Thomas W. Yee

Complements to Vector Generalized Linear and Additive Models

February 12, 2025

Springer

Preface

The beginning is the most important part of the work.
—Plato

This document contains complementary material for Yee (2015). Over time, I hope to add more and more content, especially regarding practical matters as a consequence of changes to the VGAM package.

Thomas Yee
Auckland, New Zealand

History is written by the victors.
—Winston Churchill

Contents

Part I
General Theory

Chapter 1
Complements: Introduction

1.1 New names for link functions

In January 2019 VGAM 1.1-0 renamed many link functions so that they all end in "`link`", e.g., `loglink()` is a copy of `loge()`, `logitlink()` is a copy of `logit()`. Based on Table 1.2 of Yee (2015), Table 1.1 is a summary and lists the new names next to the old ones.

Table 1.1 Some new VGAM link functions currently available. Any old names are in brackets. They are grouped approximately according to their domains. As with the entire book, all logarithms are natural: to base e.

Function	Link $g_j(\theta_j)$	Domain of θ_j	Link name
`cauchitlink()` [`cauchit()`]	$\tan(\pi(\theta - \frac{1}{2}))$	$(0, 1)$	cauchit
`cloglogloglink()` [`cloglog()`]	$\log\{-\log(1-\theta)\}$	$(0, 1)$	complementary log-log
`foldsqrtlink()` [`foldsqrt()`]	$\sqrt{2\theta} - \sqrt{2(1-\theta)}$	$(0, 1)$	folded square root
`logitlink()` [`logit()`]	$\log \dfrac{\theta}{1-\theta}$	$(0, 1)$	logit
`multilogitlink()` [`multilogit()`]	$\log \dfrac{\theta_j}{\theta_{M+1}}$	$(0, 1)^M$	multi-logit; $\sum_{j=1}^{M+1} \theta_j = 1$
`probitlink()` [`probit()`]	$\Phi^{-1}(\theta)$	$(0, 1)$	probit (for "probability unit")
`fisherzlink()` [`fisherz()`]	$\frac{1}{2} \log \dfrac{1+\theta}{1-\theta}$	$(-1, 1)$	Fisher's Z
`rhobitlink()` [`rhobit()`]	$\log \dfrac{1+\theta}{1-\theta}$	$(-1, 1)$	rhobit
`loglink()` [`loge()`]	$\log\ \theta$	$(0, \infty)$	log (logarithmic)
`logneglink()` [`logneg()`]	$\log(-\theta)$	$(-\infty, 0)$	log-negative
`negloglink()` [`negloge()`]	$-\log(\theta)$	$(0, \infty)$	negative-log
`reciprocallink()` [`reciprocal()`]	θ^{-1}	$(0, \infty)$	reciprocal
`nbcanlink()`	$\log\,(\theta/(\theta + k))$	$(0, \infty)$	NB canonical link (Sec. 11.3.3)
`extlogitlink()` [`extlogit()`]	$\log \dfrac{\theta - A}{B - \theta}$	(A, B)	extended logit
`explink()`	e^θ	$(-\infty, \infty)$	exponential
`identitylink()`	θ	$(-\infty, \infty)$	identity
`negidentitylink()` [`negidentity()`]	$-\theta$	$(-\infty, \infty)$	negative-identity
`logclink()` [`logc()`]	$\log(1-\theta)$	$(-\infty, 1)$	log-complement
`logloglink()` [`loglog()`]	$\log\log(\theta)$	$(1, \infty)$	log-log
`loglogloglink()`	$\log\log\log(\theta)$	(e, ∞)	log-log-log
`logofflink(`θ`, offset =` A`)` [`logoff()`]	$\log(\theta + A)$	$(-A, \infty)$	log with offset

Chapter 2
Complements: LMs, GLMs and GAMs

2.1 More on the Hauck-Donner Effect

Recall from Section 2.3.6.2 that the Hauck-Donner effect (HDE), put simply, is due to Wald statistics being nonmonotonic near the parameter space boundary. It was discovered by Hauck and Donner (1977).

Some recent developments include the writing of the function `hdeff()` which enables its detection, for almost all VGAM family functions. The call `summary(vglmObject)` conducts HDE detection by default (it can be suppressed by `HDEtest = FALSE`), and another function is `hdeffsev()` for measuring HDE severity.

The main paper has now appeared as Yee (2022) and because this took a long time, some subsequent results are in Yee (2021).

2.2 More on the Wald Test

Consider `wald.stat()` for testing $H_{0k} : \theta_k = \theta_{k0}$ by the Wald statistic, for variables $k = 1, 2, \ldots$. One has to be careful reading the literature because several combinations are possible: using the EIM versus the OIM, as well as evaluating these at the original MLE versus at the hypothesized values versus at values obtained by further IRLS iterations. To give some control of this choice in VGAM, the arguments `iterate.SE` and `orig.SE` operate (Table 2.1). Suppose the current coefficient being tested is the kth one.

- Argument `orig.SE`: if `TRUE` then the SE is evaluated at the MLE $\widehat{\boldsymbol{\theta}}$, i.e., the regression coefficients of the original fit are used. If `FALSE` then θ_{k0} is used and the other coefficients are determined by `iterate.SE`. Note that if `orig.SE = TRUE` then both `iterate.SE = TRUE` and `iterate.SE = FALSE` will result in the same $\widehat{\boldsymbol{\theta}}$ being used to compute the SE because of the obvious fact that further iterations from the original $\widehat{\theta}_k$ will lead to no change in the other parameter estimates: $\widehat{\widehat{\boldsymbol{\theta}}}_{[-k]} = \widehat{\boldsymbol{\theta}}_{[-k]}$. In the literature $\widehat{\widehat{\boldsymbol{\theta}}}_{[-k]}$ is called the *restricted MLE* of $\boldsymbol{\theta}_{[-k]}$ because H_{0k} imposes a restriction or constraint on the kth value of $\boldsymbol{\theta}$.
- Thus argument `iterate.SE` results in $\widehat{\widehat{\boldsymbol{\theta}}}_{[-k]}$ being computed by further IRLS iteration for the SE. If `FALSE` then those from the original object, $\widehat{\boldsymbol{\theta}}_{[-k]}$, are

used. Altogether, `wald.stat()` can return three different variants of the Wald statistic.
- The numerator of the signed Wald statistic is $\widehat{\theta}_k - \theta_{k0}$, and by default $\theta_{k0} = 0$ for all k.
- Almost all VGAM family functions use the EIM rather than the OIM. For some models they coincide.

The default is `iterate.SE = TRUE` and `orig.SE = FALSE` so that $(\theta_{k0},\ \widehat{\widehat{\boldsymbol{\theta}}}_{[-k]})$ is used for evaluating $\boldsymbol{\mathcal{I}}_E$. Setting `orig.SE = TRUE` corresponds to the same situation as `summary(vglmObject)`—where the HDE can be manifest.

Laskar and King (1997) investigate the behaviour of the *null Wald* (NW) statistic on MA(1) regression model. The NW statistic is where the variance is evaluated at the null hypothesis rather than the MLE. The results showed that the HDE was not present for the NW statistic—this is of no surprise at all. For one parameter models the NW statistic can be obtained by setting `orig.SE = FALSE`. Laskar and King (1997) attribute the NW idea to Mantel (1987); see also Goh and King (1999).

2.3 More on the Rao Score Test

Like `lrt.stat()`, function `score.stat()` for conducting the Rao score (RS) test (Rao, 1948) actually calls `wald.stat()` because of its many shared computational details. Basically, the underlying principles behind Table 2.1 hold for computing the SE for the score test. For `score.stat()`, the default is to use $\boldsymbol{U}(\theta_{k0},\ \widehat{\widehat{\boldsymbol{\theta}}}_{[-k]})$ and $\boldsymbol{\mathcal{I}}_E(\theta_{k0},\ \widehat{\widehat{\boldsymbol{\theta}}}_{[-k]})$. For computing $\boldsymbol{U}$, it is always a function of θ_{k0}—the question is what are the *other* arguments? The logical argument `iterate.score` enables this choice and operates in a similar manner to `iterate.SE`.

Some notes:

- The three arguments attempt to allow maximum flexibility, e.g., the combination $\boldsymbol{U}(\theta_{k0},\ \widehat{\boldsymbol{\theta}}_{[-k]})$ with $\boldsymbol{\mathcal{I}}_E^{-1}(\theta_{k0},\ \widehat{\widehat{\boldsymbol{\theta}}}_{[-k]})$ is obtained by `orig.SE = FALSE`, `iterate.score = FALSE`, `iterate.SE = TRUE`. Altogether, six different variants of the RS statistic can be returned by `score.stat()`. Setting `orig.SE = TRUE` will use $\boldsymbol{\mathcal{I}}_E(\widehat{\boldsymbol{\theta}})$ for the SEs.
- Of course, $\boldsymbol{U}(\widehat{\boldsymbol{\theta}}) = \boldsymbol{0}$ but note that $\boldsymbol{U}(\theta_{k0},\ \widehat{\widehat{\boldsymbol{\theta}}}_{[-k]})$ is of the form $(a, \boldsymbol{0})$ for some $a \in \mathbb{R}$. In `score.stat()` the option $\boldsymbol{U}(\theta_{k0},\ \widehat{\boldsymbol{\theta}}_{[-k]})$ has the form $(a, \boldsymbol{b})$ for some $a \neq 0$ and $\boldsymbol{b} \neq \boldsymbol{0}$ in general.
- Some useful (including historical) background to the RS test is given by Bera and Bilias (2001). The three tests are called LR, W and RS, and are implemented in VGAM by `lrt.stat()`, `wald.stat()` and `score.stat()` respectively. In econometrics especially, the RS test is known as the *Lagrange multiplier* (LM) test. A recent article on the score test failing is Karavarsamis et al. (2020).
- Some combinations can lead to the score test becoming inconsistent (Freedman, 2007), e.g., using his notation, he gives three flavours of the information matrix:

Table 2.1 How `wald.stat()` computes the SE of the kth regression coefficient. The arguments `iterate.SE` and `orig.SE` are logical. Note: the coefficients are written in the order (k, *others*) where $\widehat{\theta}_k \in \widehat{\boldsymbol{\theta}}$ which is the MLE of the original fit, and $\widehat{\widehat{\boldsymbol{\theta}}}_{[-k]}$ are the other estimated coefficients obtained upon further IRLS iteration. The ‡ denotes the default. Note: in `score.stat()` this table also applies to SEs, along with $\boldsymbol{U}(\theta_{k0},\ \textit{others})$ being specified by `iterate.score`.

	`iterate.SE`	`!iterate.SE`
`orig.SE`	$\left(\widehat{\theta}_k,\ \widehat{\boldsymbol{\theta}}_{[-k]}\right) = \widehat{\boldsymbol{\theta}}$	$\left(\widehat{\theta}_k,\ \widehat{\boldsymbol{\theta}}_{[-k]}\right) = \widehat{\boldsymbol{\theta}}$
`!orig.SE`	$\left(\theta_{k0},\ \widehat{\widehat{\boldsymbol{\theta}}}_{[-k]}\right)$ (‡)	$\left(\theta_{k0},\ \widehat{\boldsymbol{\theta}}_{[-k]}\right)$

$$S_n = \boldsymbol{U}(\widehat{\boldsymbol{\theta}}_{k0})^T \boldsymbol{\mathcal{I}}_E^{-1}(\widehat{\boldsymbol{\theta}}_{k0})\, \boldsymbol{U}(\widehat{\boldsymbol{\theta}}_{k0}), \tag{2.1}$$

$$T_n = \boldsymbol{U}(\widehat{\boldsymbol{\theta}}_{k0})^T \boldsymbol{\mathcal{I}}_O^{-1}(\widehat{\boldsymbol{\theta}}_{k0})\, \boldsymbol{U}(\widehat{\boldsymbol{\theta}}_{k0}), \tag{2.2}$$

$$U_n = \boldsymbol{U}(\widehat{\boldsymbol{\theta}}_{k0})^T \boldsymbol{\mathcal{I}}_O^{-1}(\widehat{\boldsymbol{\theta}})\, \boldsymbol{U}(\widehat{\boldsymbol{\theta}}_{k0}), \tag{2.3}$$

where $\widehat{\boldsymbol{\theta}}_{k0} \equiv (\theta_{k0},\ \widehat{\widehat{\boldsymbol{\theta}}}_{[-k]})$, hence the quantities are evaluated at the restricted MLE. He describes (2.1) as being based on the "estimated expected" information at the restricted MLE satisfying the null hypothesis—this is the conventional textbook version, governed by conventional asymptotic theory. It is the `score.stat()` default. Version (2.2) is often used when the EIM cannot be obtained in closed form, which is the usual case. Version (2.3) at the unrestricted MLE is not widely used for the score test. The statistic (2.2) may be inconsistent because the OIM at the restricted maximum may not be positive definite. By 'consistent', it should reject with high probability when the alternative hypothesis is true. Fortunately, VGAM almost always uses the EIM so that (2.2) is hardly ever a problem. Statistics (2.1) and (2.3) are okay because their information matrices are typically positive-definite. However, inconsistency may also occur due to spurious roots—by 'inconsistent' it is meant that the score test power at the true value of $\boldsymbol{\theta}$ does not approach 1 as n grows—and this problem can occur even when using the EIM. He writes that 'the score test statistic does not tend to infinity as it should' and 'lack of power at remote alternatives—especially when the expected likelihood equation has spurious roots', and these comments elude to the second result of the tipping point theorem (Yee, 2020).

- All the above work on the Rao score test is imperfect. In fact, there are 12 possibilities: 2 for U and 3×2 for the SE. This suggests that in the future there should be arguments equivalent to `iterate.score` (logical), `iterate.SE.k` (`"null"`, `"old"`, `"new"`), `Iterate.SE.others` (logical), that implements the full range of choices.
  Note that when the code is written, this might mean the default will change. If the model is intercept-only then the following should give a warning: `iterate.score == TRUE`, `iterate.SE.k == "new"`, or `Iterate.SE.others == TRUE`, because $\widehat{\widehat{\boldsymbol{\theta}}}_{[-k]}$ doesn't exist.

2.4 Relative Risk Regression

A GLM with a binomial family and log-link is called by some as 'relative risk regression'. A recent paper on this model is Schwendinger et al. (2021), which describes allied interesting topics such as the failure of half-stepping of `glm()` to converge, finiteness and uniqueness of the MLE, and detection of infinite components of the MLE.

Bibliographic Notes

Dobson and Barnett (2018) is an elementary treatment on GLMs and some allied subjects. Fox (2016) is an applied book on using R to fit GLMs. Ly et al. (2017) is a tutorial article on Fisher information, written by and of particular interest to mathematical psychologists. Yee (2020) gives details on the HDE in terms of its detection in regression models based on IRLS, tipping points and characterization of the parameter space based on the first two derivatives of the Wald statistic. Eilers and Marx (2021) looks at some practicalities of smoothing, especially with P-splines. Wang and Yan (2021) describes the splines2 package for shape-restricted regression splines.

Chapter 3
Complements: VGLMs

3.1 Iteratively Reweighted Least Squares

Giving a few more details behind (3.9) and the rest of Section 3.2 as a whole, recall that we have $\ell = \sum_{i=1}^n w_i\,\ell_i$, $(\boldsymbol{u}_i)_j = \partial\ell_i/\partial\eta_j$, $\mathbf{X}_i = \boldsymbol{x}_i^T \otimes \mathbf{I}_M$, and $\mathbf{X}_{\text{VLM}} = \mathbf{X}_{\text{LM}} \otimes \mathbf{I}_M$. For simplicity, let's assume $w_i = 1$ for $i = 1, \ldots, n$. Then

$$\begin{aligned}
\frac{\partial \ell_i}{\partial \boldsymbol{\beta}_j} &= \frac{\partial \ell_i}{\partial \eta_j}\frac{\partial \eta_j}{\partial \boldsymbol{\beta}_j} = \frac{\partial \ell_i}{\partial \eta_j}\,\boldsymbol{x}_i, \\
\frac{\partial \ell}{\partial \boldsymbol{\beta}} &= \mathbf{X}_{\text{VLM}}^T\,\boldsymbol{u}, \\
\boldsymbol{\mathcal{I}}^{(a-1)} &= \sum_{i=1}^n \mathbf{X}_i^T \mathbf{W}_i^{(a-1)}\,\mathbf{X}_i = \mathbf{X}_{\text{VLM}}^T \mathbf{W}^{(a-1)} \mathbf{X}_{\text{VLM}},
\end{aligned}$$

hence (3.9) is

$$\begin{aligned}
\boldsymbol{\beta}^{(a)} &= \boldsymbol{\beta}^{(a-1)} + \boldsymbol{\mathcal{I}}\Big(\boldsymbol{\beta}^{(a-1)}\Big)^{-1}\,\boldsymbol{u}\Big(\boldsymbol{\beta}^{(a-1)}\Big) \\
&= \Big(\mathbf{X}_{\text{VLM}}^T \mathbf{W}^{(a-1)}\,\mathbf{X}_{\text{VLM}}\Big)^{-1} \cdot \\
&\quad \Big[\mathbf{X}_{\text{VLM}}^T \mathbf{W}^{(a-1)}\,\mathbf{X}_{\text{VLM}}\,\boldsymbol{\beta}^{(a-1)} + \mathbf{X}_{\text{VLM}}^T \mathbf{W}^{(a-1)}\,\mathbf{W}^{-1(a-1)}\,\boldsymbol{u}^{(a-1)}\Big] \\
&= \Big(\mathbf{X}_{\text{VLM}}^T \mathbf{W}^{(a-1)}\,\mathbf{X}_{\text{VLM}}\Big)^{-1} \mathbf{X}_{\text{VLM}}^T \mathbf{W}^{(a-1)}\,\Big[\mathbf{X}_{\text{VLM}}\,\boldsymbol{\beta}^{(a-1)} + \mathbf{W}^{-1(a-1)}\,\boldsymbol{u}^{(a-1)}\Big] \\
&= \Big(\mathbf{X}_{\text{VLM}}^T \mathbf{W}^{(a-1)}\,\mathbf{X}_{\text{VLM}}\Big)^{-1} \mathbf{X}_{\text{VLM}}^T \mathbf{W}^{(a-1)}\,\boldsymbol{z}^{(a-1)}, \qquad (3.1)
\end{aligned}$$

where $\boldsymbol{z} = \big(\boldsymbol{z}_1^T, \ldots, \boldsymbol{z}_n^T\big)^T$ with $\boldsymbol{z}_i = \boldsymbol{\eta}_i + \mathbf{W}_i^{-1}\boldsymbol{u}_i$. One recognizes that (3.1) is the GLS solution obtained by regressing $\boldsymbol{z}^{(a-1)}$ upon $\mathbf{X}_{\text{VLM}}$ with weight matrix $\mathbf{W}^{(a-1)}$. This explains why Fisher scoring amounts to applying an IRLS algorithm.

Incidentally, Fisher scoring (as opposed to Newton-Raphson) is due to Fisher (1925).

Table 3.1 New functions in VGAM 1.0-5 concerning standard likelihood inference and the Hauck–Donner effect. The null and alternative hypotheses are $H_0 : \beta^*_{(j)k} = \beta^*_{(j)k,0}$ versus $H_1 : \beta^*_{(j)k} \neq \beta^*_{(j)k,0}$. Notes: (i) models evaluated at $\beta^*_{(j)k} = \beta^*_{(j)k,0}$ have the other elements of the parameter vector $\boldsymbol{\beta}^*$ estimated by IRLS, subject to H_0. (ii) By default the `.stat()`-type functions return the signed square root of the test statistics (so are asymptotically standard normal). (iii) The score vector is denoted by $\boldsymbol{U}$. (iv) The enumeration of the $\beta^*_{(j)k}$ have been mapped to $\theta_1, \theta_2, \ldots$ for simplicity.

Function	Description
`lrt.stat()`	LRT statistics, $\widetilde{\mathcal{W}}_L = \text{sgn}(\widehat{\theta}_s - \theta_{s0}) \cdot \sqrt{2\left[\ell(\widehat{\boldsymbol{\theta}}) - \ell(\theta_{s0})\right]}$.
`score.stat()`	Rao's score test statistics, $\widetilde{\mathcal{W}}_U = \text{sgn}(\widehat{\theta}_s - \theta_{s0}) \cdot \sqrt{\boldsymbol{U}(\theta_{s0})^T \boldsymbol{\mathcal{I}}_E^{-1}(\theta_{s0})\, \boldsymbol{U}(\theta_{s0})}$, where $\boldsymbol{U} = \sum_{i=1}^n \sum_{j=1}^M (\partial \ell_i / \partial \eta_j)\, (\partial \eta_j / \partial \theta_s)$.
`wald.stat()`	Wald test statistics, $\widetilde{\mathcal{W}}_s = \sqrt{\mathcal{W}_s} = \text{sgn}(\widehat{\theta}_s - \theta_{s0}) \cdot \sqrt{(\widehat{\theta}_s - \theta_{s0})^2 \,/\, \text{SE}^2(\theta_{s0})}$.

Table 3.2 Other new functions in VGAM (version 1.1-2 and later).

Function	Description
`add1.vglm()`	Adds all possible single terms to a VGLM.
`anova.vglm()`	Analysis of deviance for VGLMs (Types I, II and III).
`drop1.vglm()`	Drops all possible single terms from a VGLM.
`hdeff()`	Detects the HDE in VGLMs.
`nbcanlink()`	Canonical link for negative binomial regression.
`ordsup()`	Ordinal superiority measures for categorical data models.
`R2latvar()`	R^2 for latent variable models.
`step4()`	Choose a model by AIC in a stepwise algorithm (S4 generic function).

3.2 Confidence Intervals for Regression Coefficients

The stats generic function `confint()` allows the computation of confidence intervals (CIs) for regression coefficients and has three methods functions that are of relevance here.

- Function `confint.default()` assumes normality of the estimators about their true values, and requires the `coef()` and `vcov()` methods functions to work on the fitted object. The basic arguments are

```
> args(confint)

   function (object, parm, level = 0.95, ...)
   NULL
```

The CIs are based on the Wald method: an approximate $100(1-\alpha)\%$ confidence interval for θ_j is given by

$$\widehat{\theta}_j \;\pm\; z(\alpha/2)\, \text{SE}(\widehat{\theta}_j), \tag{3.2}$$

which is (A.23). These are symmetric about the point estimate, and are quick and easy to compute on a calculator (at least for common α values such as 5%, that is).

- The methods function `confint.lm()` returns CIs for each β_k of an LM (see (2.1)). Using the result

$$\widehat{\boldsymbol{\beta}} \;\sim\; N_p\Big(\boldsymbol{\beta},\; \sigma^2 \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\Big) \tag{3.3}$$

  (same as (2.8)) where σ is required to be estimated, the CI formula is based on a t_{n-p}-distribution. In fact it is simply (3.2) with $z(\alpha/2)$ replaced by $t_{n-p}(\alpha/2)$. The degrees of freedom, $n-p$, is returned by `df.residual()`.
- The methods function `confint.glm()` in MASS (written by D. M. Bates and W. N. Venables and subsequently corrected by B. D. Ripley) is based on the LRT described in Section A.1.4.1. Some of the details are as follows. Partition $\boldsymbol{\theta} = (\boldsymbol{\theta}_1^T, \boldsymbol{\theta}_2^T)^T$ where $p_j = \dim(\boldsymbol{\theta}_j)$, and treat $\boldsymbol{\theta}_2$ as a nuisance parameter. Let the *profile likelihood* for $\boldsymbol{\theta}_1$ be

$$R(\boldsymbol{\theta}_1) \;=\; \max_{\boldsymbol{\theta}_2} \; L(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)/L(\widehat{\boldsymbol{\theta}}). \tag{3.4}$$

  Then the LR subset statistic $-2\log R(\boldsymbol{\theta}_{*1}) \sim \chi^2_{p_1}$ asymptotically, therefore an approximate $100(1-\alpha)\%$ confidence region for $\boldsymbol{\theta}_1$ is the set of all $\boldsymbol{\theta}_{1*}$ such that

$$-\,2\log R(\boldsymbol{\theta}_{1*}) \;<\; \chi^2_{p_1}(\alpha). \tag{3.5}$$

  The function `confint.glm()` essentially determines (3.5) with $p_1 = 1$ for each regression coefficient. Computationally, it uses offsets and the original model's starting values to calculate values of the profile likelihood along a grid cast around the MLE of each β_k. The `approx()` function can then be used to find the confidence limits corresponding to the specified α level.

Note that `confint()`, by default, returns CIs for *each* regression coefficient in the model, therefore issues relating to multiple comparisons must be borne in mind.

Now for `"vglm"` objects, a methods function is available to return CIs for each $\beta^*_{(j)k}$ of a VGLM. It is

```
> args(confintvglm)

   function (object, parm = "(All)", level = 0.95, method = c("wald",
       "profile"), trace = NULL, ...)
   NULL
```

The default value of argument `parm` signifies that CIs for all regression coefficients are to be computed. The first value of argument `method` is its default (warning: the order of the values might change in the future). For `"wald"` the method of `confint.default()` is used. For `"profile"` the profile likelihood method of `confint.glm()` is used (indeed, the VGAM code is heavily based on the MASS code).

It is well known that CIs based on LRT tend to be more accurate than Wald CIs, especially when n is small. The profile likelihood method is computationally expensive and it is sometimes useful to set `trace = TRUE` in order to monitor the progress of the computations.

In its current implementation, models with an estimated dispersion parameter, such as `quasibinomialff()` and `quasipoissonff()`, are not handled—only full likelihood models are. When solving for (3.4) it is possible that an attempt to cross

over the boundary of the parameter space is made by $\boldsymbol{\theta}_2$, hence some warnings may be issued.

