COMPoissonReg: Usage, the Normalizing Constant, and Other Computational Details

Andrew M. Raim* Kimberly F. Sellers†

2023-11-29

Abstract

`COMPoissonReg` is an R package which supports Conway-Maxwell Poisson (CMP) and Zero-Inflated Conway-Maxwell Poisson (ZICMP) models. This vignette describes fundamental computational details, especially those involving the normalizing constant and related quantities. The CMP normalizing constant does not have a general closed form; furthermore, it requires care to handle numerically as its magnitude can vary greatly with changes in the parameters. Primary `COMPoissonReg` functions are demonstrated with examples, including those implementing basic distribution functions and regression modeling.

Contents

*andrew.raim@gmail.com, Center for Statistical Research & Methodology, U.S. Census Bureau, Washington, DC, 20233, U.S.A. **Disclaimer:** This document is released to inform interested parties of ongoing research and to encourage discussion of work in progress. Any views expressed are those of the authors and not those of the U.S. Census Bureau.

†kfs7@georgetown.edu, Center for Statistical Research & Methodology, U.S. Census Bureau and Department of Mathematics and Statistics, Georgetown University, Washington, DC, 20057, U.S.A.

1 Introduction

The R package `COMPoissonReg` supports Conway-Maxwell Poisson (CMP) and Zero-Inflated Conway-Maxwell Poisson (ZICMP) models for analysis of count data in a flexible manner, to account for data dispersion relative to a Poisson model. The package provides regression functionality in addition to basic distribution functions. Interested users can refer to Sellers and Shmueli [2010] and Sellers and Raim [2016] regarding the underlying theoretical developments for the CMP and ZICMP regressions, respectively. A full specification of the public `COMPoissonReg` interface can be found in the manual. In addition to package prerequisites `Rcpp` [Eddelbuettel, 2013] and `numDeriv` [Gilbert and Varadhan, 2019], `ggplot2` [Wickham, 2016] is also used in this vignette.

One of the challenges of working with CMP and ZICMP lies in computing the normalizing constant and related quantities. The normalizing constant does not have a simple closed form in general and can quickly increase or decrease magnitude as parameters are varied. `COMPoissonReg` takes a hybrid approach of either truncating the infinite series or making use of an approximation, depending on parameter values.

The remainder of the vignette proceeds as follows. Section 2 describes functions to support the CMP distribution, including numerical handling of the normalizing constant. Section 3 describes functions for ZICMP. Finally, Section 4 demonstrates regression functions; Sections 4.1 and 4.2 give specific examples based on CMP and ZICMP outcomes, respectively. The `COMPoissonReg` package is on CRAN at https://cran.r-project.org/package=COMPoissonReg and the source code is on Github at https://github.com/lotze/COMPoissonReg.

2 Conway-Maxwell Poisson Distribution

Let $Y \sim \text{CMP}(\lambda, \nu)$ be a Conway-Maxwell Poisson (CMP) random variable with density

$$f(y \mid \lambda, \nu) = \frac{\lambda^y}{(y!)^\nu Z(\lambda, \nu)}, \quad y \in \mathbb{N}, \quad Z(\lambda, \nu) = \sum_{r=0}^{\infty} \frac{\lambda^r}{(r!)^\nu},$$

where $\lambda > 0$, $\nu > 0$, and $\mathbb{N}$ represents the nonnegative integers $\{0, 1, 2, \ldots\}$. Three notable special cases of $\text{CMP}(\lambda, \nu)$ help to demonstrate its flexibility in count modeling.

a. The case $\nu = 1$ corresponds to Poisson(λ).

b. When $\lambda \in (0, 1)$ and $\nu = 0$, the $\text{CMP}(\lambda, \nu)$ distribution is Geometric($1 - \lambda$) with density $f(y \mid \lambda) = (1 - \lambda)\lambda^y$ for $y \in \mathbb{N}$, which is overdispersed relative to Poisson.

c. When $\nu \to \infty$, $\text{CMP}(\lambda, \nu)$ converges to a Bernoulli($\lambda/(1 + \lambda)$) distribution which is underdispersed relative to Poisson.

2.1 Normalizing Constant

The normalizing constant $Z(\lambda, \nu)$ presents some challenges in the practical use of CMP models and has been a topic of interest in the CMP literature. In general, there is no simple closed form expression for the series $Z(\lambda, \nu)$. Shmueli et al. [2005] give the approximation

$$Z(\lambda, \nu) = \frac{\exp(\nu \lambda^{1/\nu})}{\lambda^{(\nu-1)/2\nu}(2\pi)^{(\nu-1)/2}\nu^{1/2}} \left\{1 + O(\lambda^{-1/\nu})\right\}, \tag{1}$$

where the $O(\cdot)$ term vanishes as $\lambda^{-1/\nu}$ becomes small. Approximations have been further studied and refined in subsequent literature; see for example Gillispie and Green [2015], Daly and Gaunt [2016], and Gaunt et al. [2019]. The expression in (1) emphasizes that the magnitude of $Z(\lambda, \nu)$ explodes as $\nu \to 0$ when $\lambda > 1$. For example, $Z(2, 0.075) \approx e^{780.515}$ is too large to store as a double-precision floating point number, and may evaluate to infinity if care is not taken. In contrast, $Z(\lambda, \nu) \to 1/(1 - \lambda)$ when $\lambda < 1$ and $\nu \to 0$.

In practice, the `COMPoissonReg` package does not place constraints on λ and ν, except to ensure that they are positive, so that their values are driven by the data or the user's selection. A hybrid strategy motivated by (1) is taken by `COMPoissonReg`.

To compute $Z(\lambda, \nu)$, suppose we are given a small tolerance $\delta > 0$. If

$$\lambda^{-1/\nu} < \delta, \tag{2}$$

the first term of (1) dominates the second term, and we take

$$\begin{aligned} Z(\lambda, \nu) &\approx \frac{\exp(\nu \lambda^{1/\nu})}{\lambda^{(\nu-1)/2\nu} (2\pi)^{(\nu-1)/2} \nu^{1/2}} \\ &= \exp\left\{ \nu \lambda^{1/\nu} - \frac{\nu - 1}{2\nu} \log \lambda - \frac{\nu - 1}{2} \log(2\pi) - \frac{1}{2} \log \nu \right\}. \end{aligned} \tag{3}$$

as an approximation. Otherwise, the series is computed by truncating to a finite number of terms, which is described next. In either case, computations are kept on the log-scale as much as possible to accommodate numbers with potentially very large and very small magnitudes.

We approximate $Z(\lambda, \nu)$ by a finite summation $Z^{(M)}(\lambda, \nu) = \sum_{r=0}^{M} \lambda^r / (r!)^\nu$ if condition (2) fails, so that the remainder is smaller than a given tolerance. The general approach is described in Appendix B of Shmueli et al. [2005]. Robbins [1955] gives bounds for Stirling's approximation as

$$\sqrt{2\pi} n^{n+1/2} e^{-n} e^{1/(12n+1)} < n! < \sqrt{2\pi} n^{n+1/2} e^{-n} e^{1/(12n)}.$$

Noting that $e^{1/(12n+1)} \geq 1$ for $n \geq 1$ and $\sqrt{2\pi} e^{1/(12n)} \leq e$ for $n \geq 2$, we obtain simpler bounds

$$\sqrt{2\pi} n^{n+1/2} e^{-n} \leq n! \leq e n^{n+1/2} e^{-n},$$

which will be convenient in the following calculations.[1] We may then bound the truncation error for $Z^{(M)}(\lambda, \nu)$ using

$$\begin{aligned} |Z(\lambda, \nu) - Z^{(M)}(\lambda, \nu)| &= Z(\lambda, \nu) - Z^{(M)}(\lambda, \nu) \\ &= \sum_{r=M+1}^{\infty} \frac{\lambda^r}{(r!)^\nu} \\ &\leq \sum_{r=M+1}^{\infty} \frac{\lambda^r}{(2\pi)^{\nu/2} r^{\nu r + \nu/2} e^{-r\nu}} \\ &\leq \sum_{r=M+1}^{\infty} \frac{\lambda^r}{(2\pi)^{\nu/2} (M+1)^{\nu r + \nu/2} e^{-r\nu}} \\ &= (2\pi)^{-\nu/2} (M+1)^{-\nu/2} \sum_{r=M+1}^{\infty} \left(\frac{\lambda e^\nu}{(M+1)^\nu} \right)^r \\ &= (2\pi)^{-\nu/2} (M+1)^{-\nu/2} \sum_{r=0}^{\infty} \left(\frac{\lambda e^\nu}{(M+1)^\nu} \right)^{r+M+1} \\ &= (2\pi)^{-\nu/2} (M+1)^{-\nu/2} \left(\frac{\lambda e^\nu}{(M+1)^\nu} \right)^{M+1} \frac{1}{1 - \frac{\lambda e^\nu}{(M+1)^\nu}} \\ &=: \Delta_M, \end{aligned} \tag{4}$$

assuming that $|\lambda e^\nu / (M+1)^\nu| < 1$ so that the geometric series in (4) converges. To ensure this convergence we choose M at least large enough so that

$$\lambda e^\nu / (M+1)^\nu < 1 \iff M > \lambda^{1/\nu} e - 1.$$

For a small given number $\epsilon > 0$, we may consider bounding the relative error by

$$\frac{|Z(\lambda, \nu) - Z^{(M)}(\lambda, \nu)|}{Z^{(M)}(\lambda, \nu)} \leq \frac{\Delta_M}{Z^{(M)}(\lambda, \nu)} < \epsilon. \tag{5}$$

[1] These bounds are also stated at https://en.wikipedia.org/wiki/Stirling/%27s_approximation, last accessed 2022-10-09.

The second inequality of (5) can be expressed on the log-scale using

$$\log \Delta_M - \log Z^{(M)}(\lambda, \nu) < \log \epsilon, \tag{6}$$

where

$$\log \Delta_M = -\frac{\nu}{2}\log(2\pi) - \nu\left(M + \frac{3}{2}\right)\log(M+1) + (M+1)(\nu + \log\lambda) - \log\left(1 - \frac{\lambda e^{\nu}}{(M+1)^{\nu}}\right).$$

Therefore, we compute $Z^{(M)}(\lambda, \nu)$ until at least $M > \lambda^{1/\nu}e - 1$, increasing M and updating $Z^{(M)}(\lambda, \nu)$ until (6) is satisfied. This is summarized as Algorithm 1.

Algorithm 1 Compute the CMP normalizing constant using truncation approach.

Input: $\lambda > 0$ rate parameter.
Input: $\nu > 0$ dispersion parameter.
Input: $\epsilon > 0$ tolerance.
Input: $y_{\max} \in \mathbb{N}$ upper limit for M

function TRUNCATED-$Z(\lambda, \nu, \epsilon, y_{\max})$
  $M = 0$, $Z^{(0)} = 1$
  while $M \leq \lambda^{1/\nu}e - 1$ and $M \leq y_{\max}$ **do**
    $Z^{(M+1)} \leftarrow Z^{(M)} + \lambda^M/(M!)^{\nu}$
    $M \leftarrow M + 1$
  while $\log \Delta_M - \log Z^{(M)} \geq \log \epsilon$ and $M \leq y_{\max}$ **do**
    $Z^{(M+1)} \leftarrow Z^{(M)} + \lambda^M/(M!)^{\nu}$
    $M \leftarrow M + 1$
  return $\{Z^{(M)}, M\}$

The individual terms $\lambda^r/(r!)^{\nu}$ in the summation may be too large to store at their original scale. Therefore, summation is carried out at the log-scale, wherever possible, using the identity

$$\log(x + y) = \log x + \log(1 + \exp\{\log y - \log x\}); \tag{7}$$

this is especially helpful when $0 < y \ll x$, as $\log x$ may be kept on the log-scale by the first term of the right-hand side of (7), and the standard library function `log1p` may be used to accurately compute $\log(1+\phi)$ for very small $\phi > 0$.

Many of the functions in the user interface of `COMPoissonReg` take an optional `control` argument, which can be constructed as follows.

```
> control = get.control(
+   ymax = 100000,
+   hybrid.tol = 1e-2,
+   truncate.tol = 1e-6
+ )
```

The tolerances δ and ϵ are specified as `hybrid.tol` and `truncate.tol` respectively. Taking `hybrid.tol` to be a very small positive number results in use of the truncated sum $Z^{(M)}(\lambda, \nu)$, while `hybrid.tol = Inf` uses the approximation method (3), except in extreme cases where $\lambda^{-1/\nu}$ evaluates to zero or ∞ numerically. The argument `ymax` specifies upper limit M; this is a safety measure which prevents very large computations unless the user opts to allow them. When no control object is specified, a global default (via option `COMPoissonReg.control`) is used.

```
> control = getOption("COMPoissonReg.control")
> control$ymax
[1] 1e+06
> control$hybrid.tol
[1] 0.01
```

```
> control$truncate.tol
[1] 1e-06
```

The default may be replaced in your current session if desired.

```
> options(COMPoissonReg.control = control)
```

The control object contains several other useful arguments to be discussed later in the vignette.

The `ncmp` function computes the normalizing constant $Z(\lambda, \nu)$ and returns its value either on the original scale or the log-scale.

```
> ncmp(lambda = 1.5, nu = 1.2)
[1] 4.01341
> ncmp(lambda = 1.5, nu = 1.2, log = TRUE)
[1] 1.389641
> ncmp(lambda = 1.5, nu = 1.2, log = TRUE, control = get.control(hybrid.tol = 1e10))
[1] 1.373642
> ncmp(lambda = 1.5, nu = 1.2, log = TRUE, control = get.control(hybrid.tol = 1e-10))
[1] 1.389641
```

Before proceeding, let us define a function to display errors and warnings which are intentionally triggered in the remainder of the vignette.

```
> print_warning = function(x) { print(strwrap(x), quote = FALSE) }
```

The function `tcmp` returns the truncation value M obtained from Algorithm 1.

```
> nu_seq = c(1, 0.5, 0.2, 0.1, 0.05, 0.03)
> tryCatch({ tcmp(lambda = 1.5, nu = nu_seq) }, warning = print_warning)
[1] simpleWarning in y_trunc(prep$lambda, prep$nu, tol = truncate.tol, ymax
[2] = ymax): Terms of normalizing constant CMP(1.5, 0.03) could not be
[3] bounded by a geometric series when y <= 1e+06. Consider adjusting the
[4] controls ymax, hybrid.tol, and truncate.tol
```

Note that `tcmp` returns `1e6` and produces a warning for the smallest `nu` value 0.03 because Algorithm 1 has reached `ymax = 1e6` before the series could be bounded by a geometric series. Here, it is likely that support values with non-negligible mass are being left out. Let us increase `ymax` to avoid this problem.

```
> tcmp(lambda = 1.5, nu = nu_seq, control = get.control(ymax = 3e6))
[1]      11      20      56     219    9039 2013739
```

It is also possible to reach the second loop of Algorithm 1 where the geometric series can be used, but `ymax` is not large enough to satisfy (6). Here is an example where this occurs.

```
> tcmp(lambda = 1.2, nu = 0.03, control = get.control(ymax = 1200))
Warning in y_trunc(prep$lambda, prep$nu, tol = truncate.tol, ymax = ymax):
Absolute relative error 1.13902e-05 was larger than tolerance 1e-06 with
CMP(1.2, 0.03) truncated to 1200. Consider adjusting the controls ymax,
hybrid.tol, and truncate.tol
[1] 1200
```

Now that we have a somewhat robust computation for the normalizing constant, let us create a plot of the interesting behavior when $\lambda > 1$ and ν is decreasing.

```
library(ggplot2)

nu_seq = seq(0.03, 1.5, length.out = 20)
nc1 = ncmp(lambda = 0.5, nu = nu_seq, log = TRUE)
```

```
nc2 = ncmp(lambda = 1.05, nu = nu_seq, log = TRUE)
nc3 = ncmp(lambda = 1.20, nu = nu_seq, log = TRUE)
```

```
ggplot() +
    geom_point(data = data.frame(x = nu_seq, y = nc1), aes(x = x, y = y), pch = 1) +
    geom_point(data = data.frame(x = nu_seq, y = nc2), aes(x = x, y = y), pch = 2) +
    geom_point(data = data.frame(x = nu_seq, y = nc3), aes(x = x, y = y), pch = 3) +
    xlab("nu") +
    ylab("log of normalizing constant") +
    theme_bw()
```

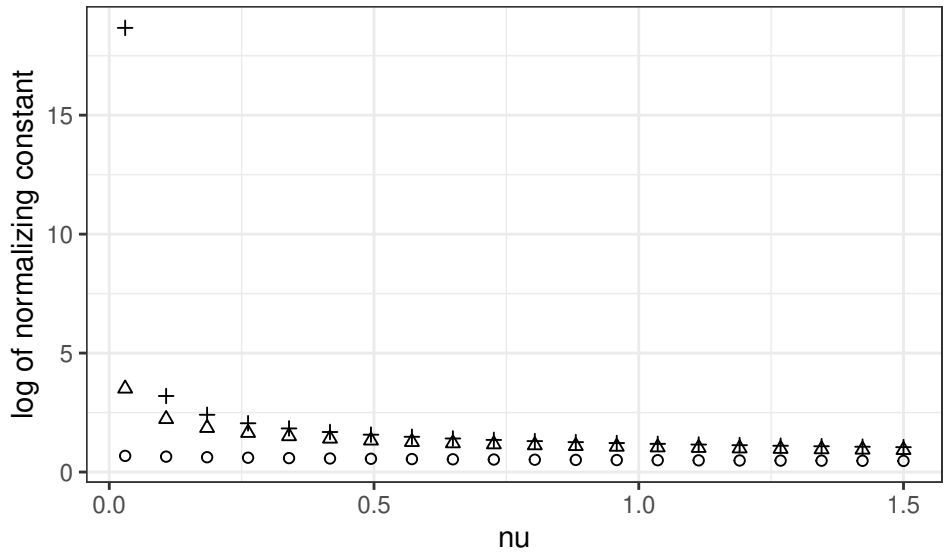


Figure 1: Log of normalizing constant for $\lambda = 0.5$ ($\circ$), $\lambda = 1.05$ (Δ), and $\lambda = 1.20$ ($+$).

We see that with $\lambda = 1.2$ a value of $Z(\lambda, 0.03) \approx e^{18.67}$ is obtained, which is an extremely large jump from the next value in the series $Z(\lambda, 0.1074) \approx e^{3.2}$.

2.2 Density, Generation, CDF, and Quantile Functions

The respective functions for CMP density, variate generation, CDF, and quantile functions are `dcmp`, `rcmp`, `pcmp` , and `qcmp`. Their usage is similar to distribution functions provided by the R `stats` package.

```
> dcmp(0, lambda = 10, nu = 0.9)
[1] 6.819476e-06
> dcmp(0:17, lambda = 10, nu = 0.9, log = TRUE)
 [1] -11.895728  -9.593143  -7.914390  -6.600556  -5.545636  -4.691545
 [7]  -4.001543  -3.450278  -3.019190  -2.694107  -2.463848  -2.319369
[13]  -2.253200  -2.259069  -2.331636  -2.466296  -2.659041  -2.906347
> dcmp(c(0, 1, 2), lambda = c(10, 11, 12), nu = c(0.9, 1.0, 1.1), log = TRUE)
[1] -11.895728  -8.602105  -6.071951
```

```
> rcmp(50, lambda = 10, nu = 0.9)
 [1] 10 13  8 18 17 12 13 15 13 10 20 13 13 14 15  5 18 14 14 17  9 20 17 10 10
[26] 16 11 13 10 12 14 12 15 14 11 14 10 11 14  8 13 18  6 13 18 18 11 13 15  9
```

```
> pcmp(0:17, lambda = 10, nu = 0.9)
 [1] 6.819476e-06 7.501423e-05 4.404609e-04 1.800073e-03 5.704532e-03
```

```
 [6] 1.487703e-02 3.316443e-02 6.490125e-02 1.137420e-01 1.813448e-01
[11] 2.664516e-01 3.647872e-01 4.698497e-01 5.742973e-01 6.714341e-01
[16] 7.563328e-01 8.263482e-01 8.810232e-01
```

```
> qq = seq(0, 0.95, length.out = 10)
> qcmp(qq, lambda = 10, nu = 0.9)
 [1]  0  8 10 11 12 13 14 15 17 19
```

The CMP distribution functions compute the normalizing constant via Section 2.1. The density `dcmp` uses the hybrid method, while `rcmp`, `pcmp`, and `qcmp` use the truncation method regardless of condition (2). Variate generation, CDF, and quantiles are then computed on CMP as if it were a discrete distribution on the sample space $\{0, \ldots, M\}$. As with `tcmp` and `ncmp`, a warning is emitted in cases where they may not produce reliable results.

```
> tryCatch({ rcmp(1, lambda = 2, nu = 0.01) }, warning = print_warning)
[1] simpleWarning in r_cmp(n, prep$lambda, prep$nu, hybrid.tol,
[2] truncate.tol, ymax): Terms of normalizing constant CMP(2, 0.01) could
[3] not be bounded by a geometric series when y <= 1e+06. Consider
[4] adjusting the controls ymax, hybrid.tol, and truncate.tol
```

The truncation method for `qcmp` can result in unreliable quantiles when the requested probability is very close to 1. For example, the actual quantile for probability 1 is ∞, which can be expressed with no computation, but the computed quantity will be the truncation value M. More generally, (5) implies that

$$\begin{aligned} \frac{Z(\lambda,\nu) - Z^{(M)}(\lambda,\nu)}{Z(\lambda,\nu)} < \epsilon &\iff \mathrm{P}(Y > M) < \epsilon \\ &\iff 1 - F(M \mid \lambda,\nu) < \epsilon \\ &\iff F^{-}(1-\epsilon \mid \lambda,\nu) < M. \end{aligned} \tag{8}$$

Therefore, it is possible that quantiles larger than $1 - \epsilon$ may be inaccurately computed with the truncated support. `COMPoissonReg` gives a warning when these are requested.

```
> tryCatch({
+   qcmp(0.9999999, lambda = 1.5, nu = 0.5)
+ }, warning = print_warning)
[1] simpleWarning in qcmp(0.9999999, lambda = 1.5, nu = 0.5): At least one
[2] requested quantile was very close to 1. In particular, 1 of the given
[3] probabilities were greater than 1 - truncate_tol = exp(-1e-06), where
[4] truncate_tol = 1e-06. Associated results may be affected by truncation.
[5] Consider adjusting the controls ymax and truncate.tol or reducing logq.
```

As a sanity check, let us ensure that the empirical density values, cumulative probabilities, and quantiles of draws from `rcmp` align with respective results computed via `dcmp`, `pcmp`, and `qcmp`.

```
library(ggplot2)

n = 100000
lambda = 0.5
nu = 0.1
x = rcmp(n, lambda, nu)

xx = seq(-1, max(x))  ## Include -1 to ensure it gets probability zero
qq = seq(0, 0.99, length.out = 100)

fx = dcmp(xx, lambda, nu)
px = pcmp(xx, lambda, nu)
```

```
qx = qcmp(qq, lambda, nu)

qx_emp = quantile(x, probs = qq)
```

```
ggplot() +
    geom_bar(data = data.frame(x = x), aes(x = x, y = ..prop..), fill = "NA",
        col = "black") +
    geom_point(data = data.frame(x = xx[-1], fx = fx[-1]), aes(x, fx)) +
    ylab("Density") +
    theme_bw()
```

```
Warning: The dot-dot notation (`..prop..`) was deprecated in ggplot2 3.4.0.
i Please use `after_stat(prop)` instead.
This warning is displayed once every 8 hours.
Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
generated.
```

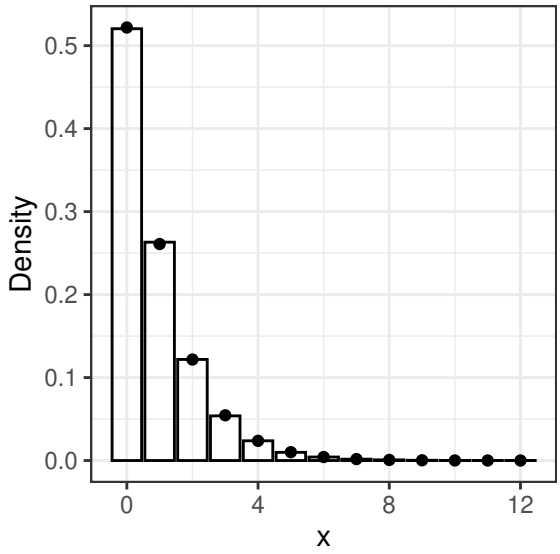


Figure 2: Empirical density of draws (histogram) versus density computed via the dcmp function (points).

```
ggplot() +
    stat_ecdf(data = data.frame(x = x), aes(x), geom = "step") +
    geom_point(data = data.frame(x = xx, px = px), aes(x, px)) +
    ylab("Probability") +
    theme_bw()
```

```
ggplot() +
    geom_point(data = data.frame(x = qq, qx_emp = qx_emp), aes(qq, qx_emp), pch = 1) +
    geom_point(data = data.frame(x = qq, qx = qx), aes(qq, qx), pch = 3) +
    xlab("Probability") +
    ylab("Quantile") +
    theme_bw()
```

2.3 Expected Value and Variance

For $Y \sim \text{CMP}(\lambda, \nu)$, we can consider computing the expectation and variance of Y in two ways. First, if there is a moderately-sized M where $\{0, \ldots, M\}$ contains all but a negligible amount of the mass of Y, we

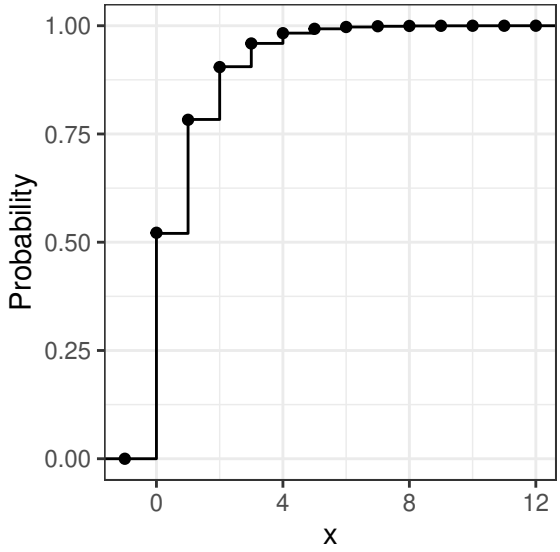


Figure 3: Empirical CDF of draws (solid line) versus CDF computed via the pcmp function (points).

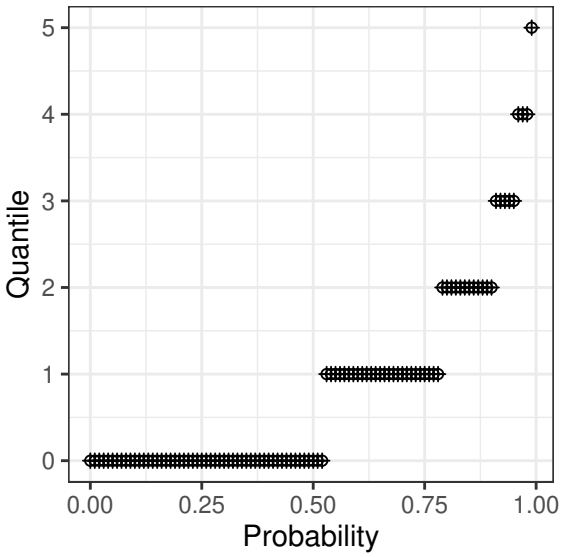


Figure 4: Empirical quantiles of draws (`o`) versus quantiles computed via the qcmp function (`+`).

can compute the moments using truncated summations

$$\mathrm{E}(Y) = \sum_{y=0}^{M} y \frac{\lambda^y}{(y!)^\nu Z(\lambda,\nu)}, \quad \mathrm{E}(Y^2) = \sum_{y=0}^{M} y^2 \frac{\lambda^y}{(y!)^\nu Z(\lambda,\nu)}, \quad \mathrm{Var}(Y) = \mathrm{E}(Y^2) - [\mathrm{E}(Y)]^2.$$

Otherwise, a different approach is taken. Notice that the expected value is related to the first derivative of the log-normalizing constant via

$$\frac{\partial}{\partial\lambda} \log Z(\lambda,\nu) = \frac{\frac{\partial}{\partial\lambda} Z(\lambda,\nu)}{Z(\lambda,\nu)} = \frac{1}{Z(\lambda,\nu)} \sum_{y=0}^{\infty} y \frac{\lambda^{y-1}}{(y!)^\nu}$$
$$\iff \mathrm{E}(Y) = \lambda \frac{\partial}{\partial\lambda} \log Z(\lambda,\nu).$$

For the second derivative,

$$\frac{\partial^2}{\partial\lambda^2} \log Z(\lambda,\nu) = \frac{Z(\lambda,\nu)\frac{\partial^2}{\partial\lambda^2} Z(\lambda,\nu) - [\frac{\partial}{\partial\lambda} Z(\lambda,\nu)]^2}{Z(\lambda,\nu)^2} = \frac{1}{Z(\lambda,\nu)} \sum_{y=0}^{\infty} y(y-1) \frac{\lambda^{y-2}}{(y!)^\nu} - \left[\frac{\mathrm{E}(Y)}{\lambda}\right]^2$$
$$\iff \lambda^2 \frac{\partial^2}{\partial\lambda^2} \log Z(\lambda,\nu) = \mathrm{E}[Y(Y-1)] - [\mathrm{E}(Y)]^2 = \mathrm{Var}(Y) - \mathrm{E}(Y)$$
$$\iff \mathrm{Var}(Y) = \lambda^2 \frac{\partial^2}{\partial\lambda^2} \log Z(\lambda,\nu) + \mathrm{E}(Y).$$

Therefore, we may use first and second derivatives of $\log Z(\lambda,\nu)$, taken with respect to λ, to compute $\mathrm{E}(Y)$ and $\mathrm{Var}(Y)$. The `ecmp` and `vcmp` functions implement this computation of the expectation and variance, respectively. The control object is used to determine whether to use the truncation or differentiation approach. Condition (2) is used to determine whether to use the truncation approach (if false) or differentiation approach (if true).

```
> ecmp(lambda = 10, nu = 1.2)
[1] 6.727397
> ecmp(lambda = 1.5, nu = 0.5)
[1] 2.815447
> ecmp(lambda = 1.5, nu = 0.05)
[1] 3334.757
> ecmp(lambda = 1.5, nu = 0.05, control = get.control(hybrid.tol = 1e-10))
[1] 3334.762
> ecmp(lambda = 1.5, nu = 0.05, control = get.control(hybrid.tol = 1e10))
[1] 3334.757
```

```
> vcmp(lambda = 10, nu = 1.2)
[1] 5.679667
> vcmp(lambda = 1.5, nu = 0.5)
[1] 4.513041
> vcmp(lambda = 1.5, nu = 0.05)
[1] 66505.13
> vcmp(lambda = 1.5, nu = 0.05, control = get.control(hybrid.tol = 1e-10))
[1] 66505.03
> vcmp(lambda = 1.5, nu = 0.05, control = get.control(hybrid.tol = 1e10))
[1] 66505.13
```

Provided that an enormously large truncation value M is not required, we may compute other moments by truncated sums using `tcmp`.

```
> M = tcmp(lambda = 1.5, nu = 0.05)
> print(M)
```

```
[1] 9039
> xx = seq(0, M)
> sum(xx^3 * dcmp(xx, lambda, nu))    # E(X^3)
[1] 7.794469
> sum(xx^4 * dcmp(xx, lambda, nu))    # E(X^4)
[1] 35.92968
```

3 Zero-Inflated Conway-Maxwell Poisson Distribution

Let $S \sim \text{Bernoulli}(p)$ and $T \sim \text{CMP}(\lambda, \nu)$ be independent random variables. Then $Y = (1-S)T$ follows a Zero-Inflated Conway-Maxwell Poisson distribution $\text{ZICMP}(\lambda, \nu, p)$ with density

$$f(y \mid \lambda, \nu, p) = (1-p)\frac{\lambda^y}{(y!)^\nu Z(\lambda, \nu)} + p \cdot \text{I}(y = 0), \quad y \in \mathbb{N}.$$

Like CMP, several interesting special cases are obtained.

a. Taking $\nu = 1$ corresponds to Zero-Inflated Poisson $\text{ZIP}(\lambda, p)$ with density $f(y \mid \lambda, p) = (1-p)e^{-\lambda}\lambda^y/y! + p \cdot \text{I}(y = 0)$.

b. When $\lambda \in (0, 1)$ and $\nu \to 0$, $\text{ZICMP}(\lambda, \nu)$ converges to a Zero-Inflated Geometric distribution with density $f(y \mid \lambda, p) = (1-p)(1-\lambda)\lambda^y + p \cdot \text{I}(y = 0)$ for $y \in \mathbb{N}$.

c. When $\nu \to \infty$, $\text{ZICMP}(\lambda, \nu, p)$ converges to a "Zero-Inflated Bernoulli" distribution with density $f(y \mid \lambda, p) = (1-p)\left[\lambda/(1+\lambda)\right]^y \left[1/(1+\lambda)\right]^{1-y} + p \cdot \text{I}(y = 0)$. which remains a Bernoulli distribution with an adjusted success probability. Here the λ and p parameters are not individually identifiable. Therefore, users may want to avoid zero-inflation in CMP data analyses with extreme underdispersion.

d. Finally, $p = 0$ yields $\text{CMP}(\lambda, \nu)$.

3.1 Density, Generation, CDF, and Quantile Functions

There is a close relationship between the CMP and ZICMP distributions, as we have seen from construction of ZICMP. The relationship between the CMP densities and variate generation mechanisms was given earlier in this section. Denote $F(x \mid \lambda, \nu)$ and $F(x \mid \lambda, \nu, p)$ as the CDFs of $\text{CMP}(\lambda, \nu)$ and $\text{ZICMP}(\lambda, \nu, p)$, respectively. We have

$$F(x \mid \lambda, \nu, p) = (1-p)F(x \mid \lambda, \nu) + p \cdot \text{I}(x \geq 0).$$

For a given probability $\phi \in [0, 1]$, the associated CMP and ZICMP quantile functions are related via

$$\begin{aligned} F^-(\phi \mid \lambda, \nu, p) &= \inf\{x \in \mathbb{N} : F(x \mid \lambda, \nu, p) \geq \phi\} \\ &= \inf\{x \in \mathbb{N} : (1-p)F(x \mid \lambda, \nu) + p \cdot \text{I}(x \geq 0) \geq \phi\} \\ &= \inf\{x \in \mathbb{N} : F(x \mid \lambda, \nu) \geq (\phi - p)/(1-p)\} \\ &= F^-\left(\frac{\phi - p}{1-p} \mid \lambda, \nu\right). \end{aligned} \tag{9}$$

The respective functions for ZICMP density, variate generation, CDF, and quantile functions are `dzicmp`, `rzicmp`, `pzicmp` , and `qzicmp`. They make use of the CMP implementation described in Section 2 such as the criteria to either truncate or approximate the normalizing constant.

```
> qq = seq(0, 0.95, length.out = 20)
> rzicmp(20, lambda = 1.5, nu = 0.2, p = 0.25)
 [1] 10 17 14  8  0 11  0 19 12 16  8  0 10  0  0 12  8 10 12  0
> dzicmp(c(0, 1, 2), lambda = 1.5, nu = 0.2, p = 0.25)
```

```
[1] 0.26630689 0.02446034 0.03194095
> pzicmp(c(0, 1, 2), lambda = 1.5, nu = 0.2, p = 0.25)
[1] 0.2663069 0.2907672 0.3227082
> qzicmp(qq, lambda = 1.5, nu = 0.2, p = 0.25)
 [1]  0  0  0  0  0  0  2  3  4  5  6  7  8  9 11 12 13 15 17 20
```

As with `qcmp`, the `qzicmp` function is computed by the truncation method, and cannot accurately compute quantiles very close to 1. More specifically, from the CMP distribution, (8) gives

$$\begin{aligned} M &> F^-(1-\epsilon \mid \lambda, \nu) \\ &= F^-\left(\frac{\phi_\epsilon - p}{1-p} \mid \lambda, \nu\right) \\ &= F^-\left(\phi_\epsilon \mid \lambda, \nu, p\right) \end{aligned}$$

where $\phi_\epsilon = (1-\epsilon)(1-p)+p$ and the last equality uses (9). This motivates a warning from `qzicmp` when the argument is larger than ϕ_ϵ.

```
> tryCatch({
+   qzicmp(0.9999999, lambda = 1.5, nu = 0.5, p = 0.5)
+ }, warning = print_warning)
[1] simpleWarning in qzicmp(0.9999999, lambda = 1.5, nu = 0.5, p = 0.5): At
[2] least one requested quantile was very close to 1. In particular, 1 of
[3] the given probabilities were greater than (1 - truncate.tol) * (1-p) +
[4] p, where truncate_tol = 1e-06. Associated results may be affected by
[5] truncation. Consider adjusting the controls ymax and truncate.tol or
[6] reducing logq.
```

Let us repeat the sanity check from Section 2.2 to ensure that the empirical density values, cumulative probabilities, and quantiles line up with the ones computed via `dzicmp`, `pzicmp`, and `qzicmp`.

```
library(ggplot2)

n = 100000
lambda = 0.5
nu = 0.1
p = 0.5
x = rzicmp(n, lambda, nu, p)

xx = seq(-1, max(x))  ## Include -1 to ensure it gets probability zero
qq = seq(0, 0.99, length.out = 100)

fx = dzicmp(xx, lambda, nu, p)
px = pzicmp(xx, lambda, nu, p)
qx = qzicmp(qq, lambda, nu, p)

qx_emp = quantile(x, probs = qq)
```

```
ggplot() +
    geom_bar(data = data.frame(x = x), aes(x = x, y = ..prop..), fill = "NA",
        col = "black") +
    geom_point(data = data.frame(x = xx[-1], fx = fx[-1]), aes(x, fx)) +
    ylab("Density") +
    theme_bw()
```

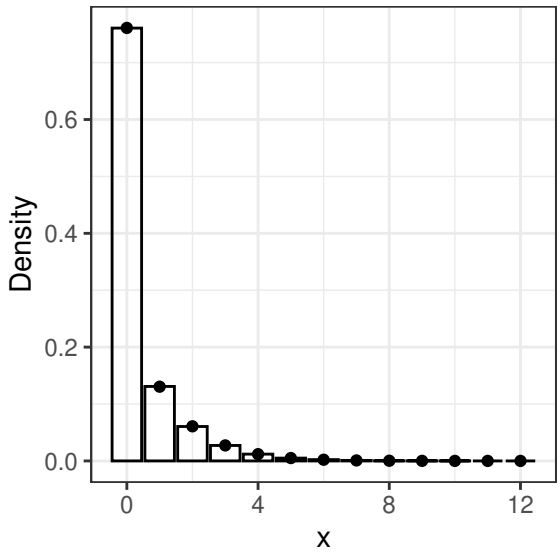


Figure 5: Empirical density of draws (histogram) versus density computed via the dzicmp function (points).

```
ggplot() +
    stat_ecdf(data = data.frame(x = x), aes(x), geom = "step") +
    geom_point(data = data.frame(x = xx, px = px), aes(x, px)) +
    ylab("Probability") +
    theme_bw()
```

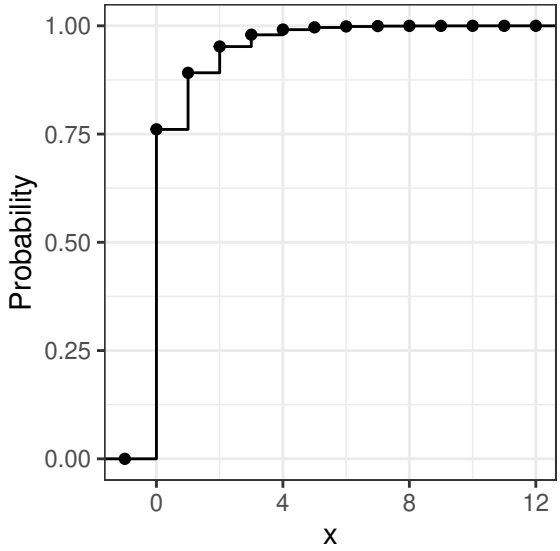


Figure 6: Empirical CDF of draws (solid line) versus CDF computed via the pzicmp function (points).

```
ggplot() +
    geom_point(data = data.frame(x = qq, qx_emp = qx_emp), aes(qq, qx_emp), pch = 1) +
    geom_point(data = data.frame(x = qq, qx = qx), aes(qq, qx), pch = 3) +
    xlab("Probability") +
    ylab("Quantile") +
    theme_bw()
```

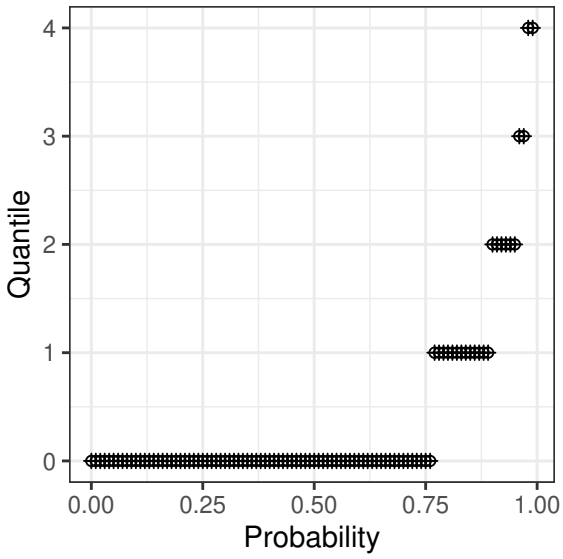


Figure 7: Empirical quantiles of draws (o) versus quantiles computed via the qzicmp function (+).

3.2 Expectation and Variance

The expected value and variance of $Y \sim \text{ZICMP}(\lambda, \nu, p)$ are apparent from the construction $Y = (1 - S)T$ given earlier in this section. Namely,

$$\text{E}(Y) = (1 - p)\,\text{E}(T) \quad \text{and} \quad \text{Var}(Y) = (1 - p)\left\{\text{Var}(T) + p[\text{E}(T)]^2\right\}$$

may be obtained using formulas for iterated conditional expectations and variances. They are evaluated in `COMPoissonReg` using the `ezicmp` and `vzicmp` functions respectively. These functions make use of the `ecmp` and `vcmp` functions described in Section 2.3 to compute $\text{E}(T)$ and $\text{Var}(T)$.

```
> ezicmp(lambda = 1.5, nu = 0.5, p = 0.1)
[1] 2.533903
> ezicmp(lambda = 1.5, nu = 0.5, p = c(0.1, 0.2, 0.5))
[1] 2.533903 2.252358 1.407724
```

```
> vzicmp(lambda = 1.5, nu = 0.5, p = 0.1)
[1] 4.775144
> vzicmp(lambda = 1.5, nu = 0.5, p = c(0.1, 0.2, 0.5))
[1] 4.775144 4.878712 4.238206
```

4 Regression Modeling with CMP and ZICMP

Suppose there are n subjects with outcomes $y_1, \ldots, y_n \in \mathbb{N}$ and covariates $\boldsymbol{x}_i \in \mathbb{R}^{d_1}$, $\boldsymbol{s}_i \in \mathbb{R}^{d_2}$, and $\boldsymbol{w}_i \in \mathbb{R}^{d_3}$ for $i = 1, \ldots, n$. The `COMPoissonReg` package fits both CMP and ZICMP regression models.

The CMP regression model assumes that

$$Y_i \overset{\text{ind}}{\sim} \text{CMP}(\lambda_i, \nu_i), \quad i = 1, \ldots, n,$$

where $\log \lambda_i = \boldsymbol{x}_i^\top \boldsymbol{\beta}$ and $\log \nu_i = \boldsymbol{s}_i^\top \boldsymbol{\gamma}$. Writing $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{\gamma})$, the likelihood is

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} \left[\frac{\lambda_i^{y_i}}{(y_i!)^{\nu_i} Z(\lambda_i, \nu_i)} \right]. \tag{10}$$

The ZICMP regression model assumes that

$$Y_i \overset{\text{ind}}{\sim} \text{ZICMP}(\lambda_i, \nu_i, p_i), \quad i = 1, \ldots, n,$$

where $\log \lambda_i = \boldsymbol{x}_i^\top \boldsymbol{\beta}$, $\log \nu_i = \boldsymbol{s}_i^\top \boldsymbol{\gamma}$, and $\operatorname{logit} p_i = \boldsymbol{w}_i^\top \boldsymbol{\zeta}$. Writing $\boldsymbol{\theta} = (\boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\zeta})$, the likelihood is

$$L(\boldsymbol{\theta}) = \prod_{i=1}^{n} \left[(1 - p_i) \frac{\lambda_i^{y_i}}{(y_i!)^{\nu_i} Z(\lambda_i, \nu_i)} + p_i \, \mathrm{I}(y_i = 0) \right]. \tag{11}$$

We will write $d = d_1 + d_2 + d_3$ for the total dimension of $\boldsymbol{\theta}$. The $n \times d_1$ design matrix whose rows consist of $\boldsymbol{x}_i$ will be denoted $\boldsymbol{X}$. Similarly, we will write $\boldsymbol{S}$ and $\boldsymbol{W}$ as the $n \times d_2$ and $n \times d_3$ design matrices constructed from $\boldsymbol{s}_i$ and $\boldsymbol{w}_i$, respectively. The `glm.cmp` function provides a formula interface to fit models of both types: (10) and (11).

```
out = glm.cmp(formula.lambda, formula.nu = ~ 1, formula.p = NULL,
    data = NULL, init = NULL, fixed = NULL, control = NULL, ...)
```

The interface contains three formulas: `formula.lambda` specifies the regression $\boldsymbol{x}_i^\top \boldsymbol{\beta}$ used for λ_i, while `formula.nu` and `formula.p` correspond to $\boldsymbol{s}_i^\top \boldsymbol{\gamma}$ for ν_i and $\boldsymbol{w}_i^\top \boldsymbol{\zeta}$ for p_i, respectively. ZICMP regression is utilized when `formula.p` is set to something other than its default `NULL` value; otherwise, CMP regression is assumed. The `data` argument is used to pass a `data.frame` explicitly rather than having the data be read from the local environment. The `init`, `fixed`, and `control` arguments and associated helper functions are described below.

The `init` argument represents an initial value for the optimizer. The following functions can be used to construct it.

```
> get.init(beta = c(1, 2, 3), gamma = c(-1, 1), zeta = c(-2, -1))
$beta
[1] 1 2 3

$gamma
[1] -1  1

$zeta
[1] -2 -1

attr(,"class")
[1] "COMPoissonReg.init"
> get.init.zero(d1 = 3, d2 = 2, d3 = 2)
$beta
[1] 0 0 0

$gamma
[1] 0 0

$zeta
[1] 0 0

attr(,"class")
[1] "COMPoissonReg.init"
```

The `fixed` argument is used to specify indices of the three coefficients which will remain fixed at their initial value during optimization.

```
> get.fixed(beta = c(1L, 2L), gamma = c(1L))
$beta
[1] 1 2

$gamma
[1] 1

$zeta
integer(0)

attr(,"class")
[1] "COMPoissonReg.fixed"
```

The specification above requests the first two elements of `beta` and the first element of `gamma` to be fixed. Notice that indices must be integers and that the default value is an empty integer vector which is interpreted as "no elements are fixed". The `fixed` argument can usually be disregarded but may be useful in some circumstances; an example is given in Section 4.2.

Specifying the elements of `init` and `fixed` may be somewhat awkward with the formula interface, as they require knowledge of how formulas will be expanded into design matrices and coefficients. It can be helpful to produce the design matrices using R's `model.matrix` function.

```
> model.matrix(formula.lambda, data = data)
> model.matrix(formula.nu, data = data)
> model.matrix(formula.p, data = data)
```

The `control` argument has been introduced in Section 2; regression modeling makes use of several additional arguments. `COMPoissonReg` uses `optim` to compute the maximum likelihood estimate (MLE) $\hat{\boldsymbol{\theta}}$ for $\boldsymbol{\theta}$ under the specified model. Several controls are provided to influence how `COMPoissonReg` invokes `optim`; here are their default values.

```
> control = getOption("COMPoissonReg.control")
> control$optim.method
[1] "L-BFGS-B"
> control$optim.control
$maxit
[1] 150
```

The element `optim.method` is a string which is passed as the `method` argument to `optim`, while `optim.control` is a list passed as the `control` argument of `optim`. Note that, for the latter, if an entry is given for `fnscale`, it will be ignored and overwritten internally by `COMPoissonReg`.

The covariance of $\hat{\boldsymbol{\theta}}$ is estimated by $\hat{\boldsymbol{V}}(\hat{\boldsymbol{\theta}}) = -[\boldsymbol{H}(\hat{\boldsymbol{\theta}})]^{-1}$, where $\boldsymbol{H}(\boldsymbol{\theta}) = \partial^2 \log L(\boldsymbol{\theta})/[\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}^\top]$ is the Hessian of the log-likelihood computed by `optim`. The standard error for coefficient θ_j in $\boldsymbol{\theta}$ is then obtained as the square root of the jth diagonal of $\hat{\boldsymbol{V}}(\hat{\boldsymbol{\theta}})$.

We will now illustrate use of the regression tools using two examples whose data are included in the package. Note that these demonstrations are not intended to be complete regression analyses, and results may be slightly different than previously published analyses due to differences in the computations.

4.1 CMP Regression

4.1.1 Freight Dataset

The `freight` dataset [Kutner et al., 2003] was analyzed using CMP regression by Sellers and Shmueli [2010] and found to exhibit underdispersion. The data describe $n = 10$ instances where 1,000 ampules were transported via air shipment. The outcome of interest is the variable `broken` which describes the number of

broken ampules in each shipment. The covariate `transfers` describes the number of times the carton was transferred from one aircraft to another during the shipment.

Let us load and view the dataset.

```
> data(freight)
> print(freight)
   broken transfers
1      16         1
2       9         0
3      17         2
4      12         0
5      22         3
6      13         1
7       8         0
8      15         1
9      19         2
10     11         0
```

Before fitting a CMP regression, let us fit a Poisson regression model $Y_i \overset{\text{ind}}{\sim} \text{Poisson}(\lambda_i)$ with

$$\log \lambda_i = \beta_0 + \beta_1 \cdot \text{transfers}_i.$$

This can be carried out with the standard `glm` function.

```
> glm.out = glm(broken ~ transfers, data = freight, family = poisson)
> summary(glm.out)

Call:
glm(formula = broken ~ transfers, family = poisson, data = freight)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  2.35295    0.13174   17.86  < 2e-16 ***
transfers    0.26384    0.07924    3.33 0.000869 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 12.5687  on 9  degrees of freedom
Residual deviance:  1.8132  on 8  degrees of freedom
AIC: 50.395

Number of Fisher Scoring iterations: 4
```

4.1.2 CMP Regression

Next, let us fit a similar CMP regression model with

$$\log \lambda_i = \beta_0 + \beta_1 \cdot \text{transfers}_i,$$
$$\log \nu_i = \gamma_0,$$

using only an intercept for ν_i.

```
> cmp.out = glm.cmp(broken ~ transfers, data = freight)
> print(cmp.out)
```

```
CMP coefficients
              Estimate     SE z-value   p-value
X:(Intercept)  13.8286 6.2405  2.2159    0.0267
X:transfers     1.4843 0.6892  2.1537   0.03127
S:(Intercept)   1.7550 0.4493  3.9065 9.365e-05
--
Transformed intercept-only parameters
   Estimate     SE
nu   5.7835 2.5982
--
Chi-squared test for equidispersion
X^2 = 9.1048, df = 1, p-value = 2.5494e-03
--
Elapsed: 0.05 sec   Sample size: 10   formula interface
LogLik: -18.6449   AIC: 43.2898   BIC: 44.1975
Optimization Method: L-BFGS-B   Converged status: 0
Message: CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH
```

The coefficients used in the λ_i formula are prefixed with an `X:` label, while an `S:` label is used for coefficients of the ν_i formula. Notice that estimates for `X:` coefficients from the CMP fit are dissimilar to those from the Poisson fit; this may occur when the estimate of ν deviates from the value of 1. Similarly to the `glm` output, the output of `glm.cmp` displays several quantities for each coefficient θ_j, $j = 1, \ldots, d$: a point estimate $\hat{\theta}_j$, an associated standard error $\widehat{\mathrm{SE}}(\hat{\theta}_j)$, a z-value $z_j = \hat{\theta}_j / \widehat{\mathrm{SE}}(\hat{\theta}_j)$, and a p-value $2\Phi(-|z_j|)$ for the test $H_0 : \theta_j = 0$ versus $H_1 : \theta_j \neq 0$. Here, Φ is the CDF of the standard normal distribution. Because an intercept-only formula was specified for ν_i, $\hat{\nu} = \exp(\hat{\gamma})$ does not vary with i and its estimate and associated standard error are added to the display. Here we see evidence of underdispersion with $\hat{\nu} > 1$. A test for equidispersion is displayed to determine whether there is a significant amount of over or underdispersion in the data. In particular, a likelihood ratio test is used to decide whether $H_0 : \boldsymbol{\gamma} = \mathbf{0}$ versus $H_0 : \boldsymbol{\gamma} \neq \mathbf{0}$. The test statistic is displayed along with the degrees of freedom and associated p-value. Here we have fairly strong evidence to reject the null hypothesis of equidispersion.

The AIC for the CMP model `cmp.out` shows some improvement over that of `glm.out`. Let us also consider a slope coefficient for the ν component using

$$\log \nu_i = \gamma_0 + \gamma_1 \cdot \mathrm{transfers}_i.$$

via the following call to `glm.cmp`.

```
> cmp2.out = glm.cmp(broken ~ transfers, formula.nu = ~ transfers, data = freight)
> print(cmp2.out)
CMP coefficients
              Estimate     SE z-value   p-value
X:(Intercept)  15.5959 7.1087  2.1939   0.02824
X:transfers     4.6322 2.3978  1.9319   0.05338
S:(Intercept)   1.8935 0.4525  4.1846 2.857e-05
S:transfers     0.1206 0.0505  2.3883   0.01693
--
Chi-squared test for equidispersion
X^2 = 11.6991, df = 2, p-value = 2.8812e-03
--
Elapsed: 0.08 sec   Sample size: 10   formula interface
LogLik: -17.3477   AIC: 42.6954   BIC: 43.9058
Optimization Method: L-BFGS-B   Converged status: 0
Message: CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH
```

Model `cmp2.out` provides a slight improvement over `cmp.out` in AIC and BIC, but we may prefer `cmp.out`

in the interest of simplicity.

4.1.3 Adjustments to Optim

To gain some insight into the optimization, we may wish to increase the trace level, which can be done as follows.

```
> control = get.control(optim.control = list(maxit = 5, trace = 3, REPORT = 1))
> cmp3.out = glm.cmp(broken ~ transfers, data = freight, control = control)
N = 3, M = 5 machine precision = 2.22045e-16
This problem is unconstrained.
At X0, 0 variables are exactly at the bounds
At iterate     0  f=       273.34  |proj g|=       260.29
At iterate     1  f =       49.923  |proj g|=           113
At iterate     2  f =       46.155  |proj g|=        44.594
At iterate     3  f =       29.883  |proj g|=        22.868
At iterate     4  f =       27.657  |proj g|=        2.6174
At iterate     5  f =       27.599  |proj g|=         4.443
At iterate     6  f =       27.359  |proj g|=        7.3357
final  value 27.359209
stopped after 6 iterations
N = 2, M = 5 machine precision = 2.22045e-16
This problem is unconstrained.
At X0, 0 variables are exactly at the bounds
At iterate     0  f=       148.87  |proj g|=       152.94
At iterate     1  f =       57.259  |proj g|=        146.63
At iterate     2  f =       53.033  |proj g|=        67.913
At iterate     3  f =       42.039  |proj g|=        42.645
At iterate     4  f =       36.942  |proj g|=        38.126
At iterate     5  f =       32.584  |proj g|=        43.852
At iterate     6  f =        24.06  |proj g|=        17.695
final  value 24.060495
stopped after 6 iterations
```

Data from the local environment may be passed to the `glm.cmp` function without explicitly using the `data` argument.

```
> y = freight$broken
> x = freight$transfers
> glm.cmp(y ~ x)
```

4.1.4 Offset Term

In a count regression model, it may be desirable to include offset terms such as

$$\log \lambda_i = \boldsymbol{x}_i^\top \boldsymbol{\beta} + \text{offx}_i, \quad \log \nu_i = \boldsymbol{s}_i^\top \boldsymbol{\beta} + \text{offs}_i.$$

An `offset` term may be used in the formula interface to accomplish this.

```
> freight$offx = 13
> freight$offs = 1
> glm.cmp(broken ~ transfers + offset(offx), data = freight)
> glm.cmp(broken ~ transfers + offset(offx), formula.nu = ~1 + offset(offs), data = freight)
```

For users who wish to bypass the formula interface and prepare the $\boldsymbol{X}$ and $\boldsymbol{S}$ design matrices manually, a "raw" interface to the regression functionality is also provided.

```
> y = freight$broken
> X = model.matrix(~ transfers, data = freight)
> S = model.matrix(~ 1, data = freight)
> offs = get.offset(x = rep(13, nrow(freight)), s = rep(1, nrow(freight)))
> cmp.raw.out = glm.cmp.raw(y, X, S, offset = offs)
```

4.1.5 Accessor Functions

Several accessors are provided to extract results from the output object.

```
> logLik(cmp.out)                ## Log-likelihood evaluated at MLE.
[1] -18.64489
> AIC(cmp.out)                   ## AIC evaluated at MLE.
[1] 43.28978
> BIC(cmp.out)                   ## BIC evaluated at MLE.
[1] 44.19754
> coef(cmp.out)                  ## Estimates of theta as a flat vector
X:(Intercept)   X:transfers S:(Intercept)
    13.828561      1.484303      1.755002
> coef(cmp.out, type = "list")   ## Estimates of theta as a named list
$beta
X:(Intercept)   X:transfers
    13.828561      1.484303

$gamma
S:(Intercept)
     1.755002
> vcov(cmp.out)                  ## Estimated covariance matrix of theta hat
              X:(Intercept) X:transfers S:(Intercept)
X:(Intercept)     38.944034   4.0877545     2.8000943
X:transfers        4.087754   0.4749988     0.2978880
S:(Intercept)      2.800094   0.2978880     0.2018301
> sdev(cmp.out)                  ## Standard deviations from vcov(...) diagonals
X:(Intercept)   X:transfers S:(Intercept)
    6.2405155     0.6892016     0.4492551
> sdev(cmp.out, type = "list")   ## Standard deviations as a named list
$beta
[1] 6.2405155 0.6892016

$gamma
[1] 0.4492551
```

The `predict` function computes several useful quantities evaluated at the estimate $\hat{\boldsymbol{\theta}}$. The argument default `type = "response"` produces a vector of estimates of the response $\hat{y}_i = \mathrm{E}(Y_i)$ for $i = 1, \ldots, n$ using the method described in Section 2.3. The argument `type = "link"` produces a `data.frame` with columns for the linked parameters λ_i and ν_i.

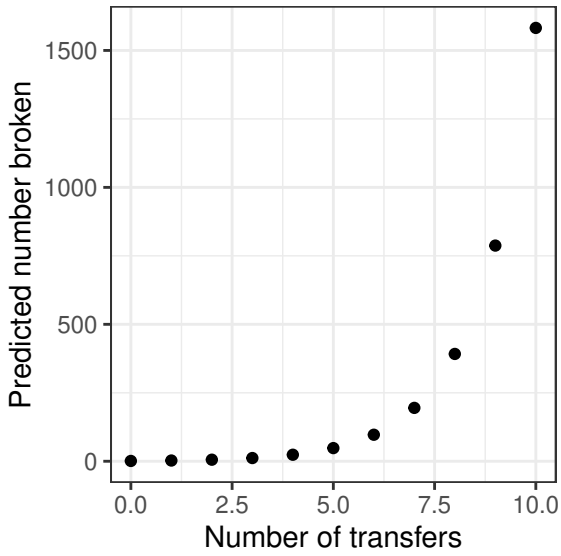
```
> predict(cmp.out)
 [1] 13.70507 10.50769 17.83752 10.50769 23.17871 13.70507 10.50769 13.70507
 [9] 17.83752 10.50769
> predict(cmp.out, type = "link")
     lambda      nu
1   4469845 5.78346
2   1013136 5.78346
3  19720454 5.78346
```

```
4   1013136 5.78346
5  87004438 5.78346
6   4469845 5.78346
7   1013136 5.78346
8   4469845 5.78346
9  19720454 5.78346
10  1013136 5.78346
```

Note that the estimated `nu` values are equal for all observations because the model assumed only an intercept term for the dispersion component.

We can also use `predict` on new covariate values. Note that models fit with the formula interface expect the new data to be provided as a `data.frame` which is interpreted using the formula used to fit the model. If the raw interface was used to fit the model, use the `get.modelmatrix` function to specify design matrices to use for prediction.

```
# Prepare new data to fit by formula interface
new.df = data.frame(transfers = 0:10)

# Prepare new data to fit by raw interface
X = model.matrix(~ transfers, data = new.df)
S = model.matrix(~ 1, data = new.df)
new.data = get.modelmatrix(X = X, S = S)

# Pass new data to model from by formula interface
y.hat.new = predict(cmp.out, newdata = new.df)

# Pass new data to model from by raw interface
y.hat.new = predict(cmp.raw.out, newdata = new.data)

# Compute predictions for links
predict.out = predict(cmp.out, newdata = new.df, type = "link")

# Plot predictions
ggplot() +
    geom_point(data = new.df, aes(transfers, y.hat.new)) +
    xlab("Number of transfers") +
    ylab("Predicted number broken") +
    theme_bw()
```




```
> print(y.hat.new)
 [1]    1.176542    2.690927    5.694026   11.715378   23.807032   48.096062
 [7]   96.889869  194.912671  391.832565  787.430083 1582.156271
> print(predict.out)
         lambda      nu
1  1.013136e+06 5.78346
2  4.469845e+06 5.78346
3  1.972045e+07 5.78346
4  8.700444e+07 5.78346
5  3.838538e+08 5.78346
6  1.693520e+09 5.78346
7  7.471622e+09 5.78346
8  3.296396e+10 5.78346
9  1.454333e+11 5.78346
10 6.416355e+11 5.78346
11 2.830824e+12 5.78346
```

The `leverage` function computes the diagonal entries of a "hat" matrix which can be formulated in CMP regression. These can be used to diagnose influential observations. For details, see Section 3.6 of Sellers and Shmueli [2010].

```
> leverage(cmp.out)
 [1] 0.1541752 0.1943925 0.2733327 0.2259140 0.6002368 0.3795226 0.2379250
 [8] 0.1117655 0.3653074 0.4574283
```

The `residuals` function provides either raw (the default) or quantile-based residuals [Dunn and Smyth, 1996]. In a CMP regression setting, raw residuals $y_i - \hat{y}_i$ generally do not work well with traditional regression diagnostics, such as Q-Q plots. Quantile-based residuals often produce interpretable diagnostics; however, a random element is used in the computation of quantile residuals for discrete distributions. This aids interpretability but gives slightly different residual values each time they are computed. See Dunn and Smyth [1996] for details.

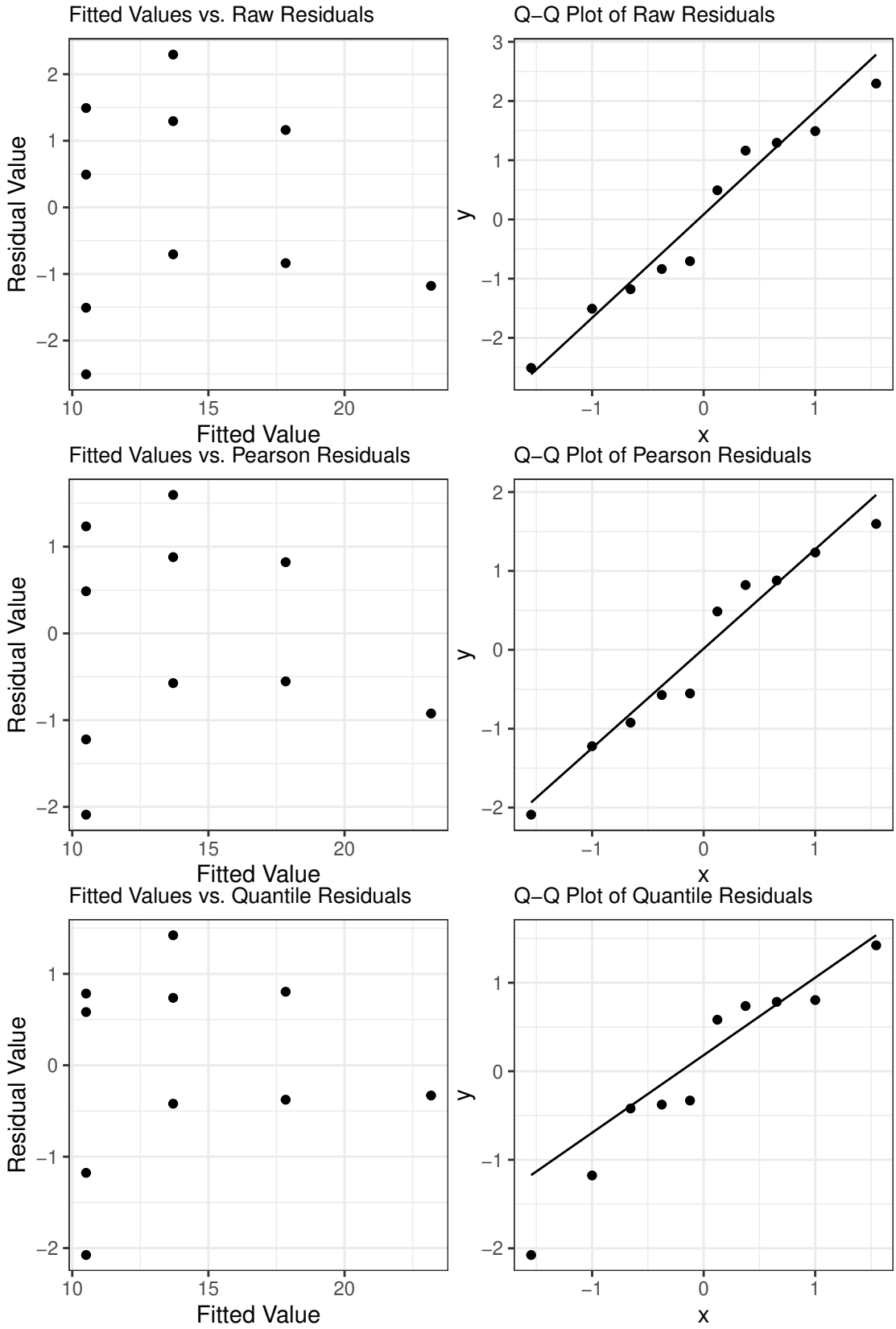
```
> res.raw = residuals(cmp.out)
> res.qtl = residuals(cmp.out, type = "quantile")
```

Pearson residuals may be preferred over raw residuals for diagnostics; these can be obtained by standardizing raw residuals using leverage values and variance estimates.

```
> link.hat = predict(cmp.out, type = "link")
> vv = vcmp(link.hat$lambda, link.hat$nu)
> hh = leverage(cmp.out)
> res.pearson = res.raw / sqrt(vv*(1-hh))
```

For each type of residual—raw, Pearson, and quantile—we now plot fitted values versus residuals and Q-Q plots.

```
plot.fit.res = function(y.hat, res) {
    ggplot(data.frame(y = y.hat, res = res)) +
        geom_point(aes(y, res)) +
        xlab("Fitted Value") +
        ylab("Residual Value") +
        theme_bw() +
        theme(plot.title = element_text(size = 10))
}

plot.qq.res = function(res) {
    ggplot(data.frame(res = res), aes(sample = res)) +
        stat_qq() +
        stat_qq_line() +
        theme_bw() +
        theme(plot.title = element_text(size = 10))
}

y.hat = predict(cmp.out)

plot.fit.res(y.hat, res.raw) +
    ggtitle("Fitted Values vs. Raw Residuals")
plot.qq.res(res.raw) +
    ggtitle("Q-Q Plot of Raw Residuals")

plot.fit.res(y.hat, res.pearson) +
    ggtitle("Fitted Values vs. Pearson Residuals")
plot.qq.res(res.pearson) +
    ggtitle("Q-Q Plot of Pearson Residuals")

plot.fit.res(y.hat, res.qtl) +
    ggtitle("Fitted Values vs. Quantile Residuals")
plot.qq.res(res.qtl) +
    ggtitle("Q-Q Plot of Quantile Residuals")
```



In this example, with only 10 observations, it is difficult to see an advantage of using quantile residuals; the benefit will be more apparent in Section 4.2. One benefit of raw residuals is that they may be used to

compute a mean-squared error.

```
> mean(res.raw^2)
[1] 2.191384
```

To access the results of the equidispersion test shown in the output of `cmp.out`, we may use the `equitest` accessor function.

```
> equitest(cmp.out)
$teststat
[1] 9.104772

$pvalue
[1] 0.002549436

$df
[1] 1
```

The `deviance` function computes the deviance quantities $D_i = -2[\log L_i(\hat{\boldsymbol{\theta}}) - \log L_i(\tilde{\boldsymbol{\theta}}_i)]$ for $i = 1, \ldots, n$, where $L_i(\boldsymbol{\theta})$ is the term of the likelihood corresponding to the ith observation, $\hat{\boldsymbol{\theta}}$ is the MLE computed under the full likelihood $L(\boldsymbol{\theta}) = \prod_{i=1}^{n} L_i(\boldsymbol{\theta})$, and $\tilde{\boldsymbol{\theta}}_i$ is the maximizer of $L_i(\boldsymbol{\theta})$.

```
> deviance(cmp.out)
 [1]  0.3376988 -0.5150035 -1.7750671 -0.6863613 -2.2081582 -1.9383878
 [7]  2.1710154 -1.1354954 -1.6584238 -2.1772063
```

The `parametric.bootstrap` function carries out a parametric bootstrap with R repetitions. Using the fitted MLE $\hat{\boldsymbol{\theta}}$, bootstrap samples $\boldsymbol{y}^{(r)} = (y_1^{(r)}, \ldots, y_n^{(r)})$ are drawn from the likelihood $L(\hat{\boldsymbol{\theta}})$ for $r = 1, \ldots, R$. Estimate $\hat{\boldsymbol{\theta}}^{(r)}$ is fitted from bootstrap sample $\boldsymbol{y}^{(r)}$. An $R \times d$ matrix is returned whose rth row is $\hat{\boldsymbol{\theta}}^{(r)}$.

```
> cmp.boot = parametric.bootstrap(cmp.out, reps = 100)
> head(cmp.boot)
     (Intercept) transfers (Intercept)
[1,]   12.435045  1.066322    1.620051
[2,]   22.895677  2.768967    2.297103
[3,]   21.989757  2.518793    2.223626
[4,]   26.788708  2.663686    2.416597
[5,]   15.550523  2.011527    1.873008
[6,]    8.892033  1.240461    1.348294
```

We used $R = 100$ in the display above to keep vignette computations small, but a larger number may be desired in practice. Bootstrap repetitions can be used, for example, to compute 95% confidence intervals for each of the coefficients.

```
> t(apply(cmp.boot, 2, quantile, c(0.025,0.975)))
                 2.5%     97.5%
(Intercept) 8.2489159 55.910246
transfers   0.8240959  5.806495
(Intercept) 1.2567221  3.135545
```

4.1.6 Large Covariates

Large covariates can present numerical difficulties in fitting CMP regression. We will briefly demonstrate the difficulties and some possible workarounds. First let us generate a new dataset based on a large covariate in the regression for λ_i.

```
set.seed(1234)
n = 200
```

```
x = rnorm(n, 500, 10)
X = cbind(intercept = 1, slope = x)
S = matrix(1, n, 1)
beta_true = c(-0.05, 0.05)
gamma_true = 2
lambda_true = exp(X %*% beta_true)
nu_true = exp(S %*% gamma_true)
y = rcmp(n, lambda_true, nu_true)
```

Notice that the generated counts $y_1, \ldots, y_n$ are relatively small compared to the covariate $x_1, \ldots, x_n$.

```
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  471.4   492.3   498.3   499.4   505.5   530.4
> summary(y)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   22.0    27.0    29.0    29.1    31.0    37.0
```

An initial attempt to fit the true data-generating model fails.

```
> tryCatch({
+   glm.cmp(y ~ x, formula.nu = ~ 1)
+ }, error = print_warning)
[1] Error in optim(par.init, loglik, method = optim.method, control =
[2] optim.control, : L-BFGS-B needs finite values of 'fn'
```

Internally, the linked rate parameter $\lambda_i = \exp(\beta_0 + \beta_1 x_i)$ may evaluate to `Inf` or become very close to zero as the optimizer moves β_1 away from zero in a positive or negative direction, respectively. Some possible ways to address this are as follows.

Standardize the covariate to have mean zero and variance one.

```
> glm.cmp(y ~ scale(x), formula.nu = ~ 1)
CMP coefficients
              Estimate     SE z-value   p-value
X:(Intercept)  26.0617 2.5565 10.1944   2.1e-24
X:scale(x)      0.5253 0.0627  8.3848 5.082e-17
S:(Intercept)   2.0416 0.0980 20.8290 2.365e-96
--
Transformed intercept-only parameters
   Estimate    SE
nu   7.7031 0.755
--
Chi-squared test for equidispersion
X^2 = 232.4625, df = 1, p-value = 1.7311e-52
--
Elapsed: 0.65 sec   Sample size: 200   formula interface
LogLik: -417.9344   AIC: 841.8689   BIC: 851.7638
Optimization Method: L-BFGS-B   Converged status: 0
Message: CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH
```

Use a logarithmic transformation on the covariate.

```
> glm.cmp(y ~ log(x), formula.nu = ~ 1)
CMP coefficients
               Estimate      SE z-value   p-value
X:(Intercept) -134.3015 17.4934 -7.6773 1.625e-14
```

```
X:log(x)        25.8123  3.1597  8.1693 3.103e-16
S:(Intercept)    2.0422  0.1017 20.0764 1.188e-89
--
Transformed intercept-only parameters
   Estimate    SE
nu   7.7074 0.784
--
Chi-squared test for equidispersion
X^2 = 232.3911, df = 1, p-value = 1.7943e-52
--
Elapsed: 0.65 sec   Sample size: 200   formula interface
LogLik: -417.9869   AIC: 841.9738   BIC: 851.8688
Optimization Method: L-BFGS-B   Converged status: 0
Message: CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH
```

Change optimization method or other `optim` arguments.

```
> control = get.control(optim.method = "BFGS", optim.control = list(maxit = 200))
> suppressWarnings({
+   cmp.out = glm.cmp(y ~ x, formula.nu = ~ 1, control = control)
+   print(cmp.out)
+ })
CMP coefficients
              Estimate     SE  z-value p-value
X:(Intercept)  12.1759 0.0467 260.6333       0
X:x            -0.0210    NaN      NaN     NaN
S:(Intercept)  -0.6747    NaN      NaN     NaN
--
Transformed intercept-only parameters
   Estimate  SE
nu   0.5093 NaN
--
Chi-squared test for equidispersion
X^2 = 254.5417, df = 1, p-value = 2.6567e-57
--
Elapsed: 0.13 sec   Sample size: 200   formula interface
LogLik: -937.3960   AIC: 1880.7921   BIC: 1890.6870
Optimization Method: BFGS   Converged status: 0
```

In this case, standardization and logarithmic transformation produce a usable fit. Changing the optimization method to `BFGS` allows the optimization to finish, but there are further numerical problems in computing the Hessian for standard errors.

4.1.7 Large Outcomes

Now consider a generated dataset with large outcomes but a relatively small covariate. This situation can also present numerical difficulties.

```
set.seed(1234)
n = 200
x = runif(n, 1, 2)
X = cbind(intercept = 1, slope = x)
S = matrix(1, n, 1)
beta_true = c(1, 1)
gamma_true = -0.95
```

```
lambda_true = exp(X %*% beta_true)
nu_true = exp(S %*% gamma_true)
y = rcmp(n, lambda_true, nu_true)
```

```
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.009   1.238   1.486   1.483   1.735   1.999
> summary(y)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  156.0   332.0   606.5   810.5  1161.2  2391.0
```

An initial attempt to fit the data-generating model fails.

```
> tryCatch({
+   glm.cmp(y ~ x, formula.nu = ~ 1)
+ }, error = print_warning)
[1] Error in optim(par.init, loglik, method = optim.method, control =
[2] optim.control, : L-BFGS-B needs finite values of 'fn'
```

Informative starting values help the optimizer to initially make progress. True data-generating parameters will not be available in a real data analysis situation but help to illustrate the idea.

```
> init = get.init(beta = beta_true, gamma = gamma_true)
> glm.cmp(y ~ x, formula.nu = ~ 1, init = init)
CMP coefficients
              Estimate     SE z-value   p-value
X:(Intercept)   0.9970 0.1050  9.4933 2.238e-21
X:x             1.0029 0.1047  9.5757 1.012e-21
S:(Intercept)  -0.9498 0.1044 -9.0936 9.584e-20
--
Transformed intercept-only parameters
   Estimate     SE
nu   0.3868 0.0404
--
Chi-squared test for equidispersion
X^2 = 133.5665, df = 1, p-value = 6.7968e-31
--
Elapsed: 0.02 sec   Sample size: 200   formula interface
LogLik: -1023.2186   AIC: 2052.4373   BIC: 2062.3322
Optimization Method: L-BFGS-B   Converged status: 0
Message: CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH
```

Without choosing an initial value, changing the optimization method to `Nelder-Mead` and increasing the maximum number of iterations also helps the optimizer find a solution.

```
> control = get.control(optim.method = "Nelder-Mead", optim.control = list(maxit = 1000))
> glm.cmp(y ~ x, formula.nu = ~ 1, control = control)
CMP coefficients
              Estimate     SE  z-value    p-value
X:(Intercept)   0.8460 0.0424  19.9591   1.25e-88
X:x             0.8518 0.0417  20.4230  1.043e-92
S:(Intercept)  -1.1133 0.0486 -22.8895 5.915e-116
--
Transformed intercept-only parameters
   Estimate    SE
nu   0.3285 0.016
```

```
--
Chi-squared test for equidispersion
X^2 = 5048.8383, df = 1, p-value = 0.0000e+00
--
Elapsed: 0.03 sec   Sample size: 200   formula interface
LogLik: -1024.1845   AIC: 2054.3691   BIC: 2064.2640
Optimization Method: Nelder-Mead   Converged status: 0
```

Note that this solution is different from the previous one; the log-likelihood of the previous one is slightly better.

4.2 ZICMP Regression

4.2.1 Couple Dataset

The `couple` dataset [Loeys et al., 2012] was analyzed with ZICMP regression in Sellers and Raim [2016] and found to exhibit overdispersion. The data concern separation trajectories of $n = 387$ couples. The variable `UPB` records the number of unwanted pursuit behavior perpetrations and is considered the outcome of interest. Included covariates are the binary variable `EDUCATION`, which is 1 if at least a bachelor's degree was attained, and a continuous variable `ANXIETY` which measures anxious attachment. A zero-inflated count model is considered for these data because 246 of the 387 records have an outcome of zero.

Let us load and view the first few records in the dataset.

```
> data(couple)
> head(couple)
  UPB EDUCATION ANXIETY
1  15         0  1.0053
2   0         0 -0.7034
3   0         1 -0.7034
4   3         1  0.6110
5  12         0  0.2167
6   0         1  2.0569
```

As a preliminary model, let us fit a standard Poisson model $Y_i \overset{\text{ind}}{\sim} \text{Poisson}(\lambda_i)$ with

$$\log \lambda_i = \beta_0 + \beta_1 \cdot \text{EDUCATION}_i + \beta_2 \cdot \text{ANXIETY}_i.$$

We may use the standard `glm` function.

```
> glm.out = glm(UPB ~ EDUCATION + ANXIETY, data = couple, family = poisson)
> summary(glm.out)

Call:
glm(formula = UPB ~ EDUCATION + ANXIETY, family = poisson, data = couple)

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.81695    0.04386  18.628   <2e-16 ***
EDUCATION   -0.21579    0.07047  -3.062   0.0022 **
ANXIETY      0.42169    0.03333  12.651   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 2478.3  on 386  degrees of freedom
```

```
Residual deviance: 2310.8  on 384  degrees of freedom
AIC: 2782.4

Number of Fisher Scoring iterations: 6
```

4.2.2 ZICMP Regression

Now consider a ZICMP regression with

$$\begin{aligned}\log \lambda_i &= \beta_0 + \beta_1 \cdot \text{EDUCATION}_i + \beta_2 \cdot \text{ANXIETY}_i,\\ \log \nu_i &= \gamma_0,\\ \operatorname{logit} p_i &= \zeta_0 + \zeta_1 \cdot \text{EDUCATION}_i + \zeta_2 \cdot \text{ANXIETY}_i.\end{aligned}$$

We use the `glm.cmp` function as follows.

```
> zicmp0.out = glm.cmp(UPB ~ EDUCATION + ANXIETY,
+   formula.nu = ~ 1,
+   formula.p = ~ EDUCATION + ANXIETY,
+   data = couple)
> print(zicmp0.out)
ZICMP coefficients
              Estimate      SE z-value   p-value
X:(Intercept)  -0.1604  0.0189 -8.4844 2.169e-17
X:EDUCATION    -0.0678  0.0325 -2.0849   0.03708
X:ANXIETY       0.0226  0.0142  1.5923    0.1113
S:(Intercept) -11.0963 59.4006 -0.1868    0.8518
W:(Intercept)   0.4176  0.1599  2.6119  0.009005
W:EDUCATION    -0.3863  0.2677 -1.4433    0.1489
W:ANXIETY      -0.5234  0.1335 -3.9200 8.856e-05
--
Transformed intercept-only parameters
   Estimate    SE
nu        0 9e-04
--
Chi-squared test for equidispersion
X^2 = 350.5662, df = 1, p-value = 3.1903e-78
--
Elapsed: 4.73 sec   Sample size: 387   formula interface
LogLik: -627.1675   AIC: 1268.3350   BIC: 1296.0440
Optimization Method: L-BFGS-B   Converged status: 0
Message: CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH
```

There are now three sets of coefficients reported in the output: the `X:`, `S:`, and `W:` prefixes label estimates for the λ_i, ν_i, and p_i formulas respectively.

4.2.3 Comments about Results

The AIC of the ZICMP model is drastically smaller than the Poisson model, indicating a greatly improved fit. However, there are signs of possible numerical issues. The estimate for γ_0 is a large negative number, but with an extremely large associated SE, which suggests that the effect may not be statistically significant. On the other hand, the estimate of ν is nearly zero with a small SE, which suggests that the dispersion parameter is indeed statistically significant. On the surface, this seems to be a contradiction.

The issue is that the Hessian of the log-likelihood becomes insensitive to small changes in γ_0 when $\lambda_i < 1$ and γ_0 is a large negative number. Let us first verify that the estimates for λ_i are indeed smaller than 1.

```
> pred.out = predict(zicmp0.out, type = "link")
> summary(pred.out$lambda)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.7719  0.8034  0.8264  0.8300  0.8560  0.9030
```

To show the insensitivity of the Hessian, let us consider a simpler setting with $Y \sim \text{CMP}(\lambda, \nu)$, $\lambda = \exp\{-0.25\} \approx 0.7788$ fixed, and $\nu = \exp\{\gamma_0\}$. We then have log-density

$$\log f(y \mid \gamma_0) = y \log \lambda - e^{\gamma_0} \log(y!) - \log Z(\lambda, e^{\gamma_0}),$$

with first derivative and second derivatives, respectively,

$$\frac{\partial}{\partial \gamma_0} \log f(y \mid \gamma_0) = -e^{\gamma_0} \log(y!) - \frac{\partial}{\partial \gamma_0} \log Z(\lambda, e^{\gamma_0}),$$
$$\frac{\partial^2}{\partial \gamma_0^2} \log f(y \mid \gamma_0) = -e^{\gamma_0} \log(y!) - \frac{\partial^2}{\partial \gamma_0^2} \log Z(\lambda, e^{\gamma_0}).$$

For a given value of y, $-e^{\gamma_0} \log(y!)$ approaches zero as γ_0 decreases. Therefore, let us focus on the function $g(\gamma_0) = -\log Z(\lambda, e^{\gamma_0})$ and its first and second derivatives. The following code illustrates their behavior.

```
library(numDeriv)
g = function(gamma0) {
    -ncmp(lambda = exp(-0.25), nu = exp(gamma0), log = TRUE)
}
dat = data.frame(gamma0 = seq(0, -13), g = NA, d_g = NA, d2_g = NA)
for (j in 1:nrow(dat)) {
    gamma0 = dat$gamma0[j]
    dat$g[j] = g(gamma0)
    dat$d_g[j] = numDeriv::grad(func = g, x = gamma0)
    dat$d2_g[j] = numDeriv::hessian(func = g, x = gamma0)
}
```

Here is the result.

```
> print(dat)
   gamma0          g          d_g          d2_g
1       0 -0.7788006 1.899077e-01 -6.055103e-02
2      -1 -0.9888736 2.197811e-01  1.939888e-03
3      -2 -1.1975605 1.883311e-01  5.754063e-02
4      -3 -1.3531805 1.206365e-01  7.009797e-02
5      -4 -1.4415733 6.002814e-02  4.795655e-02
6      -5 -1.4821920 2.534234e-02  2.308249e-02
7      -6 -1.4986634 9.858232e-03  9.528758e-03
8      -7 -1.5049621 3.704947e-03  3.658399e-03
9      -8 -1.5073136 1.373898e-03  1.364919e-03
10     -9 -1.5081834 5.069224e-04  5.060507e-04
11    -10 -1.5085040 1.866893e-04  1.865710e-04
12    -11 -1.5086220 6.870664e-05  6.869063e-05
13    -12 -1.5086654 2.527949e-05  2.527732e-05
14    -13 -1.5086814 9.300307e-06  9.300014e-06
```

Notice that $g(\gamma_0)$ approaches a limit as $\gamma_0 \to -\infty$, which coincides with the CMP distribution approaching a Geometric distribution. It may not be surprising that the first and second derivatives approach zero accordingly. This explains the large SE for γ_0 in the model `zicmp0.out`. With estimates tending to this region of the parameter space, it may be preferable to fix fix γ_0 at a value such as $-\infty$, which will be done in the next section.

4.2.4 Fixed Coefficients

Our attempt to fit the previous model strongly tended to the Zero-Inflated Geometric special case of ZICMP, but SEs computed via the Hessian become large in this region. In this section, we fix γ_0 at the extreme $-\infty$ and fit the remaining coefficients. Let us use the raw interface to do this.

```
init = coef(zicmp0.out, type = "list")
y = couple$UPB
X = model.matrix(~ EDUCATION + ANXIETY, data = couple)
S = model.matrix(~ 1, data = couple)
W = model.matrix(~ EDUCATION + ANXIETY, data = couple)
control = get.control(optim.method = "BFGS")
zicmp.out = glm.zicmp.raw(y, X, S, W,
    init = get.init(beta = c(-1,0,0), gamma = -Inf, zeta = c(-1,0,0)),
    fixed = get.fixed(gamma = 1L), control = control)
```

```
> print(zicmp.out)
ZICMP coefficients
              Estimate     SE z-value   p-value Fixed
X:(Intercept)  -0.1605 0.0188 -8.5432 1.305e-17     F
X:EDUCATION    -0.0678 0.0325 -2.0837   0.03718     F
X:ANXIETY       0.0226 0.0142  1.5896    0.1119     F
S:(Intercept)     -Inf     NA      NA        NA     T
W:(Intercept)   0.4172 0.1599  2.6090  0.009082     F
W:EDUCATION    -0.3863 0.2678 -1.4428    0.1491     F
W:ANXIETY      -0.5245 0.1336 -3.9261 8.632e-05     F
--
Transformed intercept-only parameters
   Estimate SE
nu        0  0
--
Some elements of gamma were fixed. Chi-squared test for equidispersion not defined.
--
Elapsed: 0.88 sec   Sample size: 387   raw interface
LogLik: -627.1672   AIC: 1266.3345   BIC: 1290.0850
Optimization Method: BFGS   Converged status: 0
```

Notice that an additional `Fixed` column has been added to the display, indicating that the coefficient `gamma` is fixed. Furthermore, its `Estimate` column is set to the initial value and the columns `SE`, `z-value`, and `p-value` are set to `NA`. This model achieves a similar log-likelihood value as our first attempt using `L-BFGS-B` but does not exhibit signs of numerical issues.

4.2.5 Accessor Functions

Here are several of the accessors provided to extract model outputs.

```
> logLik(zicmp.out)              ## