Writing VGAM Family Functions

T. W. Yee

November 16, 2006

Beta Version 0.6-5

Department of Statistics,
University of Auckland,
New Zealand
yee@stat.auckland.ac.nz
http://www.stat.auckland.ac.nz/∼yee

Contents

[Important note: This document and code is not yet finished, but should be completed one day . . .]

1 Introduction

This document describes some details about how you can write a VGAM family function to fit a certain distribution or model. Readers are assumed to be familiar with Chambers and Hastie (1993). VGAM has a flexible design that makes it easy for users to solve for the maximum likelihood estimates of their own problems. At its simplest, one merely creates a basic VGAM *family function* that has the appropriate derivatives and weights etc.

An outline of this document is as follows. The basics of family functions and their creation are given in the next section. Section 3 provides details on more advanced features. Section 4 gives several progressively more complicated examples of family functions.

For those wanting to solve a simple problem, reading the next section might suffice. For VGAM family functions that allow options such as input parameters the techniques exemplified in Section 4 will be necessary.

2 Basics

The essential requirements to create a VGAM family function is a model that can be estimated by iteratively reweighted least-squares (IRLS). This usually means a model whose log-likelihood $\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} w_i \, \ell_i(\boldsymbol{\theta})$, score vectors $w_i \, \dfrac{\partial \ell_i}{\partial \boldsymbol{\theta}}$, and Hessians $w_i \, \dfrac{\partial^2 \ell_i}{\partial \boldsymbol{\theta} \, \partial \boldsymbol{\theta}^T}$ (or their expected values) can be written explicitly. Here, $\boldsymbol{\theta}$ is the vector of parameters to be estimated, and the w_i are prior weights which are inputted into `vglm()`, `vgam()` etc. using the `weights` argument.

An optional requirement is some measure of the fit, e.g., a deviance function $D(\boldsymbol{\theta}) = 2\{\ell_{\max} - \ell(\boldsymbol{\theta})\}$. Sometimes an expression for $\ell_{\max}$ cannot be found so that maximizing a log-likelihood $\ell(\boldsymbol{\theta})$ is the next best choice. The user may define his own criterion—see elsewhere for details. If there is no objective function to be minimized or maximized then iterations will continue until the changes in the regression coefficients are sufficiently small.

2.1 Writing a VGAM Family Function

The following is a simple family function for the exponential distribution $f(y;\lambda) = \lambda\, e^{-\lambda y}$, where $y > 0$ and $\lambda > 0$ is the rate parameter.

```
> print(simple.exponential)

function ()
{
    new("vglmff", blurb = c("Simple Exponential distribution\n",
        "Link:    log(rate)\n"), deviance = function(mu, y, w,
        residuals = FALSE, eta, extra = NULL) {
        devy = -log(y) - 1
        devmu = -log(mu) - y/mu
        devi = 2 * (devy - devmu)
        if (residuals)
            sign(y - mu) * sqrt(abs(devi) * w)
        else sum(w * devi)
    }, initialize = expression({
        predictors.names = "log(rate)"
```

Table 1: *Slots of a typical* VGAM *family object and their purpose.*

Slot	Type	Purpose
`@blurb`	character string	Descriptive.
`@link`	`function(mu, extra=NULL)`	Returns $n \times M$ `eta` $(\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_n)^T$. Sometimes does not exist.
`@inverse`	`function(eta, extra=NULL)`	Returns n-row matrix `mu` with rows $\boldsymbol{\mu}_i^T$.
`@derivative`	`expression`	Returns $n \times M$ matrix with rows $w_i\, \partial \ell_i / \partial \boldsymbol{\eta}$.
`@initialize`	`expression`	Computes initial `mustart` and/or `etastart`; preprocesses `y` if necessary.
`@deviance`	`function(mu, y, w, resid= FALSE, eta, extra=NULL)`	Returns the deviance, or if `resid=TRUE` then a $n \times M$ matrix of deviance residuals (`NULL` if they do not exist).
`@loglikelihood`	`function(mu, y, w, resid= FALSE, eta, extra=NULL)`	Returns the log-likelihood.
`@weight`	`expression`	Computes `wz` ($= \mathbf{W}_i$; see Section 2.7). This is run directly after `@deriv`.
`@first`	`expression`	An optional expression executed at beginning of iterations.
`@last`	`expression`	An optional expression executed at end of iterations.

```
        mustart = y + (y == 0)/8
    }), inverse = function(eta, extra = NULL) exp(-eta), link = function(mu,
        extra = NULL) -log(mu), vfamily = "simple.exponential",
        deriv = expression({
            rate = 1/mu
            dl.drate = mu - y
            drate.deta = dtheta.deta(rate, "loge")
            w * dl.drate * drate.deta
        }), weight = expression({
            ed2l.drate2 = -1/rate^2
            wz = -w * drate.deta^2 * ed2l.drate2
            wz
        }))
}
<environment: namespace:VGAM>
```

What is the statistical derivation behind this function? Recall that

$$\mu \;=\; \frac{1}{\lambda} \quad \text{and} \quad \mathrm{Var}(Y) \;=\; \frac{1}{\lambda^2}\,.$$