The functions `plot.profile.glm()` and `pairs.profile.glm()` from MASS appear to work with `"vglm"` objects. Here is an example based on the GPD for simulated extremes data (Sect. 16.3), where it is well known that the shape parameter requires a lot of data in order to be estimated with any certainty.

```
> set.seed(1); Threshold <- 0; shape <- exp(-1) - 0.5
> gdata <- data.frame(x2 = runif(nn <- 1000))
> gdata <- transform(gdata, y2 = rgpd(nn, scale = exp(1 + 0.1 * x2),
                                      shape = shape))
> fit1 <- vglm(y2 ~ x2, gpd(Threshold), data = gdata)
> coef(fit1)

  (Intercept):1 (Intercept):2            x2
        0.96947      -1.01303       0.26087

> coef(fit1, matrix = TRUE)

             loglink(scale) logofflink(shape, offset = 0.5)
  (Intercept)        0.96947                          -1.013
  x2                 0.26087                           0.000

> confint(fit1, method = "wald")

                     2.5 %   97.5 %
  (Intercept):1  0.846926  1.09201
  (Intercept):2 -1.160338 -0.86571
  x2             0.077614  0.44413

> confint(fit1, method = "profile")

                     2.5 %   97.5 %
  (Intercept):1  0.844439  1.09032
  (Intercept):2 -1.169865 -0.85487
  x2             0.078201  0.44322
```

With such a large n it is not surprising that both methods yield similar CIs. Then

```
> pfit1 <- profile(fit1)
> class(pfit1)

  [1] "profile.glm" "profile"

> MASS:::plot.profile(pfit1)  # Simply plot(pfit1) might work
```

and

```
> MASS:::pairs.profile(pfit1)  # Simply pairs(pfit1) might work
```

give Figs. 3.1–3.2.

From the online help of `MASS:::plot.profile`: "the `pairs()` method shows, for each pair of parameters x and y, two curves intersecting at the MLE, which give the loci of the points at which the tangents to the contours of the bivariate profile likelihood become vertical and horizontal, respectively. In the case of an exactly bivariate normal profile likelihood, these two curves would be straight lines giving the conditional means of $y|x$ and $x|y$, and the contours would be exactly elliptical."

Profile likelihoods are described briefly and at an introductory level in Coles (2001, Sects. 2.6.5,2.6.6) and another numerical example of `confint()` can be found in Section 16.1.

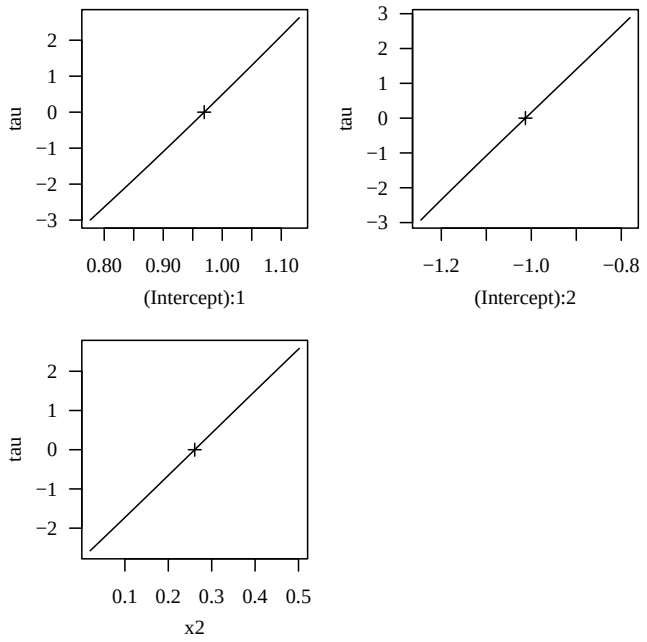


Fig. 3.1 Profile plots of a GPD model fitted to some simulated data.

3.3 Standard Errors for Regression Coefficients

When a simple VGLM is plotted using the `"vgam"` `plot()` methods function with `se = TRUE` the ± 2 SEs are 0 at the mean of that variable (for a simple term of the form $\beta^*_{(j)k}\, x_k$, that is). An example of this is Figure 8.2(a). In particular, the plotted line is

$$\widehat{\beta}^*_{(j)k}\, (x_{ik} - \overline{x}_k) \tag{3.6}$$

so that it is centred at that variable's mean. Hence the fitted line goes through $(\overline{x}_k,\ 0)$. Also, the SEs used are

$$\mathrm{SE}(\widehat{\beta}^*_{(j)k}) \cdot |x_{ik} - \overline{x}_k| \tag{3.7}$$

which `predict(vglmObject, type = "terms", se = TRUE)` returns. It is based on $\widehat{\mathrm{Var}}(\boldsymbol{x}_i^T \widehat{\boldsymbol{\beta}}^*) = \boldsymbol{x}_i^T\, \widehat{\mathrm{Var}}(\widehat{\boldsymbol{\beta}}^*)\, \boldsymbol{x}_i$ where the matrix in the middle is (3.21).

Setting `rug = TRUE` plots the location of the x_{ik} on the horizontal axis and this can be useful to see what the (jittered) distribution of the values looks like.

3.4 Variable Selection for VGLMs

This section, which concerns `add1.vglm()`, `drop1.vglm()` and `step4vglm()`, is closely related to Section 3.5. The latter function is a direct adaptation of

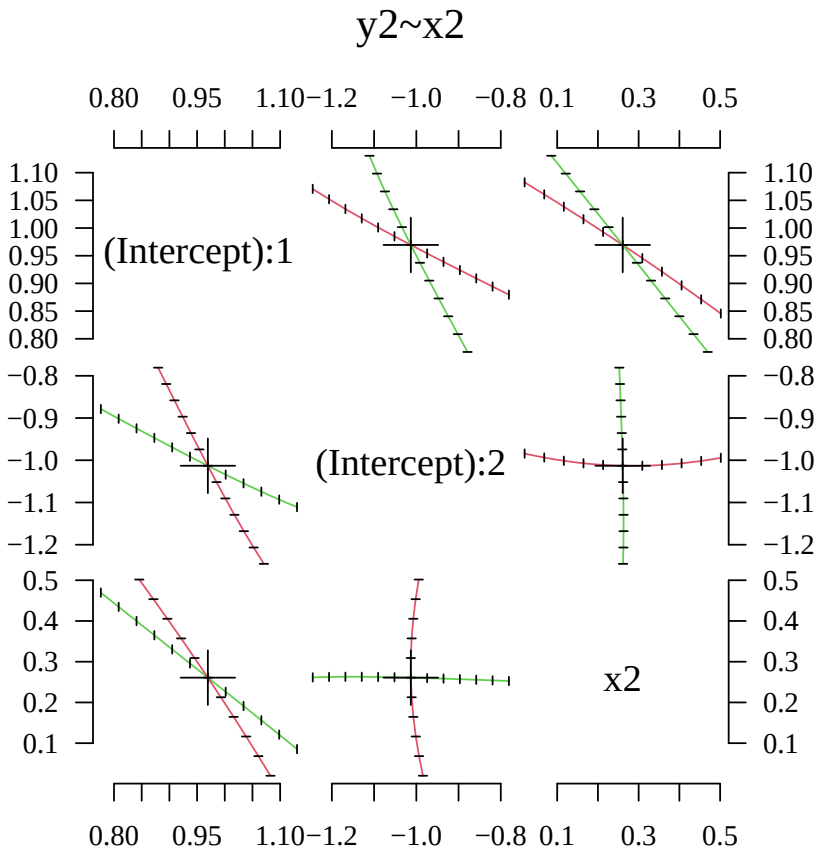


Fig. 3.2 Pairs plots of a GPD model fitted to some simulated data.

`stats:::step()` for `"vglm"` objects. Since `step()` is not generic, the name `step4()` was adopted and it *is* generic, as well as being S4 rather than S3.

It is the intent that `step4vglm()` should work as similar as possible to `step()`, which chooses a model by AIC in a stepwise algorithm. Internally it repeatedly calls `add1.vglm()` and `drop1.vglm()`. These functions add or drop one term from a `"vglm"` fit—and p-values are possible by specifying `test = "LRT"` as opposed to `"none"`.

Here are the arguments:

```
> args(add1.vglm)

   function (object, scope, test = c("none", "LRT"), k = 2, ...)
   NULL

> args(drop1.vglm)

   function (object, scope, test = c("none", "LRT"), k = 2, ...)
   NULL
```

```
> args(step4vglm)

   function (object, scope, direction = c("both", "backward", "forward"),
       trace = 1, keep = NULL, steps = 1000, k = 2, ...)
   NULL
```

As usual, where there are choices, the first value is the default choice. The first two functions were directly adapted from `add1.glm()` and `drop1.glm()`.

Here is an example. We fit a NB-1 to the `azpro` data set where some extra variables have been added.

```
> data(azpro, package = "COUNT")
> set.seed(1)
> azpro <- transform(azpro,
                     x10 = factor(round(runif(nrow(azpro), -0.5, 3.5))),
                     x11 = runif(nrow(azpro)))
> vglm.D93 <-
    vglm(los ~ procedure + sex + age75 + admit + x10 + x11,
         family = negbinomial(parallel = TRUE, zero = ""),  # NB1
         data = azpro)
>
> add1(vglm.D93,
       scope = ~ procedure * sex + age75 + admit + x10 * x11,
       test = "LRT")

   Single term additions

   Model:
   los ~ procedure + sex + age75 + admit + x10 + x11
                Df logLik   AIC  LRT Pr(>Chi)
   <none>           -9959 19938
   procedure:sex  1  -9958 19938 1.43     0.23
   x10:x11        3  -9957 19940 3.39     0.34
```

None of the interactions are needed really. Now let's try some stepwise regression.

```
> ans <- step4(vglm.D93,
               scope = ~ procedure + sex + age75 + admit + x10 + x11 +
                         hospital)

   Start:  AIC=19938
   los ~ procedure + sex + age75 + admit + x10 + x11

               Df logLik   AIC
   - x10        3  -9960 19934
   - x11        1  -9959 19936
   <none>          -9959 19938
   + hospital   1  -9959 19940
   - age75      1  -9977 19973
   - sex        1  -9979 19976
   - admit      1 -10100 20218
   - procedure  1 -11056 22130

   Step:  AIC=19934
   los ~ procedure + sex + age75 + admit + x11

               Df logLik   AIC
   - x11        1  -9960 19933
   <none>          -9960 19934
```

```
   + hospital   1  -9960 19936
   + x10        3  -9959 19938
   - age75      1  -9979 19969
   - sex        1  -9980 19973
   - admit      1 -10101 20214
   - procedure  1 -11057 22126

   Step:  AIC=19933
   los ~ procedure + sex + age75 + admit

               Df logLik   AIC
   <none>          -9960 19933
   + x11        1  -9960 19934
   + hospital   1  -9960 19935
   + x10        3  -9959 19936
   - age75      1  -9979 19968
   - sex        1  -9981 19971
   - admit      1 -10101 20212
   - procedure  1 -11057 22124

> ans


   Call:
   vglm(formula = los ~ procedure + sex + age75 + admit, family = negbinomial(parallel = TRUE,
       zero = ""), data = azpro)


   Coefficients:
   (Intercept):1 (Intercept):2     procedure           sex         age75
         1.47683       1.14215       0.94695      -0.11474       0.11609
           admit
         0.30326

   Degrees of Freedom: 7178 Total; 7172 Residual
   Log-likelihood: -9960.4

> ans@post$anova  # Results placed here

      Step Df Deviance Resid. Df Resid. Dev   AIC
   1       NA       NA      7168      19918 19938
   2 - x10  3  2.51845      7171      19920 19934
   3 - x11  1  0.44374      7172      19921 19933
```

Note that the final model is placed in the `post` slot, with component name `anova`. The final model here happens to drop the two junk variables that were created—this is a good thing.

3.4.1 The `update()` Function

Incidentally, the generic function `update()` works for `"vglm"` objects. For example,

```
> update(vglm.D93, . ~ . - x10 - x11)


   Call:
   vglm(formula = los ~ procedure + sex + age75 + admit, family = negbinomial(parallel = TRUE,
```

```
    zero = ""), data = azpro)


Coefficients:
(Intercept):1 (Intercept):2     procedure           sex         age75
      1.47683       1.14215       0.94695      -0.11474       0.11609
        admit
      0.30326

Degrees of Freedom: 7178 Total; 7172 Residual
Log-likelihood: -9960.4
```

Fortunately, `update.default()` and `update.formula()` were written so generally that no new code in VGAM is needed to get this going!

3.5 Analysis of Deviance for VGLMs

The methods function `anova.vglm()` produces analysis of deviance tables for VGLM fits. The function borrows ideas from `anova.glm()` in `stats` and `Anova.glm()` in `car`. The former implements Type I hypothesis tests only, and the latter implements Type II and III tests only (but not exactly as the SAS definition). By *analysis of deviance*, it is meant loosely that if the deviance of the model is not defined or implemented, then twice the difference between the log-likelihoods of two nested models is asymptotically chi-squared distributed with degrees of freedom equal to the difference in the number of parameters of the two models. This is because most VGAM family functions do not have a deviance that is defined or implemented, so we use $2(\ell - \ell_0)$ to loosely be called the deviance between the two models. This is "`2 * LogLik Diff.`" in the output. See Section A.1.4.2 for the overall relevant background material.

The `anova()` methods function for `"vglm"` objects has a `type` argument which allows Type I, II, and III tests to be conducted for the terms in the formula of the models.

```
> args(anova.vglm)

function (object, ..., type = c("II", "I", "III", 2, 1, 3), test = c("LRT",
    "none"), trydev = TRUE, silent = TRUE)
NULL
```

It is seen that Type II tests are the (current) default, and LRTs are performed as opposed to no test at all. Some justification for `type = "II"` being the default is given below.

Although they are more difficult test to understand than the other two, Type II tests do not suffer from the marginality problem of Type III, and according to the online help of `car:::Anova.glm()` Type I tests rarely test interesting hypotheses in unbalanced designs. However, Type II are inappropriate when there are significant interactions. It can be shown that when there is no interaction, Type II tests have more statistical power than Type III, however, when there is an interaction, they are inappropriate (Lewsey et al., 2001; Langsrud, 2003). In terms of statistical software, Type III is the default for, e.g., Minitab, SAS, SPSS and Stata; and Type I is the default for Genstat and `stats:::anova()` in R. Type II is the

default for `car:::Anova()` and `anova.vglm()`. A simple reference on the above issues is Fox and Weisberg (2011).

For `anova(fit, type = 1)`, specifying a single object gives a sequential analysis of deviance table for that fit. Of course, the usual regularity conditions are assumed to hold. For the analysis of deviance table, the reductions in the residual deviance as each term of the formula is added in turn are given in as the rows of a table, plus the residual deviances themselves.

Also for `type = 1`, if more than one object is specified then the table has a row for the residual degrees of freedom and deviance for each model. For all but the first model, the change in degrees of freedom and deviance is also given. (This only makes statistical sense if the models are nested.) It is conventional to list the models from smallest to largest, but this is up to the user.

Setting the argument `test = "none"` means that no p-values are returned whereas `test = "LRT"` conducts a likelihood ratio test. It is hoped that soon in the future `test = "Rao"` will conduct Rao's score test—see `score.stat()`. The function `lrtest()` provides an alternative method to compare nested models.

3.5.1 Types I, II, and III

This section gives a few details about the different types of tests implemented. It was SAS software that popularized the notion of Type I, II, and III sum of squares (SS) for hypothesis testing, especially in the context of LMs and ANOVA. We use the same notions here for VGLMs. Whereas the notes here correspond to $E(Y)$ in a LM and η in a GLM, it corresponds to $(\eta_1, \ldots, \eta_M)$ in VGLMs because a variable x_k can be potentially found in every η_j.

Note that Type II and III for `anova.vglm()` are the same as `car:::Anova.glm()`, and the latter has definitions that do not precisely match the SAS definitions. A full treatment would involve discussion of missing values and estimable functions—something not given here.

Also note that the topic of Type I, II, and III SS is controversial amongst statisticians and there is no general consensus about which is the best in general (Hector et al., 2010; Madsen and Thyregod, 2011). Their differences can be illustrated in terms of two factors called A and B, say, so that their interaction $A * B = 1 + A + B + A : B$ is the sum of the intercept, two main effects and the interaction term. It always to pays to test for the interaction terms before the main effects because main effects are rarely interpretable in the presence of interactions. If there was another factor C, say, then $A*B*C = 1+A+B+C+A : B + A : C + B : C + A : B : C$. The data is called *balanced* of there are an equal number of observations in each cell of the contingency table, e.g., at each level of A, $A : B$, etc. In ANOVA, if the data is unbalanced, then there are several ways to calculate sums of squares, hence the common three types. It transpires that Type I, II, and III all coincide with balanced data because the factors are orthogonal.

3.5.1.1 Type I Tests

These are called *sequential* SS and *incremental* SS. For this, the order of the terms is important, and the each term is added sequentially from first to last. Computationally, Type I SSs are the most easily computed (Table 3.3).

According to Nelder (1994) and others, Type I and II sums are the only appropriate ones for testing ANOVA effects; however, see also the discussion of Nelder's article, including Searle (1995) and Rodriguez et al. (1995).

Table 3.3 Type I tests in a LM with $A * B$. It is sequential from first to last. Notationally, $\text{SS}(\mu, A, B)$ is the sum of squares of the model comprising 1, A and B, while $\text{SS}(A|\mu, B)$ is the additional sum of squares due to adding A to the model comprising 1 and B, etc.

Source	Type I SS
μ	$\text{SS}(\mu)$ also known as the `NULL` model
A	$\text{SS}(A\|\mu) = \text{SS}(\mu,\ A) - \text{SS}(\mu)$
B	$\text{SS}(B\|\mu,\ A) = \text{SS}(\mu,\ A,\ B) - \text{SS}(\mu,\ A)$
$A:B$	$\text{SS}(A:B\|\mu,\ A,\ B) = \text{SS}(\mu,\ A,\ B,\ A:B) - \text{SS}(\mu,\ A,\ B)$

3.5.1.2 Type III Tests

These are described next as they are easy to understand. Type III SS are called the *partial* SS approach. Here, every effect is adjusted for *all* other effects, so that a particular term is entered *last* in a Type I analysis. If the model has interaction terms then this means that care must be taken, e.g., for $A * B$, we have a p-value for A, given a model with 1, B and $A:B$. Usually it does not make sense to test for a main effect given an interaction term, hence Type III tests should be used with care. Type III tests violate marginality—see Section 3.5.1.3. In fact, the help file of `car:::Anova.glm` gives a warning to be careful of type-III tests. Table 3.4 gives a breakdown of the Type III SS for the two-factor case.

Table 3.4 Type III tests in a LM with $A * B$. Each term is entered last.

Source	Type I SS
A	$\text{SS}(A\|\mu,\ B,\ A:B) = \text{SS}(\mu,\ A,\ B,\ A:B) - \text{SS}(\mu,\ B,\ A:B)$
B	$\text{SS}(B\|\mu,\ A,\ A:B) = \text{SS}(\mu,\ A,\ B,\ A:B) - \text{SS}(\mu,\ A,\ A:B)$
$A:B$	$\text{SS}(A:B\|\mu,\ A,\ B) = \text{SS}(\mu,\ A,\ B,\ A:B) - \text{SS}(\mu,\ A,\ B)$

3.5.1.3 Type II Tests

These have been described as *hierarchical* or *partially sequential* tests. As the `car:::Anova.glm` help file says, Type II tests are calculated according to the principle of marginality: higher-order terms are not included when adding a particular term.

According to SAS, Type II SS are the reduction in error SS due to adding the term after all other terms have been added to the model except terms that contain the effect being tested. An effect is contained in another effect if it can be derived by deleting variables from the latter effect, e.g., the main effect of A is not adjusted for terms such as $A : B$, $A : C$ or $A : B : C$. For example, A and B are both contained in $A : B$, hence for the model $A * B$, the Type II SS are given by the reduced SS given in Table 3.5. Thus the p-value for A is based on a regression on 1 and B because $A : B$ contains A. As another example, for three factors, $A : B$ is contained in $A : B : C$, therefore adding $A : B$ gives the Type II $\text{SS}(A : B|\mu,\ A,\ B,\ C,\ A : C,\ B : C) = \text{SS}(\mu,\ A,\ B,\ C,\ A : B,\ A : C,\ B : C) - \text{SS}(\mu,\ A,\ B,\ C,\ A : C,\ B : C)$.

It can be shown that when there is no interaction, Type II tests have more statistical power than Type III tests. However, when there is an interaction, Type II are inappropriate.

Table 3.5 Type II tests in a LM with $A * B$. Higher-order terms are not included when adding a particular term to the model.

Source	Type II SS
A	$\text{SS}(A|\mu,\ B) = \text{SS}(\mu,\ A,\ B) - \text{SS}(\mu,\ B)$
B	$\text{SS}(B|\mu,\ A) = \text{SS}(\mu,\ A,\ B) - \text{SS}(\mu,\ A)$
$A : B$	$\text{SS}(A : B|\mu,\ A,\ B) = \text{SS}(\mu,\ A,\ B,\ A : B) - \text{SS}(\mu,\ A,\ B)$

3.5.2 On `anova()` and `Anova()`

Here are some thoughts on `stats:::anova()` and `car:::Anova()`, both from a developer's and user's point of view.

The generic function `Anova()` in `car` has several methods functions for various types of models, such as those produced by `lm()` (univariate and multivariate responses), `glm()`, `polr()` in MASS, `multinom()` in `nnet`. The functions computes Type II or Type III analysis-of-deviance tables, and they offer new capabilities above the standard R function `anova()`. In particular, `anova()` fits Type I only, whereas `Anova()` fits Type II and III only, with Type II being its default.

While the methods functions for `Anova()` increases its applicability, there are dangers that casual users need to be aware of, for example, `Anova.polr()` only handles the default logit link for cumulative link models fitted by `MASS:::polr()`, and feeding in a cumulative probit model results in nonsense output and does not even issue a warning message (In fact, this limitation is not even mentioned in the online help file!).

Each methods function of `anova()` handles a series of fits, via the `...` argument. However, `Anova()` only handles a single model. Thus

```
> Anova(fit.logit2, fit.logit)
```

ignores the second model. This is justified because Type I tests are not implemented by `Anova()`.

Currently `anova.vglm()` implements Types I, II, III, so can be thought of as a combination of `stats:::anova()` and `car:::Anova()`. Indeed, `anova.vglm()` tries to offer a selection of the good points from both functions. Currently `type = "II"` is the default, but that might possibly change in the future. So it is safest to specify it explicitly. And although LRT p-values are computed, one day it is hoped that Rao's score tests be conducted too. And `anova.vglm()` can handle a series of fits, e.g.,

```
> anova(fit.logit2, fit.logit, type = 1)
```

It is necessary to specify `type = "I"` here.

