd edition, 2002.

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

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

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

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

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

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

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

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

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

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

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

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