Because $\lambda > 0$ it is natural for a log link on the parameter to be taken. Thus $\log\,\lambda = \eta$, which equals $-\log\,\mu$. Now $w_i\,\ell_i(\lambda_i; y_i) \;=\; w_i\,\log\,f(y_i;\lambda_i) = w_i\,(\log\,\lambda_i - \lambda_i\,y_i)$ so that

$$w_i\,\frac{\partial\,\ell_i}{\partial\,\lambda_i} \;=\; w_i\,\left(\frac{1}{\lambda_i} - y_i\right) \;=\; w_i\,(\mu_i - y_i) \quad \text{and} \quad w_i\,\frac{\partial^2\,\ell_i}{\partial\,\lambda_i^2} \;=\; -\,w_i\,\frac{1}{\lambda_i^2} \;=\; w_i\,E\left(\frac{\partial^2\,\ell_i}{\partial\,\lambda_i^2}\right).$$

These relationships coupled with (omitting subscripts i)

$$\begin{aligned}\frac{\partial\ell}{\partial\eta} &= \frac{\partial\ell}{\partial\lambda}\,\frac{\partial\lambda}{\partial\eta},\\ E\left(\frac{\partial^2\ell}{\partial\eta^2}\right) &= \left(\frac{\partial\lambda}{\partial\eta}\right)^2\,\frac{\partial^2\ell}{\partial\lambda^2}\end{aligned}$$

explain `@deriv` and `@weight` respectively (the expected Hessian was used rather than the observed Hessian. That is, Fisher scoring is used rather than Newton-Raphson). The MLE is given by setting $\partial\ell_i/\partial\lambda_i = 0$ giving $\hat{\lambda}_i = y_i^{-1}$ so that $\ell_{i,\max} = -\log\,y_i - 1$.

2.2 Further notes

Here are some further notes about VGAM family functions.

1. Family functions in VGAM have the form

   ```
   my.family.function <- function(<argument list>)
   {
       new("vglmff", ...)
   }
   ```

   where the slots of the `"vglmff"` object are described in Table 1.

2. Table 2 lists the variables used by `vglm()` and `vgam()`. Some of these need to be assigned values; others can be modified with care. The variables are particularly important in `@initialize`, `@deriv` and `@weight`.

3. `wz` and the output of `@deriv` have been multiplied by `w` because

$$\ell \;=\; \sum_{i=1}^{n} w_i \, \ell_i \; .$$

4. `@link` is optional since for some models it is not possible to compute $\boldsymbol{\eta}_i$ given $\boldsymbol{\mu}_i$. Indeed, for some models, $\boldsymbol{\mu}_i$ does not even exist!

5. `care.exp()` is sometimes preferable to `exp()` as the former handles over- and under-flow.

6. Variable names that should not be used include `R`, `iter`, `effects`, ...

7. Some VGAM family functions cannot be used by `vgam()` with `s()`. In particular, nonlinear regression using Gauss-Newton has the design matrix changing from iteration to iteration, therefore smoothing is not possible.

2.3 Link Functions

The following formulae are useful in order to program family functions.

$$\frac{\partial \ell}{\partial \eta_j} \;=\; \frac{\partial \ell}{\partial \theta_j} \frac{\partial \theta_j}{\partial \eta_j}, \tag{1}$$

$$\frac{\partial^2 \ell}{\partial \eta_j^2} \;=\; \frac{\partial \ell}{\partial \theta_j} \frac{\partial^2 \theta_j}{\partial \eta_j^2} + \left(\frac{\partial \theta_j}{\partial \eta_j}\right)^2 \frac{\partial^2 \ell}{\partial \theta_j^2}, \tag{2}$$

$$\frac{\partial^2 \ell}{\partial \eta_j \, \partial \eta_k} \;=\; \left\{ \frac{\partial^2 \ell}{\partial \theta_j \, \partial \theta_k} - \frac{\partial \ell}{\partial \theta_k} \frac{\partial \theta_k}{\partial \eta_k} \frac{\partial^2 \eta_k}{\partial \theta_j \, \partial \theta_k} \right\} \frac{\partial \theta_j}{\partial \eta_j} \frac{\partial \theta_k}{\partial \eta_k}, \quad j \neq k, \tag{3}$$

Table 2: *Variables in* `vglm.fit()` *and* `vgam.fit()` *and their properties.*

Variable	Purpose
`M`	M, the number of linear/additive predictors.
`constraints`	A list containing named constraint matrices.
`eta`	$n \times M$ matrix of linear predictors. May be a vector if $M = 1$.
`fit`	the VGAM object returned.
`predictors.names`	To be assigned a vector of M names for the linear predictors.
`mu`	n row matrix of fitted values.
`minimize.criterion`	Logical variable—is the convergence criterion to be minimized?
`n`	Number of observations in the data set; is n.
`extra`	List containing useful data or intermediate computations.
`etastart`	To be assigned a matrix of linear/additive predictors in `@initialize`.
`w`	Prior weights; the `weights` argument; By default it is unweighted or `rep(1,n)`.
`wz`	Matrix-band representation of $\mathbf{W}_i$; see Section 2.7 for details.
`x`	LM model matrix, is $n \times p$.
`y`	Response. May be a vector or a matrix.

$$
\begin{aligned}
\frac{\partial y}{\partial x} &= \left(\frac{\partial x}{\partial y}\right)^{-1}, \\
\frac{\partial^2 y}{\partial x^2} &= -\left(\frac{\partial y}{\partial x}\right)^3 \frac{\partial^2 x}{\partial y^2}, \\
E\left(\frac{\partial \ell}{\partial \boldsymbol{\theta}}\right) &= \mathbf{0}.
\end{aligned} \tag{4}
$$

The significance of (4) is that certain terms in (2) and (3) vanish if one wishes to use the expected information matrix (EIM) rather than the observed information matrix (OIM; i.e., Fisher scoring versus Newton-Raphson). We use EIM and OIM to denotes these quantities. The variable `wz` computed in `@weight` consists of

$$
(\mathbf{W}_i)_{jk} = -w_i \frac{\partial^2 \ell_i}{\partial \eta_j \, \partial \eta_k} \quad \text{or} \quad -w_i \, E\left(\frac{\partial^2 \ell_i}{\partial \eta_j \, \partial \eta_k}\right)
$$

whereas `@deriv` returns

$$
(\boldsymbol{d}_i)_j = w_i \frac{\partial \ell_i}{\partial \eta_j}.
$$

That is,

$$
\mathbf{W}_i = -w_i \frac{\partial^2 \ell_i}{\partial \boldsymbol{\eta} \, \partial \boldsymbol{\eta}^T} \quad \text{or} \quad -w_i \, E\left(\frac{\partial^2 \ell_i}{\partial \boldsymbol{\eta} \, \partial \boldsymbol{\eta}^T}\right)
$$

and

$$
\boldsymbol{d}_i = w_i \frac{\partial \ell_i}{\partial \boldsymbol{\eta}}.
$$

The $\mathbf{W}_i$ are referred to as the *working weight matrices*. In general, Fisher scoring is preferred over Newton-Raphson where possible. This is mainly because the $\mathbf{W}_i$ are usually positive-definite over a larger parameter space.

For most models (3) simplifies to

$$
\frac{\partial^2 \ell}{\partial \eta_j \, \partial \eta_k} = \frac{\partial^2 \ell}{\partial \theta_j \, \partial \theta_k} \frac{\partial \theta_j}{\partial \eta_j} \frac{\partial \theta_k}{\partial \eta_k} \quad j \neq k,
$$

because η_j is only a function of θ_j and not θ_k. An example where this does not occur is a negative binomial distribution parameterized by p and k with $\eta_1 = \operatorname{logit} p = \log\left(k/(\mu + k)\right)$ and $\eta_2 = \log k$ so that η_1 is a function of both parameters.

2.4 Starting Values

One of the functions of `@initialize` is to compute starting values. That is, all VGAM family functions should be *self-starting*. It suffices to assigning to the variable `etastart` a $n \times M$ matrix of linear predictors $(\boldsymbol{\eta}_1, \ldots, \boldsymbol{\eta}_n)^T$.

Choosing good starting values is not always easy, and is often crucial to the success of the scoring algorithm. For simple univariate models choosing $\hat{\eta}_i = g(y_i)$ often works. This follows as it can be shown for GLMs that

$$\begin{aligned} g(y_i) &\approx g(\mu_i) + g'(\mu_i)\,(y_i - \mu_i) \\ &= \eta_i + \frac{\partial \eta}{\partial \mu_i}\,(y_i - \mu_i) \\ &= z_i \end{aligned}$$

which tells us that the adjusted dependent variable z_i is a local approximation to $g(y_i)$. More generally, choosing $\boldsymbol{\mu}_i$ equalling its MLE is a good idea. However, sometimes $g(y_i)$ is undefined for certain values of y_i and this must be taken care of, e.g., $\log(0)$ is undefined for the Poisson distribution.

2.5 Arguments in VGAM Family Functions

Although there are no arguments in `simple.exponential()`, almost all VGAM family functions offer a variety such as a

- link, e.g., `cumulative(link="logit")`, `cratio(link=probit)`;
- initial value, e.g.,

  ```
  > args(betaff)

  function (link = "loge", earg = list(), i1 = NULL, i2 = NULL,
      trim = 0.05, A = 0, B = 1, zero = NULL)
  NULL
  ```

  Here, `i1` and `i2` are optional initial values.

- constraint, e.g.,

  ```
  > args(cratio)

  function (link = "logit", earg = list(), parallel = FALSE, reverse = FALSE,
      zero = NULL)
  NULL
  ```

  The arguments `parallel` and `zero` refer to linear constraints-on-the-functions. For more details see Section 2.8.

This richer set of options contrasts with the single argument of a typical `glm()` family function which is just `link`.

2.6 Other Notes about VGAM Family Functions

1. Ideally at least one of `@deviance` and `@loglikelihood` should exist. VGAM uses `@deviance` if it exists, otherwise `@loglikelihood` for testing convergence. It is sometimes a good idea to program both. Having either enables half-stepping to occur—each IRLS iteration is guaranteed to be an improvement if the next step happens to 'overshoot'. Half-stepping means that a half-step will be taken until the change in deviance or loglikelihood is an improvement over the current iteration. Half-stepping is not possible with `crit="coeff"` since there is no objective function.

2. To facilitate the variety of VGAM link functions available, some useful functions are

   - `dtheta.deta(theta, link, earg = list())` and `d2theta.deta2(theta, link, earg = list())`, which compute $d^k\theta/d\eta^k$, $k = 1, 2$, for a variety of link functions. For example, `dtheta.deta(lambda, "loge")` and `d2theta.deta2(lambda, "loge")` both return `lambda` because $\eta = \log\,\lambda \Rightarrow \lambda = e^\eta \Rightarrow d^k\lambda/d\eta^k = e^\eta = \lambda$.
   - Note: `dtheta.deta()` should apply to only the parameter and not a function of the parameter. For example, to compute $\partial(1-p)/\partial\eta$ don't use `dtheta.deta(1-p, link)` but rather use $-\partial p/\partial\eta$ with `-dtheta.deta(p, link)`.
   - `theta2eta(theta, link, earg = list())` and `eta2theta(eta, link, earg = list())` which facilitate `@link` and `@inverse`. For example, `dtheta.deta(prob, "logit")` and `d2theta.deta2(prob, "logit")` return `prob*(1-prob)` and `prob*(1-prob)*(1-2*prob)` respectively. This is because $\eta = \mathrm{logit}\,p \Rightarrow p = e^\eta/(1+e^\eta) \Rightarrow dp/d\eta = e^\eta/(1+e^\eta)^2 = p(1-p)$ etc.
   - The function `namesof()` is useful for `@blurb`. For example,

     ```
     > namesof("lambda", "loge")
     [1] "log(lambda)"
     ```

     and

     ```
     > namesof("prob", "logit", tag = FALSE)
     [1] "logit(prob)"
     ```

     The arguments of `namesof()` are:

     ```
     > args(namesof)
     function (theta, link, earg = list(), tag = FALSE, short = TRUE)
     NULL
     ```

3. With VGAM family functions corresponding to a certain type of data (e.g., `binom2.or()` and `binom2.rho()`), it is important to allow for a uniform type of processing of response `y`. Currently, models operating on categorical data should call `process.categorical.data.vgam` from `@initialize`—this does preprocessing, e.g., convert a factor into the appropriate matrix of counts. Similarly, `process.binomial2.data.vgam` for `binom2.or()` and `binom2.rho()` families. The deviance for categorical data is `Deviance.categorical.data.vgam` etc.

2.7 The wz Data Structure

The variable wz stores the working weight matrices $\mathbf{W}_i$ in a special format called the *matrix-band* format. This format comprises a $n \times M^*$ matrix where

$$M^* \;=\; \sum_{i=1}^{\mathit{hbw}} (M - i + 1) \;=\; \frac{1}{2}\mathit{hbw}\,(2\,M - \mathit{hbw} + 1)$$

is the number of columns. Here, *hbw* refers to the *half-bandwidth* of the matrix, which is an integer between 1 and M inclusive. A diagonal matrix has unit half-bandwidth, a tridiagonal matrix has half-bandwidth 2 etc.

Suppose $M = 4$. Then wz will have $M^* = 10$ columns enumerating the unique ($\mathbf{W}_i$ is symmetric) elements of $\mathbf{W}_i$ as follows:

$$\mathbf{W}_i \;=\; \begin{pmatrix} 1 & 5 & 8 & 10 \\ & 2 & 6 & 9 \\ & & 3 & 7 \\ & & & 4 \end{pmatrix}. \tag{5}$$

That is, the order is firstly the diagonal, then the band above that, followed by the second band above the diagonal etc.

Why is such a format adopted? For this example, if $\mathbf{W}_i$ is diagonal then only the first 4 columns of wz are needed. If $\mathbf{W}_i$ is tridiagonal then only the first 7 columns of wz are needed. As well as reducing the size of wz itself in most cases, the matrix-band format often makes the computation of wz very simple and efficent, e.g., see the VGAM family functions involving categorical data. A final reason is that sometimes we want to input $\mathbf{W}_i$ into VGAM. If wz is $M \times M \times n$ then `vglm(..., weights=wz)` will result in an error, whereas if wz being an $n \times M^*$ matrix will work.

If $\mathbf{W}_i$ *is* banded then wz needs not have $\frac{1}{2}M(M+1)$ columns; only M^* columns suffice, and the rest of the elements of $\mathbf{W}_i$ are implicitly zero.

2.7.1 Auxiliary functions

To facilitate the use of the matrix-band format, the functions listed in Table 3 are distributed with VGAM. Writers of VGAM family functions should look at the online help for examples and further details about those functions. Here are some common examples of usage.

Table 3: *Auxiliary functions for the matrix-band format of* wz*. Some are not yet working.*

Function name	Purpose
`dimm(M, hbw=M)`	Returns M^* given M. `dimm` stands for **dim**ension-of-**m**atrix.
`iam(j,k,M,both=` `TRUE,diag=TRUE)`	Either maps (j,k) onto $\{1,2,\ldots,M^*\}$ or supplies the indices of the array. `iam` stands for **i**ndex-of-**a**rray-to-**m**atrix.
`m2a(x)`	Converts a matrix in matrix-band form into a $M \times M \times n$ array. Alternatively, `x` may be a VGAM object, in which case `weights(x, type="working")` is the matrix to be converted.
`a2m(x)`	Converts a $M \times M \times n$ array into a matrix of matrix-band form.

1. In the 4×4 example above,

```
> iam(NA, NA, M = 4, both = TRUE, diag = TRUE)

$row.index
 [1] 1 2 3 4 1 2 3 1 2 1

$col.index
 [1] 1 2 3 4 2 3 4 3 4 4
```

   returns the indices for the respective array coordinates for successive columns of matrix-band format (see (5)). If `diag=FALSE` then the first 4 elements in each vector are omitted. Note that the first two arguments of `iam()` are not used here and have been assigned `NA`s for simplicity.

2. If $M > 1$ then a typical use is as follows. Consider the VGAM family function `normal1()` for the univariate normal distribution. It has $\boldsymbol{\eta} = (\mu, g(\sigma))^T$, and using its EIM, $(\mathbf{W}_i)_{11} = \sigma_i^{-2}$, $(\mathbf{W}_i)_{12} = 0$, and $(\mathbf{W}_i)_{22} = 2\,\sigma_i^{-2}$. Then

```
> normal1()@weight

expression({
    wz = matrix(as.numeric(NA), n, 2)
    ed2l.dmu2 = -1/sd^2
    ed2l.dsd2 = -2/sd^2
    wz[, iam(1, 1, M)] = -w * ed2l.dmu2 * dmu.deta^2
    wz[, iam(2, 2, M)] = -w * ed2l.dsd2 * dsd.deta^2
    wz
})
```

   Note that `M` is already set to 2, and `wz` has two columns. An alternative is to have `wz <- matrix(0, n, dimm(M))` in the top line.

3. For the multinomial logit model, one has $(\mathbf{W}_i)_{jj} = w_i\,\mu_{ij}(1-\mu_{ij}),\ j = 1, \ldots, M$, and $(\mathbf{W}_i)_{jk} = -w_i\,\mu_{ij}\mu_{ik},\ j \neq k$. One can program this using

```
> multinomial()@weight

expression({
    tiny = (mu < .Machine$double.eps^0.5) | (mu > 1 - .Machine$double.eps^0.5)
    if (M == 1)
        wz = mu[, 1] * (1 - mu[, 1])
    else {
        index = iam(NA, NA, M, both = TRUE, diag = TRUE)
        wz = -mu[, index$row] * mu[, index$col]
        wz[, 1:M] = wz[, 1:M] + mu[, 1:M]
    }
    atiny = (tiny %*% rep(1, ncol(mu))) > 0
    if (any(atiny)) {
        if (M == 1)
            wz[atiny] = wz[atiny] * (1 + .Machine$double.eps^0.5) +
```

```
                .Machine$double.eps
        else wz[atiny, 1:M] = wz[atiny, 1:M] * (1 + .Machine$double.eps^0.5) +
            .Machine$double.eps
    }
    w * wz
})
```

Alternatively, something like

```
        wz <- mu[,1:M] * (1 - mu[,1:M])
        if(M > 1) {
            index <- iam(NA, NA, M=M, both=TRUE, diag=FALSE)
            wz <- cbind(wz, -mu[,index$row] * mu[,index$col])
        }
```

uses the same idea. See also

```
> dirichlet()@weight
```

```
expression({
    index = iam(NA, NA, M, both = TRUE, diag = TRUE)
    wz = matrix(trigamma(sumshape), nrow = n, ncol = dimm(M))
    wz[, 1:M] = wz[, 1:M] - trigamma(shape)
    wz = -w * wz * dsh.deta[, index$row] * dsh.deta[, index$col]
    wz
})
```

2.8 Implementing Constraints Within Family Functions

For many models certain types of linear constraints-on-the-functions are common and thus should be easily used by the user. Here are some examples:

1. the proportional odds model has a `parallel` option to constrain $\beta_{(1)k} = \cdots = \beta_{(M)k}$ for a covariate x_k,

2. the bivariate logit model (`binom2.or()`) has an `exchangeable` option to allow $\eta_1 = \eta_2$ for exchangeable data,

3. the shape parameter of a generalized Pareto distribution (GPD) is often just a scalar and not a function of the $\boldsymbol{x}$:

```
> args(gpd)

function (threshold = 0, lscale = "loge", lshape = "logoff",
    escale = list(), eshape = if (lshape == "logoff") list(offset = 0.5) else if (lsh
        "elogit") list(min = -0.5, max = 0.5) else NULL, percentiles = c(90,
        95), iscale = NULL, ishape = NULL, tshape0 = 0.001, method.init = 1,
    zero = 2)
NULL
```

Writers of VGAM family functions should consider adding arguments to aid the creation of constraint matrices corresponding to common constraints anticipated for that model. VGAM allows this to easily be done using the functions `cm.vgam()` and `cm.zero.vgam()`. The former applies to logical vectors such as `parallel` and `exchangeable` and have a common format[1]. The latter applies just to the `zero` constraint. We will discuss the two functions separately below. Both functions are to be put in `@constraints` or `@initialize`. A typical usage is

```
> print(multinomial()@constraints)

expression({
    constraints = cm.vgam(matrix(1, M, 1), x, FALSE, constraints,
        intercept.apply = FALSE)
    constraints = cm.zero.vgam(constraints, x, NULL, M)
    constraints = cm.nointercept.vgam(constraints, x, NULL, M)
})
```

The first argument of `cm.vgam()` is the constraint matrix; here $\mathbf{1}_M$ corresponds to the parallelism constraint. The second argument is always the LM design matrix `x`. the fourth argument is the constraint vector itself (i.e., always the variable `constraints`). The function `cm.zero.vgam()` is even simpler. The line containing `cm.zero.vgam()` should be pasted in its entirety as above. If possible, programmers should use the same names for their constraint argument as existing VGAM family functions. Currently these are `parallel`, `exchangeable`, `zero`.

For further information about `substitute()` see Section 4.

Here are some further notes.

1. What if the user has

[1] Recall that valid examples are `TRUE`, `FALSE`, `c(TRUE,x2=FALSE,f=FALSE)`, `c(FALSE,x2=TRUE,x5=TRUE,x6=FALSE)`, `c("(Intercept)"=TRUE,FALSE,x2=TRUE)`

```
cm <- diag(M)
fit <- vglm(ymat ~ x2 + x3, myVGAMfamfun(parallel=TRUE),
            constraints=list("(Intercept)"=cm, x2=cm, x3=cm))
```

i.e, which constraint is applied to x_2, say? Answer: it depends on the model and what the programmer feels has higher precedence. However, since it is more difficult to specify `constraints=`, this should override values of the constraint vector `parallel`. Note that, at present, VGAM requires the constraints list to be fully specified, i.e., all terms must be included in the list.

2. `cm.vgam()` has an argument `intercept.apply = FALSE` which means that by default the constraint is not applied to the intercept. Also, `overwrite = FALSE` meaning it will not overwrite any value in the `constraints` argument of `vglm()`. Technically, it adds constraint matrices to the list `constraints` to `vglm()`/`vgam()`. The function `cm.zero.vgam()` deletes certain columns off the existing constraint matrix.

3. The VGAM family function is free to apply or not apply the constraint to the intercept. For example, for the proportional odds model, $\eta_j = \boldsymbol{\beta}_j^T \boldsymbol{x}$ so it shouldn't, whereas with the bivariate logit model, $\eta_1 = \eta_2$ for exchangeable data, therefore it should.

2.9 The `extra` argument

VGAM modelling functions `vglm()`, `vgam()`, etc. have an argument called `extra` which allows extra information to be passed into the VGAM family function. After the model is fitted the `extra` variable is saved in the `extra` slot of the fitted model.

3 More Advanced Features

3.1 VGAM Link Functions

VGAM provides a number of links and some of them are summarized in Table 4. One can easily write a new VGAM link function. As a guideline just look at a simple link function such as `logit()` and `loge()`[2]. Some link functions require a specified parameter (or parameters), e.g., a log with an offset or a power transformation. This can also be handled; see `logoff()` and `powl()`.

Consider the logit link:

```
> print(logit)

function (theta, earg = list(), inverse = FALSE, deriv = 0, short = TRUE,
    tag = FALSE)
{
    if (is.character(theta)) {
        string <- if (short)
            paste("logit(", theta, ")", sep = "")
        else paste("log(", theta, "/(1-", theta, "))", sep = "")
        if (tag)
            string <- paste("Logit:", string)
        return(string)
    }
    if (!inverse && is.list(earg) && length(earg$bval)) {
        theta[theta <= 0] <- earg$bval
        theta[theta >= 1] <- 1 - earg$bval
    }
    if (inverse) {
        if (deriv > 0) {
            1/Recall(theta = theta, earg = earg, inverse = FALSE,
                deriv = deriv)
```

[2] `loge()` and `powl()` are used instead of `log` and `power` because, of course, they are already used.

Table 4: *Some* VGAM *link functions which are currently available.*

Link	$g(\theta)$	Range of θ
`cloglog`	$\log\{-\log(1-\theta)\}$	$(0,1)$
`fisherz`	$\frac{1}{2}\log_e\{(1+\theta)/(1-\theta)\}$	$(-1,1)$
`fsqrt`	$\sqrt{2\theta} - \sqrt{2(1-\theta)}$	$(0,1)$
`identity`	θ	$(-\infty,\infty)$
`loge`	$\log_e(\theta)$	$(0,\infty)$
`logc`	$\log_e(1-\theta)$	$(-\infty,1)$
`logit`	$\log_e(\theta/(1-\theta))$	$(0,1)$
`logoff`	$\log_e(\theta+A)$	$(-A,\infty)$
`probit`	$\Phi^{-1}(\theta)$	$(0,1)$
`powl`	θ^p	$(0,\infty)$
`rhobit`	$\log_e\{(1+\theta)/(1-\theta)\}$	$(-1,1)$

```
        }
        else {
            eta <- care.exp(theta)
            eta/(1 + eta)
        }
    }
    else {
        switch(deriv + 1, {
            log(theta/(1 - theta))
        }, theta * (1 - theta), theta * (1 - theta) * (1 - 2 *
            theta))
    }
}
<environment: namespace:VGAM>
```

This function should be easily understood, given that $\eta = \log\{\theta/(1-\theta)\}$, $\theta = e^\eta/(1+e^\eta)$, $d\theta/d\eta = \theta(1-\theta)$, $d^2\theta/d\eta^2 = \theta(1-\theta)(1-2\theta)$.

3.1.1 Making Use of Link Functions

To make life easier for writers of VGAM family functions, Table 6 lists the auxiliary functions that extract the labels and derivatives from VGAM link functions.

3.2 Control Functions

VGAM fits a very large class of models, therefore it is clear that the algorithmic constants supplied by `vglm.control()` and `vgam.control()` will not be suitable for all VGAM family functions. For this reason, each VGAM family function is allowed to have its own control function or functions. For example, if the family function is `fred()`, then VGAM will evaluate `vglm.control.fred()` or `vgam.control.fred()` if it exists, in order to get some or all of its algorithmic constants. This facility is useful for models that are inherantly ill-conditioned, so that relaxation of some parameters are warranted. In general, VGAM family functions have the classes, and these are evaluated in reverse order. For example, if `fred@vfamily=c("fred","circular")` then `vgam(..., fam=fred)` would evaluate (in order) `vgam.control()`, `vgam.control.circular()` and then `vgam.control.fred()`. Here,

Table 5: *Arguments in a* VGAM *link function.*

Argument	Function
`theta`	θ—the parameter to be estimated. May be numerical or character. If character then only `short` and `tag` are accessed.
`earg`	An extra argument that allows other parameters, e.g., an offset value or power value. It is a list.
`inverse`	Logical. Use the inverse link function?
`deriv`	Either 0, 1 or 2. Returns $d^k\theta/d\eta^k$. If `inverse=TRUE` then see the code.
`short`	Logical. If `TRUE` then returns a short label, else a long label. e.g., `"logit(theta)"` and `"log(theta/(1-theta)"`.
`tag`	If true, add a descriptor to the label, e.g., adds `"Logit:  "` to the front of the label.

fred() is one of several family functions for circular data, so it might suffice to have just vglm.control.circular() and vgam.control.circular() instead of two for each family function.

The following is similar to glm.control():

```
> vglm.control()

$backchat
[1] FALSE

$checkwz
[1] TRUE

$convergence
expression({
    switch(criterion, coefficients = if (iter == 1)
        iter < maxit
    else (iter < maxit && max(abs(new.crit - old.crit)/(abs(old.crit) +
        epsilon)) > epsilon), abs(old.crit - new.crit)/(abs(old.crit) +
        epsilon) > epsilon && iter < maxit)
})

$criterion
[1] "deviance"

$epsilon
[1] 1e-07

$half.stepsizing
[1] TRUE

$maxit
[1] 30

$min.criterion
     deviance loglikelihood           AIC    Likelihood           rss
         TRUE         FALSE          TRUE         FALSE          TRUE
 coefficients
         TRUE
```

Table 6: *Functions that make use of a* VGAM *link function.*

Function	Returns
dtheta.deta(theta, link, earg = list())	$d\theta/d\eta$
d2theta.deta2(theta, link, earg = list())	$d^2\theta/d\eta^2$
eta2theta(theta, link, earg = list())	$\theta = g^{-1}(\eta)$
namesof(theta, link, earg = list(), tag = FALSE, short = TRUE)	A description of the link function
theta2eta(theta, link, earg = list())	$\eta = g(\theta)$

```
$save.weight
[1] FALSE

$stepsize
[1] 1

$trace
[1] FALSE

$wzepsilon
[1] 1.818989e-12

$xij
NULL
```

3.3 Example

```
> print(fff.control)

function (save.weight = TRUE, ...)
{
    list(save.weight = save.weight)
}
<environment: namespace:VGAM>
```

Any algorithmic constants not supplied by `fff.control()` will be obtained by `vglm.control()` or `vgam.control()`.

3.4 Classes of Family Functions

`glm()` and `gam()` use family functions that evaluates to a family object, e.g.,

```
> class(binomial())

[1] "family"
```

VGAM use family functions have

```
> class(micmen())

[1] "vglmff"
attr(,"package")
[1] "VGAM"
```

but allow some VGAM family function-specific actions via

```
> micmen()@vfamily

[1] "micmen"     "vnonlinear"
```

What features does this add?

1. Algorithmic constants for this could be supplied by `vnonlinear.control()`. This is highly advantageous, since there are scores of nonlinear regression models under the `"vnonlinear"` umbrella. This saves a `.control` function for each VGAM family function if `vglm.control()` is unsuitable.

2. Note that `print.vnonlinear()` would then need to be modified to handle a fitted object or simply a family function.

4 Examples

This section gives successive examples of increasing complexity. Prospective writers of family functions are also encouraged to examine these in detail, as well as some of the family objects and generator functions provided with VGAM via `print.default()`.

4.1 Extending the Exponential Distribution

We will extend the family function `simple.exponential()` described previously in three ways. Firstly, suppose one wishes to introduce a known[3] location parameter a, i.e., $f(y;\lambda) = \lambda\, e^{-\lambda\,(y-a)}$, where $y \geq a$. One then has

$$E(Y) \;=\; \mu \;=\; a + \frac{1}{\lambda} \quad \text{and} \quad \mathrm{Var}(Y) \;=\; \frac{1}{\lambda^2} \;=\; (\mu - a)^2 \,.$$

The default value of a is zero. Secondly, we will allow a choice between the EIM and the OIM. The VGAM family function, which is considerably longer, is

```
> print(exponential)

function (link = "loge", earg = list(), location = 0, expected = TRUE)
{
    if (!is.Numeric(location, allow = 1))
        stop("bad input for argument \"location\"")
    if (mode(link) != "character" && mode(link) != "name")
        link = as.character(substitute(link))
    if (!is.list(earg))
        earg = list()
    new("vglmff", blurb = c("Exponential distribution\n\n", "Link:     ",
        namesof("rate", link, tag = TRUE), "\n", "Mean:     ",
        "mu =", location, "+ 1 / ", namesof("rate", link, tag = TRUE,
            earg = earg), "\n", "Variance: ", if (location ==
            0) "Exponential: mu^2" else paste("(mu-", location,
            ")^2", sep = "")), initialize = eval(substitute(expression({
        if (ncol(cbind(y)) != 1) stop("response must be a vector or a one-column matrix")
        extra$loc = .location
        if (any(y <= extra$loc)) stop(paste("all responses must be greater than",
            extra$loc))
        predictors.names = namesof("rate", .link, tag = FALSE)
        mu = y + (y == extra$loc)/8
        if (!length(etastart)) etastart = theta2eta(1/(mu - extra$loc),
            .link, earg = .earg)
    }), list(.location = location, .link = link, .earg = earg))),
        inverse = eval(substitute(function(eta, extra = NULL) extra$loc +
            1/eta2theta(eta, .link, earg = .earg), list(.link = link,
            .earg = earg))), last = eval(substitute(expression({
            misc$location = extra$loc
            misc$link = c(rate = .link)
```

[3]The location parameter of an exponential distribution must be treated as known. Due to the memoryless property, it would be unestimable if unknown.

```
            misc$earg = list(rate = .earg)
        }), list(.link = link, .earg = earg))), link = eval(substitute(function(mu,
            extra = NULL) theta2eta(1/(mu - extra$loc), .link,
            earg = .earg), list(.link = link, .earg = earg))),
        deviance = eval(substitute(function(mu, y, w, residuals = FALSE,
            eta, extra = NULL) {
            devy = -log(y - .location) - 1
            devmu = -log(mu - .location) - (y - .location)/(mu -
                .location)
            devi = 2 * (devy - devmu)
            if (residuals) sign(y - mu) * sqrt(abs(devi) * w) else sum(w *
                devi)
        }, list(.location = location, .earg = earg))), vfamily = c("exponential"),
        deriv = eval(substitute(expression({
            rate = 1/(mu - extra$loc)
            dl.drate = mu - y
            drate.deta = dtheta.deta(rate, .link, earg = .earg)
            w * dl.drate * drate.deta
        }), list(.link = link, .earg = earg))), weight = eval(substitute(expression({
            d2l.drate2 = -((mu - extra$loc)^2)
            wz = -(drate.deta^2) * d2l.drate2
            if (!.expected) {
                d2rate.deta2 = d2theta.deta2(rate, .link, earg = .earg)
                wz = wz - dl.drate * d2rate.deta2
            }
            w * wz
        }), list(.link = link, .expected = expected, .earg = earg))))
}
<environment: namespace:VGAM>
```

Here are some notes.

1. The S function `substitute()` is used to substitute the value of arguments in a VGAM family function into an S expression that will be evaluated later. Note `exponential()` repeatedly substitutes the argument `link`. Rather than substitute the location parameter into *all* the components, the above code has the value placed into `extra` in `@initialize`, and other components of the function use it through `extra`. This technique is particularly beneficial if the size of the argument is large because substituting a large structure can be inefficient. For example, if one had

   ```
   fit <- vglm(y ~ x1, exponential(location=1:length(y)))
   ```

   where `y` was a large vector. However, the value of `location` is directly substituted into `@deviance` for the obvious reason that the VGAM family function used is attached to the fitted object.

2. If more than one variable needs to be passed into `@inverse`, `@link` etc., then `extra` should be a list.

5 Distributing Your Family Functions

People are encouraged to write VGAM family functions and distribute them separately as an R package. This then should 'Depends' on VGAM in the `DESCRIPTION` file.

Exercises

1. Modify `exponential()` so that it has an additional argument called `irate=NULL` which is an optional initial value for the rate parameter. If inputted by the user, it should override all other self-starting initial values.

References

Chambers, J. M., Hastie, T. J. (Eds.), 1993. Statistical Models in S. Chapman & Hall, New York.