3.5.3 Examples

3.5.3.1 Proportional Odds Model

Here is an example of fitting a full-interaction proportional odds model involving three factors.

```
> data("backPain", package = "VGAM")
> backPain$x1 <- factor(backPain$x1)  # It's really a factor variable
> backPain$x2 <- factor(backPain$x2)  # Ditto
> backPain$x3 <- factor(backPain$x3)  # Ditto
> summary(backPain)  # To check

    x1     x2     x3                       pain
    1:39   1:21   1:64   worse                : 5
    2:62   2:52   2:37   same                 :14
           3:28          slight.improvement   :18
                         moderate.improvement:20
                         marked.improvement   :28
                         complete.relief      :16

> fitlogit <- vglm(pain ~ x1 * x2 * x3, propodds, data = backPain)
> coef(fitlogit)

   (Intercept):1 (Intercept):2 (Intercept):3 (Intercept):4 (Intercept):5
        5.627426      4.033720      3.024353      2.008484      0.172164
             x12           x22           x23           x32       x12:x22
      -1.849842     -1.475770      0.054328     -1.466637      0.953756
        x12:x23       x12:x32       x22:x32       x23:x32   x12:x22:x32
      -1.465196      1.492585      0.538289     -0.468558     -1.866090
    x12:x23:x32
      -0.025652

> anova(fitlogit)

   Analysis of Deviance Table (Type II tests)

   Model: 'cumulative', 'VGAMordinal', 'VGAMcategorical'

   Links: 'logitlink'

   Response: pain

            Df Deviance Resid. Df Resid. Dev Pr(>Chi)
```

```
   x1        1    13.22       495        329  0.00028 ***
   x2        2     5.20       497        321  0.07430 .
   x3        1     7.55       495        321  0.00599 **
   x1:x2     2     3.56       493        316  0.16892
   x1:x3     1     0.62       492        313  0.43060
   x2:x3     2     0.36       493        313  0.83720
   x1:x2:x3  2     1.29       491        312  0.52434
   ---
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> anova(fitlogit, type = "I")

   Analysis of Deviance Table (Type I tests: terms added sequentially from
   first to last)

   Model: 'cumulative', 'VGAMordinal', 'VGAMcategorical'

   Links: 'logitlink'

   Response: pain

            Df Deviance Resid. Df Resid. Dev Pr(>Chi)
   NULL                       500        343
   x1        1    15.94       499        327  6.5e-05 ***
   x2        2     4.54       497        323    0.103
   x3        1     6.18       496        316    0.013 *
   x1:x2     2     3.16       494        313    0.206
   x1:x3     1     0.45       493        313    0.504
   x2:x3     2     0.36       491        312    0.837
   x1:x2:x3  2     1.29       489        311    0.524
   ---
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> anova(fitlogit, type = "III")

   Analysis of Deviance Table (Type III tests: each term added last)

   Model: 'cumulative', 'VGAMordinal', 'VGAMcategorical'

   Links: 'logitlink'

   Response: pain

            Df Deviance Resid. Df Resid. Dev Pr(>Chi)
   x1        1     2.60       490        314     0.11
   x2        2     3.74       491        315     0.15
   x3        1     1.75       490        313     0.19
   x1:x2     2     3.96       491        315     0.14
   x1:x3     1     0.81       490        312     0.37
   x2:x3     2     0.42       491        312     0.81
   x1:x2:x3  2     1.29       491        312     0.52
```

Naïvely, one can see that the p-values for the main effects can be quite different. Starting with the highest-order interactions, one concludes that `x1:x2:x3` is not needed, nor any of the pairwise interactions. Then let's fit main effects only:

```
> fitlogit2 <- vglm(pain ~ x1 + x2 + x3, propodds, data = backPain)
> coef(fitlogit2)

   (Intercept):1 (Intercept):2 (Intercept):3 (Intercept):4 (Intercept):5
```

```
       5.410242      3.836542      2.838690      1.859782      0.096801
            x12           x22           x23           x32
      -1.465704     -1.031782     -1.102121     -0.924080

> summary(fitlogit2, presid = FALSE)


   Call:
   vglm(formula = pain ~ x1 + x2 + x3, family = propodds, data = backPain)

   Coefficients:
                Estimate Std. Error z value Pr(>|z|)
   (Intercept):1   5.4102     0.7247    7.47  8.3e-14 ***
   (Intercept):2   3.8365     0.5955    6.44  1.2e-10 ***
   (Intercept):3   2.8387     0.5479    5.18  2.2e-07 ***
   (Intercept):4   1.8598     0.5080    3.66  0.00025 ***
   (Intercept):5   0.0968     0.4757    0.20  0.83877
   x12            -1.4657     0.3968   -3.69  0.00022 ***
   x22            -1.0318     0.4839   -2.13  0.03298 *
   x23            -1.1021     0.5372   -2.05  0.04023 *
   x32            -0.9241     0.3804   -2.43  0.01513 *
   ---
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

   Number of linear predictors:  5

   Names of linear predictors: logitlink(P[Y>=2]), logitlink(P[Y>=3]),
   logitlink(P[Y>=4]), logitlink(P[Y>=5]), logitlink(P[Y>=6])

   Residual deviance: 316.4 on 496 degrees of freedom

   Log-likelihood: -158.2 on 496 degrees of freedom

   Number of Fisher scoring iterations: 5

   No Hauck-Donner effect found in any of the estimates


   Exponentiated coefficients:
       x12     x22     x23     x32
   0.23092 0.35637 0.33217 0.39690

> anova(fitlogit2)

   Analysis of Deviance Table (Type II tests)

   Model: 'cumulative', 'VGAMordinal', 'VGAMcategorical'

   Links: 'logitlink'

   Response: pain

      Df Deviance Resid. Df Resid. Dev Pr(>Chi)
   x1  1    14.08       497        330  0.00018 ***
   x2  2     5.13       498        322  0.07708 .
   x3  1     6.18       497        323  0.01295 *
   ---
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> anova(fitlogit2, type = "I")
```

```
   Analysis of Deviance Table (Type I tests: terms added sequentially from
   first to last)

   Model: 'cumulative', 'VGAMordinal', 'VGAMcategorical'

   Links: 'logitlink'

   Response: pain

        Df Deviance Resid. Df Resid. Dev Pr(>Chi)
   NULL                  500        343
   x1    1    15.94       499        327  6.5e-05 ***
   x2    2     4.54       497        323    0.103
   x3    1     6.18       496        316    0.013 *
   ---
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> anova(fitlogit2, type = "III")

   Analysis of Deviance Table (Type III tests: each term added last)

   Model: 'cumulative', 'VGAMordinal', 'VGAMcategorical'

   Links: 'logitlink'

   Response: pain

      Df Deviance Resid. Df Resid. Dev Pr(>Chi)
   x1  1    14.08       497        330  0.00018 ***
   x2  2     5.13       498        322  0.07708 .
   x3  1     6.18       497        323  0.01295 *
   ---
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The results suggests that `x2` could possibly be dropped.

```
> fitlogit3 <- vglm(pain ~ x1 + x3, propodds, data = backPain)
> coef(fitlogit3)

   (Intercept):1 (Intercept):2 (Intercept):3 (Intercept):4 (Intercept):5
         4.55944       3.00054       2.01011       1.05992      -0.63074
             x12           x32
        -1.58899      -0.87114

> summary(fitlogit3, presid = FALSE)


   Call:
   vglm(formula = pain ~ x1 + x3, family = propodds, data = backPain)

   Coefficients:
                Estimate Std. Error z value Pr(>|z|)
   (Intercept):1    4.559      0.597    7.64  2.2e-14 ***
   (Intercept):2    3.001      0.442    6.78  1.2e-11 ***
   (Intercept):3    2.010      0.390    5.16  2.5e-07 ***
   (Intercept):4    1.060      0.352    3.02   0.0026 **
   (Intercept):5   -0.631      0.347   -1.82   0.0690 .
   x12             -1.589      0.396   -4.01  6.1e-05 ***
   x32             -0.871      0.377   -2.31   0.0208 *
   ---
```

```
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

   Number of linear predictors:  5

   Names of linear predictors: logitlink(P[Y>=2]), logitlink(P[Y>=3]),
   logitlink(P[Y>=4]), logitlink(P[Y>=5]), logitlink(P[Y>=6])

   Residual deviance: 321.53 on 498 degrees of freedom

   Log-likelihood: -160.76 on 498 degrees of freedom

   Number of Fisher scoring iterations: 5

   Warning: Hauck-Donner effect detected in the following estimate(s):
   '(Intercept):1'


   Exponentiated coefficients:
       x12     x32
   0.20413 0.41848

> anova(fitlogit3)

   Analysis of Deviance Table (Type II tests)

   Model: 'cumulative', 'VGAMordinal', 'VGAMcategorical'

   Links: 'logitlink'

   Response: pain

      Df Deviance Resid. Df Resid. Dev Pr(>Chi)
   x1  1    16.94       499        338  3.9e-05 ***
   x3  1     5.59       499        327    0.018 *
   ---
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> anova(fitlogit3, type = "I")

   Analysis of Deviance Table (Type I tests: terms added sequentially from
   first to last)

   Model: 'cumulative', 'VGAMordinal', 'VGAMcategorical'

   Links: 'logitlink'

   Response: pain

        Df Deviance Resid. Df Resid. Dev Pr(>Chi)
   NULL                  500        343
   x1    1    15.94       499        327  6.5e-05 ***
   x3    1     5.59       498        322    0.018 *
   ---
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> anova(fitlogit3, type = "III")

   Analysis of Deviance Table (Type III tests: each term added last)

   Model: 'cumulative', 'VGAMordinal', 'VGAMcategorical'
```

```
   Links: 'logitlink'

   Response: pain

      Df Deviance Resid. Df Resid. Dev Pr(>Chi)
   x1  1    16.94       499        338  3.9e-05 ***
   x3  1     5.59       499        327    0.018 *
   ---
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3.5.3.2 Bivariate Normal

Here is an example from a bivariate normal distribution where no deviance is implemented.

```
> set.seed(123); nn <- 1000
> bdata <- data.frame(x2 = runif(nn), x3 = runif(nn))
> bdata <- transform(bdata, y1 = rnorm(nn, 1 + 2 * x2 + 0.1 * x3),
                            y2 = rnorm(nn, 3 + 4 * x2))
> fit1 <- vglm(cbind(y1, y2) ~ x2 + x3,
               binormal(eq.sd = TRUE), data = bdata, trace = FALSE)
> coef(fit1, matrix = TRUE)

                mean1     mean2 loglink(sd1) loglink(sd2) rhobitlink(rho)
   (Intercept) 1.02837  2.965324   -0.0067316   -0.0067316        0.052149
   x2          2.04200  4.097529    0.0000000    0.0000000        0.000000
   x3          0.08636 -0.068109    0.0000000    0.0000000        0.000000

> anova(fit1, type = 1)

   Analysis of Deviance Table (Type I tests: terms added sequentially from
   first to last)

   Model: 'binormal'

   Links: 'identitylink', 'identitylink', 'loglink', 'loglink', 'rhobitlink'

   Response: cbind(y1, y2)

        Df 2 * LogLik Diff. Resid. Df LogLik Pr(>Chi)
   NULL                          4996  -3374
   x2    2             1099      4994  -2825   <2e-16 ***
   x3    2                1      4992  -2824      0.6
   ---
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

> # Drop x3 manually... and call the model fit2
> fit2 <- vglm(cbind(y1, y2) ~ x2,
               binormal(eq.sd = TRUE), data = bdata, trace = FALSE)
> anova(fit2, fit1, type = 1)  # More than one object specified

   Analysis of Deviance Table

   Model 1: cbind(y1, y2) ~ x2
   Model 2: cbind(y1, y2) ~ x2 + x3
     Resid. Df LogLik Df 2 * LogLik Diff. Pr(>Chi)
   1      4994  -2825
   2      4992  -2824  2             1.02      0.6
```

```
> lrtest(fit1, fit2)  # An alternative way of testing for x3, given x2

  Likelihood ratio test

  Model 1: cbind(y1, y2) ~ x2 + x3
  Model 2: cbind(y1, y2) ~ x2
    #Df LogLik Df Chisq Pr(>Chisq)
  1 4992  -2824
  2 4994  -2825  2  1.02        0.6
```

Although the truth is that `x3` has a small effect, the data suggests that that variable can be dropped.

3.6 GLM Residuals and Diagnostics

This section might better belong to Chapter 2, however it is hoped that this work be extended to VGLMs in the future.

3.6.1 Randomized Quantile Residuals

Dunn and Smyth (1996) propose *randomized quantile residuals* for continuous and discrete distributions. They have some nice advantages over other types of residuals:

(i) They have an exact standard normal distribution regardless of whether the distribution is continuous or discrete. In contrast, deviance and Pearson residuals may contain distracting patterns. This standard normality arises if $\boldsymbol{\theta}$ are consistently estimated and holds apart from the sampling variability in $\widehat{\boldsymbol{\theta}}$.

(ii) They are very easily implemented when a p-type function exists for that distribution, i.e., the CDF. This is often the case for most distributions in VGAM.

(iii) They can be used where trends and patterns are of interest because $y_i < \widehat{\mu}_i$ does not necessarily imply that $r_{iq} < 0$. In fact, the authors state that their best use is under these circumstances.

For continuous distributions the quantile residuals are defined by

$$r_{iq} = \Phi^{-1}[F(y_i; \widehat{\boldsymbol{\theta}})]. \tag{3.8}$$

For example, for a default `exponential()` object it is `qnorm(pexp(y, rate = 1 / fitted(object)))`.

For discrete distributions the *randomized* quantile residuals are

$$r_{iq} = \Phi^{-1}(U_i) \tag{3.9}$$

where $U_i \sim \text{Unif}(a_i, b_i)$, $a_i = \lim_{y \to y_i^-} F(y; \widehat{\boldsymbol{\theta}})$, $b_i = F(y_i; \widehat{\boldsymbol{\theta}})$. Actually, the authors write $(a_i, b_i]$ but with `runif()` this is effectively the same as the completely open interval. As an example, for the Poisson distribution, this is of the form `qnorm(runif(length(y), ppois(y-1, mu), ppois(y, mu)))`. The authors recommend four replications of the quantile residuals with discrete distributions

because they have a random component, and any features not preserved across all four sets of residuals are considered artifacts of the randomization.

The following is a simple illustration of their use.

```
> nn <- 200
> set.seed(123)
> pdata <- data.frame(x2 = rnorm(nn))
> pdata <- transform(pdata, y1 = rpois(nn, exp(1 + x2)))
> fit3 <- vglm(y1 ~ x2, poissonff, data = pdata)
> coef(fit3, matrix = TRUE)

             loglink(lambda)
(Intercept)          1.02777
x2                   0.98706
```

```
> rqres <- resid(fit3, type = "rquantile")
> hist(rqres, prob = TRUE, main = "(a)")
> qqnorm(rqres, main = "(b)", col = "blue")
> qqline(rqres)
```

This gives Fig. 3.3. Not surprisingly, the standard normal distribution expected of the residuals is largely obtained.

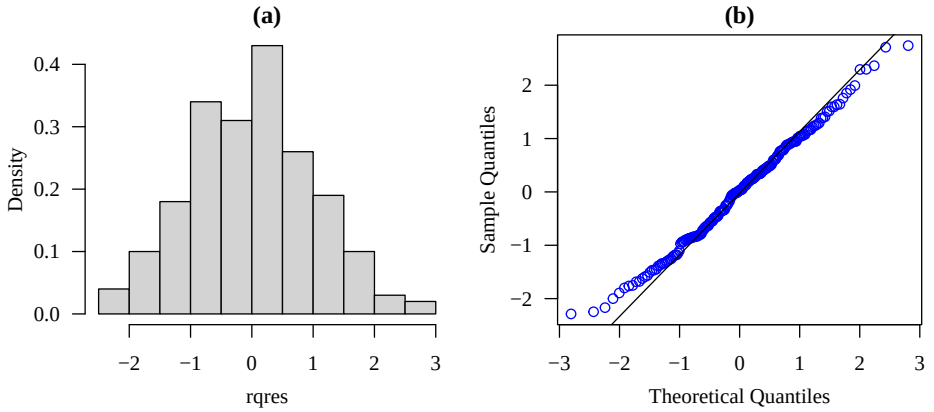


Fig. 3.3 Randomized quantile residuals for simulated Poisson data. (a) Histogram, (b) normal QQ plot.

3.6.2 Standardized Residuals

Agresti (2013, p.141) describes *standardized residuals* for GLMs, which are of the form

$$r_i^{\text{std}} \quad = \quad \frac{y_i - \widehat{\mu}_i}{\text{SE}(y_i - \widehat{\mu}_i)}. \tag{3.10}$$

The standardized residuals for LMs, (2.12), are a special case.

Using results from Section 3.7.5, for GLMs,

$$\begin{aligned} \mathrm{Cov}(\boldsymbol{y} - \widehat{\boldsymbol{\mu}}) &= \mathbf{V}^{1/2} \left[\mathbf{I}_n - \mathbf{H}\right] \mathbf{V}^{1/2}, & (3.11) \\ \mathbf{H} &= \mathbf{W}^{1/2} \mathbf{X} \left(\mathbf{X}^T \mathbf{W} \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{W}^{1/2} & (3.12) \end{aligned}$$

(cf. (3.63) where $\mathbf{U} = \mathbf{W}^{1/2}$), so that (3.10) becomes

$$r_i^{\mathrm{std}} = \frac{y_i - \widehat{\mu}_i}{\sqrt{V(\widehat{\mu}_i)\,(1 - h_{ii})}}. \quad (3.13)$$

The proof of this result depends on the delta method (Agresti, 2013, p.142). For the Poisson model this is simply $r_i^{\mathrm{std}} = (y_i - \widehat{\mu}_i)/\sqrt{\widehat{\mu}_i(1 - h_{ii})}$.

The call `residuals(fit, type = "stdres")` returns these residuals for certain GLMs, e.g., `poissonff`. Here is a very simple example.

```
> set.seed(123)
> pdata <- data.frame(x2 = rnorm(nn <- 100))
> pdata <- transform(pdata, y1 = rpois(nn, exp(3 + x2)))
> fit1 <- vglm(y1 ~ x2, poissonff, data = pdata)
> coef(fit1, matrix = TRUE)

              loglink(lambda)
   (Intercept)         3.01517
   x2                  0.97756

> stem(resid(fit1, type = "stdres"))


    The decimal point is at the |

    -2 | 4
    -1 | 655
    -1 | 4433222110000
    -0 | 9999988877766555
    -0 | 4444442221111111000000
     0 | 0011122223333344444
     0 | 555667899
     1 | 113334
     1 | 5688999
     2 | 003
     2 | 5
```

The standardized residuals *do* appear to be approximately standard normal distributed.

Bibliographic Notes

Wiley and Wiley (2019) provides a general introduction to regression modelling with GLMs and VGLMs, including some other topics such as GAMs, machine learning, missing values and GLMMs.

The estimation of θ from $N(\theta, \theta^2)$ has been considered by several authors at least in the context of likelihood theory and associated topics such as the bootstrap,

e.g., Young and Smith (2005, p.209) and Severini (2000, p.186). Section 3.3.1.4 (Yee, 2015, p.102) fits this model as a VGLM using constraint matrices.

Exercises

Ex. 3.1. Simple Constraints—Poisson Distribution

(a) Suppose that $Y_1 \sim \text{Pois}(\mu_1)$ and $Y_2 \sim \text{Pois}(\mu_2 = \kappa \cdot \mu_1)$ independently, for positive μ_1 and κ. Generate 100 random variates each of Y_1 and Y_2, where $\mu_1 = 2$ and $\kappa = e \approx 2.7128$, say.
(b) Estimate μ_1 and κ using `poissonff()`.
(c) Estimate μ_1 and κ using `glm()` and `poisson()`.
(d) Suppose now that κ is known. Estimate μ_1 using all the data and `poissonff()`.
(e) Suppose that $\mu_2 = \mu_1 + \kappa$ with μ_1 and κ as in (a). Generate 100 random variates each of Y_1 and Y_2. Then repeat (b). And then repeat (d).

Ex. 3.2. Coefficient of Variation

The *coefficient of variation* (CV) is the ratio the standard deviation σ to the mean μ: σ/μ. Suppose that Y is normally distributed with some known CV. Generate $n = 100$ observations from $N(\mu,\ \sigma^2)$ where CV$= \frac{1}{4}$ is known, $\mu = 10$ is unknown, and estimate μ.

Ex. 3.3. Type III SS for Three Factors

Construct the equivalent of Table 3.5 but for three factors $A,\ B,\ C$, i.e., for $A * B * C$. Test out your answer empirically for a few terms using some artificial data set.

Chapter 4
Complements: VGAMs

Bibliographic Notes

A book soon to appear or has appeared is Wood (2017). There are other R packages for fitting GAMs, e.g., gamlss (which concentrates on models having location, scale and/or shape parameters; Stasinopoulos et al. (2017)) and R2BayesX (which is based on Bayesian methods).

Chapter 5
Complements: Reduced-Rank VGLMs

5.1 Time Series

This section shows that the VGLM and RR-VGLM infrastructure can be used to fit some time series models This section might be better placed in Section 10.2, however we position it here because the nested reduced-rank autoregressive model of Ahn and Reinsel (1988) appears in Chapter 5 of the book.

Consider the multivariate autoregressive AR(L) model

$$\boldsymbol{Y}_t \;=\; \sum_{j=1}^{L} \boldsymbol{\Phi}_j \boldsymbol{Y}_{t-j} + \boldsymbol{\varepsilon}_t, \qquad \boldsymbol{\varepsilon}_t \sim (\boldsymbol{0}, \boldsymbol{\Omega}) \text{ independently}, \quad t = 1, \ldots, n, \tag{5.1}$$

where $\boldsymbol{Y}_t$ is $M \times 1$, and $\boldsymbol{\Phi}_j$ is $M \times M$ and to be estimated. When the number of lags $L = 1$ it is possible to fit some special types of models, especially when $M = 2$.

5.1.1 Cointegration

This section is based on Murray (1994). If a linear combination of several nonstationary time series (random variables) results in a stationary time series (random variable) then we say the combined random variables are *cointegrated*. This was proposed by Granger (1981); see also Granger (1987) for their relationship with *error correction models*.

Let's follow the simple example of Murray (1994, Eqns. (3)–(4)). Suppose that

$$y_{t,1} - y_{t-1,1} = c\,(y_{t-1,2} - y_{t-1,1}) + \varepsilon_{t,1}, \tag{5.2}$$
$$y_{t,2} - y_{t-1,2} = d\,(y_{t-1,1} - y_{t-1,2}) + \varepsilon_{t,2}, \tag{5.3}$$

where the two elements of $\boldsymbol{\varepsilon}_t$ are stationary white noise steps at each time period. The actual scenario considered by Murray (1994). are the steps of a drunk woman and her puppy dog going out for a walk. The positions are on the real line and the dog is unleashed. The walk of both are not quite random walks because at every time point she calls out and the dog barks, and then they move toward each other. The result is that the two paths are nonstationary but the distance between them is stationary.

Now rearrange (5.2)–(5.3) to give

$$\begin{pmatrix} y_{t,1} \\ y_{t,2} \end{pmatrix} = \begin{pmatrix} 1-c & c \\ d & 1-d \end{pmatrix} \begin{pmatrix} y_{t-1,1} \\ y_{t-1,2} \end{pmatrix} + \boldsymbol{\varepsilon}_t. \tag{5.4}$$

Here, c and d are parameters to be estimated.

Write (5.4) as

$$\boldsymbol{Y}_t \;=\; \boldsymbol{\Phi}_1 \boldsymbol{Y}_{t-1} + \boldsymbol{\varepsilon}_t, \qquad \boldsymbol{\varepsilon}_t \sim N_2(\boldsymbol{0},\; \boldsymbol{\Omega}) \;\text{ independently}, \quad t = 1, \ldots, n, \tag{5.5}$$

where normality is now assumed. Attention is drawn to the following four cases. The normality assumption means that the family function `binormal()` can be used for all of them.

1. Firstly, suppose that $c + d = 1$ so that $\boldsymbol{\Phi}_1$ is of unit rank. This corresponds to a VGLM with offsets and a constraint matrix $(1, 1, 0, 0, 0)^T$ for the variable $y_{t-1,2} - y_{t-1,1}$. It is a special case of the next model.
2. Secondly, if $c + d \neq 1$ then one can fit (5.4) as a VGLM using offsets. This is because

$$\begin{pmatrix} y_{t,1} \\ y_{t,2} \end{pmatrix} = \begin{pmatrix} y_{t-1,1} \\ y_{t-1,2} \end{pmatrix} + \begin{pmatrix} c & 0 \\ 0 & d \end{pmatrix} \begin{pmatrix} y_{t-1,2} - y_{t-1,1} \\ y_{t-1,1} - y_{t-1,2} \end{pmatrix} + \boldsymbol{\varepsilon}_t. \tag{5.6}$$

   One can think of this as the 'proper' solution to this cointegration problem.
3. Thirdly, if $\boldsymbol{\Phi}_1$ was a general matrix without having the structure imposed by (5.4) then this might be fitted by regressing the $\boldsymbol{Y}_t$ with $\boldsymbol{Y}_{t-1}$ as an ordinary VGLM. This particular model is a vector autoregressive model of order-1, commonly written as VAR(1).
4. Fourthly, suppose we stipulate that $\boldsymbol{\Phi}_1$ is of rank-1. Then we can fit this as a RR-VGLM. Like the third model, this model is not cointegrated.

As a numerical example, we select two responses from the four time series considered in Ahn and Reinsel (1988). These concern the monthly averages of grain prices in the United States for wheat flour, corn, wheat and rye for the period January 1961–October 1972. The units are dollars per 100 pound sack for wheat flour, and per bushel for corn, wheat and rye. We shall look at wheat and rye only. The entire data set can be seen by

```
> year <- seq(1961 + 1/12, 1972 + 10/12, by = 1/12)
> for (j in 1:4)
    plot(grain.us[, j] ~ year, main = names(grain.us)[j],
         type = "b", pch = "*", ylab = "", col = "blue")
```

This produces Fig. 5.1.

To start off with, let's get the data prepared.

```
> cgrain.df <- scale(grain.us, scale = FALSE)  # Centre the time series only
> grain.df <- subset(cgrain.df, select = c(wheat, rye))
> N <- nrow(grain.df)
> grain.df <- transform(grain.df,
                        wheat.lag1 = c(NA, wheat[-N]),
                        rye.lag1   = c(NA, rye[-N]))
> grain.df <- grain.df[-1, ]
```

The first model can be fitted by

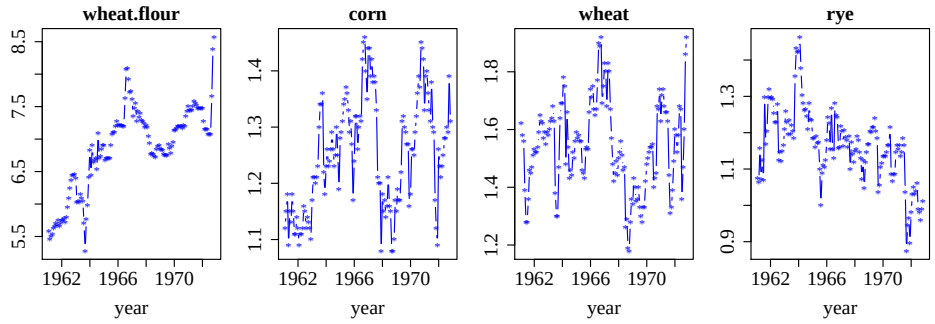


Fig. 5.1 Monthly average prices of Grain series, January 1961–October 1972, in data frame `grain.us`.

```
> grain.df <- transform(grain.df,
                        zedd  = rye.lag1 - wheat.lag1,
                        zilch = 0)
> M1 <- 5  # For binormal()
> Hlist1 <- list(
    "(Intercept)" = diag(M1)[, -(1:2)],
    zedd = rbind(1, 1, 0, 0, 0))
> grain.fit1 <-
      vglm(cbind(wheat, rye) ~ zedd,
           offset = cbind(wheat.lag1, rye.lag1, zilch, zilch, zilch),
           constraints = Hlist1,
           binormal, data = grain.df)
> coef(grain.fit1, matrix = TRUE)

                  mean1      mean2 loglink(sd1) loglink(sd2) rhobitlink(rho)
  (Intercept)  0.0000000  0.0000000      -2.4202      -2.8497         0.79465
  zedd        -0.0095059 -0.0095059       0.0000       0.0000         0.00000

> constraints(grain.fit1, matrix = TRUE)

                 (Intercept):1 (Intercept):2 (Intercept):3 zedd
  mean1                      0             0             0    1
  mean2                      0             0             0    1
  loglink(sd1)               1             0             0    0
  loglink(sd2)               0             1             0    0
  rhobitlink(rho)            0             0             1    0
```

Then $\widehat{c} = -0.0095$.

The second general cointegrated model can be fitted by

```
> grain.df <- transform(grain.df,
                        zedd1 = rye.lag1 - wheat.lag1,
                        zedd2 = wheat.lag1 - rye.lag1)
> Hlist2 <- list(
    "(Intercept)" = diag(M1)[, -(1:2)],
    zedd1 =  rbind(1, 0, 0, 0, 0),
    zedd2 =  rbind(0, 1, 0, 0, 0))
> grain.fit2 <-
      vglm(cbind(wheat, rye) ~ zedd1 + zedd2,
           offset = cbind(wheat.lag1, rye.lag1, zilch, zilch, zilch),
           constraints = Hlist2,
```

```
          binormal, data = grain.df)
> coef(grain.fit2, matrix = TRUE)

                mean1    mean2 loglink(sd1) loglink(sd2) rhobitlink(rho)
(Intercept) 0.00000 0.000000      -2.4334      -2.8514         0.83046
zedd1       0.08195 0.000000       0.0000       0.0000         0.00000
zedd2       0.00000 0.031068       0.0000       0.0000         0.00000

> constraints(grain.fit2, matrix = TRUE)

                 (Intercept):1 (Intercept):2 (Intercept):3 zedd1 zedd2
mean1                        0             0             0     1     0
mean2                        0             0             0     0     1
loglink(sd1)                 1             0             0     0     0
loglink(sd2)                 0             1             0     0     0
rhobitlink(rho)              0             0             1     0     0
```

Some of the output here matches (5.6), viz. $\widehat{c} = 0.0819$ and $\widehat{d} = 0.0311$.

The third general VAR(1) model (not cointegrated) can be fitted by

```
> Hlist3 <- list(
    "(Intercept)" = diag(M1)[, -(1:2)],
    wheat.lag1 = diag(M1),
    rye.lag1 = diag(M1))
> grain.fit3 <-
     vglm(cbind(wheat, rye) ~ wheat.lag1 + rye.lag1,
          constraints = Hlist3,
          binormal, data = grain.df)
> coef(grain.fit3, matrix = TRUE)

                mean1      mean2 loglink(sd1) loglink(sd2) rhobitlink(rho)
(Intercept)  0.00000  0.0000000      -2.4611      -2.8891         0.73678
wheat.lag1   0.86763 -0.0074515       0.0000       0.0000         0.00000
rye.lag1    -0.08697  0.8398803       0.0000       0.0000         0.00000

> constraints(grain.fit3, matrix = TRUE)

                 (Intercept):1 (Intercept):2 (Intercept):3 wheat.lag1:1
mean1                        0             0             0            1
mean2                        0             0             0            0
loglink(sd1)                 1             0             0            0
loglink(sd2)                 0             1             0            0
rhobitlink(rho)              0             0             1            0
                 wheat.lag1:2 rye.lag1:1 rye.lag1:2
mean1                       0          1          0
mean2                       1          0          1
loglink(sd1)                0          0          0
loglink(sd2)                0          0          0
rhobitlink(rho)             0          0          0
```

The fourth (not cointegrated) model can be fitted by

```
> Hlist4 <- Hlist3  # Same as the previous model
> grain.fit4 <-
    rrvglm(cbind(wheat, rye) ~ wheat.lag1 + rye.lag1,
           constraints = Hlist4,
           str0 = 3:5,  # The var-cov matrix elts are intercept-only
           binormal, data = grain.df)
> coef(grain.fit4, matrix = TRUE)

                mean1    mean2 loglink(sd1) loglink(sd2) rhobitlink(rho)
```

```
   (Intercept)  0.00000  0.00000      -2.0551       -2.431          1.8412
   wheat.lag1   0.49761 -0.27139       0.0000        0.000          0.0000
   rye.lag1    -0.71654  0.39080       0.0000        0.000          0.0000

> coef(grain.fit4)

   (Intercept):1 (Intercept):2 (Intercept):3    wheat.lag1      rye.lag1
        -2.05513      -2.43101       1.84117       0.49761      -0.71654

> constraints(grain.fit4, matrix = TRUE)

                  (Intercept):1 (Intercept):2 (Intercept):3 wheat.lag1 rye.lag1
   mean1                      0             0             0    1.00000  1.00000
   mean2                      0             0             0   -0.54539 -0.54539
   loglink(sd1)               1             0             0    0.00000  0.00000
   loglink(sd2)               0             1             0    0.00000  0.00000
   rhobitlink(rho)            0             0             1    0.00000  0.00000
```

It is conceivable that a VGAM family function might be written to estimate the parameters of a $N_3(\boldsymbol{\mu},\ \boldsymbol{\Sigma})$ distribution, called `trinormal()` say. If so then one could fit cointegration models to a set of three times series using the basic VGAM infrastructure presented above.

Bibliographic Notes

Some recent work on RR-VGLMs include the following. Bura et al. (2016) develop RRR for models in the exponential family; basing their work on Bura and Yang (2011) and making use of the alternating algorithm, two asymptotic tests for the dimension R are described. Bura et al. (2018) develops asymptotic theory for RR-VGLMs, based on M-estimation for concave criterion functions maximized over non-convex and non-closed parameter spaces; the consistency and asymptotic distribution of MLEs for RR-VGLMs are derived.

Recently, Powers et al. (2018) propose a nuclear penalized multinomial regression model—it is somewhat similar to the stereotype model but uses a different type of RRR. They apply it to predicting bat outcomes in baseball.

Chapter 8
Complements: Using the VGAM Package

8.1 Introduction

This chapter looks at some more topics related to using the VGAM package.

8.1.1 On Fitted Values

Some VGAM family functions have an argument called `type.fitted` which allows different types of 'fitted values' to be returned by the `fitted()` generic. This argument is assigned a vector of possible values, and the first is taken as the default. Usually the default is `"mean"` to signify the mean. Another common alternative is to return quantiles (`"quantiles"` or `"percentiles"`), in which case the argument `percentiles` is relevant and can accept a vector of percentiles (values in $[0, 100]$, although the values 0 and 100 are not recommended in general).

Suppose `fit` is a fitted model whose family function has the `type.fitted` argument. Then the following calls should work:

```
> fitted(fit1, type.fitted = "quantiles", percentiles = c(5, 25, 80))
> predict(fit1, newdata = head(ndata), type = "response",
          type.fitted = "quantiles",
          percentiles = c(33+1/3, 66+2/3))
> predict(fit1, type = "response",
          type.fitted = "quantiles",
          percentiles = c(33+1/3,  66+2/3))
```

In the above the call to `fitted()` passes the new `percentile` values into the `@linkinv` slot using the `@extra` slot of the object. Assigning any acceptable value of the family function's `type.fitted` should work, i.e., any of the possible values specific to that family function.

The remainder of this section concerns the labelling of the fitted values. Currently, a vector response or a 1-column matrix response results in the internal variable `y` in `vglm()` being a vector (due to `model.response()` being called), hence `colnames(y)` returns a `NULL`. Consequently for many VGAM family functions, when the fitted values of the fitted model are obtained using `fitted()` then it is not possible to label the 1-column matrix response with the name of the response. Here is an example.

```
> fit1 <- vglm(y1 ~ 1, zoabetaR, data = odata)
> fit1 <- vglm(cbind(y1) ~ 1, zoabetaR, data = odata)  # Same as previous
> fit2 <- vglm(cbind(y1, y2) ~ 1, zoabetaR, data = odata)
> fitted(fit1)  # No colnames
> fitted(fit2)  # Does have colnames labelling
```

Multi-column responses should not have any labelling problems.

With multiple responses, currently the fitted values for `type.fitted = "quantiles"` are enumerated in an order that makes its use with respect to the response matrix easier. Here is an example.

```
> set.seed(1)
> ndata <- data.frame(x2 = runif(nn <- 200))
> ndata <- transform(ndata, y1 = rnbinom(nn, mu = exp(1+x2), size = exp(1)))
> ndata <- transform(ndata, y2 = rnbinom(nn, mu = exp(2+x2), size = exp(1)))
> fit1  <- vglm(cbind(y1, y2) ~ x2, negbinomial, data = ndata)
> head(fitted(fit1, type.fitted = "quantiles", percentiles = c(5, 25, 80)))

      5%y1 5%y2 25%y1 25%y2 80%y1 80%y2
[1,]    0    1     1     5     6    14
[2,]    0    2     2     5     6    16
[3,]    0    2     2     7     7    21
[4,]    1    4     3    11     9    31
[5,]    0    1     1     4     5    13
[6,]    1    4     3    11     9    31

> predict(fit1, newdata = head(ndata), type = "response",
          type.fitted = "quantiles",
          percentiles = c(33+1/3, 66+2/3))

  33.333%y1 33.333%y2 66.667%y1 66.667%y2
1         2         6         4        11
2         2         7         5        13
3         3         9         5        16
4         3        13         7        24
5         2         5         4        10
6         3        13         7        24

> head(
  predict(fit1, type = "response",
          type.fitted = "quantiles",
          percentiles = c(33+1/3,  66+2/3))
  )

      33.333%y1 33.333%y2 66.667%y1 66.667%y2
[1,]         2         6         4        11
[2,]         2         7         5        13
[3,]         3         9         5        16
[4,]         3        13         7        24
[5,]         2         5         4        10
[6,]         3        13         7        24

> myres <- c(depvar(fit1)) - fitted(fit1, type.fitted = "quantiles")
> colMeans(myres)  # 'Residuals'

   25%y1  25%y2  50%y1  50%y2  75%y1  75%y2
   2.605  6.935  0.735  1.945 -1.810 -4.840
```

These types of 'residuals' are easily computed by recycling.

8.1.2 Automating calls using for loops

The following code fits 4 types of cumulative link models. These are combinations of parallel and non-parallel, and 2 choices of link functions. Currently, it is necessary to do some slightly more advanced programming involving `substitute()` and `parse()` in order to get this to work. In the future the relevant VGAM internals may change, therefore this solution might change too.

```
> data("pneumo")
> pneumo <- transform(pneumo, let = log(exposure.time))
>
> for (par in c(TRUE, FALSE)) {
    for (lnk in c("logitlink", "clogloglink")) {
      cat("\n\n\n")
      cat("link:", lnk, ",   parallel:", par, "\n")
      my.call <- eval(substitute(expression({ paste(
                 "vglm(cbind(normal, mild, severe) ~ let, ",
                 "cumulative(link = '",  .lnk , "', ",
                             "parallel = ", .par ,
                             ", reverse = TRUE), ",
                 "data = pneumo)", sep = "")
                 }), list( .par = par, .lnk = lnk )))
      emc <- eval(my.call)
      fit <- eval(parse(text = emc))
      print(coef(fit, matrix = TRUE))
    }
  }



  link: logitlink ,   parallel: TRUE
              logitlink(P[Y>=2]) logitlink(P[Y>=3])
  (Intercept)            -9.6761           -10.5817
  let                     2.5968             2.5968



  link: clogloglink ,   parallel: TRUE
              clogloglink(P[Y>=2]) clogloglink(P[Y>=3])
  (Intercept)              -8.5988              -9.3547
  let                       2.2094               2.2094



  link: logitlink ,   parallel: FALSE
              logitlink(P[Y>=2]) logitlink(P[Y>=3])
  (Intercept)            -9.5933           -11.1048
  let                     2.5713             2.7435



  link: clogloglink ,   parallel: FALSE
              clogloglink(P[Y>=2]) clogloglink(P[Y>=3])
  (Intercept)              -8.5090              -10.5706
  let                       2.1819                2.5507
```

The estimated **B** matrices of each fit is printed out. [1]

8.1.3 The save.weights argument

The `save.weights` argument in `vglm.control()` specifies whether the working weight matrices of the fitted object are saved on the object. When `TRUE` the object can be much larger, because a matrix (of size up to $nM(M+1)/2$ doubles) is assigned to the `@weights` slot. For models where SFS is used one wants to have `save.weights = TRUE` because of reproducibility: one wants functions such as `vcov()` to return results corresponding exactly to the fit and not have to obtain another SFS estimate at a post-fit stage. For those models estimated solely by SFS the family function should have its own control function that assigns `save.weights = TRUE` by default. Typically, the function is called something like `famfun.control()`.

But what about family functions which use SFS optionally? For example, `negbinomial()` allows direct computation and SFS for the working weights, and there are arguments that control which algorithm is used. Then VGAM will save the working weights on the object if SFS is used at all, i.e., `save.weights` is ignored. If the direct algorithm is used then `save.weights` is used.

Bibliographic notes

Yee (2020) demonstrates the use of VGAM for the typical user, using negative binomial regression as the main vehicle. Some emphasis is placed on newer features since Yee (2015).

[1] Thanks to Max Kuhn for motivating this problem and solution.

Exercises

In general, any form of exercise, if pursued continuously, will help train us in perseverance.
—Mao Zedong

Part II
Some Applications

Chapter 11
Complements: Univariate Discrete Distributions

11.1 Introduction

This chapter looks at some more topics related to discrete distributions, especially as related to the VGAM package.

11.2 More on Negative Binomial Regression

11.2.1 Hypothesis Testing

A common test when performing negative binomial regression is a test of the Poisson assumption, that is, testing $H_0 : k = \infty$. Some results for this are summarized in Dean and Lawless (1989) and are summarized further here. As this is a test of whether a parameter is on the boundary of the parameter space, the results of, e.g., Moran (1971) apply. When $k = \infty$, the distribution of $Z = \sqrt{n}\,\widehat{k}^{-1}\, i(\widehat{\boldsymbol{\beta}}_1, \infty)^{1/2}$ asymptotically has a half-normal distribution for $Z > 0$ and a probability mass of $\frac{1}{2}$ at 0. Here, $\widehat{\boldsymbol{\beta}}_1$ is the MLE of $\boldsymbol{\beta}_1$ obtained under H_0 (i.e., a Poisson regression), and i the expected information. Alternatively, one can use analogous results of Chernoff (1954), which show that the LRT statistic for testing H_0 is asymptotically like a random variable having a probability mass of $\frac{1}{2}$ at 0 and a $\frac{1}{2}\chi^2_1$ distribution above 0. What this means in practice is that one can divide the usual LRT p-value by 2. The following illustrates the test on the `V1` data set.

```
> poisfit <- vglm(hits ~ 1, poissonff,   weights = ofreq, data = V1)
> nbdfit  <- vglm(hits ~ 1, negbinomial, weights = ofreq, data = V1)
> Coef(poisfit)

    lambda
   0.93229

> Coef(nbdfit)  # 'size' is quite large but is it Inf?

        mu     size
   0.93229 24.95898

> # P-value:
> pchisq(2 * (logLik(nbdfit) - logLik(poisfit)), df = 1,
         lower = FALSE) / 2
```

```
[1] 0.26088
```

(One cannot apply `lrtest()`, so the p-value is computed manually.) The p-value is large, therefore there is no evidence against the null hypothesis of the data coming from a Poisson distribution. This seems to confirm the belief that the guidance system of the doodle bugs was so primitive that essentially it was random about the intended target (central London—maybe Buckingham Palace or Churchill's bedroom?).

11.3 Marginal Effects

Section 14.1.4 talks about marginal effects for categorical regression models. This quantity is used by econometricians especially (and it is related to another quantity known as the *elasticity*). Even easier are marginal effects for certain count distributions. For Poisson regression it is almost trivial to show that

$$\frac{\partial \mu_i}{\partial x_k} \;=\; \frac{\partial e^{\eta_i}}{\partial \eta_i}\frac{\partial \eta_i}{\partial x_k} \;=\; \mu_i\,\beta_k$$

where the linear predictor is written here simply as $\eta = \sum_d \beta_d x_d$. Note this is crucially based on a log link—the code does not handle otherwise.

For `negbinomial()`, the first linear predictor of each response is also $\eta = \log \mu$, hence its computation is the same. It is also true for `posnegbinomial()` models.

Information about the VGAM software implementation for marginal effects is given in Section 18.3.1. Some very introductory material about marginal effects for Poisson regression can be found in Hilbe (2011, pp.125–34).

11.4 New VGAM Family Functions

Table 11.1 summarizes some new VGAM family functions for discrete distributions. Here are some skeleton details for some of them.

eq:dgenpois0

11.4.1 The Bell Distribution

Castellares et al. (2018) propose the Bell distribution for count regression. This section is based on that paper.

The Bell distribution is based on the expansion

$$\exp(e^x - 1) \;=\; \sum_{t=0}^{\infty} \frac{B_t}{t!}\, x^t, \tag{11.1}$$

for real x (Bell, 1934b,a), where B_t is the tth Bell number defined by

$$B_t \;=\; e^{-1} \sum_{i=0}^{\infty} \frac{i^t}{i!}. \tag{11.2}$$

The first few values of the Bell series are $B_0 = B_1 = 1$, $B_2 = 2$, $B_3 = 5$, $B_4 = 15$, $B_5 = 52$, $B_6 = 203$, $B_7 = 877$. From these, one can define the Bell distribution as

$$\Pr(Y = y; s) \;=\; \frac{s^y \exp(1 - e^s)\, B_y}{y!}, \quad y = 0(1)\infty, \quad 0 < s. \tag{11.3}$$

Castellares et al. (2018) summarize and derive some properties of this distribution, e.g.,

- it is a member of the 1-parameter exponential family;
- the Bell numbers B_t are the tth moments of the Poisson distribution;
- the distribution is strongly unimodal and infinitely divisible;
- the mean is $E(Y) = se^s$ (the fitted values of the family function `bellf()`), and $\mathrm{Var}(Y) = s(1 + s)e^s$;
- having an index of dispersion $\mathrm{Var}(Y)/E(Y) = 1+s$, it can model overdispersion (but not undispersion), although it has limited capabilities in this area because the amount of overdispersion accommodated is constrained by the mean;
- they show that although the Poisson is not a special case, it corresponds to a special case of the multiple Poisson process, and the distribution approaches the Poisson as $s \to 0$;
- $Y = A_1 + \cdots + A_N \sim \mathrm{Bell}(s)$ where $N \sim \mathrm{Pois}(e^s - 1)$ and $A_t \sim \mathrm{Positive} - \mathrm{Pois}(s)$ are i.i.d. This serves the basis of `rbell()`.

For one observation, its EIM is $(1+s)e^s/s$. The family function `bellff()` estimates the distribution by Fisher scoring.

An alternative parameterization involves the Lambert W function so that $\eta = \log \mu$ is theoretically possible. This arises because $\mu = se^s$ so that $s = W_0(\mu)$ and

$$\Pr(Y = y; s) \;=\; \exp\{1 - e^{W_0(\mu)}\} \frac{W_0(\mu)^y\, B_y}{y!}, \quad y = 0(1)\infty, \quad 0 < s \tag{11.4}$$

is an alternative to (11.3). However, currently $\eta = \log s$ is the default linear predictor of `bellff()`.

Currently, because the Bell numbers rapidly increase, in practice the y_i should not exceed 218 in value. Thus the regression method is limited to relatively small counts.

11.4.2 Differenced Zeta Distribution

The parameter s is the positive shape parameter, and a is the argument `start` of the VGAM family function `diffzeta()`. The quantity $\mathcal{A}$ used for the fitted value is

$$\mathcal{A} \;=\; \sum_{i=1}^{a} \frac{1}{i^s}.$$

According to Moreno-Sánchez et al. (2016), this model fits quite well to about 40 percent of all the English books in the Project Gutenberg data base (about 30,000 texts). Like most VGAM family functions, multiple responses are handled.

Bibliographic notes

Testing whether a given data set reasonably comes from a specified distribution is not given much emphasis in the chapter. A book on this important problem is Thas (2010), which is mainly concerned about goodness-of-fit tests, including tests for the one-sample problem where we wish test the hypothesis that the sample observations have a hypothesized distribution.

Some multivariate count distributions such as the negative-multinomial and the generalized Dirichlet-multinomial can be fitted by iteratively reweighted Poisson regressions (IRPR). This algorithm is simple and has good properties such as stability and favourable convergence properties. IRPR was proposed in Zhang et al. (2017) and has advantages over IRLS because the EIMs are expensive to compute. The VGAM package could be adapted to perform IRPR.

A very introductory book for the practitioner on modelling counts is Hilbe (2014).

Some count distributions for underdispersed data are described in Sellers and Morris (2017).

Exercises

Ex. 11.1. Show that the negative binomial distribution is *strictly unimodal*, i.e., that the first derivative of the PMF with respect to y has only one root.

Ex. 11.2. Consider the EBBD PMF (**??**). If $\rho = \mu = \frac{1}{2}$, show that the PMF becomes a discrete uniform distribution.

Ex. 11.3. Use Euler's difference formula

$$\sum_{n=0}^{k} (-1)^{k-n} \binom{k}{n} (A + Bn)^p \;=\; \begin{cases} 0, & 0 \le p < k, \\ B^k\, k!, & p = k. \end{cases}$$

to show that

$$\sum_{y=0}^{\infty} \frac{(\theta + y\lambda)^y}{y!}\, e^{-y\lambda-\theta} \;=\; \frac{1}{1-\lambda}, \qquad -\lambda_0 < \lambda < 1,$$

where λ_0 solves $\lambda\, e^{\lambda} = \exp(-1)$, i.e., $\lambda_0 \approx 0.278$ (Tuenter, 2000).

Distribution	PMF $f(y;\boldsymbol{\theta})$	Support	Range of $\boldsymbol{\theta}$	Mean	VGAM family
Differenced zeta	$\left(\frac{a}{y}\right)^s - \left[\frac{a}{1+y}\right]^s$	$a(1)\infty$	$(0,\infty)$	$a^s\left[\zeta(s) - \mathcal{A} + \frac{1}{a^{s-1}}\right]$	`diffzeta(dpqr)`

Table 11.1 New VGAM family functions for discrete distributions.

Chapter 12
Complements: Univariate Continuous Distributions

12.1 Introduction

This chapter looks at some updates since Yee (2015) on some more topics related to continuous distributions, especially as related to the VGAM package.

Distribution	PDF $f(y;\boldsymbol{\theta})$	Support	Range of $\boldsymbol{\theta}$	Mean (or median $\widetilde{\mu}$)	VGAM family
Generalized secant hyperbolic	$\dfrac{(c_1/b)\ \exp(c_2 z)}{\exp(2\,c_2 z) + 2C_3 \exp(c_2 z) + 1}$	$(-\infty, \infty)$	$0 < b$, $-\pi < s$	a	`gensh(dpqr)`
Chi	$\dfrac{y^{\nu-1}\, e^{-y^2/2}}{2^{\nu/2-1}\, \Gamma(\nu/2)}$	$(0, \infty)$	$0 < \nu$	$\dfrac{\sqrt{2}\, \Gamma((1+\nu)/2)}{\Gamma(\nu/2)}$	`chisq(squared = FALSE)`
Topp-Leone	$2s(1-y)[y(2-y)]^{s-1}$	$(0, 1)$	$0 < s < 1$	$1 - \dfrac{4^s\, [\Gamma(1+s)]^2}{\Gamma(2+2s)}$	`topple(dpqr)`
Hüsler-Reiss angular surface	$\dfrac{s\ \exp(-\{2 + s^2\ \mathrm{logit} y\}^2/[8s^2])}{4\,\sqrt{2\pi}\, y(1-y)^2}$	$(0, 1)$	$0 < s$	—	`hurea(d)`

Table 12.1 Univariate continuous distributions implemented in VGAM with support on (A, B), for finite A and B. For `gensh()`, $z = (y - a)/b$, constants C_3, c_1 and c_2 are functions of s. See also Table 12.11 for distributions related to the beta distribution.

Chapter 14
Complements: Categorical Data Analysis

14.1 Introduction

This chapter looks at some more topics related to categorical data analysis, especially as related to the VGAM package.

14.1.1 Some Jargon

In the literature the proportional odds model is known the *ordered logit model*. It can be fitted with the VGAM family function `propodds()`. The *generalized ordered logit model* is the nonproportional odds model, also know as the *nonparallel cumulative logit model*; it can be fitted with the VGAM family function `cumulative(reverse = TRUE)`. Here, we use `reverse = TRUE` to make the signs of the regression coefficients the same between the two type of models. The ordered logit model is a special case of the generalized ordered logit model, as is the partial proportional odds model too.

On the nonproportional odds model McCullagh and Nelder (1989, p.155) writes "The usefulness of non-parallel regression models is limited to some extent by the fact that the lines must eventually intersect. Negative fitted values are then unavoidable for some values of $\boldsymbol{x}$, though perhaps not in the observed range. If such intersections occur in a sufficiently remote region of the $\boldsymbol{x}$-space, this flaw in the model need not be serious." With `vglm(..., family = cumulative)` the half-stepping and `@validparams` features should stop the $\eta_j(\boldsymbol{x}_i)$ from actually intersecting inside the data set's $\boldsymbol{x}$-space (but approaching it, to machine precision). Hence it is highly recommended that users set `trace = TRUE` in order to monitor convergence. Some warnings may also be issued. Any nonstandard convergence behaviour is suggestive of the intersecting-η_j problem.

Also, profile likelihood methods may fail when applied to `cumulative()` models because the $\eta_j(\boldsymbol{x}_i)$ may intersect a little beyond their MLE. The functions to be vigilant of include `profile()`, `vplot.profile()`, `vpairs.profile()`, `confint(... , method = "profile")`.

14.1.2 The R2latvar() Function

VGAM has the `R2latvar()` utility function which returns a measure of predictive power for some types of cumulative link models. In a nutshell, it treats the model like a LM and computes R^2 on the η-scale. The following description draws from Agresti (2019, Sec. 6.3.7).

Consider a cumulative link model with the parallelism assumption applying to all η_j. This makes $\text{Var}(\eta_{ij})$ the same for all values of j. If the link is a `logitlink`, `probitlink` or `cloglog link` then the η-scale corresponds the standard logistic, standard normal and standard extreme value (log-Weibull) distributions respectively, according to the latent variable interpretation (see, e.g., Section 14.4.1.1 of Yee (2015), Agresti (2019, Sec. 6.2.6), McCullagh and Nelder (1989, Sec. 5.2.2)). That is, the link function corresponds to the inverse of the CDF of those distributions. These distributions have variances $\pi^2/3$, 1, and $\pi^2/6$, respectively—these are $\text{Var}(\varepsilon)$ in (14.18).

Consider computing the coefficient of determination R^2 of (14.18), treated as a LM. Recall for a LM that $R^2 = 1 - \text{ResSS}/\text{TotSS} = 1 - FVU$, where FVU is the fraction of variance unexplained. Since $R^2 = \text{RegSS}/\text{TotSS}$, we can compute

$$R^2_\eta \;=\; \frac{\text{Var}(Y')}{\text{Var}(Y') + \text{Var}(\varepsilon)}. \tag{14.1}$$

The subscript η here is used to emphasize that the scale is on the latent variable or η scale (possibly, using a subscript ν would be more in keeping with the rest of the book). Since the linear predictors are all parallel, we can choose the first one η_1, say, to represent the η_j scale. The latent variable scores are η_{i1} for $i = 1, \ldots, n$. Then (14.1) can be estimated using sample variances by

$$\widehat{R}^2_\eta \;=\; \frac{\widehat{\text{Var}}(\eta_{i1})}{\widehat{\text{Var}}(\eta_{i1}) + \text{Var}(\varepsilon)}. \tag{14.2}$$

Incidentally, some software such as Stata call the quantity the McKelvey–Zavoina R-squared, which was proposed in McKelvey and Zavoina (1975) for measuring the goodness of fit in cumulative probit models.

Here is a numerical example, mimicking Agresti (2019). Note: as of mid-2022, the following call to `read.table()` doesn't actually work, however the file can be downloaded manually. And the problem is encountered more than once in this chapter.

```
> Polviews2 <-
    read.table("http://users.stat.ufl.edu/~aa/cat/data/Polviews2.dat",
               header = TRUE)
```

```
> fitlogit <- vglm(ordered(ideology) ~ factor(party) + factor(gender),
                   cumulative(parallel = TRUE), data = Polviews2)
> fitprobit <- vglm(ordered(ideology) ~ factor(party) + factor(gender),
                    cumulative(link = "probitlink", parallel = TRUE),
                    data = Polviews2)
> R2latvar(fitlogit)

   [1] 0.48699

> R2latvar(fitprobit)
```

```
[1] 0.49452
```

For `fitlogit` Agresti (2019) says that we predict that 48.7% of the variability in the political ideology latent variable is explained by the two explanatory variables, and that this value is 'moderately large'.

One can compute the above manually, as follows.

```
> eta1 <- predict(fitlogit)[, 1]  # Use the 1st linear predictor, say
> var(eta1) / (var(eta1) + (pi^2)/3)

   [1] 0.48699

> eta2 <- predict(fitprobit)[, 2]  # Use the 2nd linear predictor, say
> var(eta2) / (var(eta2) + 1)

   [1] 0.49452
```

14.1.3 The `ordsup()` Function

Agresti and Kateri (2017) propose 'ordinal superiority' measures for the linear model and cumulative link models. These involve the probability that an observation from one distribution falls above an independent observation from the other distribution, adjusted for explanatory variables in a model. In fact it allows two groups to be compared without supplementary explanatory variables. Let Y_1 and Y_2 be independent random variables from groups A and B, say, for a quantitative ordinal categorical scale. Then

$$\Delta \;=\; \Pr(Y_1 > Y_2) - \Pr(Y_2 > Y_1) \tag{14.3}$$

summarizes their relative size. A second quantity is

$$\gamma \;=\; \Pr(Y_1 > Y_2) - \frac{1}{2}\Pr(Y_2 = Y_1). \tag{14.4}$$

Then it is easily shown that they are interrelated by

$$\Delta \;=\; 2 \times \gamma - 1, \tag{14.5}$$
$$\gamma \;=\; (\Delta + 1)/2. \tag{14.6}$$

The range of γ is $[0, 1]$, while for Δ it is $[-1, 1]$.

Note that the notation defining groups A and B is that there is a variable (call it x_2, say) such that $x_2 = 1$ for group A (aka Y_1) and $x_2 = 0$ for group B (aka Y_2). Some sketch details for the cumulative probit model are as follows: letting $\eta^* = \beta^*_{(1)2}\, x_2 + \boldsymbol{x}^T\boldsymbol{\beta}^*$, then the latent variable $\nu^* \sim N(\eta^*, 1)$ and hence

$$\begin{aligned}\gamma \;&=\; \Pr[Y_1 > Y_2] \;=\; \Pr[\nu_1^* > \nu_2^*] \\ &=\; \Pr\left[\frac{\nu_1^* - \nu_2^* - \beta^*_{(1)2}}{\sqrt{2}} > \frac{-\beta^*_{(1)2}}{\sqrt{2}}\right] \;=\; \Phi\left(\frac{\beta^*_{(1)2}}{\sqrt{2}}\right).\end{aligned}$$

For the above quantities γ and Δ, the `ordsup()` function is currently implemented for a very limited number of specific models—`cumulative()` with `link =`

`"logitlink"` or `link = "probitlink"`, and `uninormal()` with the default settings to handle the LM. By default only binary variables are chosen from all the explanatory variables. Confidence intervals are also available.

The following mimics the example from Agresti and Kateri (2017). It concerns a data set with $n = 40$ having a four-category response variable measuring mental impairment (1 = well, 2 = mild symptom formation, 3 = moderate symptom formation, 4 = impaired) to a binary indicator of socioeconomic status (`ses`: 0 = low, 1 = high) and a quantitative life-events (`life`) index taking values from the set `0:9`.

```
> Mental <-
   read.table("http://users.stat.ufl.edu/~aa/glm/data/Mental.dat",
                        header = TRUE)
```

```
> Mental$impair <- ordered(Mental$impair)  # It is really ordinal
> summary(with(Mental, impair))

    1  2  3  4
   12 12  7  9

> pfit3 <- vglm(impair ~ ses + life, data = Mental,
                cumulative(link = "probitlink", reverse = TRUE,
                           parallel = TRUE))
> coef(pfit3, matrix = TRUE)

              probitlink(P[Y>=2]) probitlink(P[Y>=3]) probitlink(P[Y>=4])
   (Intercept)             0.16118            -0.74563            -1.33917
   ses                    -0.68336            -0.68336            -0.68336
   life                    0.19535             0.19535             0.19535

> unlist(ordsup(pfit3))  # The 'ses' variable is binary

   gamma.ses Delta.ses
     0.31447  -0.37105
```

According to Agresti and Kateri (2017, p.216), one can intepret $\widehat{\gamma}$ as follows. To compare the two levels of `ses` using $\widehat{\beta}^*_{(1)2} = -0.68336$, we can use $\widehat{\gamma} \approx 0.314$. The ordinal superiority measure $\widehat{\gamma}$ has the interpretation that at any particular value for `life` events, there is about a $1/3$ chance of lower mental impairment at low `ses` than at high `ses`. The 95% profile likelihood confidence interval for $\beta^*_{(1)2}$ yields confidence intervals $(0.161, 0.507)$ for γ. Such CIs can be obtained as follows (Wald intervals not used):

```
> unlist(ordsup(pfit3, confint = TRUE, method = "profile"))

        gamma.ses       Delta.ses lower.gamma.ses upper.gamma.ses Lower.Delta.ses
         0.314475       -0.371050        0.160801        0.507490       -0.678398
   Upper.Delta.ses
         0.014981
```

For illustration's sake only, now fit a crude LM to these data:

```
> fit7 <- vglm(as.numeric(impair) ~ ses + life, uninormal, Mental)
> coef(fit7, matrix = TRUE)  # Parameter 'sd' is estimated by MLE
```

```
                  mean loglink(sd)
(Intercept)  1.91974   -0.012378
ses         -0.64501    0.000000
life         0.17778    0.000000

> ordsup(fit7)

$gamma
    ses
0.32212

$Delta
     ses
-0.35575

> ordsup(fit7, all.vars = TRUE)  # Some output may not be meaningful

$gamma
    ses    life
0.32212 0.55064

$Delta
     ses     life
-0.35575  0.10128
```

This example is quite crude because it treats `as.numeric(impair)` as normal about a fitted multiple linear regression plane.

14.1.4 More on Ordinal Categorical Data

The following is drawn from Agresti and Tarantola (2018).

There are some marginal effect variants, which are described in Long (1997), Long and Freese (2014), Greene (2018). The *average marginal effect* (AME) is the marginal effect of x_k at each of the n sample values of the explanatory variables and then averages them out. An alternative is the *marginal effect at the mean* (MEM) which computes the marginal effect at $\overline{\boldsymbol{x}}$, i.e., each explanatory variable set at its mean. A third marginal effect is known as *marginal effect at representative values* (MER) by setting all the explanatory variables at values of interest.

Here are some musings based on Agresti (2019, Sec. 6.3.4).

```
> Mental <-
      read.table("http://users.stat.ufl.edu/~aa/cat/data/Mental.dat",
                 header = TRUE)
```

```
> Mfit <- vglm(ordered(impair) ~ life + ses, propodds, data = Mental)
> meMfit <- margeff(Mfit)
> dimnames(meMfit)

[[1]]
[1] "(Intercept)" "life"        "ses"

[[2]]
[1] "1" "2" "3" "4"
```

```
   [[3]]
    [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "15"
   [16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28" "29" "30"
   [31] "31" "32" "33" "34" "35" "36" "37" "38" "39" "40"

> meMfit[-1, "1", ]  # Look at one of them; it is a matrix

             1         2         3         4         5         6         7
   life -0.07474 -0.032458 -0.06744 -0.075878 -0.079394 -0.064982 -0.072935
   ses   0.26047  0.113113  0.23502  0.264429  0.276684  0.226457  0.254175
              8         9        10        11        12        13        14
   life -0.079394 -0.079394 -0.050516 -0.072935 -0.064982 -0.069103 -0.028747
   ses   0.276684  0.276684  0.176045  0.254175  0.226457  0.240821  0.100182
             15        16        17        18        19        20        21
   life -0.079394 -0.072935 -0.041027 -0.078989 -0.036728 -0.069103 -0.032458
   ses   0.276684  0.254175  0.142975  0.275271  0.127993  0.240821  0.113113
             22        23        24        25        26        27        28
   life -0.055549 -0.079394 -0.07474 -0.078153 -0.075878 -0.055549 -0.012545
   ses   0.193586  0.276684  0.26047  0.272359  0.264429  0.193586  0.043718
             29        30        31        32        33        34        35
   life -0.060228 -0.045834 -0.055549 -0.041027 -0.078989 -0.050516 -0.036728
   ses   0.209889  0.159730  0.193586  0.142975  0.275271  0.176045  0.127993
             36        37        38        39        40
   life -0.045834 -0.045834 -0.041027 -0.016737 -0.012545
   ses   0.159730  0.159730  0.142975  0.058326  0.043718

> # Approximate AME
> rowMeans(meMfit[-1, "1", ])  # Almost the same (ses is slightly different)

       life       ses
   -0.057155  0.199181

> rowMeans(meMfit[-1, "4", ])  # Almost the same (ses is slightly different)

       life       ses
    0.047745 -0.166387

> apply(meMfit[-1, "1", ], 1, sd)  # Differs from book's SE

      life      ses
   0.020377 0.071011

> apply(meMfit[-1, "4", ], 1, sd)  # Differs from book's SE

      life      ses
   0.021232 0.073994
```

Now compare with the results in the book:

```
> library("MASS")  # To get polr()
> library("erer", quietly = TRUE)  # To get ocME()
```

```
> pfit <- polr(factor(impair) ~ life + ses, method="logistic", Mental)
> ocME(pfit)  # Marginal effects at the mean


Re-fitting to get Hessian

       effect.1 effect.2 effect.3 effect.4
   life   -0.062   -0.014    0.027    0.049
   ses     0.208    0.053   -0.084   -0.176
```

One might try obtaining the MEM as follows. It involves replacing the first row by the sample mean. The `fitted.values` slot is also assigned in case it is used by `margeff()`.

```
> x.lm <- model.matrix(Mfit, type = "lm")
> x.lm[1, ] <- colMeans(x.lm)  # Replace 1st row by the sample mean
> Mfit@x <- x.lm  # Replace
>
> # This does not always work:
> fv.temp <- predict(Mfit, data.frame(x.lm[1, -1, drop = FALSE]),
                     type = "response")
>
> Mfit@fitted.values[1, ] <- fv.temp  # Replace
> meMfit <- margeff(Mfit)
> meMfit[-1, "1", 1]  # Unfortunately not the same as the book

      life      ses
  -0.07474  0.26047

> meMfit[-1, "4", 1]  # Unfortunately not the same as the book

       life       ses
   0.014383 -0.050122
```

The trick has failed—the answer here is not the same as the book.

14.2 Constraints on the Intercepts

For ordinal models, argument `thresholds` implements some constraints on the intercepts which may be useful for some data sets, e.g., `thresholds = "symm0"`. For instance, they may be equidistant, or symmetric about the origin.

Here are some examples.

- `"equid"`, e.g.,

```
> CM.equid(4)

      [,1] [,2]
 [1,]    1    0
 [2,]    1    1
 [3,]    1    2
 [4,]    1    3

> CM.equid(5, Trev = TRUE, Tref = 3)

      [,1] [,2]
 [1,]    1    2
 [2,]    1    1
 [3,]    1    0
 [4,]    1   -1
 [5,]    1   -2
```

  Hence the first regression coefficient corresponds to the baseline value and the second to the distance (better, displacement, since it may be positive or negative depending on the direction) moving away from the baseline value.
- `"symm1"`, e.g.,

```
> CM.symm1(4)

     [,1] [,2] [,3]
[1,]    1    0   -1
[2,]    1   -1    0
[3,]    1    1    0
[4,]    1    0    1

> CM.symm1(5)

     [,1] [,2] [,3]
[1,]    1    0   -1
[2,]    1   -1    0
[3,]    1    0    0
[4,]    1    1    0
[5,]    1    0    1
```

The median corresponds to the first column (and middle row if M is odd), and the displacements are the other columns.

- `"symm0"`, e.g.,

```
> CM.symm0(4)

     [,1] [,2]
[1,]    0   -1
[2,]   -1    0
[3,]    1    0
[4,]    0    1

> CM.symm0(5)

     [,1] [,2]
[1,]    0   -1
[2,]   -1    0
[3,]    0    0
[4,]    1    0
[5,]    0    1
```

The median intercept is 0 by definition so is not estimated. The remaining intercepts are estimated and comprise pairs that differ by sign only. Symmetry about the origin corresponds to deleting the first column from the `"symm1"` case.

Of course, these type of $\mathbf{H}_1$ can be seen by typing something like `constraints(fit)[[1]]`.

14.3 On the Conditional Logit Model

While Melbourne and Sydney fight about who wears Australia's cultural crown, Canberra just gets on with it.
—Judy Horacek

Section 14.2.1 applies the `xij` argument to the multinomial logit model and illustrates the idea on the `TravelMode` data frame in AER. Unfortunately the smoothing method there was wrong, as explained below. The section is concerned with the mode choice for travel between the Australian cities Sydney and Melbourne. Recall that there are 210 people's choice of transportation for travel between the two cities. Four choices of travel mode are `air`, `trn` (train), `bus` and `car`. The data set arises from case-control data: almost an equal number of each choice is represented. The explanatory variables are x_2 = `gcost` (a measure of the gener-

alized cost of the travel), $x_3 =$ `wait` (the terminal waiting time, 0 for car), and $x_4 =$ household `income`. The variables `gcost` and `wait` clearly differ for each travel mode. In contrast, variable `income` is individual-specific so that every person has the same fixed household income regardless what choice he/she made. It is stated that the reason for subtracting `wait` and `gcost` of the `car`s option from the others is because cars are the baseline group, cf. (3.37).

We now give some details behind (3.37). For $j = 1, 2, 3 = M$,

$$\begin{aligned}
\eta_j &= \log \frac{\Pr(Y=j)}{\Pr(Y=M+1)} \\
&= \log \left[\frac{\exp(\beta_{(j)1} + \beta^*_{(1)2}\, x_{i2j} + \beta^*_{(1)3}\, x_{i3j} + \beta_{(1)4}\, x_{i4}) / \sum_k \exp(\eta_k)}{\exp(\beta_{(4)1} + \beta^*_{(1)2}\, x_{i24} + \beta^*_{(1)3}\, x_{i34} + \beta_{(4)4}\, x_{i4}) / \sum_k \exp(\eta_k)} \right] \\
&= \beta^*_{(j)1} + \beta^*_{(1)2}\,(x_{i2j} - x_{i24}) + \beta^*_{(1)3}\,(x_{i3j} - x_{i34}) + \beta^*_{(1)4}\, x_{i4}.
\end{aligned} \tag{14.7}$$

The lastline is (3.37). Note that we can only subtract the covariate values of the baseline group when the component function is linear.

More generally and from first principles, suppose that

$$\Pr(Y=j) = \frac{\exp[\beta_{(j)1} + f^*_{(1)2}(x_{i2j}) + \beta^*_{(1)3} x_{i3j} + \beta^*_{(1)4} x_{i4}]}{\sum_{k=1}^4 \exp(\eta_k)}$$

for some smooth function $f^*_{(1)2}$. That is, we allow the effect of x_2 to be nonlinear. Then we have, for $j = 1, \ldots, 3$,

$$\begin{aligned}
\eta_j &= \beta^*_{(j)1} + f^*_{(1)2}(x_{i2j}) - f^*_{(1)2}(x_{i24}) + \beta^*_{(1)3}\, x_{i3j} - \beta^*_{(1)3}\, x_{i34} + \beta^*_{(1)4}\, x_{i4} \\
&= \beta^*_{(j)1} + g^*_{(1)2}(x_{i2j}, x_{i24}) + \beta^*_{(1)3}\,(x_{i3j} - x_{i34}) + \beta^*_{(1)4}\, x_{i4}, \text{ say,} \\
&\neq \beta^*_{(j)1} + h^*_{(1)2}(x_{i2j} - x_{i24}) + \beta^*_{(1)3}\,(x_{i3j} - x_{i34}) + \beta^*_{(1)4}\, x_{i4}, \text{ say.}
\end{aligned}$$

Unfortunately the h^* function here is estimated in Section 3.4.2 and this is erroneous; what we want to fit is the g^* function.

The following code fits the g^* functions correctly. It relies on the property that the regression splines are a linear combination of some B-spline basis functions. Also, it is important that the term in the main formula representing the $\boldsymbol{x}_{ij}$ term (called the placeholder) can be used for plotting the component function later. Hence the knots of the placeholder must be correct. Using something like `NS(gcost)` as a placeholder would not be good since its knots would be incorrect.

```
> data("TravelMode", package = "AER")
> air.df <- subset(TravelMode, mode == "air")  # Form 4 smaller data frames
> trn.df <- subset(TravelMode, mode == "train")
> bus.df <- subset(TravelMode, mode == "bus")
> car.df <- subset(TravelMode, mode == "car")
> TravelMode2 <- data.frame(income     = air.df$income,
                            wait.air   = air.df$wait - car.df$wait,
                            wait.trn   = trn.df$wait - car.df$wait,
                            wait.bus   = bus.df$wait - car.df$wait,
                            gcost.air  = air.df$gcost,  # No subtraction here
                            gcost.trn  = trn.df$gcost,  # No subtraction here
                            gcost.bus  = bus.df$gcost,  # No subtraction here
                            gcost.car  = car.df$gcost,
                            gcost      = air.df$gcost,  # Value unimportant
                            wait       = air.df$wait)   # Value unimportant
```

```
> TravelMode2$mode <- subset(TravelMode, choice == "yes")$mode  # Response
```

```
> NS <- function(x, ..., df = 3)
    ns(c(x, ...), df = df)[1:length(x), , drop = FALSE]
>
> tfit2 <-
    vglm(mode ~ NS(gcost.air, gcost.bus, gcost.trn, gcost.car) +
                wait + income, trace = TRUE,
         multinomial(parallel = FALSE ~ 1), data = TravelMode2,
         xij = list(NS(gcost.air, gcost.bus, gcost.trn, gcost.car) ~
                    I(NS(gcost.air, gcost.bus, gcost.trn, gcost.car) -
                      NS(gcost.car, gcost.air, gcost.bus, gcost.trn)) +
                    I(NS(gcost.trn, gcost.car, gcost.bus, gcost.air) -
                      NS(gcost.car, gcost.air, gcost.bus, gcost.trn)) +
                    I(NS(gcost.bus, gcost.trn, gcost.car, gcost.air) -
                      NS(gcost.car, gcost.air, gcost.bus, gcost.trn)),
                    wait ~ wait.air + wait.trn + wait.bus),
         form2 = ~  NS(gcost.air, gcost.bus, gcost.trn, gcost.car) +
                    wait + income +
                    I(NS(gcost.air, gcost.bus, gcost.trn, gcost.car) -
                      NS(gcost.car, gcost.air, gcost.bus, gcost.trn)) +
                    I(NS(gcost.trn, gcost.car, gcost.bus, gcost.air) -
                      NS(gcost.car, gcost.air, gcost.bus, gcost.trn)) +
                    I(NS(gcost.bus, gcost.trn, gcost.car, gcost.air) -
                      NS(gcost.car, gcost.air, gcost.bus, gcost.trn)) +
                      wait.air + wait.trn + wait.bus)

   Iteration 1: deviance = 391.32835
   Iteration 2: deviance = 384.16662
   Iteration 3: deviance = 383.95106
   Iteration 4: deviance = 383.95072
   Iteration 5: deviance = 383.95072
```

Let's look at the coefficients:

```
> coef(tfit2, matrix = TRUE)

                                                    log(mu[,1]/mu[,4])
   (Intercept)                                            7.102414
   NS(gcost.air, gcost.bus, gcost.trn, gcost.car)1       -2.474438
   NS(gcost.air, gcost.bus, gcost.trn, gcost.car)2       -6.605724
   NS(gcost.air, gcost.bus, gcost.trn, gcost.car)3       -3.298851
   wait                                                  -0.097368
   income                                                -0.025886
                                                    log(mu[,2]/mu[,4])
   (Intercept)                                            5.086642
   NS(gcost.air, gcost.bus, gcost.trn, gcost.car)1       -2.474438
   NS(gcost.air, gcost.bus, gcost.trn, gcost.car)2       -6.605724
   NS(gcost.air, gcost.bus, gcost.trn, gcost.car)3       -3.298851
   wait                                                  -0.097368
   income                                                -0.025886
                                                    log(mu[,3]/mu[,4])
   (Intercept)                                            4.392911
   NS(gcost.air, gcost.bus, gcost.trn, gcost.car)1       -2.474438
   NS(gcost.air, gcost.bus, gcost.trn, gcost.car)2       -6.605724
   NS(gcost.air, gcost.bus, gcost.trn, gcost.car)3       -3.298851
   wait                                                  -0.097368
   income                                                -0.025886
```

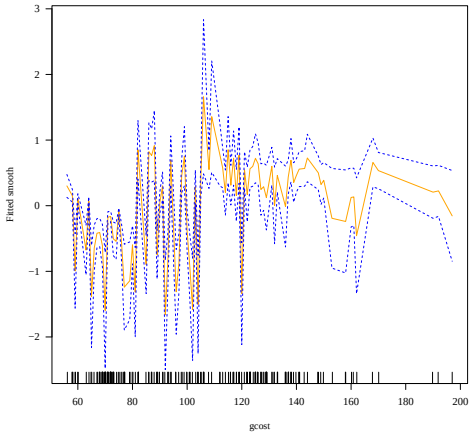


Fig. 14.1 Estimated component function with the $\boldsymbol{x}_{ij}$ facility. The function is not 'correct' for the reason explained in the text.

The estimated component function does not display properly when

```
> plot(as(tfit2, "vgam"), se = TRUE, lcol = "orange", scol = "blue",
       which.term = 1, xlab = "gcost", ylab = "Fitted smooth",
       noxmean = TRUE)
```

is used (Figure 14.1) because the term in the model's formula is the *difference* of two function values, not the function itself. That is, when the `"vgam"` plotting methods function is used it operates on the difference of the function values rather than the function itself.

One needs to do some processing in order to see what the function really looks like. Here is some quick-and-dirty code to plot the estimated function. It is not exactly generalizable, but it gives the idea on it can be done.

```
> X.lm  <- model.matrix(tfit2, type = "lm")
> ooo <- with(TravelMode2, sort.list(gcost.air))
> TravelMode3 <- TravelMode2[ooo, ]
```

Then

```
> X.vlm <- model.matrix(tfit2, type = "vlm")
> ind.start <-
    which(colnames(X.vlm) == "NS(gcost.air, gcost.bus, gcost.trn, gcost.car)1")
> ind.stop  <-  # This is rather manual
    which(colnames(X.vlm) == "NS(gcost.air, gcost.bus, gcost.trn, gcost.car)3")
> ind2  <- ind.start:ind.stop
> X.pred <-
    model.matrix( ~ -1 + NS(gcost.air, gcost.trn, gcost.bus, gcost.car),
                  data = TravelMode2)
> # Sort wrt the covariate, so that lines() effectively works
> X.pred <- X.pred[ooo, ]
> # For checking purposes
> fv <- X.pred %*% coef(tfit2)[ind2]
> plot(fv ~ gcost.air, data = TravelMode3, type = "l", col = "blue",
       ylab = "Smooth function", xlab = "gcost")
> with(TravelMode3, rug(gcost.air))
```

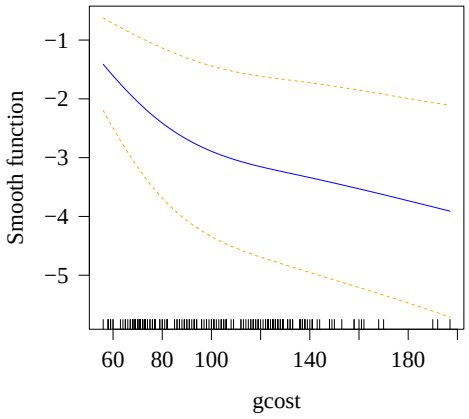


Fig. 14.2 Estimated component function with the $\boldsymbol{x}_{ij}$ facility. The (uncentred) function is 'correct'. The bands are pointwise ± 2 SEs about the estimate.

yields Figure 14.2. Actually, the figure includes pointwise ± 2 SE bands, and this is left as an exercise to the reader. The smooth function does seem to confirm that the function is linear. The negative slope agrees with intuition because if the cost of alternatives to car are more expensive than the cost by car then the alternatives become less likely to be chosen. In conclusion it is argued that Figure 14.2 is superior to Figure 3.1.

