VGAM Family Functions for Generalized Linear and Additive Models

T. W. Yee

February 20, 2008

Beta Version 0.7-4

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 in detail VGAM family functions for fitting generalized linear models (GLMs) and generalized additive models (GAMs). This provides another alternative because both R and S-PLUS have a `glm()` and `gam()` function (in R `gam()` can be found in the mgcv library). Therefore one of the purposes of this document is to highlight the *differences* that exist between VGAM and the S-PLUS `glm()` and `gam()` functions. Savvy users of `glm()` and `gam()` may therefore merely wish to read Section 5 and look through the examples in Section 7. Other users should read the entire document. For general theory and instructions about the VGAM software library see the VGAM *User Manual*. S-PLUS's `glm()` and `gam()` are described in Chambers and Hastie (1993). The general framework for VGLMs and VGAMs is given in Yee and Wild (1996).

There are many books on GLMs. McCullagh and Nelder (1989) is the most comprehensive, and some other books are Aitkin et al. (1989), Azzalini (1996), Crawley (1993), Dobson (2001),

Fahrmeir and Tutz (2001), Firth (1991), Krzanowski (1998), Lindsey (1997), Myers et al. (2002). The standard reference for GAMs is Hastie and Tibshirani (1990); see also Schimek and Turlach (2000). Venables and Ripley (2002) is also generally useful. G. Smyth's web site at `http://www.maths.uq.edu.au/~gks` has many GLM resources.

Table 1 summarized the GLM models currently implemented by VGAM. Family functions such as `binomial()` distributed in S-PLUS and R do *not* work with `vglm()` and `vgam()`. In general, GLM-type VGAM family functions differ from existing family functions by ending in "`ff`", which stands for (VGAM) "family function".

Table 1: Models currently implemented by VGAM. The "`ff`" stands for "family function", and is used to avoid clashes with other functions.

Family function	Canonical link $\theta(\mu)$	Variance function $b''(\theta)$
`gaussianff()`	μ	1
`binomialff()`	$\log(\mu/(1-\mu))$	$\mu(1-\mu)$
`poissonff()`	$\log \mu$	μ
`gammaff()`	$-1/\mu$	μ^2
`inverse.gaussianff()`	$-1/(2\mu^2)$	μ^3
`quasiff()`	$g(\mu)$	$V(\mu)$

1.1 GLM Basics

GLMs were proposed by Nelder and Wedderburn (1972), and provide a unifying framework for a number of important models. One consequence of this is that a single algorithm (*iteratively reweighted least squares*; IRLS) can be used to fit all models.

Suppose we have independent sample data $(\boldsymbol{x}_i, y_i),\ i = 1, \ldots, n$, where y_i is the response, n is the sample size and $\boldsymbol{x}_i = (x_{i1}, \ldots, x_{ip})^T$ is a vector of p explanatory variables (including the intercept if there is one). A GLM is composed of three parts:

(i) a *random component* $f(y; \mu)$ specifying the distribution of Y,

(ii) a *systematic component* $\eta = \boldsymbol{\beta}^T \boldsymbol{x}$ specifying the variation in Y accounted for by known covariates, and

(iii) a *link function* $g(\mu) = \boldsymbol{\beta}^T \boldsymbol{x}$ that ties the two together.

The η is known as the *linear predictor*, and the random component $f(y; \mu)$ is typically an exponential family distribution with $E(Y|\boldsymbol{x}) = \mu(\boldsymbol{x})$ being the *mean function*. GLMs thus fit

$$g(\mu(\boldsymbol{x}_i)) \;=\; \eta_i \;=\; \boldsymbol{\beta}^T \boldsymbol{x}_i \;=\; \beta_1\, x_{i1} + \cdots + \beta_p\, x_{ip} \tag{1}$$

where the link function g is known, strictly monotone and twice-differentiable. The vector of regression coefficients $\boldsymbol{\beta}$ is usually estimated by maximum likelihood estimation using Newton-Raphson with the expected Hessian. This is known as *Fisher-scoring* and can be fitted by IRLS. In fact, if a canonical link is used, then Fisher-scoring is equivalent to the Newton-Raphson algorithm.

The density of the exponential family can be written

$$f(y_i; \theta_i, \phi) \;=\; \exp\left[\frac{y_i\,\theta_i - b(\theta_i)}{\phi/A_i} + c\left(y_i, \frac{\phi}{A_i}\right)\right] \tag{2}$$

where ϕ is a possibly known *dispersion parameter* (or *scale parameter*), b and c are known functions, and A_i is a *known* prior weight. When ϕ is known, the distribution of Y is a one parameter canonical exponential family member. When ϕ is unknown, it is often a nuisance parameter and estimated by the method of moments. The prior weights A_i are entered into VGAM via the `weights` argument. The role of ϕ is curious: it is often treated as an unknown constant but not as a parameter.

Noting that $\ell(\mu; y) = \log\, f(y; \mu)$, the log-likelihood is

$$\ell(\theta, \phi; y) \;=\; \sum_{i=1}^{n} \frac{y_i\,\theta_i - b(\theta_i)}{\phi/A_i} + c\left(y_i, \frac{\phi}{A_i}\right), \tag{3}$$

and the score function is

$$U(\theta) \;=\; \sum_{i=1}^{n} \frac{y_i - b'(\theta_i)}{\phi/A_i}. \tag{4}$$

Then

$$\begin{aligned} E(Y_i) &= b'(\theta), \\ \mathrm{Var}(Y_i) &= \frac{\phi}{A_i}\, b''(\theta). \end{aligned}$$

Distribution	Density/probability function $f(y)$	Range of y	Range of parameters	$E(Y)$	Var(Y)	ϕ	$b(\theta)$
Gaussian	$(2\pi\sigma^2)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(y-\mu)^2/\sigma^2\right\}$	$(-\infty, \infty)$	$-\infty < \mu < \infty$, $\sigma > 0$	$\mu = \theta$	σ^2/A	σ^2	$\theta^2/2$
Binomial	$\binom{A}{Ay} p^{Ay}(1-p)^{A(1-y)}$	$0(1/A)1$	$0 < p < 1$	$p = e^\theta/\left(1+e^\theta\right)$	$p(1-p)/A$	1	$\log\left(1+e^\theta\right)$
Poisson	$\exp\{-\lambda\}\lambda^y/y!$	$0(1)\infty$	$\lambda > 0$	$\lambda = e^\theta$	λ/A	1	e^θ
Gamma	$(k/\mu)^k\, y^{k-1} \exp\{-ky/\mu\}/\Gamma(k)$	$(0, \infty)$	$\mu > 0$, $k > 0$	$\mu = -1/\theta$	$\mu^2/(Ak)$	$1/k$	$-\log(-\theta)$
Inverse Gaussian	$\left(\frac{\lambda}{2\pi\, y^3}\right)^{\frac{1}{2}} \exp\left\{-\frac{\lambda}{2\,\mu^2}\frac{(y-\mu)^2}{y}\right\}$	$(0, \infty)$	$\mu > 0$, $\lambda > 0$	$\mu = 1/\sqrt{-2\theta}$	$\mu^3/(\lambda A)$	$1/\lambda$	$-\sqrt{-2\theta}$

Table 2: Summary of GLMs supported by VGAM. A is a known prior weight.

The *variance function* is $V(\mu) = b''(\theta(\mu))$, i.e., the variance as a function of the mean. If ϕ is known then $\mathbf{A} = \mathrm{diag}(A_i)$.

IRLS forms the core algorithm behind VGAM. For GLMs, it involves regressing at each iteration (the ath iteration, say) the *adjusted dependent variable* (or *modified* or *pseudo response*)

$$z_i^{(a)} \;=\; \eta_i^{(a)} + \frac{y_i - \mu_i^{(a)}}{d\mu_i^{(a)}/d\eta_i} \tag{5}$$

with weights

$$w_i^{(a)} \;=\; \frac{A_i}{V(\mu_i^{(a)})} \left(\frac{d\mu_i^{(a)}}{d\eta_i}\right)^2$$

against $\boldsymbol{x}_i$. This is achieved by weighted least squares: $\mathbf{X}$ is the model matrix and $\mathbf{W}^{(a)}$ is $\mathrm{diag}(w_1^{(a)}, \ldots, w_n^{(a)})$. The above weights are actually obtained by the *expected* Hessian (rather than the *observed*), so this is Fisher scoring. For some models this equates to Newton-Raphson. With the new $\boldsymbol{\beta}^{(a+1)}$, a new $\boldsymbol{\eta}^{(a+1)}$ and $\boldsymbol{\mu}^{(a+1)}$ are computed, and the cycle is continued till convergence.

When close to the solution, the convergence rate is quadratic, meaning that the number of correct decimal places doubles at each iteration. At convergence, we have

$$\widehat{\mathrm{Var}}(\hat{\boldsymbol{\beta}}) \;=\; \hat{\phi}\left(\mathbf{X}^T\mathbf{W}^{(a)}\mathbf{X}\right)^{-1}. \tag{6}$$

Following from the MASS library, VGAM comes with the function `vcov()` which returns (6).

Initial values for IRLS are chosen by the specific family functions in `@initialize`. They rarely fail for GLMs, however if they do, users may input values into one of the `coefstart=`, `etastart=`, and `mustart=` arguments of `vglm()` and `vgam()`.

1.2 Inference

There is a elegant bed of theory for inference pertaining to GLM models that naturally extend that of ordinary linear theory. Under certain conditions (e.g., grouped binary data), the *deviance* $D(\boldsymbol{y}; \boldsymbol{\mu})$ can be used to measure goodness of fit of a model. The *scaled deviance* satisfies

$$\frac{D(\boldsymbol{y}; \boldsymbol{\mu})}{\phi} \;=\; -\,2\left\{\ell(\boldsymbol{\mu}; \boldsymbol{y}) - \ell(\boldsymbol{y}; \boldsymbol{y})\right\}. \tag{7}$$

The term $\ell(\boldsymbol{y}; \boldsymbol{y})$ corresponds to a *saturated model*—one where $\boldsymbol{\mu}$ maximizes ℓ over $\boldsymbol{\mu}$ unconstrained. For GLMs, a saturated model has $b'(\hat{\theta}_i) = \hat{\mu}_i = y_i$. If $\phi = 1$ then

$$D \;=\; 2\sum_{i=1}^{n} A_i \left[\{y_i\,\theta(y_i) - b(\theta(y_i))\} - \left\{y_i\hat{\theta}_i - b(\hat{\theta}_i)\right\}\right] \qquad \left(=\; \sum_{i=1}^{n} d_i, \mathrm{say},\right) \tag{8}$$

where $\hat{\theta}$ is the MLE. This is called the deviance of a model even when the scale parameter is unknown or is known to have a value other than one. The scaled deviance is sometimes called the *residual deviance*.

Smaller values of D indicate a better fit. The deviance, a generalization of the residual sum of squares in ordinary regression, is a function of the data and of the fitted values, and when divided by a dispersion parameter ϕ, is asymptotically χ^2 (as A_i or $n \to \infty$). More generally, to test if a smaller model (i.e., one with fewer variables) is applicable given a larger model, it is only necessary to examine the increase in the deviance, and to compare it to a χ^2 distribution

with degrees of freedom equal to the difference in the numbers of independent parameters in the two models (as each parameter or term has 1 degree of freedom). In the model building process, this enables a test to be carried out as to which variables can be deleted to form a smaller model, or which variables need to be added to form a larger model.

For a Gaussian family with identity link, ϕ is the variance and D is the residual sum of squares, i.e.,

$$D \;=\; \sum_{i=1}^{n} A_i (y_i - \mu_i)^2.$$

Hence

$$D/\phi \;\sim\; \chi^2_{n-p},$$

leading to the unbiased estimator

$$\hat{\phi} \;=\; D/(n-p) \tag{9}$$

because $E(\chi^2_\nu) = \nu$.

An alternative estimator is the sum of squares of the standardized residuals divided by the residual degrees of freedom:

$$\widetilde{\phi} \;=\; \frac{1}{n-p} \sum_{i=1}^{n} \frac{(y_i - \hat{\mu}_i)^2}{V(\hat{\mu}_i)/A_i}. \tag{10}$$

If allowed by `dispersion=0`, VGAM uses this formula to estimate the scale parameter ϕ. It has much less bias than (9). For the Gaussian errors $\widetilde{\phi} = \hat{\phi}$.

If M_0 is a submodel within a model M (that is, nested) with $q < p$ parameters, and if ϕ is known, then

$$\frac{D_{M_0} - D_M}{\phi} \;\dot{\sim}\; \chi^2_{p-q}.$$

If ϕ is unknown, then

$$\frac{D_{M_0} - D_M}{\phi\,(p-q)} \;\dot{\sim}\; F_{p-q,n-p}.$$

1.3 Residuals

The `resid()` (or `residuals()`) method function returns four types of residuals for `glm()` and `gam()` objects:

(i) *deviance residuals* are

$$r_i^D \;=\; \mathrm{sign}(y_i - \hat{\mu}_i)\sqrt{d_i}, \quad \text{where} \quad D \;=\; \sum_{i=1}^{n} d_i. \tag{11}$$

(cf. (8)). This residual type is the default, and is the most useful for diagnostic purposes. In particular, they are used when plotting `vgam()` objects with residuals, i.e., `plot(vgamobject, resid=T)`.

(ii) *working residuals* are

$$r_i^W \;=\; (y_i - \hat{\mu}_i)\,\frac{\partial \hat{\eta}_i}{\partial \hat{\mu}_i}. \tag{12}$$

They arise from the final IRLS iteration (cf. (5)).

(iii) *Pearson residuals* are related to the working residuals and are

$$r_i^P = \frac{y_i - \hat{\mu}_i}{\sqrt{V(\hat{\mu}_i)}}. \quad \text{Note that} \quad X^2 = \sum_{i=1}^{n} \left(r_i^P\right)^2 = \sum_{i=1}^{n} \frac{(y_i - \hat{\mu}_i)^2}{V(\hat{\mu}_i)}$$

is the Pearson chi-squared statistic.

(iv) *response residuals* are simply $r_i^R = y_i - \hat{\mu}_i$.

One uses `predict(object, type="deviance")` or `type="working"`, `type="pearson"`, `type="response"`. For the Gaussian family all four types of residuals coincide.

1.4 Quasi-likelihood Models

There are situations where one is uncertain about the use or the validity of the full distribution (2) of the response Y, e.g., when the variance of the data is greater than the model suggests (*over-dispersion*). Wedderburn (1974) proposed the use of the quasi-likelihood to alleviate this problem. Specifically, it replaces the assumptions tied in with (2) by the weaker variance assumption that

$$\mathrm{Var}(Y) \;=\; \phi\, V(\mu) \tag{13}$$

where ϕ is assumed constant across samples. This is extremely useful in applied work when the data are limited and information on the distribution of Y is lacking. However, one may have enough data to reliably estimate a *relationship* between the first two moments as required by the quasi-likelihood model.

In the absence of a likelihood function, one may estimate $\boldsymbol{\beta}$ by solving

$$\boldsymbol{U}_{\boldsymbol{\beta}} \;=\; \sum_{i=1}^{n} A_i \frac{\partial \mu_i^T}{\partial \boldsymbol{\beta}} \mathrm{Var}(Y_i)^{-1} (y_i - \mu_i) \;=\; \mathbf{0}. \tag{14}$$

What is the justification for using this? Consider

$$U_i \;=\; \frac{Y_i - \mu_i}{\phi\, V(\mu_i)}.$$

Then $E(U_i) = 0$ and $\mathrm{Var}(U_i) = 1/\{\phi\, V(\mu_i)\} = -E(\partial\, U_i/\partial\, \mu_i)$. These are the same properties that a log-likelihood derivative has. This suggests that

$$Q(\mu; y) \;=\; \int_y^{\mu} \frac{y - t}{\phi\, V(t)}\, dt$$

may behave like a log-likelihood function for μ. So we fit the *estimating equation*

$$\sum_{i=1}^{n} A_i \frac{\partial\, Q(\mu_i; y_i)}{\partial\, \boldsymbol{\beta}} \;=\; \mathbf{0}, \tag{15}$$

which is equivalent to (14). This contrasts with

$$\sum_{i=1}^{n} A_i \frac{\partial\, \ell(\theta_i; y_i)}{\partial\, \boldsymbol{\beta}} \;=\; \mathbf{0}, \tag{16}$$

with ordinary GLMs. A list of some quasi-likelihood functions is given in Table 3.

Table 3: Some quasi-likelihood functions.

Distribution	$V(\mu)$	$\phi\, Q(\mu; y)$
Gaussian	1	$-(y-\mu)^2/2$
binomial	$\mu(1-\mu)$	$y \,\mathrm{logit}\, \mu + \log(1-\mu)$
Poisson	μ	$y \log \mu - \mu$
gamma	μ^2	$-y/\mu - \log \mu$
Inverse Gaussian	μ^3	$-y/(2\mu^2) + 1/\mu$

1.5 Double Exponential Models

Overdispersion is a common characteristic of data and one method for handling it is to consider the class of *double exponential distributions*. These were proposed by Efron (1986) and the basic idea is that an ordinary one-parameter exponential family allows the addition of a second parameter θ which varies the dispersion of the family without changing the mean. The extended family behaves like the original family with sample size changed from n to $n\theta$. Then $0 < \theta < 1$ corresponds to overdispersion. The extended family is an exponential family in μ when n and θ are fixed, and an exponential family in θ when n and μ are fixed.

The formal definition

$$\tilde{f}_{\mu,\theta,n}(y) \;=\; c(\mu,\theta,n)\,\theta^{1/2}\,\{g_{\mu,n}(y)\}^{\theta}\,\{g_{y,n}(y)\}^{1-\theta}\,[dG_n(y)] \tag{17}$$

is called a double exponential family with parameters μ, θ and n. Here,

$$g_{\mu,n}(y) \;=\; e^{n[\eta y-\psi(\mu)]}\cdot[dG_n(y)] \tag{18}$$

is the density of a given exponential family. The expectation parameter $\mu = \int_{-\infty}^{\infty} y\, g_{\mu,n}(y)\, dG_n(y)$, y is the natural statistic, η is the canonical or natural parameter, $\psi(\mu)$ is a normalizing function, and $G_n(y)$ is the *carrier measure* for the exponential family.

The normalizing constant $c(\mu,\theta,n)$ in (18) nearly equals 1 so (18) is approximated by

$$f_{\mu,\theta,n}(y) \;\approx\; \theta^{1/2}\,\{g_{\mu,n}(y)\}^{\theta}\,\{g_{y,n}(y)\}^{1-\theta}\,[dG_n(y)]\,. \tag{19}$$

Approximately, the mean of Y is μ. The *effective sample size* is the dispersion parameter multiplied by the original sample size, i.e., $n\theta$.

All VGAM family functions for double exponential families are based on the approximation (19). Maximum likelihood estimation is used for the two (mean and dispersion) parameters. In fact, Fisher scoring is used, and the two estimates are asymptotically independent because the expected information matrix (EIM) is diagonal. Indeed, it is approximated by $\mathrm{diag}(n\theta/V(\mu), \frac{1}{2}\theta^{-2})$ where $\mathrm{Var}(Y) = V(\mu)$.

At present only one double exponential family is implemented and that is `dexpbinomial()`. The default model fitted is $\eta_1 = \mathrm{logit}(\mu)$ and $\eta_2 = \mathrm{logit}(\theta)$. This restricts both parameters to lie between 0 and 1, although the dispersion parameter can be modelled over a larger parameter space by assigning the arguments `ldispersion` and `edispersion`, e.g., using the link function `elogit()`. The approximate double binomial family based on (19) gives

$$f_{\mu,\theta}(y) \;\propto\; \theta^{1/2}\,\mu^{ny\theta}(1-\mu)^{n(1-y)\theta}y^{ny(1-\theta)}(1-y)^{n(1-y)(1-\theta)}, \quad y = 0, \frac{1}{n}, \frac{2}{n}, \ldots, 1. \tag{20}$$

A numerical example mimicking those in Efron (1986) is given in Section 7.2.