14.4 Derivatives of the Multinomial Logit Model

Using the multinomial logit model as an example, we now illustrate how to obtain derivatives such as $\partial \ell_i / \partial \eta_{ij}$ and $\partial \ell_i / \partial \boldsymbol{\beta}_j^{*T}$.

Firstly, note that the individual ℓ_i values can be obtained by calling `logLik(..., summation = FALSE)`. For example,

```
> pneumo <- transform(pneumo, let = log(exposure.time))
> fit <- vglm(cbind(normal, mild, severe) ~ let, multinomial, pneumo)
> logLik(fit, summation = FALSE)

         1        2        3        4        5        6        7        8
  -0.71306 -2.34168 -3.91040 -4.02642 -4.01892 -4.05910 -3.50547 -2.67549
```

Now to get $\partial \ell_i / \partial \boldsymbol{\beta}_j^{*T}$, consider the following code snippet.

```
> mu <- predict(fit, type = "response")
> w <- weights(fit, type = "prior")  # A 1-column matrix
> w <- as.vector(w)  # Convert into a vector
> extra <- fit@extra
>
> # Choose  which eta here
> jay <- 2  # Any value from the set 1:M where M == npred(fit)
> jay <- 1  # Any value from the set 1:M where M == npred(fit)
>
> y <- depvar(fit)
```

```
> dl.deta <- eval(fit@family@deriv)  # Needs "y", "w", "extra", etc.
>
> deta.dbetaj <-
    vlm2lm.model.matrix(model.matrix(fit, type = "vlm"),
                        Hlist = constraints(fit),
                        which.linpred = jay)
>
> (dl.dbetaj <- dl.deta[, jay] * deta.dbetaj)

      (Intercept):1    let:1
  1:1      0.710472  1.24891
  2:1      0.619607  1.67793
  3:1     -2.502267 -7.67709
  4:1     -0.929622 -3.08094
  5:1     -0.608315 -2.13613
  6:1      2.728082 10.02925
  7:1     -0.078339 -0.29993
  8:1      0.060382  0.23800

> colSums(dl.dbetaj)  # For checking purposes; should be all 0s

  (Intercept):1         let:1
     6.4459e-11    1.5420e-10
```

Certain variables such as `y`, `extra`, `mu` need to be assigned before the `@deriv` slot is evaluated. The function `vlm2lm.model.matrix()` chooses a subset of the big model matrix $\mathbf{X}_{\text{VLM}}$ depending on the value of the argument `which.linpred` (which specifies j). The matrix `deta.betaj` is therefore a subset of $\mathbf{X}_{\text{VLM}}$. The matrix `dl.deta` is $n \times M$. The chain rule is used to obtain the derivatives with respect to the $\beta^*_{(j)k}$s.

14.5 A Constrained Multinomial Logit Model

Suppose we want to constrain the probabilities of a multinomial logit model to be bounded. How might this be done? The answer is that constraint matrices and offsets can be combined. Suppose that $0 < p_j < p_{\max}$ is desired for all $j = 1, \ldots, M$.

The solution presented here only applies to *one* value of j rather than them all. However, this disadvantage can be weakened by choosing j to be level corresponding to the highest fitted value.

The formulas to use are

$$\begin{aligned} \omega &= \log\left(\frac{1}{p_{j,\max}} - 1\right) - \log(M-1), && (14.8)\\ \boldsymbol{\Omega} &= \omega \cdot \left(\boldsymbol{e}_j^T \otimes \mathbf{1}_n\right), && (14.9)\\ \mathbf{H}_k &= \mathbf{1}_M, \quad k = 1, \ldots, p, && (14.10) \end{aligned}$$

so that $0 < p_j < p_{\max}$. It can be seen that there is a parallelism assumption applied to all the η_j, and that the matrix of offsets, $\boldsymbol{\Omega}$, has all columns equal to $\omega \mathbf{1}_n$ except for the jth column, which is a column of 0s. The justification for (14.8)–(14.10) is that

$$p_j = \frac{e^{\eta_j}}{e^{\eta_j}\left[(M-1)\,e^{\omega}+1\right]+1},$$

and letting $\eta \to \infty$ leads to

$$p_j \;\to\; \frac{1}{(M-1)\,e^{\omega}+1}$$

which can be solved for ω. However, one consequence is

$$p_s \;\to\; \frac{e^{\omega}}{(M-1)\,e^{\omega}+1}$$

as $\eta \to \infty$ which can be undesirable ($s \neq j$).

As an illustration, for example, for $p_{\max} = 0.8$ then some values of ω are

```
> omega <- function(M = 1) log(1 / 0.8 - 1) + log(M-1)
> c(omega(2), omega(3), omega(4))

   [1] -1.38629 -0.69315 -0.28768
```

Here is the above illustrated using the `pneumo` data set.

```
> data("pneumo")
> pneumo <- transform(pneumo, let = log(exposure.time))
> my0 <- rep( 0.0 , nrow(pneumo))
> M <- 2
> myoffset <- rep(omega(M), nrow(pneumo))
> fit1 <- vglm(cbind(normal, mild, severe) ~
               offset(cbind(my0, myoffset)) + let,
               multinomial(parallel = TRUE), data = pneumo)
> coef(fit1, matrix = TRUE)

               log(mu[,1]/mu[,3]) log(mu[,2]/mu[,3])
   (Intercept)            10.7091            10.7091
   let                    -2.6935            -2.6935

> par(mfrow = c(1, 1))
> matplot(with(pneumo, let), fitted(fit1), type = "b",
          ylab = "Fitted value", col = 1:3, las = 1,
          xlab = "Log exposure time", main = "")
```

This produces Fig. 14.3. The maximum probability allowed is about 80% whereas it is almost 100% in the constrained model (cf. Fig. 14.3(b)). The undesirable feature here is that p_2 is much larger than its unconstrained value (0.2 versus almost 0).

14.5.1 A Variant Solution

Another possible solution is as follows.

The formulas to use are

$$\begin{aligned}
\omega &= \text{logit}\, p_{j,\max} + \log(M-1), && (14.11)\\
\boldsymbol{\Omega} &= \omega \cdot \left(\mathbf{1}_M^T \otimes \mathbf{1}_n\right), && (14.12)\\
\mathbf{H}_k &= \mathbf{1}_M, \qquad k = 1, \ldots, p, && (14.13)
\end{aligned}$$

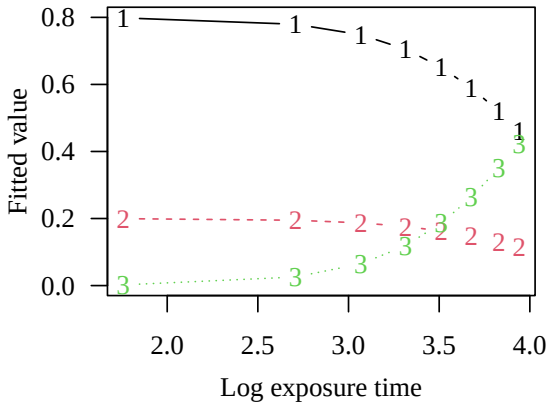


Fig. 14.3 Constraining the probabilities by an upper bound.

so that $0 < p_j < p_{\max}$. As before, there is a parallelism assumption applied to all the η_j, and that the matrix of offsets, $\boldsymbol{\Omega}$, has all columns equal to $\mathbf{0}$ except for the jth column, which is $\omega \mathbf{1}_n$. The justification for (14.11)–(14.13) is that

$$p_j \;=\; \frac{e^{\omega+\eta_j}}{e^{\eta_j}\left[M-1+e^{\omega}\right]+1},$$

and letting $\eta \to \infty$ leads to

$$p_j \;\to\; \frac{e^{\omega}}{M-1+e^{\omega}}$$

which can be solved for ω. However, one consequence is

$$p_s \;\to\; \frac{1}{M-1+e^{\omega}}$$

as $\eta \to \infty$ which can be undesirable ($s \neq j$).

As an illustration, for example, for $p_{\max} = 0.8$ then some values of ω are

```
> omega2 <- function(M = 1) logitlink(0.8) + log(M-1)
> c(omega2(2), omega2(3), omega2(4))

   [1] 1.3863 2.0794 2.4849
```

Here is the above illustrated using the `pneumo` data set.

```
> data("pneumo")
> pneumo <- transform(pneumo, let = log(exposure.time))
> my0 <- rep( 0.0 , nrow(pneumo))
> M <- 2
> myoffset <- rep(omega2(M), nrow(pneumo))
> fit2 <- vglm(cbind(normal, mild, severe) ~
               offset(cbind(myoffset, my0)) + let,
               multinomial(parallel = TRUE), data = pneumo)
> coef(fit2, matrix = TRUE)

               log(mu[,1]/mu[,3]) log(mu[,2]/mu[,3])
```

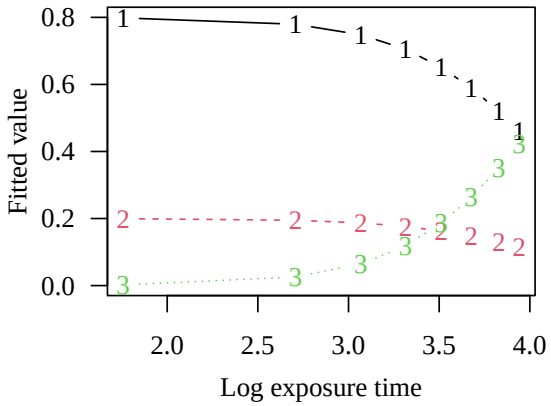


Fig. 14.4 Constraining the probabilities by an upper bound—using a variant method.

```
   (Intercept)             9.3228             9.3228
   let                    -2.6935            -2.6935

> par(mfrow = c(1, 1))
> matplot(with(pneumo, let), fitted(fit2), type = "b",
          ylab = "Fitted value", col = 1:3, las = 1,
          xlab = "Log exposure time", main = "")
```

This produces Fig. 14.4. The maximum probability allowed is about 80% whereas it is almost 100% in the constrained model (cf. Fig. 14.3(b)). The undesirable feature here is that p_2 is much larger than its unconstrained value (0.2 versus almost 0).

Bibliographic notes

Hensher et al. (2015) is a large applied book on choice modelling, albeit based on the software Nlogit. Fullerton and Xu (2016) describe a typology of categorical models involving parallelism and reduced-rank regression. Agresti (2019) is an introductory book on categorical data analysis and Fagerland et al. (2017) looks at the analysis of contingency tables.

Package gofcat computes a several goodness-of-fit measures for categorical response models. They include the Brant, Hosmer-Lemeshow, and Lipsitz tests, as well as the LRT. See Ugba (2022).

Package EffectStars2 plots the estimated coefficients in the form of a star-like graphic according to the different groups (Tutz and Schauberger, 2013).

Exercises

In general, any form of exercise, if pursued continuously, will help train us in perseverance.
—Mao Zedong

Ex. 14.1. To compare two groups, a useful summary refers to latent variables y_1^* and y_2^* that underlie responses for the two groups. Suppose these are independent, made at any particular setting of the explanatory variables. The summary measure is $\Pr(Y_2^* > Y_1^*)$. When the indicator variable in the model takes value 0 for group 1 and 1 for group 2 and has estimated coefficient $\widehat{\beta}$ in a `propodds()` model show that the probability can be estimated by [Agresti and Kateri (2017)]

$$\widehat{\Pr}(Y_2^* > Y_1^*) \;=\; \frac{\exp(\widehat{\beta}/\sqrt{2})}{1+\exp(\widehat{\beta}/\sqrt{2})}. \tag{14.14}$$

Ex. 14.2. Obtain the SE bands of Figure 14.2.

Chapter 15
Complements: Quantile and Expectile Regression

15.1 Introduction

This chapter looks at some more topics related to quantile and expectile regression, especially as related to the VGAM package.

15.1.1 Fitted Values of the LMS-BCN Model

Given an `lms.bcn()` model, extracting different quantiles from the quantiles used when first fitting the model can be obtained easily. However, one has to be careful whether the object is of class `"vgam"` or `"vglm"`. Below, we obtain the 10% and 80% quantiles of a 60 year old, noting that the default fitted quantiles are 25%, 50% and 75%.

```
> fit <- vgam(BMI ~ s(age, df = c(4, 2)), trace = FALSE,
              lms.bcn(zero=1), data = bmi.nz)
>
> # Correct for "vgam" objects, but not very elegant
> predict(fit, newdata = data.frame(age = 60))

      lambda     mu loglink(sigma)
   1 -0.65896 27.009        -1.8427

> # Correct for "vgam" objects, but not very elegant
> fit@family@linkinv(eta = predict(fit, data.frame(age = 60)),
                     extra = list(percentiles = c(10, 80)))

        10%    80%
   1 22.324 31.052
```

Note that `predict()` gives an incorrect answer when a `"vgam"` object is coerced into a `"vglm"` object. This is because `"vgam"` objects have each η_j made up of the sum of parametric and nonparametric (linear and nonlinear) components, and the latter is ignored upon the conversion.

```
> # Incorrect for "vgam" objects
> predict(as(fit, "vglm"), newdata = data.frame(age = 60))

      lambda     mu loglink(sigma)
```

```
   1 -0.65896 26.371        -1.8313

> # Incorrect for "vgam" objects
> predict(as(fit, "vglm"), percentiles = c(10, 50),
          newdata = data.frame(age = 60), type = "response")

        10%    50%
   1 21.752 26.371
```

Here is an example involving regression splines.

```
> fit2 <- vglm(BMI ~ bs(age, df = 4), trace = FALSE,
               lms.bcn(zero = c(1, 3)), data = bmi.nz)
>
> # Correct for "vglm" objects, but not very elegant
> predict(fit2, newdata = data.frame(age = 60))

       lambda     mu loglink(sigma)
   1 -0.64529 27.085        -1.8439

> # Correct for "vglm" objects
> predict(fit2, percentiles = c(10, 80),
          newdata = data.frame(age = 60), type = "response")

        10%   80%
   1 22.386 31.13
```

Incidentally, this is for people from the original fit:

```
> head(fitted(fit2, percentiles = c(10, 80)), 3)

          10%    80%
   [1,] 21.048 29.270
   [2,] 21.716 30.198
   [3,] 22.031 30.637

> head(fitted(fit2), 3)  # Default quantiles

        25%    50%    75%
   1 22.969 25.466 28.443
   2 23.698 26.274 29.345
   3 24.042 26.656 29.772
```

15.1.2 Qlink Link Functions for Parametric QRl

In the usual quantile regression setting the distribution of the response given the explanatory variables is unspecified. In Miranda-Soberanis and Yee (2019) the distribution is specified and they introduce new link functions to *directly* model specified quantiles of seven 1–parameter continuous distributions. They transform certain prespecified quantiles to become linear or additive predictors. This is an example of parametric quantile regression. The quantile crossing problem can be avoided by enforcing parallelism constraint matrices. The new link functions are in VGAMextra 0.0-2 or higher. The distributions have support on $(0, \infty)$, $(0, 1)$ or $(-\infty, \infty)$, therefore there are three links currently implemented and they are $\eta_\tau = \log \xi_\tau$, $\text{logit}\, \xi_\tau$ and ξ_τ.

The names of the link functions end in `"Qlink"`, and one needs `Q.reg()` to preprocess the response. The distributions currently implemented include `benini1()`, `exponential()`, `gamma1()`, `maxwell()`, `rayleigh()`. `topple()`, `normal1sdff()`. The last one resides in VGAMextra. Some of these distributions have η_j which is parallel with respect to $x_2, \ldots, x_d$ so that τ_j only affects the intercept of η_j. Regardless, setting `parallel = FALSE` ~ 1 for all the models means that the linear/additive predictors are parallel with respect to $x_2, \ldots, x_d$, hence there is no quantile crossing problem.

Here is a simple example.

```
> set.seed(1)
> maxdata <- data.frame(x2 = sort(runif(n <- 200)))  # Sorted for plotting
> ratefun <- function(x) exp(2 -  6 * sin(2 * x - 0.2) / (x + 0.5)^2)
> # Generate the data:
> maxdata <- transform(maxdata, y = rmaxwell(n, rate = ratefun(x2)))
> my.tau <- c(0.25, 0.50, 0.75)  # Use these quantiles
```

```
> library("VGAMextra")
> mydof <- 4  # Effective degrees of freedom of the smoothing spline
> fit1 <-
    vgam(Q.reg(y, pvector = my.tau) ~ s(x2, df = mydof), data = maxdata,
         maxwell(link = maxwellQlink(p = my.tau),
                 type.fitted = "Qlink"))
```

```
> plot(y ~ x2, maxdata, main = "", xlab = expression(x[2]), las = 1,
       ylab = "y")
> with(maxdata, matlines(x2, fitted(fit1), col = "blue", lwd = 1.5,
                         lty = 1))
```

This gives Figure 15.1. The empirical proportions are

```
> 100 * colMeans(depvar(fit1, drop = TRUE) < fitted(fit1))

  [1] 26.0 50.0 73.5
```

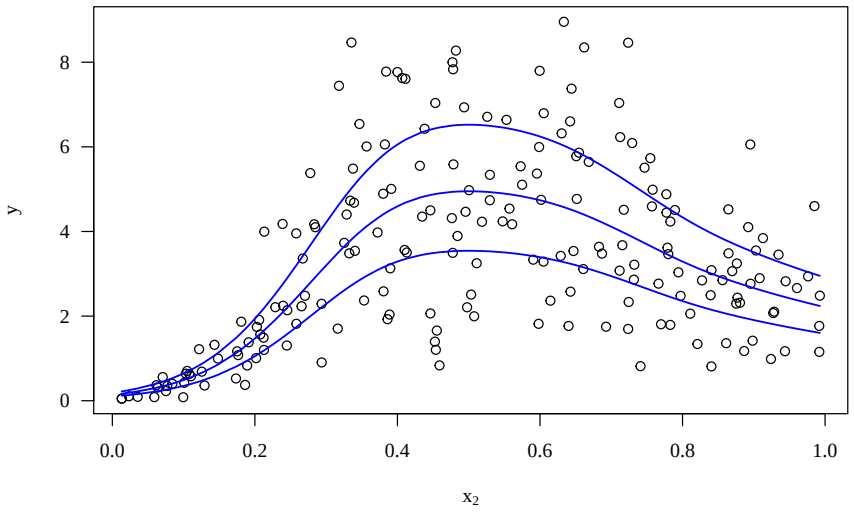
which agree well with `my.tau`.

If `fit` is a `"Qlink"`-type object then `fitted(fit)` and `predict(fit, type = "response")` are the same.

Bibliographic notes

A recent book on quantile regression is Koenker et al. (2018). Some further information about expectiles can be found in Schnabel and Eilers (2009), De Rossi and Harvey (2009), Schnabel (2011).

Fig. 15.1 Simulated Maxwell data including the fitted quantile functions from `fit1`.



Exercises

Ex. 15.1. Parallelism

(a) For `exponential()` using `expQlink()` show that the η_j are parallel. Hint: for $Y \sim$ exponential(λ), with $\lambda > 0$ a rate parameter, the density and CDF are given by $f(y;\lambda) = \lambda\, e^{-\lambda y}$ and $F(y;\lambda) = 1 - e^{-\lambda y}$.

(b) For `maxwell()` using `maxwellQlink()` show that the η_j are parallel. Hint: this distribution has density $f(y;a) = \sqrt{2/\pi}\, a^{3/2}\, y^2 \exp(-a\, y^2/2)$ and CDF

$$F(y;a) = \sqrt{\frac{2}{a} \cdot \texttt{qgamma}(\tau, 1.5)}.$$

(c) For `rayleigh()` using `rayleighQlink()` are the η_j are parallel? Show your working. Hint: this distribution has CDF $F(y;b) = b\sqrt{-2\log(1-\tau)}$.

(d) For `benini1()` using `benini1Qlink()` are the η_j are parallel? Show your working. Hint: this distribution has CDF

$$F(y;s) = y_0 \cdot \exp\left(\sqrt{\frac{-\log(1-\tau)}{s}}\right),$$

where y_0 is known (given) and (y_0, ∞) is the support of the distribution.

Chapter 16
Complements: Extremes

16.1 Using Confidence Intervals Based on Profile Likelihoods

Section 3.2 describes `confint()` based on profile likelihoods as an alternative to Wald intervals. Here, we mimic the GPD analysis given in Coles (2001, Sect. 4.4.1) based on some rainfall data in south-west England during the 20th century. The analysis is based on 152 exceedances of the the threshold value of 30.

Firstly let's obtain an appropriate subset of the data in the form of a data frame.

```
> data(rain, package = "ismev")
> mythresh <- 30
> rain30 <- data.frame(y = rain[rain > mythresh])
> summary(unlist(rain30))  # Only one variable here

      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      30.2    32.0    35.3    39.1    42.0    86.6

> dim(rain30)

   [1] 152   1
```

To keep things simple, let's use identity links to estimate the two parameters.

```
> gpdfit <- vglm(y ~ 1, gpd(threshold = mythresh, lscale = "identitylink",
                            lshape = "identitylink"),
                 crit = "coef", data = rain30)
> coef(gpdfit, matrix = TRUE)

               scale  shape
   (Intercept) 7.4403 0.1845
```

The MLEs agree. As for the SEs, there is a slight difference, possibly because Coles (2001) might use OIMs instead of EIMs:

```
> round(vcov(gpdfit), dig = 4)

                (Intercept):1 (Intercept):2
   (Intercept):1        0.8628       -0.0580
   (Intercept):2       -0.0580        0.0092
```

Now to compute approximate 95% CIs for the shape parameter ξ, these are

```
> confint(gpdfit, "(Intercept):2", method = "wald")  # Cf. [-0.014, 0.383]

                     2.5 % 97.5 %
  (Intercept):2 -0.0038056 0.3728

> confint(gpdfit, "(Intercept):2", method = "profile")  # Cf. [0.019, 0.418]

     2.5 %  97.5 %
  0.01362 0.41545
```

There is some discrepancy in both types of intervals with Coles (2001), nevertheless this is not major. Regardless, it appears that $0 < \xi$.

Bibliographic Notes

Belzile et al. (2022) is a recent review of EVA software with a focus on the numerical challenges involved.

Chapter 17
Complements: Generally Altered, Inflated, Truncated and Deflated Regression

This chapter is restricted to count responses; GAITD regression for continuous Y is currently under development and is not yet described here.

17.1 The ZMP

This section highlights the fact that some of the earliest VGAM family functions can be used to fit the zero-*modified* (or maybe the zero-*deflated*) variant even though it seemingly only fits the zero-*inflated* version. We will use the Poisson as the parent distribution here but the idea can apply to others such as the NBD.

The zero-*modified* Poisson (ZMP) distribution may be written

$$f_\mu(y) \;=\; \nu\, f_\pi(y;\boldsymbol{\theta}_\pi) + (1-\nu)\cdot I(y=0), \qquad y=0,1,\ldots, \tag{17.1}$$

where $f_\pi(y)$ is the usual Poisson PMF. Now $0 \le \nu \le 1 + 1/[1-f_\pi(0)]$ means that it is a combination of the ZIP and *zero-deflated Poisson* (ZDP). In VGAM the key is to set the link function to be something like `identitylink` or `extlogitlink()` with argument `max` set manually to some suitable value. Some notes:

- The PMF (17.1) has a term $\nu\, f_\pi(y)$ which is better than using $\phi\, f_\pi(y)$ because, even though $\phi = 1-\nu$, one doesn't want $\phi\, f_\pi(y) < 0$ so this implies $0 \le \phi$ is needed. That is, for `zipoisson()`, the PMF written as

$$f_\iota(y) \;=\; (1-\phi)\, f_\pi(y;\boldsymbol{\theta}_\pi) + \phi\cdot I(y=0), \qquad y=0,1,\ldots, \tag{17.2}$$

  has the disadvantage that it must have $1-\phi \ge 0$ in order to avoid negative probabilities when $y \ne 0$. Thus using (17.1) is better than (17.2).
- Care is needed when fitting the ZMP using `zipoisson()` and `zipoissonff()` because problems might occur when iterations get close to or go past the boundary of the parameter space, especially when there are covariates. Regarding the previous bullet point, (17.1) has ν matching with the quantity/argument `onempstr0`; hence it can be argued that maybe `zipoissonff()` is more suitable than `zipoisson()`.
- The ZDP can be written

$$f_\delta(y) \;=\; (1+\psi)\, f_\pi(y;\boldsymbol{\theta}_\pi) - \psi\cdot I(y=0), \qquad y=0,1,\ldots, \tag{17.3}$$

where $0 \leq \psi \leq f_\pi(0)/[1 - f_\pi(0)]$. That is, $\nu = 1 + \psi$.

- Summarizing, and writing "*" for the parent or base distribution, the ZM* contains the ZI*, ZD* and ZT* as special cases. In fact, one can think of these three models as nonoverlapping and their combination is the ZM*. Specifically,
  - $\nu = 0$ is a degenerate distribution at 0;
  - $\nu \in (0, 1)$ is the ZI*;
  - $\nu = 1$ is the usual parent distribution *;
  - $\nu \in (1, 1/[1 - f_\pi(0)])$ is the ZD*; and
  - $\nu = 1/[1 - f_\pi(0)]$ is the ZT*.
- In the GAITD regression method below, the user must specify the direction: either inflation or deflation. Hence strictly speaking, one cannot fit the 'modified' model because this combines both operators into one operator. Instead, one can fit the equivalent of the 'modified' model by manually supplying the direction. And in the case of truncation, one can specify values of a set $\mathcal{T}$ of truncated values so that this requires manual specification too.

Here is a numerical example involving simulated data illustrating `zipoissonff()` fitting the ZDP.

```
> set.seed(1)
> nn <- 1000; lambda <- 1
> (deflat.limit <- -1 / expm1(lambda))  # ZDP boundary

   [1] -0.58198

> pstr0 <- deflat.limit / 2  # Moderate deflation
> 1 - pstr0  # \nu; this is estimated below; aka onempstr0

   [1] 1.291

> deflatpoisdata <-
      data.frame(y1 = rzipois(nn, lambda, pstr0 = pstr0))
> zdpfit1 <- vglm(y1 ~ 1, data = deflatpoisdata,
                  zipoissonff(lonempstr0 = "identitylink"))
> coef(zdpfit1, matrix = TRUE)

              loglink(lambda) onempstr0
   (Intercept)        0.023934    1.2732

> Coef(zdpfit1)

      lambda onempstr0
      1.0242    1.2732

> coef(summary(zdpfit1))

                Estimate Std. Error  z value    Pr(>|z|)
   (Intercept):1 0.023934   0.042419  0.56421  5.7261e-01
   (Intercept):2 1.273161   0.036413 34.96461 7.7669e-268
```

Now change the link function to `extlogitlink()`:

```
> zdpfit2 <-
      vglm(y1 ~ 1, data = deflatpoisdata,
           zipoissonff(lonempstr0 = "extlogitlink(max = 1.4)"))
> coef(zdpfit2, matrix = TRUE)

              loglink(lambda) extlogitlink(onempstr0, min = 0, max = 1.4)
```

```
   (Intercept)        0.023934                                      2.3063

> Coef(zdpfit2)

      lambda onempstr0
      1.0242    1.2732

> coef(summary(zdpfit2))

                Estimate Std. Error z value   Pr(>|z|)
   (Intercept):1 0.023934   0.042419 0.56421 5.7261e-01
   (Intercept):2 2.306340   0.315648 7.30669 2.7381e-13
```

Both fits have identical estimates of ν and λ.

17.2 Heaped and Seeped Data

A very common aberration in retrospective self-reported survey data is *digit preference* (*heaping*) whereby multiples of 10 or 5 upon rounding are measured in excess, creating spikes in spikeplots. Handling this problem requires great flexibility. This section serves only as motivation for a new technique called *GAITD regression* described in Section 17.3. GAITD regression applies to data not necessarily contaminated by measurement error, hence it is only one possible cause of such data.

17.3 GAITD Regression

VGAM 1.1-6 and higher has functions which implement GAITD regression.

GAITD regression is an attempt to generalize four popular models most commonly based on the Poisson distribution and known by the acronyms ZIP, ZAP, ZTP and ZDP: the zero-inflated, zero-altered, zero-truncated (positive) and zero-deflated Poisson respectively. Over the past two decades they have gained wide acceptance and popularity among practitioners with count responses, e.g., Kleiber and Zeileis (2008), Zuur et al. (2012), Cameron and Trivedi (2013), Agresti (2015). Much of this has been driven by the realization that excess 0s are commonly encountered in real-life data, and to a lesser extent, a deficiency or the impossibility of recording 0 values in other types of data.

All four types of operators ("A", "I", "T" and "D") have found rich applications in both Poisson and binomial distribution forms. The ZIP has been attributed to Lambert (1992), and the ZAP is often described as a hurdle model. In capture–recapture experiments the absence of 0s leads to conditional models (e.g., Otis et al., 1978; Yee et al., 2015) such as the positive Bernoulli distribution or zero-truncated binomial (ZTB); occupancy models (e.g., MacKenzie et al., 2002) also make use of them.

Let $\mathcal{R}$ be the support of the parent (base) distribution, e.g., $\{0, 1, \ldots\}$ for the Poisson. GAITD regression extends previous work such as the above in three directions:

(I) Any subset of the support can be altered, inflated, deflated or truncated, cf. treating only the singleton $\{0\}$ as special. The first three are denoted $\mathcal{A}$ and $\mathcal{I}$ and $\mathcal{D}$ with finite cardinality. The truncation set $\mathcal{T}$ may be innumerable so it is merely a proper subset of $\mathcal{R}$.

(II) Rather than allowing only one of $\mathcal{A}$, $\mathcal{I}$ $\mathcal{T}$ and $\mathcal{D}$, the four operators are combined into a single model and are allowed to operate concurrently. This confers greater versatility and a holistic approach. The $\mathcal{A}$, $\mathcal{I}$, $\mathcal{T}$ and $\mathcal{D}$ are mutually disjoint.

(III) Utilizing (I) and (II) on $\mathcal{A}$, $\mathcal{I}$ and $\mathcal{D}$, parametric and nonparametric forms are spawned, hence there are 7 types of special values. These are further combined into a single super-mixture model, called the *GAITD combo* for modesty.

(IV) Although we develop (I)–(II) mainly for 1-parameter count parent distributions (Poisson, logarithmic and zeta) our work applies to other distributions such as the 2-parameter negative binomial and not necessarily discrete.

These directions allow an important unification of the four operators into a single model. A finite mixture distribution approach is taken for this.

Applications that utilize (I) can arise from many situations. For example, generally-truncated count distributions can arise from a surprising range of scenarios and may be justified when certain support values have no possibility of occurring, for example, tetraphobia in East Asian culture and triskaidekaphobia in Western culture that create structural absences in certain sampling units. Buildings that omit the 4th floor and public passenger seating that omit row 13 are everyday examples.

Bibliographic notes

This chapter summarizes details from Yee and Ma (2024) and Yee et al. (2025). It is anticipated that content here will be put into the second edition of Yee (2015) to replace Chapter 17 on ZI-, ZA- and ZT-distributions.

An overview of ZI-models with particular emphasis on the ZIP is given in Wagh and Kamalja (2018).

Exercises

. . . the East is rising and the West is declining. . .
—Xi Jinping

Ex. 17.1. An alternative parameterization of the zero-inflated Poisson (ZIP) is

$$\Pr(Y = 0) = p, \tag{17.4}$$

$$\Pr(Y = y) = \frac{1-p}{1-e^{-\lambda}} \cdot \frac{e^{-\lambda}\lambda^y}{y!}, \quad y = 1, 2, \ldots. \tag{17.5}$$

(a) Derive the expected information matrix.

(b) Based on (17.4)–(17.5), write the form of the PMF for a 0-inflated 1-parameter count distribution in general. How does its expected information matrix change?

Ex. 17.2. By carefully choosing special values and probabilities, use the software to spikeplot a GAITD count distribution that is bimodal. Then do the same for trimodality and up to 7 modes.

Chapter 18
Complements: On VGAM Family Functions

18.1 Character Input for the zero Argument

VGAM 1.0-1 and higher allows the `zero` argument to have *character* input. For example,

```
> set.seed(123); n <- 1000
> ldata <- data.frame(x2 = runif(n))
> ldata <- transform(ldata, y1 = rlogis(n, loc = 0+5*x2, scale = exp(2)))
> ldata <- transform(ldata, y2 = rlogis(n, loc = 0+5*x2, scale = exp(0+1*x2)))
> ldata <- transform(ldata, w1 = runif(n))
> ldata <- transform(ldata, w2 = runif(n))
> fit2 <- vglm(cbind(y1, y2) ~ x2,
  #            logistic(zero = "location1"),
  #            logistic(zero = "location2"),
  #            logistic(zero = "location"),  # All "location" parameters
  #            logistic(zero = "scale1"),
  #            logistic(zero = "scale2"),
  #            logistic(zero = "scale*"),  # Wildcards do not work
               logistic(zero = c("location", "scale2")),
  #            logistic(zero = c("LOCAT", "scale2")),
  #            logistic(zero = c("LOCAT")),
  #            trace = TRUE,
  #            weights = cbind(w1, w2),
               weights = w1,
               data = ldata)
> coef(fit2, matrix = TRUE)

             location1 loglink(scale1) location2 loglink(scale2)
  (Intercept)    2.7247         1.90498    2.2067         0.67507
  x2             0.0000         0.15383    0.0000         0.00000
```

In the above, all the various `zero` examples work. Those with `LOCAT` issue a warning that that value is unmatched. Importantly, the parameter names are `c("location1", "scale1", "location2", "scale2")` because there are 2 responses. Note that `zero` does not accept wildcards (cf. Linux operating system), e.g., `"location*"` does not work. However, `"location"` does work; it means that *all* location parameters are intercept-only.

Yee (2015) described `zero` for only numerical input. Allowing character input is particularly important when the number of parameters cannot be determined without having the actual data first. For example, with time series data, an ARMA(p, q)

process might have parameters $\theta_1, \ldots, \theta_p$ which should be intercept-only by default. Then specifying a numerical default value for `zero` would be too difficult (there are the drift and scale parameters too). However, it is possible with the character representation: `zero = "theta"` would achieve this.

Here are some further notes.

1. Many VGAM family functions have had their `zero` default value converted to the character representation—the advantage being that it is more readable.
2. It is not advised to mix numeric with character input, e.g., `c("location1", 3)` is transformed by R into `c("location1", "3")` which is probably not what the user really intends.
3. When programming a VGAM family function that allows character input, the variable `predictors.names` must be assigned correctly. This is done in the `initialize` slot. For example,

```
> logistic()@initialize

   expression({
       temp5 <- w.y.check(w = w, y = y, ncol.w.max = Inf, ncol.y.max = Inf,
           out.wy = TRUE, colsyperw = 1, maximize = TRUE)
       w <- temp5$w
       y <- temp5$y
       ncoly <- ncol(y)
       M1 <- 2
       extra$ncoly <- ncoly
       extra$M1 <- M1
       M <- M1 * ncoly
       mynames1 <- param.names("location", ncoly, skip1 = TRUE)
       mynames2 <- param.names("scale", ncoly, skip1 = TRUE)
       parameters.names <- c(mynames1, mynames2)[interleave.VGAM(M,
           M1 = M1)]
       predictors.names <- c(namesof(mynames1, "identitylink", earg = list(
           theta = , inverse = FALSE, deriv = 0, short = TRUE, tag = FALSE),
           tag = FALSE), namesof(mynames2, "loglink", earg = list(
           theta = , bvalue = NULL, inverse = FALSE, deriv = 0,
           short = TRUE, tag = FALSE), tag = FALSE))[interleave.VGAM(M,
           M1 = M1)]
       if (!length(etastart)) {
           if (1 == 1) {
               locat.init <- y
               scale.init <- sqrt(3) * apply(y, 2, sd)/pi
           }
           else {
               locat.init <- scale.init <- NULL
               for (ii in 1:ncoly) {
                   locat.init <- c(locat.init, median(rep(y[, ii],
                     w[, ii])))
                   scale.init <- c(scale.init, sqrt(3) * sum(w[,
                     ii] * (y[, ii] - locat.init[ii])^2)/(sum(w[,
                     ii]) * pi))
               }
           }
           locat.init <- matrix(if (length(NULL))
               NULL
           else locat.init, n, ncoly, byrow = TRUE)
           if ("identitylink" == "loglink")
               locat.init <- abs(locat.init) + 0.001
           scale.init <- matrix(if (length(NULL))
```

```
            NULL
        else scale.init, n, ncoly, byrow = TRUE)
        etastart <- cbind(theta2eta(locat.init, "identitylink",
            earg = list(theta = , inverse = FALSE, deriv = 0,
                short = TRUE, tag = FALSE)), theta2eta(scale.init,
            "loglink", earg = list(theta = , bvalue = NULL, inverse = FALSE,
                deriv = 0, short = TRUE, tag = FALSE)))[, interleave.VGAM(M,
            M1 = M1)]
    }
})
```

In the past, most family functions checked that `zero`, if it was not a `NULL`, was numeric; this block of code should be commented out.

4. The `zero` argument also accepts `""` and `NA` as alternatives to `NULL` (to mean 'none').

18.2 Link Functions

Calls of the form `linkfun(`θ`, deriv = 2, inverse = TRUE)` return the following:

$$\frac{\partial^2\theta}{\partial\eta^2} = -\left(\frac{\partial\eta}{\partial\theta}\right)^{-3}\frac{\partial^2\eta}{\partial\theta^2}. \tag{18.1}$$

This follows from the basic property

$$\frac{\partial\theta}{\partial\eta} = \left(\frac{\partial\eta}{\partial\theta}\right)^{-1}.$$

All link functions in VGAM have been converted to give the output as summarized in Table 18.1.

18.3 Writing Some Methods Functions

The generic function `summary()`, when applied to a `"vglm"` object, calls the methods function `summaryvglm()`, which computes quantities such as the SEs and Wald statistics, then these are printed by the methods function `show.summary.vglm()`. The output that appears from this is the same for all 150+ VGAM family functions. However, for many types of models, it would be useful for certain model-specific quantities to be printed out in the `summary()` too. Here are some examples.

- In a `binom2.or(zero = 3)` model the estimated odds ratio is usually of particular interest and should be printed out.
- For a proportional odds model, $\exp\{\beta^*_{(1)k}\}$ is the odds ratio for $\Pr(Y \le j)$, from a change in x_k to $x_k + 1$, keeping all other variables fixed. Hence exponentiating the regression coefficients is useful to some practitioners.
- For the N_2 distribution, the default is to model the correlation parameter ρ using a `"rhobit"` link and as intercept-only. If so, then it is informative to print out $\widehat{\rho}$.

Table 18.1 Calls to a link function in VGAM, called `linkfun()` here. Notes: (1) Cases 1–2 are inverses, cases 3–4 are reciprocals, cases 5–6 are *not* reciprocals but are related by (18.1). (2) Some big changes occurred for VGAM version 0.9-9 (2015-07); previous to that cases 3–4 were switched, and cases 5–6 used to be reciprocals of each other (a bug). (3) Case 2 is the only one where the argument `theta` is actually η.

Case	Call	Returns	Old case
1.	`linkfun(`θ`)`	$\eta = g(\theta)$	
2.	`linkfun(`η`, inverse = TRUE)`	$\theta = g^{-1}(\eta)$	
3.	`linkfun(`θ`, deriv = 1)`	$\frac{d\eta}{d\theta}$	4.
4.	`linkfun(`θ`, deriv = 1, inverse = TRUE)`	$\frac{d\theta}{d\eta}$	3.
5.	`linkfun(`θ`, deriv = 2)`	$\frac{d^2\eta}{d\theta^2}$	
6.	`linkfun(`θ`, deriv = 2, inverse = TRUE)`	$\frac{d^2\theta}{d\eta^2}$	5.
7.	`linkfun(`θ`, deriv = 3)`	$\frac{d^3\eta}{d\theta^3}$	
8.	`linkfun(`θ`, deriv = 3, inverse = TRUE)`	$\frac{d^3\theta}{d\eta^3}$	

- In capture–recapture models such as `posbernoulli.tb()` the population size estimate $\widehat{N}$ and its SE is often the final goal of the analysis.

Fortunately, it is also possible for programmers to write methods functions that print extra output for functions such as `summary()`. This section describes how this may be done. We take as two examples the functions `summary()` and `margeff()`, when applied to regression models for categorical responses. We have

```
> multinomial()@vfamily
   [1] "multinomial"     "VGAMcategorical"
> acat()@vfamily
   [1] "acat"            "VGAMordinal"     "VGAMcategorical"
> cratio()@vfamily
   [1] "cratio"          "VGAMordinal"     "VGAMcategorical"
> sratio()@vfamily
   [1] "sratio"          "VGAMordinal"     "VGAMcategorical"
> cumulative()@vfamily
   [1] "cumulative"      "VGAMordinal"     "VGAMcategorical"
```

which is FYI only. What is really implemented is

```
setClass("VGAMcategorical",     contains = "vglmff")

setClass("VGAMordinal",         contains = "VGAMcategorical")
setClass("multinomial",         contains = "VGAMcategorical")

setClass("acat",                contains = "VGAMordinal")
setClass("cumulative",          contains = "VGAMordinal")
setClass("cratio",              contains = "VGAMordinal")
setClass("sratio",              contains = "VGAMordinal")
```

This establishes the line of inheritance. These 4 ordinal models and 1 nominal model use S4 dispatch methods such as the ones below to exploit these relationships. We'll see that the class of the `VGAMff` argument is the key to the line of inheritance and not the class of the `object` argument.

18.3.1 Marginal Effects

To compute the marginal effects of several regression models, we have

```
> margeff

   function(object, subset = NULL, ...) {


     try.this <- findFirstMethod("margeffS4VGAM",
                                 object@family@vfamily)
     if (length(try.this)) {
       margeffS4VGAM(object = object,
                subset = subset,
                VGAMff = new(try.this),
             ...)
     } else {
       stop("Could not find a methods function for ",
            "'margeffS4VGAM' emanating ",
            "from '", object@family@vfamily[1], "'")
     }
   }
   <bytecode: 0x55b78579a4a0>
   <environment: namespace:VGAM>
```

The function `findFirstMethod()` looks at the model's `vfamily` slot to see if there is a methods function for computing the marginal effects. If so, then it starts off by calling it. If no such methods function exists, then nothing happens, because it is all optional.

Note that `margeff()` is not a generic function. The S4 OOP is done using `margeffS4VGAM()`, which `margeff()` calls. The class of the argument `VGAMff` is what the S4 dispatch operates on. Currently, the suffix "`S4VGAM`" is used to denote a generic function that programmers can write methods functions for.

Then one should write, e.g.,

```
setMethod("margeffS4VGAM",  signature(VGAMff = "multinomial"),
  function(object, subset = NULL, VGAMff, ...) {
...
  object <- callNextMethod(VGAMff = VGAMff, object = object,
```

```
                            subset = subset, ...)
...
  return(answer)
})

setMethod("margeffS4VGAM",  signature(VGAMff = "VGAMcategorial"),
  function(object, subset = NULL, VGAMff, ...) {
...
})
```

The call to `callNextMethod()` in the methods function for `"multinomial"` calls the methods function for `"VGAMcategorial"`—this is the inheritance idea.

Another example is

```
setMethod("margeffS4VGAM",  signature(VGAMff = "acat"),
  function(object, subset = NULL, VGAMff, ...) {
  object <- callNextMethod(VGAMff = VGAMff,
                           object = object,
                           subset = subset,
                           ...)
...
  return(answer)
})

setMethod("margeffS4VGAM",  signature(VGAMff = "VGAMordinal"),
  function(object, subset = NULL, VGAMff, ...) {
...  # Compute hdot
  object@post$hdot <- hdot
...
  return(object)
})
```

Here, the methods function for `"VGAMordinal"` computes a quantity called `hdot` and assigns it to the `post` slot—this quantity is used by several ordinal regression models such as `"acat"`.

18.3.2 Show

The methods functions `show.vglm()` and `show.vgam()` are called whenever the name of an S4 object of class `"vglm"` or `"vgam"` is typed in at the command prompt. (This is similar to the `print()` generic being applied to an S3 object). Sometimes it is useful to print a little more about the fitted model than the usual output produced by `show()`. Fortunately, `show.vglm()` and `show.vgam()` enable this to occur. As a simple example, consider the following code.

```
setMethod("showvglmS4VGAM",
          signature(VGAMff = "acat"),
  function(object, VGAMff, ...) {
  cat("\nThis is an adjacent categories model with",
      1 + object@misc$M, "levels\n")
  invisible(object)
  })

setMethod("showvgamS4VGAM",
```

```
          signature(VGAMff = "acat"),
  function(object, VGAMff, ...) {
  cat("\nThis is an adjacent categories model with",
      1 + object@misc$M, "levels\n")
  invisible(object)
  })
```

Thus we get, as an example,

```
> pneumo <- transform(pneumo, let = log(exposure.time))
> acatfit <- vglm(cbind(normal, mild, severe) ~ let, acat, data = pneumo)
> acatfit


   Call:
   vglm(formula = cbind(normal, mild, severe) ~ let, family = acat,
       data = pneumo)


   Coefficients:
   (Intercept):1 (Intercept):2         let:1         let:2
        -8.93603      -3.03906       2.16537       0.90209

   Degrees of Freedom: 16 Total; 12 Residual
   Residual deviance: 5.3474
   Log-likelihood: -25.251

   This is an adjacent categories model with 3 levels
```

The reason this works is because the following code fragment appears at the end of `show.vglm()`:

```
  try.this <- findFirstMethod("showvglmS4VGAM",
                              object@family@vfamily)
  if (length(try.this)) {
    showvglmS4VGAM(object = object,
                   VGAMff = new(try.this))
  }
```

The same holds for `show.vgam()` too.

18.3.3 Summary

The `summary()` generic is slightly more complicated than `margeff()` because it computes quantities and returns it as an object of a different class, e.g., `"summary.vglm"`, and then is printed by another methods function, e.g., `show.summary.vglm()`. The `summary()` generic for `"vglm"` objects searches for any methods function for `"summaryvglmS4VGAM"` matched on a value from the object's `vfamily` slot. If any values exist, then the first value is chosen as the starting point, and often this is the name of the VGAM family function itself.

```
setMethod("summaryvglmS4VGAM",
          signature(VGAMff = "cumulative"),
  function(object, VGAMff, ...) {
   object@post <- callNextMethod(VGAMff = VGAMff,
```

```
                                  object = object, ...)
  object@post$reverse <- object@misc$reverse
  object@post
})

setMethod("showsummaryvglmS4VGAM",
          signature(VGAMff = "cumulative"),
  function(object, VGAMff, ...) {
 if (object@post$reverse)
     cat("Reversed\n") else cat("Not reversed\n")
 cat("\n")
})
```

It is recommended that all quantities computed after estimation be placed in the object's `post` slot, which is a list.

Here's an example.

```
> coalminers <- transform(coalminers, Age = (age - 42) / 5)
> coalfit <- vglm(cbind(nBnW, nBW, BnW, BW) ~ Age,
                  binom2.or, data = coalminers, trace = TRUE)

   VGLM    linear loop  1 :  deviance = 50.65692
   VGLM    linear loop  2 :  deviance = 50.56779
   VGLM    linear loop  3 :  deviance = 50.56779

> summary(coalfit, presid = FALSE)


   Call:
   vglm(formula = cbind(nBnW, nBW, BnW, BW) ~ Age, family = binom2.or,
       data = coalminers, trace = TRUE)

   Coefficients:
                Estimate Std. Error z value Pr(>|z|)
   (Intercept):1 -2.26244    0.02983   -75.8   <2e-16 ***
   (Intercept):2 -1.48903    0.02057   -72.4   <2e-16 ***
   (Intercept):3  2.83253    0.05598    50.6   <2e-16 ***
   Age:1          0.51470    0.01198    43.0   <2e-16 ***
   Age:2          0.32672    0.00887    36.8   <2e-16 ***
   ---
   Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

   Names of linear predictors: logitlink(mu1), logitlink(mu2), loglink(oratio)

   Residual deviance: 50.568 on 22 degrees of freedom

   Log-likelihood: -110.62 on 22 degrees of freedom

   Number of Fisher scoring iterations: 3

   No Hauck-Donner effect found in any of the estimates


   Odds ratio:  16.988
```

When η_3 is intercept-only, the odds ratio is computed and printed at the bottom of the summary.

Another example is `posbernoulli.tb()`. Special cases of this model, such as `posbernoulli.b()` and `posbernoulli.t()`, inherit from this model.

18.3.4 Further Comments

The reader is reminded that all S4 methods ought to be documented in an `.Rd` file. Any generic function that is already supported by VGAM can be found declared in the `NAMESPACE` file and identified by the suffix "`S4VGAM`".

Table 18.2 Additional slots of a typical VGAM family object (of S4 class `"vglmff"`), and their purposes. These have been introduced since VGAM 1.0-2.

Slot	Type	Purpose
`@validparams`	`function(eta, y, extra = NULL)`	Returns a single logical: whether all the parameters are within the parameter space or an approximation to the parameter space.
`@validfitted`	`function(eta, y, extra = NULL)`	Returns a single logical: whether all the fitted values are valid, e.g., are `NA`s allowed?

18.4 A New Slot or Two

VGAM family functions now have an additional slot or two. They are `@validparams` and `@validfitted`. Their introduction was motivated by the `glm()` equivalent; the purpose is to make sure that each IRLS iteration is a valid one, therefore makes the estimation procedure more likely to avoid crashing and to converge. Currently `@validparams` appears in a few VGAM family functions while `@validfitted` is undeveloped.

Table 18.2 summarizes these. Examples of their use include:

- `negbinomial()` and variants. For the NBD,

$$V(\mu) \;=\; \mu\left(1+\frac{\mu}{k}\right) \;\approx\; \mu \;\text{ when }\; \frac{\mu}{k} \;\approx\; 0,$$

  so an artificial boundary of the parameter space is when the distribution is 'too' close to a Poisson (possibly the data may be also be underdispersed relative to a Poisson distribution). Note that major overdispersion, as when $k \approx 0$, is a less frequent problem. Having the artificial boundary stops the estimate of k at iteration a, $k^{(a)}$, from a floating point overflow.
- The support of the GEV and GPD in Table 16.1 depends on the value of the shape parameter ξ.
- Some of the statistical size distributions tabulated in Table 12.14 have constraints such as $-ap < 1 < aq$.

When `@validparams` returns a `FALSE` then `vglm.fit()` will issue a warning and invoke code to do half-stepping. The latter keeps the estimates within the parameter space.

The astute reader will notice that one of the arguments of the functions is `y` despite the well-known result that the usual MLE regularity conditions do not hold if the support of distribution depends on the parameters (Section A.1.2.2). This is done for convenience—otherwise the response would have to be passed in via the `extra` list.

Bibliographic notes

For R programming, Braun and Murdoch (2016) is the second edition of Braun and Murdoch (2008).

Exercises

> The noblest exercise of the mind within doors, and most befitting a person of quality, is study.
> —William Ramsay

Ex. 18.1. Suppose one wants to implement `linkfun(`θ`, deriv = 3, inverse = TRUE)`. Show that would entail return the following:

$$\frac{\partial^3 \theta}{\partial \eta^3} = 3\left(\frac{\partial \eta}{\partial \theta}\right)^{-5}\left(\frac{\partial^2 \eta}{\partial \theta^2}\right)^2 - \left(\frac{\partial \eta}{\partial \theta}\right)^{-4}\frac{\partial^3 \eta}{\partial \theta^3}. \tag{18.2}$$

Hint: (18.1) should help.

Appendix A
Background Material

A.1 A Bit More on Inference

A.1.1 Likelihood Ratio Statistic

Here are a few more details to fill in some missing gaps in Section A.1.4.2.

For simplicity, suppose that θ is a single parameter and that there is a single observation. Firstly, to show that

$$\widehat{\theta} \;\sim\; N(\theta,\; [\boldsymbol{\mathcal{I}}_{O1}(\widehat{\theta})]^{-1}), \tag{A.1}$$

use the Taylor series

$$\begin{aligned} \ell(\theta) &\approx \ell(\widehat{\theta}) - \ell'(\widehat{\theta})\left(\theta - \widehat{\theta}\right) - \frac{1}{2}\left[-\ell''(\widehat{\theta})\right]\left(\theta - \widehat{\theta}\right)^2 \\ &= \ell(\widehat{\theta}) - \frac{1}{2}\boldsymbol{\mathcal{I}}_{O1}(\widehat{\theta})\left(\theta - \widehat{\theta}\right)^2. \end{aligned}$$

This implies that

$$L(\theta) \;\approx\; K \cdot \exp\left\{-\frac{1}{2}\boldsymbol{\mathcal{I}}_{O1}(\widehat{\theta})\left(\theta - \widehat{\theta}\right)^2\right\},$$

which corresponds to the likelihood of obtaining a single observation $\widehat{\theta}$ from the distribution in (A.1).

Secondly, the (Wilk's) LRT statistic is

$$2\log\frac{\ell(\widehat{\theta})}{\ell(\theta)} \;=\; \boldsymbol{\mathcal{I}}_{O1}(\widehat{\theta}) \cdot \left(\widehat{\theta} - \theta\right)^2 \xrightarrow{\mathcal{D}} \chi^2_1.$$

Thus a LRT is possible based on this to test $H_0 : \theta = \theta_0$ versus $H_1 : \theta \neq \theta_0$. This justifies the 'vertical distance' mentioned regarding Figure A.2.

A.1.2 More on Probabilities

For some sequence of real numbers a_n, we write $X_n = o_p(a_n)$ if X_n/a_n converges in probability to 0.

In (A.33) we defined what it meant by $X_n = O_p(a_n)$. This is said to be *stochastically bounded* because X_n/a_n cannot grow arbitrarily large in magnitude. Here are some consequences.

- If $X_n \xrightarrow{\mathcal{D}} X$ then $X_n = O_p(1)$.
- If $X_n \xrightarrow{\mathcal{P}} a$ then $X_n = a + o_p(1)$. Alternatively, one could write $X_n = O_p(1)$, however this is less informative.

Based on these definitions, *Slutsky's Theorem* states the following results for a random variable $Y_n \xrightarrow{\mathcal{D}} Y$ and $X_n \xrightarrow{\mathcal{P}} a$.

- $$Y_n + X_n \xrightarrow{\mathcal{D}} Y + a.$$
- $$Y_n X_n \xrightarrow{\mathcal{D}} a \cdot Y.$$
- If $a \neq 0$ then
$$\frac{Y_n}{X_n} \xrightarrow{\mathcal{D}} \frac{Y}{a}.$$

A.2 On Some More Special Functions

A.2.1 Lambert W Function

The Lambert W function is the root of the equation

$$W(z)\, e^{W(z)} \;=\; z \tag{A.2}$$

for complex z. It is multi-valued if z is real and $z < -1/e$. For real $-1/e \leq z < 0$ it has two possible real values, and currently only the upper branch is computed. The function `lambertW()` computes W, and further details are at Corless et al. (1996). Its use is for the Makeham distribution. See also Goerg (2011, 2014).

A.2.2 The Lerch Φ Function

The VGAM package includes `lerch()` for computing the Lerch transcendental function

$$\Phi(x, s, v) \;=\; \sum_{n=0}^{\infty} \frac{x^n}{(n+v)^s}. \tag{A.3}$$

This can be written (see Erdélyi, 1981, eqn 3, p.27)

$$\Phi(x, s, v) \;=\; \frac{1}{\Gamma(s)} \int_0^\infty \frac{t^{s-1} \exp(-vt)}{1 - z \exp(-t)}\, dt \tag{A.4}$$

for $|z| < 1$ and for $v \neq 0, -1, -2, \ldots$.

We have

```
> args(lerch)

  function (x, s, v, tolerance = 1e-10, iter = 100)
  NULL
```

Some special cases of the Lerch function include: $\zeta(s) = \Phi(1, s, 1)$, ${}_2F_1 = \Phi(\cdot, s = 1, \cdot)$.

A.2.3 The Hurwitz Zeta Function

The Hurwitz ζ function is defined for complex arguments and is

$$\zeta(s, q) \;=\; \sum_{n=0}^{\infty} (n + q)^{-s}, \qquad \Re(s) > 1, \tag{A.5}$$

with $\Re(q) > 0$. Hence $\zeta(s, 1)$ is the Riemann zeta function (A.56). Because its computation is also amenable to the Euler-Maclaurin series (Johansson, 2015), it is computed by `zeta()` in VGAM.

For a positive integer m, the mth derivative of the polygamma function is

$$\psi^{(m)}(z) \;=\; (-1)^{m+1}\, m!\, \zeta(m + 1, z),$$

e.g., $\psi'(z) = \zeta(2, z)$ for the trigamma function (Section A.4.1). The Hurwitz zeta function is a special case of the Lerch function (A.3): $\Phi(x = 1, s, v) = \zeta(s, v)$.

This special function can be used to define generalizations of the ordinary zeta and Zipf distributions. For example, Moreno-Sánchez et al. (2016) consider a random variable defined on $a(1)\infty$ based on $\zeta(s, a)$—although usually $a = 1$ it is not always so with word-studies data.

A.2.4 Bernoulli Numbers and Polynomials

Bernoulli numbers are $B_n = B_n(0)$ where the Bernoulli polynomials are defined by

$$\frac{t\, e^{xt}}{e^t - 1} \;=\; \sum_{n=0}^{\infty} B_n(x)\, \frac{t^n}{n!}, \qquad |t| < 2\pi. \tag{A.6}$$

We have $B_0 = 1$, $B_1 = -1/2$, $B_2 = 1/6$, $B_4 = -1/30$, $B_6 = 1/42$, $B_8 = -1/30$, $B_{10} = 5/66$, $B_{12} = -691/2730$, $B_{14} = 7/6$, $B_{16} = -3617/510$, $B_{18} = 43867/798$, $B_{20} = -174611/330$, as the first few Bernoulli numbers, with $B_{2n+1} = 0$ for $n = 1(1)\infty$. The Riemann zeta function can be expressed in terms of an infinite series involving Bernoulli numbers.

The first few Bernoulli polynomials are

$$\begin{aligned} B_0(x) &= 1, \\ B_1(x) &= -\frac{1}{2} + x, \\ B_2(x) &= \frac{1}{6} - x + x^2, \\ B_3(x) &= \frac{1}{2}x - \frac{3}{2}x^2 + x^3. \end{aligned}$$

Some properties are:

$$\sum_{k=1}^{m} k^n = \frac{B_{n+1}(m+1) - B_{n+1}}{n+1}, \qquad m, n = 1(1)\infty, \tag{A.7}$$

$$\int_a^x B_n(t)\,dt = \frac{B_{n+1}(x) - B_{n+1}(a)}{n+1}, \qquad n = 1(1)\infty, \tag{A.8}$$

$$\int_0^1 B_m(t)\,B_n(t)\,dt = (-1)^{n-1}\,\frac{m!\,n!\,B_{m+n}}{(m+n)!}, \qquad m, n = 1(1)\infty, \tag{A.9}$$

$B_n'(x) = n\,B_{n-1}(x)$ for $n = 1(1)\infty$, $B_n(1-x) = (-1)^n\,B_n(x)$ for $n = 0(1)\infty$. It follows from (A.9) that Bernoulli polynomials are orthogonal on $[0, 1]$ for odd $m+n$. Other properties can be found in Abramowitz and Stegun (1964, Chap. 23) and Olver et al. (2010, Chap. 24).

A.2.5 Euler-Maclaurin Summation Formula

Suppose that $f \in \mathcal{C}^{2m}[a, b]$, $\lfloor x \rfloor$ is the floor function, B_r are the Bernoulli numbers, and $B_n(x)$ are the Bernoulli polynomials. Then

$$\begin{aligned} \sum_{k=a}^{b-1} f(k) &= \int_a^b f(x)\,dx - \frac{1}{2}\{f(b) - f(a)\} + \\ &\quad \sum_{r=1}^{m} \frac{B_{2r}}{(2r)!}\left\{f^{(2r-1)}(b) - f^{(2r-1)}(a)\right\} + R_{2m} \end{aligned} \tag{A.10}$$

where

$$\begin{aligned} R_{2m} &= \frac{-1}{(2m)!}\int_a^b B_{2m}(x - \lfloor x \rfloor)\,f^{(2m)}(x)\,dx \\ &= (-1)^m\,2\int_a^b \left\{\sum_{s=1}^{\infty} \frac{\cos(2\pi s x)}{(2\pi s)^{2m}}\right\} f^{(2m)}(x)\,dx. \end{aligned}$$

The remainder $R_{2m} = O(1/(2m)!)$, therefore it is considered negligible for some sufficiently large value of m.

A.3 Some More Series Expansions

For $p \neq 1$,

$$1 + p + p^2 + \cdots + p^{n-1} = \frac{1 - p^n}{1 - p}$$

is a geometric series. And the algebraic series

$$\begin{aligned}
\sum_{i=1}^{n} i &= \frac{n(n+1)}{2}, \\
\sum_{i=1}^{n} i^2 &= \frac{n(n+1)(2n+1)}{6}, \\
\sum_{i=1}^{n} i^3 &= \frac{n^2(n+1)^2}{4}.
\end{aligned}$$

Exercises

There are no big problems, there are just a lot of little problems.
—Henry Ford

Ex. A.1. **CDF and Expected Value**
Suppose that a random variable Y has support on $a(1)\infty$ and has a $\log(y + \alpha)$ term in its log-likelihood, where $0 < \alpha < \infty$.

(a) Show that its EIM involves calculating

$$\psi'(a + \alpha) - E[\psi'(Y + \alpha)] \quad (= \mathcal{A}_\infty, \text{ say}). \tag{A.11}$$

(b) Show that

$$\mathcal{A}_\infty = \sum_{y=a}^{\infty} \frac{\Pr(Y \geq y + 1)}{(y + \alpha)^2}. \tag{A.12}$$

(c) † Suppose we approximate (A.12) by a finite sum:

$$\mathcal{A}_U \equiv \sum_{y=a}^{U-1} \frac{\Pr(Y \geq y + 1)}{(y + \alpha)^2} \approx \mathcal{A}_\infty, \tag{A.13}$$

for some suitable upperbound U. How might U be chosen?

Ex. A.2. **Digamma Difference**
Apply the Euler-Maclaurin summation formula to $\psi(y + k) - \psi(k)$, a term in the log-likelihood of the negative binomial distribution. Under what conditions would the approximation be accurate? Can it avoid catastrophic cancellation?

References

Abramowitz, M. and I. A. Stegun (Eds.) (1964). *Handbook of Mathematical Functions*. New York, USA: Dover.

Agresti, A. (2013). *Categorical Data Analysis* (Third ed.). Hoboken, NJ, USA: Wiley.

Agresti, A. (2015). *Foundations of Linear and Generalized Linear Models*. NJ, USA: Wiley.

Agresti, A. (2019). *An Introduction to Categorical Data Analysis* (Third ed.). New York, USA: Wiley.

Agresti, A. and M. Kateri (2017). Ordinal probability effect measures for group comparisons in multinomial cumulative link models. *Biometrics 73*(1), 214–219.

Agresti, A. and C. Tarantola (2018). Simple ways to interpret effects in modeling ordinal categorical data. *Statistica Neerlandica 72*(3), 210–223.

Ahn, S. K. and G. C. Reinsel (1988). Nested reduced-rank autoregressive models for multiple time series. *J. Amer. Statist. Assoc. 83*(403), 849–856.

Bell, E. T. (1934a). Exponential numbers. *Amer. Math. Monthly 41*(7), 411–419.

Bell, E. T. (1934b). Exponential polynomials. *Ann. Math. 35*(2), 258–277.

Belzile, L. R., C. Dutang, P. J. Northrop, and T. Opitz (2022). A modeler's guide to extreme value software.

Bera, A. K. and Y. Bilias (2001). Rao's score, Neyman's $C(\alpha)$ and Silvey's LM tests: An essay on historical developments and some new results. *Journal of Statistical Planning and Inference 97*(1), 9–44.

Braun, W. J. and D. J. Murdoch (2008). *A First Course in Statistical Programming with R*. Cambridge, UK: Cambridge University Press.

Braun, W. J. and D. J. Murdoch (2016). *A First Course in Statistical Programming with R* (Second ed.). New York, USA: Cambridge University Press.

Bura, E., S. Duarte, and L. Forzani (2016). Sufficient reductions in regressions with exponential family inverse predictors. *Journal of the American Statistical Association 111*(515), 1313–1329.

Bura, E., S. Duarte, L. Forzani, E. Smucler, and M. Sued (2018). Asymptotic theory for maximum likelihood estimates in reduced-rank multivariate generalized linear models. *Statistics 52*(5), 1005–1024.

Bura, E. and J. Yang (2011). Dimension estimation in sufficient dimension reduction: A unifying approach. *Journal of Multivariate Analysis 102*(1), 130–142.

Cameron, A. C. and P. K. Trivedi (2013). *Regression Analysis of Count Data* (Second ed.). Cambridge: Cambridge University Press.

Castellares, F., S. L. P. Ferrari, and A. J. Lemonte (2018). On the Bell distribution and its associated regression model for count data. *Applied Mathematical Modelling 56*, 172–185.

Chernoff, H. (1954). On the distribution of the likelihood ratio. *Ann. Math. Statist. 25*(3), 573–578.

Coles, S. (2001). *An Introduction to Statistical Modeling of Extreme Values*. London: Springer-Verlag.

Corless, R. M., G. H. Gonnet, D. E. G. Hare, D. J. Jeffrey, and D. E. Knuth (1996). On the Lambert W function. *Adv. Comput. Math. 5*(4), 329–359.

De Rossi, G. and A. Harvey (2009). Quantiles, expectiles and splines. *J. Econometrics 152*(2), 179–185.

Dean, C. and J. F. Lawless (1989). Tests for detecting overdispersion in Poisson regression models. *J. Amer. Statist. Assoc. 84*(406), 467–472.

Dobson, A. J. and A. G. Barnett (2018). *An Introduction to Generalized Linear Models* (Fourth ed.). Boca Raton, FL, USA: CRC Press, Taylor & Francis.

Dunn, P. K. and G. K. Smyth (1996). Randomized quantile residuals. *J. Comput. Graph. Statist. 5*(3), 236–244.

Eilers, P. H. C. and B. D. Marx (2021). *Practical Smoothing: The Joys of P-Splines*. Cambridge: Cambridge University Press.

Erdélyi, A. (1981). *Higher Transcendental Functions*, Volume 1. Melbourne, FL, USA: Robert E. Krieger Publishing Company.

Fagerland, M. W., S. Lydersen, and P. Laake (2017). *Statistical Analysis of Contingency Tables*. Boca Raton, FL, USA: CRC Press.

Fisher, R. A. (1925). Theory of statistical estimation. *Mathematical Proceedings of the Cambridge Philosophical Society 22*(5), 700–725.

Fox, J. (2016). *Applied Regression Analysis and Generalized Linear Models* (Third ed.). Thousand Oaks, CA, USA: Sage Publications.

Fox, J. and S. Weisberg (2011). *An R Companion to Applied Regression* (Second ed.). Thousand Oaks, CA, USA: Sage Publications.

Freedman, D. A. (2007). How can the score test be inconsistent? *Amer. Statist. 61*(4), 291–295.

Fullerton, A. S. and J. Xu (2016). *Ordered Regression Models: Parallel, Partial, and Non-Parallel Alternatives*. Boca Raton, FL, USA: Chapman & Hall/CRC.

Goerg, G. M. (2011). Lambert W random variables—a new family of generalized skewed distributions with applications to risk estimation. *Ann. Appl. Stat. 5*(3), 2197–2230.

Goerg, G. M. (2014). Usage of the Lambert W function in statistics. *Ann. Appl. Stat. 8*(4), 2567.

Goh, K.-L. and M. L. King (1999). A correction for local biasedness of the Wald and null Wald tests. *Oxford Bulletin of Economics and Statistics 61*(3), 435–450.

Granger, C. W. J. (1981). Some properties of time series data and their use in econometric model specification. *Journal of Econometrics 16*(1), 121–130.

Granger, C. W. J. (1987). Cointegraion and error correction: Representation, estimation, and testing. *Econometrica 55*, 251–276.

Greene, W. H. (2018). *Econometric Analysis* (8th ed.). New York, NY, USA: Pearson.

Hauck, W. W. and A. Donner (1977). Wald's test as applied to hypotheses in logit analysis. *J. Amer. Statist. Assoc. 72*(360), 851–853.

Hector, A., S. Von Felten, and B. Schmid (2010). Analysis of variance with unbalanced data: an update for ecology & evolution. *Journal of Animal Ecology 79*(2), 308–316.

Hensher, D. A., J. M. Rose, and W. H. Greene (2015). *Applied Choice Analysis* (Second ed.). Cambridge, U.K.: Cambridge University Press.

Hilbe, J. M. (2011). *Negative Binomial Regression* (Second ed.). Cambridge, UK; New York, USA: Cambridge University Press.

Hilbe, J. M. (2014). *Modeling Count Data*. New York, USA: Cambridge University Press.

Johansson, F. (2015). Rigorous high-precision computation of the Hurwitz zeta function and its derivatives. *Numerical Algorithms 69*(2), 253–270.

Karavarsamis, N., G. Guillera-Arroita, R. M. Huggins, and B. J. T. Morgan (2020). The score test for the two-sample occupancy model. *Aus. & N. Z. J. Statist. 62*(1), 95–115.

Kleiber, C. and A. Zeileis (2008). *Applied Econometrics with R*. New York, USA: Springer.

Koenker, R., V. Chernozhukov, X. He, and L. Peng (Eds.) (2018). *Handbook of Quantile Regression*. Boca Raton, FL, USA: CRC/Taylor & Francis.

Lambert, D. (1992). Zero-inflated Poisson regression, with an application to defects in manufacturing. *Techn. 34*(1), 1–14.

Langsrud, Ø. (2003). ANOVA for unbalanced data: Use Type II instead of Type III sums of squares. *Statistics and Computing 13*(2), 163–167.

Laskar, M. R. and M. L. King (1997). Modified Wald test for regression disturbances. *Econ. Let. 56*(1), 5–11.

Lewsey, J. D., W. P. Gardiner, , and G. Gettinby (2001). A study of Type II and Type III power for testing hypotheses from unbalanced factorial designs. *Communications in Statistics—Simulation and Computation 30*, 597–609.

Long, J. S. (1997). *Regression Models for Categorical and Limited Dependent Variables*. Thousand Oaks, CA, USA: Sage Publications.

Long, J. S. and J. Freese (2014). *Regression Models for Categorical Dependent Variables Using Stata* (Third ed.). College Station, TX, USA: Stata Press.

Ly, A., M. Marsman, J. Verhagen, R. P. P. P. Grasman, and E.-J. Wagenmakers (2017). A tutorial on Fisher information. *J. Math. Psych. 80*, 40–55.

MacKenzie, D. I., J. D. Nichols, G. B. Lachman, S. Droege, J. Royle, and C. A. Langtimm (2002). Estimating site occupancy rates when detection probabilities are less than one. *Ecology 83*(8), 2248–55.

Madsen, H. and P. Thyregod (2011). *Introduction to General and Generalized Linear Models*. Boca Raton, FL, USA: CRC Press/Taylor & Francis.

Mantel, N. (1987). Understanding Wald's test for exponential families. *The American Statistician 41*(2), 147–148.

McCullagh, P. and J. A. Nelder (1989). *Generalized Linear Models* (Second ed.). London: Chapman & Hall.

McKelvey, R. D. and W. Zavoina (1975). A statistical model for the analysis of ordinal level dependent variables. *The Journal of Mathematical Sociology 4*(1), 103–120.

Miranda-Soberanis, V. F. and T. W. Yee (2019). New link functions for distribution–specific quantile regression based on vector generalized linear and additive models. *Journal of Probability and Statistics 5*, 1–11.

Moran, P. A. P. (1971). Maximum-likelihood estimation in non-standard conditions. *Math. Proc. Cambridge Philos. Soc. 70*(3), 441–450.

Moreno-Sánchez, I., F. Font-Clos, and A. Corral (2016). Large-scale analysis of Zipf's law in English texts. *PLoS ONE 11*(1), 1–19.

Murray, M. P. (1994). A drunk and her dog: An illustration of cointegration and error correction. *The American Statistician 48*(1), 37–39.

Nelder, J. A. (1994). The statistics of linear models: Back to basics. *Statistics and Computing 4*(4), 221–234.

Olver, F. W. J., D. W. Lozier, R. F. Boisvert, and C. W. Clark (Eds.) (2010). *NIST Handbook of Mathematical Functions*. New York, USA: National Institute of Standards and Technology, and Cambridge University Press.

Otis, D. L., K. P. Burnham, G. C. White, and D. R. Anderson (1978). Statistical inference from capture data on closed animal populations. *Wild. Monog. 62*, 3–135.

Powers, S., T. Hastie, and R. Tibshirani (2018). Nuclear penalized multinomial regression with an application to predicting at bat outcomes in baseball. *Statistical Modelling 18*(5–6), 1–23.

Rao, C. R. (1948). Large sample tests of statistical hypotheses concerning several parameters with applications to problems of estimation. *Math. Proc. Camb. Philos. Soc. 44*(1), 50–57.

Rodriguez, R., R. Tobias, and R. Wolfinger (1995). Comments on: The statistics of linear models: Back to basics. *Statistics and Computing 5*(2), 97–101.

Schnabel, S. K. (2011). *Expectile smoothing: new perspectives on asymmetric least squares. An application to life expectancy*. Ph. D. thesis, Universiteit Utrecht, Utrecht, Netherlands.

Schnabel, S. K. and P. H. C. Eilers (2009). Optimal expectile smoothing. *Computational Statistics & Data Analysis 53*(12), 4168–4177.

Schwendinger, F., B. Gr un, and K. Hornik (2021). A comparison of optimization solvers for log binomial regression including conic programming. *Computational Statistics 36*, 1721–1754.

Searle, S. R. (1995). Comments on: The statistics of linear models: Back to basics. *Statistics and Computing 5*(2), 103–107.

Sellers, K. F. and D. S. Morris (2017). Underdispersion models: Models that are "under the radar". *Comm. Statist. Theory Meth. 46*(24), 12075–12086.

Severini, T. A. (2000). *Likelihood Methods in Statistics*. New York, USA: Oxford University Press.

Stasinopoulos, M. D., R. A. Rigby, G. Z. Heller, V. Voudouris, and F. De Bastiani (2017). *Flexible Regression and Smoothing: Using GAMLSS in R*. Boca Raton, FL, USA: Chapman & Hall/CRC.

Thas, O. (2010). *Comparing Distributions*. Springer Series in Statistics. New York, USA: Springer.

Tuenter, H. J. H. (2000). On the generalized Poisson distribution. *Statistica Neerlandica 54*(3), 374–376.

Tutz, G. and G. Schauberger (2013). Visualization of categorical response models: From data glyphs to parameter glyphs. *J. Comput. Graph. Statist. 22*(1), 156–177.

Ugba, E. R. (2022). gofcat: An R package for goodness-of-fit of categorical response models. *Journal of Open Source Software 7*(76), 4382.

Wagh, Y. S. and K. K. Kamalja (2018). Zero-inflated models and estimation in zero-inflated Poisson distribution. *Comm. Statist. Sim. Comput. 47*(8), 2248–2265.

Wang, W. and J. Yan (2021). Shape-restricted regression splines with R package splines2. *Journal of Data Science 19*(3), 498–517.

Wiley, M. and J. F. Wiley (2019). *Advanced R Statistical Programming and Data Models: Analysis, Machine Learning, and Visualization*. New York, USA: Apress Media, Springer.

Wood, S. N. (2017). *Generalized Additive Models: An Introduction with R* (Second ed.). London: Chapman & Hall/CRC.

Yee, T. W. (2015). *Vector Generalized Linear and Additive Models: With an Implementation in R*. New York, USA: Springer.

Yee, T. W. (2020). The VGAM package for negative binomial regression. *Aust. N. Z. J. Stat. 62*(1), 116–131.

Yee, T. W. (2021). Some new results concerning the Hauck–Donner effect. *In preparation*.

Yee, T. W. (2022). On the Hauck-Donner effect in Wald tests: Detection, tipping points, and parameter space characterization. *J. Amer. Statist. Assoc. 117*(540), 1763–1774.

Yee, T. W., L. Frigau, and C. Ma (2025). Heaping and seeping, GAITD regression and doubly constrained reduced-rank vector generalized linear models, in smoking studies. *Under review*.

Yee, T. W. and C. Ma (2024). Generally altered, inflated, truncated and deflated regression. *Statist. Sci. 39*, 568–588.

Yee, T. W., J. Stoklosa, and R. M. Huggins (2015). The VGAM package for capture–recapture data using the conditional likelihood. *J. Statist. Soft. 65*(5), 1–33.

Young, G. A. and R. L. Smith (2005). *Essentials of Statistical Inference*. Cambridge: Cambridge University Press.

Zhang, Y., H. Zhou, J. Zhou, and W. Sun (2017). Regression models for multivariate count data. *Journal of Computational and Graphical Statistics 26*(1), 1–13.

Zuur, A. F., A. A. Saveliev, and E. N. Ieno (2012). *Zero Inflated Models and Generalized Linear Mixed Models with R*. Newburgh, UK: Highland Statistics Ltd.