Currently the approximate double Poisson family based on (19) is not implemented. Its density is

$$f_{\mu,\theta}(y) \;=\; \theta^{1/2}\,e^{-\theta\mu}\left(\frac{e^{-y}\,y^{y}}{y!}\right)\left(\frac{e\mu}{y}\right)^{\theta y}, \quad y = 0, 1, 2, \ldots. \tag{21}$$

For this, the dispersion parameter is numerically unstable.

1.6 Reduced-rank Vector GLMs

It is not uncommon to have a multivariate response where each response is binary or Poisson. Let the number of response variables be M, say. If each of the M GLMs are fitted separately then there will be Mp regression coefficients altogether. This will be large when M and p are both modest in size. The regression coefficients can be stored in a $M \times p$ matrix $\mathbf{B}$, say, and this will be large and dense in general. So a good idea is to approximate $\mathbf{B}$ by a low-rank matrix. This gives rise to the class of *reduced-rank vector generalized linear models* (RR-VGLMs). For further details see Yee and Hastie (2003) and the VGAM documentation for fitting RR-VGLMs.

2 Generalized Additive Models

GAMs are a nonparametric extension of GLMs, and provide a powerful data-driven class of models for exploratory data analysis. GAMs extend (1) to

$$g(\mu(\boldsymbol{x}_i)) \;=\; \eta_i \;=\; \beta_0 + f_1(x_{i1}) + \cdots + f_p(x_{ip}), \tag{22}$$

a sum of smooth functions of the individual covariates. For identifiability the f_k are centered.

The S-PLUS `gam()` function offers two smoothers for estimating the f_k: cubic smoothing splines via `s()` and local regression (LOESS) via `lo()`. In contrast, `vgam()` only offers `s()` for vector smoothing splines, which are a natural extension of cubic smoothing splines. Plotting the estimated f_k's versus x_k is often revealing. GAMs can be justified by a penalized likelihood argument (see, e.g., Hastie and Tibshirani (1990)[pp.149–151]; Green and Silverman (1994)).

Internally, `vgam()` is built on the same FORTRAN code that `gam()` is built on, viz., LINPACK for numerical linear algebra, F. O'Sullivan's BART for spline smoothing, and C. de Boor's routines for B-splines. The vector backfitting algorithm is a vector extension of the ordinary backfitting algorithm used by `gam()`. Actually, modified vector backfitting is used, which is where the linear components are firstly estimated and then a vector additive model is fitted to the partial residuals to estimate the nonlinear components.

VGAM's `s()` function accepts arguments `df` and `spar` to regulate the amount of smoothness. However, `df` ≥ 1 only is allowed. A value of unity corresponds to a linear fit.

3 Implementational Details

This section describes some internal details inside VGAM. It should not be of interest to most users.

- Initial values for IRLS are chosen by the specific family functions in `@initialize`. They rarely fail for GLMs, however if they do, users may input values into one of the `coefstart=`, `etastart=`, and `mustart=` arguments of `vglm()` and `vgam()`.
- The S-PLUS version uses backchatting to increase efficiency. This is an undocumented feature. In the R version, the function `vlm.wfit()` is repeatedly called to fit the weighted least-squares regressions. It is the workhorse of `vlm()` .

3.1 Not Working Yet

The following lists things yet to be done

1. The link `powl()` for power isn't implemented for `quasiff()`.

4 Specific Details About Certain Models

Before giving details about specific models, here are some general notes.

- VGAM handles the estimation of the dispersion parameter differently from S-PLUS and R. Each GLM family function in VGAM has a `dispersion` argument. For example, `binomialff(..., dispersion=1)`. If `dispersion=0` then it is estimated using (10) in `@last`, else users may input a numerical value for `dispersion`, e.g., `binomialff(..., dispersion=2)`. By default, the binomial and Poisson models have `dispersion=1`, and `dispersion=0` for the other models.

- To cater for hardened R users, `quasibinomialff()` and `quasipoissonff()` have been written. These differ in that the dispersion parameter is unknown and to be estimated. Note that the call
  `quasibinomialff(...)` is equivalent to `binomialff(..., dispersion=0)`, and similarly, `quasipoissonff(...)` is equivalent to `poissonff(..., dispersion=0)`. All this is done to mimic the R functions `quasibinomial()` and `quasipoisson()` for which ϕ is always estimated, and `binomial()` and `poisson()` never allow ϕ to be estimated.

- In contrast to `glm()` and `gam()`, any estimated dispersion parameter is computed at fitting time (`glm()`, `gam()`, `rrvglm`) and not in `summary()`. It is placed in `object@misc$dispersion`, and is used by `summary()` and `predict(..., se.fit=T)`. Also, `object@misc$default.dispersion` stores the default value of the family function's dispersion parameter, and a value of zero indicates it is estimated by default. Finally, `object@misc$estimated.dispersion` is a logical indicating whether the dispersion parameter was a value given by the user or estimated—this is needed when computing P-values for nonlinearity in `vgam()` objects.

Most of the following distributions can also be fitted by VGAM using other family functions. Often, these other family functions estimate the parameters by full maximum likelihood estimation, which has the advantage that standard errors are available. Also, many of following distributions have been extended to make them more useful or general.

4.1 Gaussian

To solve the Gaussian case by full maximum likelihood estimation, i.e., $\eta_1 = \mu$ and $\eta_2 = \log \sigma$. This related to the MADAM model of Rigby and Stasinopoulpos (1996).

4.2 Binomial

The argument `mv` specifies whether the response is multivariate. The default is `FALSE`, meaning a single binary response. In this case, the input can take one of three formats:

1. a factor (first level taken as success),

2. a vector of proportions of success. If these lie in $(0, 1)$ then it is necessary to specify `weights` (see below),

3. a 2-column matrix of counts (first column has the number of successes) .

In general, the argument `weights` in the modelling function can or must be also be specified. In particular, for a general vector of proportions, you will need to specify `weights` because the number of trials is needed.

If `mv` is `TRUE`, then the M-column response matrix must consist of 1's and 0's only.

Models for more than two levels of a response factor are handled more generally with functions such as `cumulative`, `multinomial`, `acat` etc. See the separate documentation regarding categorical data.

Two popular models for handling over-dispersion are the negative binomial (`neg.binomial()`) and beta binomial (`rbinomial.beta()`). The latter doesn't work satisfactorily. Also, there is the `abbott()` family function, but this is not finished yet. A VGAM family function has been written for the problem where n is unknown and p is known. It is not an easy problem to solve because of numerical difficulties. Lastly, `positive.binomial()` is available for the positive binomial model.

4.3 Poisson

The VGAM family function `zipoisson()` implements the zero-inflated Poisson distribution; see Thas and Rayner (2005). See also `pospoisson()` for the positive Poisson model where $P(Y = 0)$ is zero and the other probabilities rescaled. One generalization of the Poisson distribution is the *generalized Poisson* distribution; see the Exercises.

4.4 Gamma

When $k = 1$ this corresponds to the exponential distribution. A VGAM family function has been written for this (`exponential()`), as well as for the two parameter gamma distribution (called `gamma2()`). These estimate all parameters by full maximum likelihood estimation.

Note that, from Tables 1 and 2, that the MLE of the mean of a Gamma GLM does not depend on the shape parameter k, so `glm()` fits the MLE of μ for any fixed shape. An estimate of the reciprocal of the shape parameter can be found in `vglmobject@misc$dispersion`, for example,

```
y = rgamma(n=100, shape=5)
v = vglm(y ~ 1, gammaff)
```

should result in `v@misc$dispersion` near $1/5 = 0.2$. While this estimate isn't the MLE, as argued by McCullagh and Nelder (1989)[Section 8.3], it is preferable to the MLE as the latter is not robust. If you do want the MLE it is very easy to calculate it or approximate it from the deviance, which is also supplied. For multivariate responses `vglmobject@misc$dispersion` is computed for each response so is a vector of the appropriate length.

4.5 Inverse Gaussian

S-PLUS and R use $g(\mu) = 1/\mu^2$ as their default link, whereas VGAM uses $g(\mu) = -1/(2\mu^2)$ because it is the theoretical canonical link. Consequently, `coef(glmobject) + 2*coef(vglmobject)` is zero in theory. However, `summary(glmobject)` and `summary(vglmobject)` have equal standard errors ... this is a bug!

5 Summary of Differences

Here is a brief summary of the main differences between `vglm()` and `vgam()` and/or `glm()` and `gam()` for GLM and GAM models.

1. Table 1 summarized the GLM models currently implemented by VGAM. Family functions for `glm()` and `gam()` do *not* work with `vglm()` and `vgam()`. In general, GLM-type VGAM family functions differ from existing family functions only by beginning with an uppercase letter, except for `gammaff()`.

2. Example: `binomialff(link=probit)` and `binomialff(link="probit")` are ok, but `binomialff(link=pr)` will not work, neither will `binomialff(link="pr")` work.

3. `vgam()` supports `s()` but not `lo()`. The argument `df` must be assigned a value ≥ 1 (1 for linear).

4. Each GLM family function in VGAM has a `dispersion` argument. For example, `binomialff(..., dispersion=1)`. If `dispersion=0` then it is estimated using (10), else users may input a numerical value for `dispersion`. In contrast to `glm()` and `gam()`, the dispersion parameter is computed at fitting time (`glm()`, `gam()`) and not in `summary()`. It is placed in `object@misc$dispersion`, and is used by `summary()` and `predict(..., se.fit=T)`.

5. `vcov(object)` which returns the estimated variance-covariance matrix of $\hat{\boldsymbol{\beta}}$ (Equation (6)).

6. Owing mainly to differences in smoothing parameter selection and choice of knots, results from `gam()` and `vgam()` usually differ slightly (`vgam()` usually chooses fewer knots than `gam()`).

7. More generally, VGAM has additional features, e.g., `plot.vgam()` has the `deriv` argument for plotting the derivatives of the fitted smooths.

8. S-PLUS implements "safe prediction" whereas VGAM implements "smart prediction". The latter is documented in a website that can be accessed starting from the webpage given at the beginning of this document. In brief, users needed to type `predict.gam(fit)` if `fit` was a `"lm"`, `"glm"` or `"gam"` object. Smart prediction is simpler in that `predict(fit)` ought to be used for `"vglm"` and `"vgam"` objects, and furthermore, it is more efficient and true to the original fit. Smart prediction requires data-dependent functions such as `poly()`, `bs()`, `ns()`, to be *smart functions*. See the documentation for advantages, examples and provisos.

9. Initial values may be set using the arguments `coefstart`, `etastart`, and `mustart`, which are available in `vglm()` and `vgam()`.

10. There is no `robust()` or equivalent. Currently not to be used on `vglm()` or `vgam()` objects: `anova()`, `addterm()`, `drop()`, `dropterm()`, `effects()`, `step1()`, `stepAIC()`.

11. The canonical links in Table 1 are default. Note that these differ by a factor of -1 and $-\frac{1}{2}$ for gamma and inverse Gaussian models respectively.

12. The null model (intercept only) of a model is not computed in VGAM.

13. There is the `powl()` link. Its power is always inputted into the argument `exponent`. But it doesn't work yet.

14. Although `vgam()` objects are computed in a similar manner to `gam()` objects, their components are quite different. See the VGAM *User Manual* for details.

5.1 How VGAM Handles Dispersion Parameters

Suppose `famfun()` is a VGAM family function that is used by `vglm()` or `vgam()` to produce `object`. Also, let `nonzero` be a nonzero numeric variable. The following are general VGAM conventions.

1. If `famfun(dispersion=0)` is used, then `object@misc$dispersion` may be assigned by the family function a dispersion parameter $\hat{\phi}$ estimated at fitting time. Any formula may be used to compute $\hat{\phi}$. If not calculated by `famfun()`, then `object@misc$dispersion` is assigned 0, in which case it is computed by `summary()` using (23).

2. If `famfun(dispersion=nonzero)` is used, then `object@misc$dispersion` is assigned `nonzero`.

3. A family function without a `dispersion` argument has `object@misc$dispersion` equalling `NULL`.

4. `summary(..., dispersion=NULL)` is default. This means

   (a) use $\hat{\phi}$ = `object@misc$dispersion` if it is numeric and nonzero,

   (b) use $\hat{\phi} = 1$ if `object@misc$dispersion` is `NULL`,

   (c) estimate ϕ by (23) if `object@misc$dispersion` is 0.

5. If `summary(..., dispersion=0)` or `object@misc$dispersion==0` then use

$$\hat{\phi} \;=\; \hat{\sigma}^2 = RSS/(nM-p), \tag{23}$$

   where *RSS* is the residual sum of squares of the vector linear model. This formula may possibly be the same formula as the one used in `famfun()`, but usually it is different.

6. `summary(..., dispersion=nonzero)` means use `nonzero` as $\hat{\phi}$. This overrides any numerical value of `object@misc$dispersion`.

7. `predict(..., dispersion=NULL)` is default. The `dispersion` argument is passed onto `summary()`, and the conventions follow.

The above conventions may seem complicated but they are actually straightforward in practice. The only problem is if the `dispersion` argument is assigned a nonzero value in `summary()`, then it must be assigned the same value in `predict()` and `vcov()` too in order to give consistent values. Note that `summary()@dispersion==summary()@sigma`^{2}.

6 Dispersion Parameters for Matrix Responses

When `mv` is `TRUE` for Poisson and binomial models, there will be M dispersion parameters; one for each response. At the moment, the generic functions `summary()` and `predict` cannot handle these cases. One would like the option of allowing for a single dispersion parameter, especially if `parallel = TRUE`, say. Possibly, using something like `binomialff(dispersion=rep(0, ncol(y)))` might be necessary.

Computing multiple dispersion parameters would have to be done within the family function because it would be too difficult for `summary.vglm()` or `summary.vlm()` to do so.

7 Tutorial Examples

In this section we illustrate using the software to fit some GLMs and GAMs etc.

7.1 Kyphosis Data

Hastie and Tibshirani (1990) describe a kyphosis data set. It is available online in S-PLUS. The commands

```
> library(rpart)
> data(kyphosis)
> kyphosis = kyphosis
> detach("package:rpart")
> v = vgam(Kyphosis ~ s(Age, 2) + s(Start), binomialff, kyphosis)
> summary(v)

Call:
vgam(formula = Kyphosis ~ s(Age, 2) + s(Start), family = binomialff,
    data = kyphosis)

Number of linear predictors:    1

Name of linear predictor: logit(mu)

(Default) Dispersion Parameter for binomialff family:   1

Residual Deviance:  50.24194 on 74.062 degrees of freedom

Log-likelihood: -25.12097 on 74.062 degrees of freedom

Number of Iterations:  8

DF for Terms and Approximate Chi-squares for Nonparametric Effects

            Df Npar Df Npar Chisq   P(Chi)
(Intercept)  1
s(Age, 2)    1       1     5.4607 0.018360
s(Start)     1       3     7.3882 0.059277
```

Then Figure 1 was produced by

```
> par(mfrow = c(2, 1))
> plot(v, se = TRUE)
```

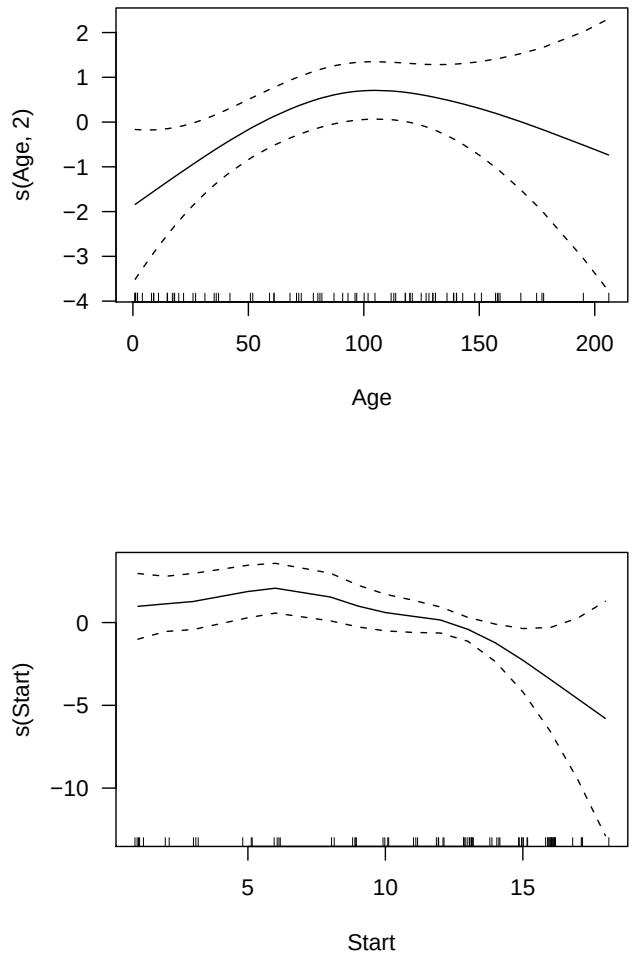

Figure 1: GAM fitted to kyphosis data (v) with standard errors: fitted functions.

Then Figure 2 was produced by

```
> par(mfrow = c(2, 1))
> plot(v, deriv = 1)
```

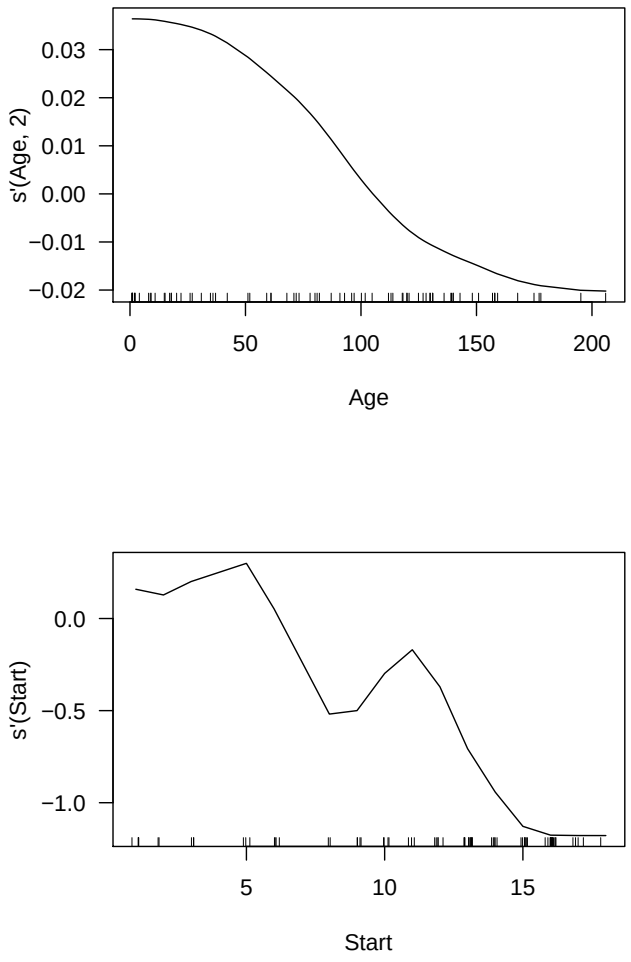

Figure 2: First derivatives of `v`.

There is some evidence of nonlinearity in the second function. In light of the standard error bands, the second function looks like it could be replaced by a quadratic. This is confirmed by the plot of the first derivatives, which looks a little rough. Rather than fitting a quadratic, let's try fitting a model called `v1`. This produces Figure 3.

```
> par(mfrow = c(2, 1))
> v1 = vgam(Kyphosis ~ s(Age, 2) + s(Start, 2), binomialff,
+     kyphosis)
> plot(v1, se = TRUE, lcol = "blue", scol = "darkgreen")
```

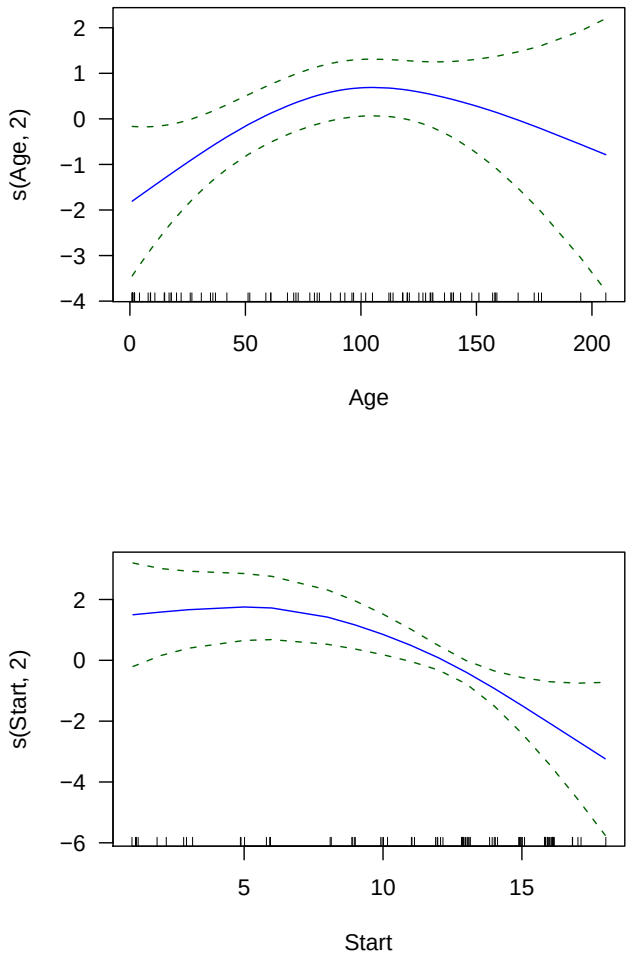

Figure 3: GAM fitted to kyphosis data (`v1`): fitted functions.

Finally this produces Figure 4.

```
> par(mfrow = c(2, 1))
> plot(v1, deriv = 1, lcol = "blue")
```

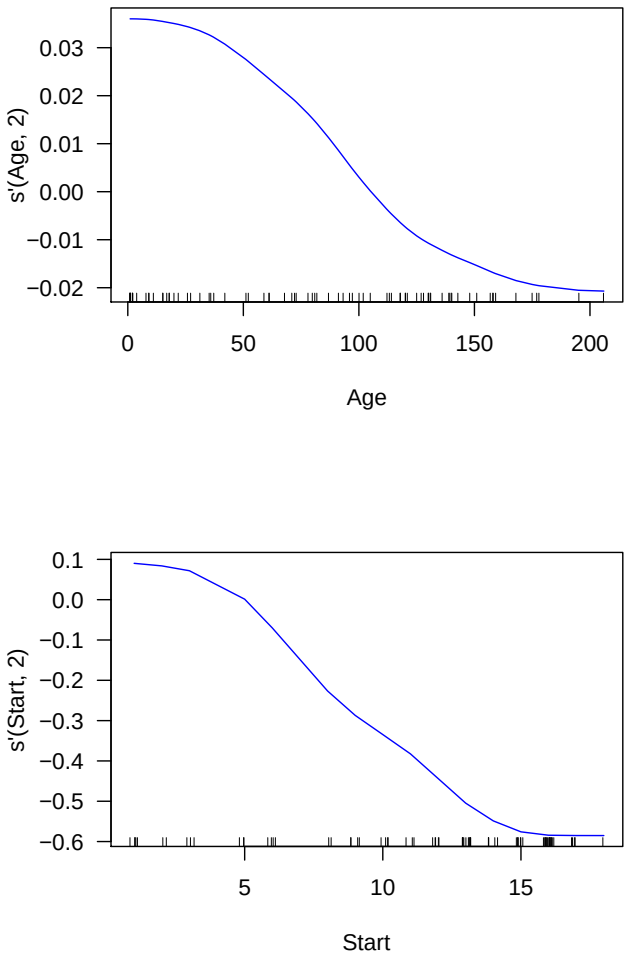

Figure 4: GAM fitted to kyphosis data (`v1`): first derivatives.

To fit a quadratic model and to test "smart prediction", one can use

```
> v2 = vgam(Kyphosis ~ s(Age, 2) + poly(Start, 2), binomialff,
+     kyphosis)
> small = kyphosis[1:5, ]
> (pred = predict(v2, small, type = "response"))

[1] 0.693425638 0.072327346 0.736954850 0.086578389 0.004539825

> fitted(v2)[1:5]

[1] 0.693425638 0.072327346 0.736954850 0.086578389 0.004539825
```

They are the same! Lastly, a fully parametric model is

```
> v3 = vglm(Kyphosis ~ poly(Age, 2) + poly(Start, 2), binomialff,
+     kyphosis)
> vcov(v3)

               (Intercept) poly(Age, 2)1 poly(Age, 2)2 poly(Start, 2)1
(Intercept)      0.4112355    -0.6657321     1.7370642        1.916741
poly(Age, 2)1   -0.6657321    18.7191651    -0.1694668       -3.639702
poly(Age, 2)2    1.7370642    -0.1694668    27.5987413        1.974187
poly(Start, 2)1  1.9167409    -3.6397024     1.9741866       19.646441
poly(Start, 2)2  1.4784134    -2.0086512     3.2776372       11.021943
               poly(Start, 2)2
(Intercept)           1.478413
poly(Age, 2)1        -2.008651
poly(Age, 2)2         3.277637
poly(Start, 2)1      11.021943
poly(Start, 2)2      14.889030
```

7.2 Double Exponential Binomial Example

Here we mimic the results of Efron (1986) relating to some toxoplasmosis data in 34 cities in El Salvador. The response is the proportions of subjects testing positive for the disease toxoplasmosis. Of interest is the effect of rainfall on the response.

In the following, equation numbers etc. refer to those in that paper, and the results here differ slightly. First we read in the data and scale the variables.

```
> data(toxop)
> toxop = transform(toxop, phat = positive/ssize, srainfall = scale(rainfall),
+     sN = scale(ssize))
```

The fit in Section 6 of Efron (1986) should be as below. In order to get the same coefficients we initially refrain from using `poly()` and manually create powers of the (scaled) rainfall variable. We use $M = 1.25$ here, as in (5.3).

```
> cmlist = list("(Intercept)" = diag(2), "I(srainfall)" = rbind(1,
+     0), "I(srainfall^2)" = rbind(1, 0), "I(srainfall^3)" = rbind(1,
+     0), "I(sN)" = rbind(0, 1), "I(sN^2)" = rbind(0, 1))
> dlist = list(min = 0, max = 1.25)
> fit = vglm(phat ~ I(srainfall) + I(srainfall^2) + I(srainfall^3) +
+     I(sN) + I(sN^2), fam = dexpbinomial(ldisp = "elogit",
+     idisp = 0.2, edisp = dlist, zero = NULL), data = toxop,
+     weight = ssize, trace = FALSE, constraints = cmlist)
```

Now look at the results.

```
> coef(fit)

 (Intercept):1  (Intercept):2   I(srainfall) I(srainfall^2)
    -0.0871606      0.6298892     -0.6863776     -0.1672244
I(srainfall^3)          I(sN)        I(sN^2)
     0.2765311      0.4528456     -0.5870306

> round(coef(fit, matrix = TRUE), dig = 2)

               logit(mu) elogit(dispersion, earg=list(min=0, max=1.25))
(Intercept)        -0.09                                           0.63
I(srainfall)       -0.69                                           0.00
I(srainfall^2)     -0.17                                           0.00
I(srainfall^3)      0.28                                           0.00
I(sN)               0.00                                           0.45
I(sN^2)             0.00                                          -0.59
```

This can be compared to (6.5) and (6.6).

The fitted means $\widehat{\mu}_i$ are (cf. Table 4)

```
> fitted(fit)[1:4, ]

[1] 0.5406688 0.4501046 0.3886074 0.4132617
```

More output is

```
> summary(fit)

Call:
vglm(formula = phat ~ I(srainfall) + I(srainfall^2) + I(srainfall^3) +
    I(sN) + I(sN^2), family = dexpbinomial(ldisp = "elogit",
    idisp = 0.2, edisp = dlist, zero = NULL), data = toxop, weights = ssize,
    constraints = cmlist, trace = FALSE)

Pearson Residuals:
                                                    Min       1Q
logit(mu)                                       -2.3021 -0.67997
elogit(dispersion, earg=list(min=0, max=1.25)) -3.2560 -0.23071
                                                  Median      3Q     Max
logit(mu)                                       -0.13975 0.60399 2.35202
elogit(dispersion, earg=list(min=0, max=1.25))  0.29820 0.63697 0.70649

Coefficients:
                  Value Std. Error  t value
(Intercept):1  -0.08716   0.141732 -0.61497
(Intercept):2   0.62989   0.872360  0.72205
I(srainfall)   -0.68638   0.232207 -2.95589
I(srainfall^2) -0.16722   0.109301 -1.52994
I(srainfall^3)  0.27653   0.086433  3.19937
I(sN)           0.45285   1.039671  0.43557
I(sN^2)        -0.58703   0.534191 -1.09892

Number of linear predictors:  2

Names of linear predictors:
logit(mu), elogit(dispersion, earg=list(min=0, max=1.25))

Dispersion Parameter for dexpbinomial family:   1

Log-likelihood: -471.5745 on 61 degrees of freedom

Number of Iterations: 12

> round(vcov(fit), dig = 3)

               (Intercept):1 (Intercept):2 I(srainfall) I(srainfall^2)
(Intercept):1          0.020         0.000        0.002         -0.009
(Intercept):2          0.000         0.761        0.000          0.000
I(srainfall)           0.002         0.000        0.054          0.009
I(srainfall^2)        -0.009         0.000        0.009          0.012
I(srainfall^3)         0.001         0.000       -0.017         -0.006
I(sN)                  0.000         0.633        0.000          0.000
I(sN^2)                0.000        -0.376        0.000          0.000
               I(srainfall^3)  I(sN) I(sN^2)
(Intercept):1           0.001 0.000   0.000
(Intercept):2           0.000 0.633  -0.376
```

```
I(srainfall)           -0.017  0.000   0.000
I(srainfall^2)         -0.006  0.000   0.000
I(srainfall^3)          0.007  0.000   0.000
I(sN)                   0.000  1.081  -0.523
I(sN^2)                 0.000 -0.523   0.285
```

```
> sqrt(diag(vcov(fit)))
```

```
 (Intercept):1  (Intercept):2   I(srainfall) I(srainfall^2)
    0.14173223     0.87235988     0.23220660     0.10930138
I(srainfall^3)          I(sN)        I(sN^2)
    0.08643295     1.03967108     0.53419106
```

The effective sample size (not quite the last column of Table 1) is given by

```
> predict(fit)[1:4, ]
```

```
   logit(mu) elogit(dispersion, earg=list(min=0, max=1.25))
1  0.1630352                                      -0.008424175
2 -0.2002479                                       0.295309555
3 -0.4531696                                       0.047882222
4 -0.3504979                                       0.295309555
```

```
> Dispersion = elogit(predict(fit)[, 2], earg = dlist, inverse = TRUE)
> c(round(weights(fit, type = "prior") * Dispersion, dig = 1))
```

```
 [1]  2.5  7.2  3.2  7.2  1.2  3.2  5.5 15.3  3.9  7.2 20.0  0.6 25.2
[14] 18.1  0.6  8.0  0.6 34.1  6.3 14.4  8.9  0.6  8.0 12.8 34.7 12.5
[27]  8.2  9.8 33.5 15.0  9.8  7.2  3.9 30.4
```

Here, the inverse of the extended logit link is used to compute the dispersion parameter.

The ordinary logistic regression (gives same results as (6.5)) is

```
> ofit = vglm(phat ~ I(srainfall) + I(srainfall^2) + I(srainfall^3),
+     fam = binomialff, data = toxop, weight = ssize)
> ofit
```

```
Call:
vglm(formula = phat ~ I(srainfall) + I(srainfall^2) + I(srainfall^3),
    family = binomialff, data = toxop, weights = ssize)

Coefficients:
   (Intercept)   I(srainfall) I(srainfall^2) I(srainfall^3)
    0.09939316    -0.44846047    -0.18726958     0.21342057

Degrees of Freedom: 34 Total; 30 Residual
Residual Deviance: 62.6346
Log-likelihood: -477.1735
```

Now we obtain the same as `fit` but using `poly()`—this can be plotted (cf. Figure 1)

```
> cmlist2 = list("(Intercept)" = diag(2), "poly(srainfall, 3)" = rbind(1,
+     0), "poly(sN, 2)" = rbind(0, 1))
> fit2 = vglm(phat ~ poly(srainfall, 3) + poly(sN, 2), fam = dexpbinomial(ldisp = "elogi
+     idisp = 0.2, edisp = dlist, zero = NULL), data = toxop,
+     weight = ssize, trace = FALSE, constraints = cmlist2)
```

Hence Figure 1 is mimicked by

```
> par(mfrow = c(1, 2))
> o = with(toxop, sort.list(rainfall))
> with(toxop, plot(rainfall[o], fitted(fit2)[o], type = "l",
+     col = "blue", las = 1, ylim = c(0.3, 0.65), main = "(a)"))
> with(toxop, points(rainfall[o], fitted(ofit)[o], col = "darkgreen",
+     type = "b", pch = 19, main = "(b)"))
> o = with(toxop, sort.list(ssize))
> with(toxop, plot(ssize[o], Dispersion[o], type = "l", col = "blue",
+     las = 1, xlim = c(0, 100)))
```

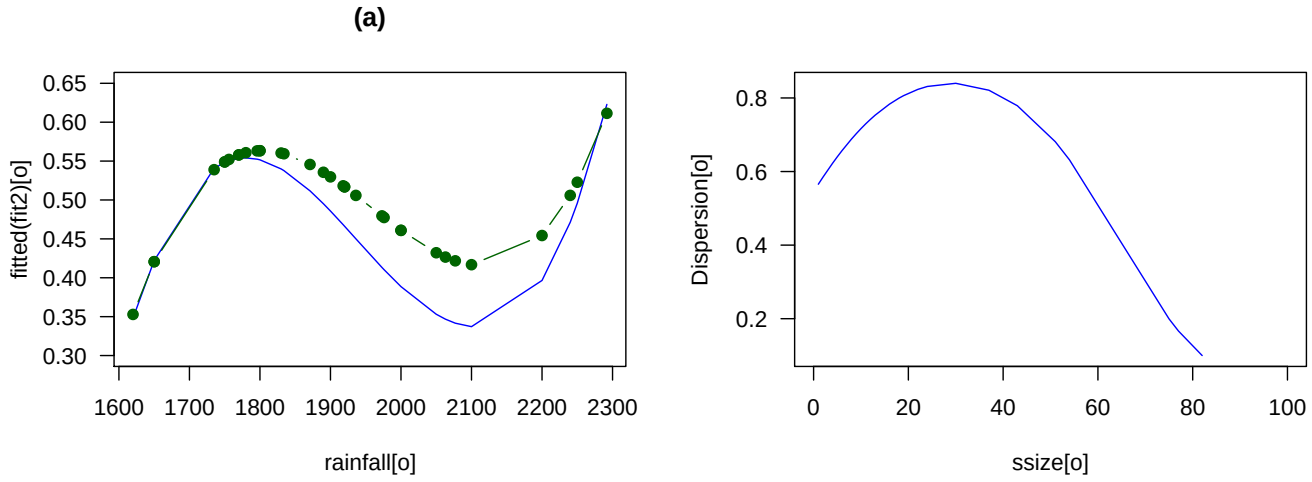

Figure 5: A mimic of Figure 1 of Efron (1986).

The two additive/linear predictors are given by

```
> par(mfrow = c(1, 2))
> plotvgam(fit2, se = TRUE, lcol = "blue", scol = "darkgreen")
```

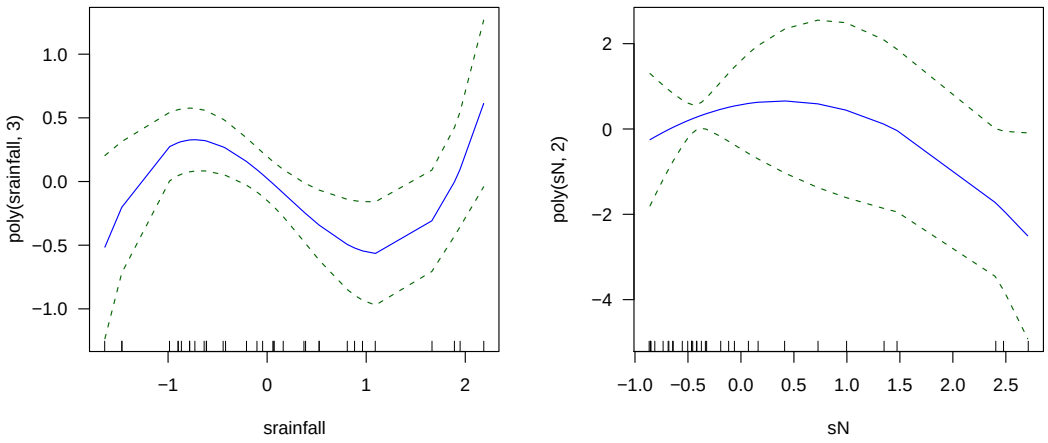

Figure 6: Estimated component functions of `fit2`.

It is left to the reader to fit a GAM to these data and compare the results with this plot.

8 Matched Case-Control Studies

This section is slightly off the beaten path but is included here because I haven't put it in the right place yet.

In a *matched case-control* study, each case ($Y = 1$) is matched with one or more controls ($Y = 0$) having the same or similar values of some set of potential confounding *(matching) variables*. For example, a 60 year old female might be matched with another 60 year old female. Comparison between people from the same *matched set* helps to control for the potential confounders (age and gender, here). One advantage of a matched case-control study is the ability to estimate the effect of other risk factors $\boldsymbol{x}^*$, which are of interest, after eliminating the effects of the confounders. A disadvantage is that one cannot explore very well the effect of the potential confounding variables—interactions between $\boldsymbol{x}^*$ and the matching variables only allow a limited picture. Another disadvantage is that the results may not be generalizable to a population since the matches may not form a random sample of any population.

Consider a matched case-control study with C matched sets. The sets are determined by values of matching variables, $\boldsymbol{m}$, there being C distinct values $\boldsymbol{m}_1, \ldots, \boldsymbol{m}_C$. Examples of matching variables in $\boldsymbol{m}_c$ are age, gender, geographic locality and race. In general, there can be a variable number of cases and controls in each set. A $1 : c_0$ design has c_0 controls matched for every case.

It is common to fit the logistic regression model

$$\text{logit } P(Y = 1|\boldsymbol{x}^*, \boldsymbol{m}_c) \;=\; \text{logit } p_c(\boldsymbol{x}^*) \;=\; \beta_{(c)1} + \boldsymbol{\beta}^T \boldsymbol{x}^*. \tag{24}$$

Here, $\beta_{(c)1}$ is an intercept term specific to the cth matched set. It represents the effect of $\boldsymbol{m}_c$ within each matched set. The parameters of interest, $\boldsymbol{\beta}$, are assumed to be constant across matched sets. Then the odds of an individual with exposure $\boldsymbol{x}^*$ developing the disease relative to an individual with baseline exposure $\boldsymbol{x}_0$ is

$$\psi(\boldsymbol{x}^*, \boldsymbol{x}_0) \;=\; \frac{p_c(\boldsymbol{x}^*)/\{1 - p_c(\boldsymbol{x}^*)\}}{p_c(\boldsymbol{x}_0)/\{1 - p_c(\boldsymbol{x}_0)\}} \;=\; \exp\{\boldsymbol{\beta}^T(\boldsymbol{x}^* - \boldsymbol{x}_0)\} \tag{25}$$

because the $\beta_{(c)1}$ cancel out. The odds ratio (25) is interpreted much like a relative risk.

The VGAM framework can handle this model, (24), using a trick or two. Firstly one has $M = C$ so that $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_C)^T$. We will enumerate all the response data as y_i for $i = 1, \ldots, n$ as usual, assume $y_i = 1$ or 0. The LM model matrix $\mathbf{X}$ has $C(c_0 + c_1)$ rows, the first column equal to $\mathbf{1}$, followed by $C - 1$ columns for `mfactor`, then p columns for $\boldsymbol{x}^*$. Hence we set up constraint matrices so that $\mathbf{H}_1 = \mathbf{1}_M$, $\mathbf{H}_c = \boldsymbol{e}_c$ for $c = 2, \ldots, C$, and $\mathbf{H}_{C+k} = \mathbf{1}_M$ for $k = 1, \ldots, p$. That is, the constraint matrix for the intercept term allows each matched set to have its own intercept, and the $C - 1$ columns due to the matching variable gives rise to $\boldsymbol{e}_c$ constraint matrices. Finally, there is a parallelism assumption made for the variables in $\boldsymbol{x}^*$. Then

$$\boldsymbol{\eta} \;=\; \begin{pmatrix} \alpha^* + \boldsymbol{\beta}^T \boldsymbol{x}^* \\ \alpha^* + \alpha_2^* + \boldsymbol{\beta}^T \boldsymbol{x}^* \\ \vdots \\ \alpha^* + \alpha_C^* + \boldsymbol{\beta}^T \boldsymbol{x}^* \end{pmatrix}, \tag{26}$$

which is the default in VGAM. This is a slightly different parameterization of (24). In VGAM the usage is something like

```
fit = vglm(y ~ mfactor + x2 + x3, mbinomial(mvar = ~ -1 + mfactor), mydata)
```

The -1 means to suppress the default intercept from the formula so that argument mvar only has mfactor as its single term. The variable mfactor is a factor with C levels; it tells what matched set each observation is in. The variables x2 and x3 denote the variables comprising $\boldsymbol{x}^*$ which are of interest. The VGAM family function mbinomial() acts in a similar way to binomialff().

8.1 Technical details

Since each y_i only affects one η_j we create a prior weight of ε for the other η_j so as to make its effect negligible. Here, ε is some very small positive number, e.g., 10^{-12} since most computers have double precision accuracy $1.0 + E = 1.0$ for approximately $|E| < 10^{-16}$.

The EIM for observation i is of the form $\mathrm{diag}(\varepsilon, \ldots, \varepsilon, \mu_{ic}(1-\mu_{ic}), \varepsilon, \ldots, \varepsilon)$ where the $\mu_{ic}(1-\mu_{ic})$ is in the c-c position. This effectively means that if y_i belongs to the cth matched set then it only contributes to the cth element of $\boldsymbol{\eta}$.

Although elegant, there are disadvantages of this algorithm. The biggest one is that *all* C intercepts are estimated. These are nuisance parameters and the cost is a lot of memory and computational time. In particular, both the memory requirements and computational time of this algorithm grows very quickly with respect to the number of matched sets, C. For example, the large model matrix of a data set with 100 matched sets consisting of one case and one control per set will take up at least (about) 20Mb of memory. Suppose there are C matched sets, each of which consists of c_1 cases and c_0 controls. Then the LM $\mathbf{X}$ is $(c_1+c_0)C \times (C+p^*)$ in dimension. And the big VLM $\mathbf{X}$ is $(c_1+c_0)C^2 \times (C+p^*)$ in dimension. This storage requirement is $O(C^3)$. For $1:1$ matching this is $2C^2 \times (C+p^*)$. Since the QR algorithm for a $m \times n$ matrix takes $O(2mn^2)$ flops, this means the computational time includes $O(2(c_1+c_0)C^2(C+p^*)^2)$ flops. With $1:1$ matching this is $O(4C^4)$ flops which grows very quickly in C.

Section 8.2 of Hastie and Tibshirani (1990) give details on how to fit the additive model extension of (24), viz.,

$$\text{logit } p_c(\boldsymbol{x}^*) \;=\; \beta_{(c)1} + \sum_{k=2}^{p} f_{(1)k}(x_k), \tag{27}$$

however their framework is smaller and so their algorithm is more complicated. It is easier within the VGAM framework and can easily be fitted using vgam(). The usage is something like

```
fit = vgam(y ~ mfactor + s(x2) + s(x3),
           mbinomial(mvar = ~ -1 + mfactor), mydata)
```

For more about matched case-control studies see Section 8.2 of Hastie and Tibshirani (1990), Section 2.12 of Faraway (2006) and Pregibon (1984). A detailed account is Chapter 7 of Breslow and Day (1980). Also, for matched pairs, one can fit an ordinary logistic regression with covariates $\boldsymbol{x}_1 - \boldsymbol{x}_0$ (see Holford et al. (1978)).

Yet to be done: see Hastie and Pregibon (1987) and Hastie and Pregibon (1988). They discuss $\mathbf{W}_i$ are approximated by its diagonal and generalized inverses. Also handling ordinal Y would be great.

8.2 Example

We use simulation to illustrate mbinomial(). First create the data:

```
> set.seed(123)
> n = 200
```

```
> mydat = data.frame(x2 = rnorm(n), x3 = rep(rnorm(n/2),
+     each = 2))
> mydat = transform(mydat, eta = -0.1 + 0.2 * x2 + 0.3 *
+     x3)
> etamat = with(mydat, matrix(eta, n/2, 2))
> condmu = exp(etamat[, 1])/(exp(etamat[, 1]) + exp(etamat[,
+     2]))
```

Here, $C = n/2 =$ 100, and the variable x3 is a confounder (the matching variable). We want to estimate the regression coefficient of x2. The variable condmu contains the conditional probabilities that within the matched set, the assignment of the c_0 controls is as observed is

$$\mu_{0c} \;=\; \frac{\exp(\boldsymbol{x}^T\boldsymbol{\beta})}{\sum_{M_c}\exp(\boldsymbol{x}^T\boldsymbol{\beta})}.$$

Given the conditional probabilities we can now generate case and control.

```
> y1 = ifelse(runif(n/2) < condmu, 1, 0)
> mydat = transform(mydat, y = c(y1, 1 - y1), ID = factor(c(row(etamat))))
```

The variable ID specifies the matched sets. Now fit the model

```
> fit = vglm(y ~ 1 + ID + x2, trace = TRUE, fam = mbinomial(mvar = ~ID -
+     1), data = mydat)

VGLM    linear loop  1 :  loglikelihood = -133.3849
VGLM    linear loop  2 :  loglikelihood = -133.2843
VGLM    linear loop  3 :  loglikelihood = -133.2842
VGLM    linear loop  4 :  loglikelihood = -133.2842
```

There are many coefficients, and these can be seen with

```
> print(summary(fit), presid = FALSE)

Call:
vglm(formula = y ~ 1 + ID + x2, family = mbinomial(mvar = ~ID -
    1), data = mydat, trace = TRUE)

Coefficients:
                Value Std. Error    t value
(Intercept)  0.448976    1.42196  0.3157437
ID2         -0.458411    2.00940 -0.2281331
ID3         -0.912485    2.07287 -0.4402029
ID4         -0.351106    2.00622 -0.1750088
ID5         -0.158464    2.01935 -0.0784728
ID6         -1.038966    2.07635 -0.5003802
ID7         -0.334518    2.02762 -0.1649804
ID8          0.587193    2.01165  0.2918956
ID9         -0.071999    2.00195 -0.0359646
ID10        -0.616196    2.03928 -0.3021639
ID11        -0.678161    2.06333 -0.3286738
ID12        -0.790873    2.01720 -0.3920639
```

ID13	-0.018996	2.06560	-0.0091965
ID14	-0.468450	2.00637	-0.2334808
ID15	-0.436105	2.02332	-0.2155389
ID16	-1.186647	2.07029	-0.5731781
ID17	-0.662190	2.01390	-0.3288101
ID18	0.472137	2.03369	0.2321580
ID19	-0.396568	2.04240	-0.1941680
ID20	0.079855	2.00526	0.0398225
ID21	-0.113299	2.02280	-0.0560109
ID22	-0.037247	2.00872	-0.0185428
ID23	0.086794	2.00502	0.0432881
ID24	-0.101002	2.00410	-0.0503977
ID25	-0.879561	2.11790	-0.4152989
ID26	0.377218	2.02077	0.1866708
ID27	-0.828107	2.02340	-0.4092648
ID28	-0.530702	2.00761	-0.2643447
ID29	0.292908	2.00302	0.1462329
ID30	-0.866732	2.04687	-0.4234416
ID31	-1.109968	2.04758	-0.5420882
ID32	-0.504241	2.01553	-0.2501783
ID33	-0.779773	2.02715	-0.3846648
ID34	-0.609944	2.03637	-0.2995243
ID35	-0.013854	2.13613	-0.0064857
ID36	-1.091937	2.03357	-0.5369565
ID37	-0.128650	2.06573	-0.0622784
ID38	-0.688513	2.02242	-0.3404394
ID39	-1.015333	2.10447	-0.4824655
ID40	0.195534	2.01909	0.0968427
ID41	-0.451477	2.03627	-0.2217174
ID42	-0.282895	2.00244	-0.1412751
ID43	0.553468	2.00962	0.2754099
ID44	-0.680123	2.24094	-0.3034984
ID45	-0.309935	2.13203	-0.1453706
ID46	0.135353	2.00630	0.0674640
ID47	0.209763	2.01908	0.1038902
ID48	-0.527144	2.02833	-0.2598905
ID49	-1.466447	2.08128	-0.7045908
ID50	0.035157	2.02319	0.0173772
ID51	-0.816761	2.02173	-0.4039911
ID52	-0.710578	2.02310	-0.3512332
ID53	-0.551192	2.01028	-0.2741869
ID54	-0.576237	2.09978	-0.2744266
ID55	-0.327016	2.00325	-0.1632425
ID56	-0.885656	2.07119	-0.4276075
ID57	-0.100727	2.07217	-0.0486094
ID58	-0.523934	2.02167	-0.2591593
ID59	-0.837876	2.02949	-0.4128499
ID60	-0.392933	2.00973	-0.1955153
ID61	-0.954997	2.03050	-0.4703262

```
ID62         0.099137    2.00527  0.0494379
ID63         0.113926    2.01416  0.0565625
ID64        -1.234127    2.34939 -0.5252970
ID65         0.076932    2.00721  0.0383280
ID66        -0.661565    2.01149 -0.3288929
ID67        -0.832207    2.01850 -0.4122904
ID68        -0.296792    2.00710 -0.1478714
ID69        -0.957391    2.02614 -0.4725188
ID70        -1.303577    2.08723 -0.6245501
ID71        -0.199416    2.00255 -0.0995809
ID72         0.343739    2.09438  0.1641244
ID73        -0.792248    2.03315 -0.3896643
ID74        -0.950369    2.15398 -0.4412156
ID75         0.055982    2.00048  0.0279841
ID76        -0.424097    2.07703 -0.2041845
ID77        -0.361722    2.00531 -0.1803822
ID78        -0.127409    2.03788 -0.0625200
ID79        -0.667242    2.01269 -0.3315173
ID80        -0.237978    2.00339 -0.1187878
ID81        -0.075362    2.01845 -0.0373363
ID82        -1.031345    2.03924 -0.5057508
ID83        -0.194506    2.00132 -0.0971886
ID84        -0.370854    2.03988 -0.1818016
ID85        -0.287610    2.00246 -0.1436282
ID86        -0.496530    2.01099 -0.2469081
ID87        -1.228579    2.03851 -0.6026848
ID88        -0.632653    2.01245 -0.3143701
ID89        -0.600223    2.02777 -0.2960014
ID90        -0.678437    2.05493 -0.3301508
ID91        -0.875720    2.02924 -0.4315507
ID92        -0.528009    2.01936 -0.2614735
ID93        -0.566730    2.00885 -0.2821161
ID94         0.089163    2.00167  0.0445444
ID95        -0.466588    2.12203 -0.2198775
ID96        -0.942491    2.13155 -0.4421624
ID97        -1.433934    2.09107 -0.6857409
ID98        -0.548368    2.13448 -0.2569098
ID99        -0.149796    2.00312 -0.0747813
ID100        0.332443    2.00356  0.1659256
x2           0.706559    0.22577  3.1295882

Number of linear predictors:  100

Dispersion Parameter for mbinomial family:   1

Log-likelihood: -133.2842 on 19899 degrees of freedom

Number of Iterations: 4
```

The fitted intercept is $\widehat{\alpha}^*$ in (26), and `ID2` is $\widehat{\alpha}_2^*$, etc. Although the standard error of the `x2` variable is smaller than those of the intercepts, it is still quite large. One would need a more larger value of C to estimate it more accurately.

In terms of the object sizes (in megabytes),

```
> objsizemb = function(object) round(object.size(object)/2^20,
+     dig = 2)
> objsizemb(fit)

[1] 1

> LMX = model.matrix(fit, type = "lm")
> dim(LMX)

[1] 200 101

> objsizemb(LMX)

[1] 0.17

> rm(LMX)
> VLMX = model.matrix(fit, type = "vlm")
> dim(VLMX)

[1] 20000   101

> objsizemb(VLMX)

[1] 16.2

> rm(VLMX)
```

9 Closing Comments

This documentation covers how VGAM can be used to fit standard GLMs and GAMs. However, the VGAM software library was more generally written to fit models from the *vector generalized linear model* (VGLM) and *vector generalized additive model* (VGAM) classes. It is hoped that users will utilize the full potential available in VGAM.

Exercises

1. For $p \neq 1$, 2, and $V(\mu) = \mu^p$, show that $Q(\mu_i; y_i) = y_i\,\mu_i^{1-p}/(1-p) - \mu_i^{2-p}/(2-p)$. What is the adjusted dependent variable and its weight?

2. Show that the normal equations (16) lead to

$$\boldsymbol{S}_{\boldsymbol{\beta}} \;=\; \sum_{i=1}^{n} A_i\,\boldsymbol{x}_i\left(y_i - \boldsymbol{x}_i^T\boldsymbol{\beta}\right) \;=\; \boldsymbol{0}$$

   for multiple linear regression, and

$$\boldsymbol{S}_{\boldsymbol{\beta}} \;=\; \sum_{i=1}^{n} \boldsymbol{x}_i\left(y_i - \frac{\exp\{\boldsymbol{x}_i^T\boldsymbol{\beta}\}}{1+\exp\{\boldsymbol{x}_i^T\boldsymbol{\beta}\}}\right) \;=\; \boldsymbol{0}$$

   for logistic regression ($A_i\,Y_i \sim \mathrm{Bin}(A_i, \mu_i)$).

3. Given the first four columns of Table 2, verify the rest of the table. What are the c's? Also verify Table 1 from Table 2.

4. The quasi-geometric distribution has $\sigma^2 V(\mu) = \sigma^2\mu(\mu+1)$. How could one fit this model? If necessary, write some code to do this.

5. The *generalized Poisson* distribution (Consul and Jain (1973); see also Jørgensen (1997) and Consul and Famoye (2006)) is

$$P(Y=y) \;=\; \frac{\theta(\theta+y\lambda)^{y-1}}{y!}\,\exp(-y\lambda-\theta), \qquad y=0,1,\ldots, \qquad 0\le\lambda<1, \qquad \theta>0.$$

   Derive the score vector and Hessian. Can you derive the mean and variance?

6. Suppose $Y \sim$ Poisson(μ). Find an expression for

$$\frac{\partial\, P(Y \ge k)}{\partial\,\mu}$$

   for a positive integer k.

References

Aitkin, M., Anderson, D., Francis, B., Hinde, J., 1989. Statistical Modelling in GLIM. Clarendon Press, Oxford.

Azzalini, A., 1996. Statistical Inference – Based on the Likelihood. Chapman & Hall, London.

Breslow, N. E., Day, N. E., 1980. Statistical Methods in Cancer Research I: The Analysis of Case-Control Studies. Vol. 1. International Agency for Research on Cancer, Lyon.

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

Consul, P. C., Famoye, F., 2006. Lagrangian Probability Distributions. Birkhäuser, Boston.

Consul, P. C., Jain, G. C., 1973. A generalization of the Poisson distribution. Technometrics 15 (4), 791–799.

Crawley, M. J., 1993. GLIM for ecologists. Blackwell Scientific Publications, Boston.

Dobson, A. J., 2001. An Introduction to Generalized Linear Models, 2nd Edition. Chapman & Hall/CRC Press, Boca Raton, FL.

Efron, B., 1986. Double exponential families and their use in generalized linear regression. Journal of the American Statistical Association 81 (395), 709–721.

Fahrmeir, L., Tutz, G., 2001. Multivariate Statistical Modelling Based on Generalized Linear Models, 2nd Edition. Springer-Verlag, New York.

Faraway, J. J., 2006. Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models. Chapman and Hall/CRC, Boca Raton, FL, USA.

Firth, D., 1991. Generalized linear models. In: Hinkley, D. V., Reid, N., Snell, E. J. (Eds.), Statistical Theory and Modelling. In Honour of Sir David Cox, FRS. Chapman & Hall, London, pp. 55–82.

Green, P. J., Silverman, B. W., 1994. Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach. Chapman & Hall, London.

Hastie, T., Pregibon, D., 1988. A new algorithm for matched case-control studies with applications to additive models. In: COMPSTAT 1988 (Copenhagen, 1988). Physica, Heidelberg, pp. 279–284.

Hastie, T. J., Pregibon, D., 1987. A new algorithm for matched case-control studies with applications to additive models. Tech. rep., AT&T Bell Laboratories, Murray Hill, NJ 07974, USA.

Hastie, T. J., Tibshirani, R. J., 1990. Generalized Additive Models. Chapman & Hall, London.

Holford, T. R., White, C., Kelsey, J. L., 1978. Multivariate analysis for matched case-control studies. American Journal of Epidemiology 107 (3), 245–256.

Jørgensen, B., 1997. The Theory of Dispersion Models. Chapman & Hall, London.

Krzanowski, W. J., 1998. An Introduction to Statistical Modelling. Arnold, London.

Lindsey, J. K., 1997. Applying Generalized Linear Models. Springer-Verlag, New York.

McCullagh, P., Nelder, J. A., 1989. Generalized Linear Models, 2nd Edition. Chapman & Hall, London.

Myers, R. H., Montgomery, D. C., Vining, G. G., 2002. Generalized Linear Models With Applications in Engineering and the Sciences. Wiley-Interscience, New York.

Nelder, J. A., Wedderburn, R. W. M., 1972. Generalized linear models. Journal of the Royal Statistical Society, Series A, General 135 (3), 370–384.

Pregibon, D., 1984. Data analytic methods for matched case-control studies. Biometrics 40 (3), 639–651.

Rigby, R. A., Stasinopoulpos, M. D., 1996. Mean and dispersion additive models. In: Härdle, W., Schimek, M. G. (Eds.), Statistical Theory and Computational Aspects of Smoothing: Proceedings of the COMPSTAT '94 Satellite Meeting held in Semmering, Austria, 27–28 August 1994. Physica-Verlag, Heidelberg, pp. 215–230.

Schimek, M. G., Turlach, B. A., 2000. Additive and generalized additive models. In: Schimek, M. G. (Ed.), Smoothing and Regression: Approaches, Computation, and Application. Wiley, New York, pp. 277–327.

Thas, O., Rayner, J. C. W., 2005. Smooth tests for the zero-inflated Poisson distribution. Biometrics 61 (3), 808–815.

Venables, W. N., Ripley, B. D., 2002. Modern Applied Statistics With S, 4th Edition. Springer-Verlag, New York.

Wedderburn, R. W. M., 1974. Quasi-likelihood functions, generalized linear models, and the Gauss-Newton method. Biometrika 61 (3), 439–447.

Yee, T. W., Hastie, T. J., 2003. Reduced-rank vector generalized linear models. Statistical Modelling 3 (1), 15–41.

Yee, T. W., Wild, C. J., 1996. Vector generalized additive models. Journal of the Royal Statistical Society, Series B, Methodological 58 (3), 481–493.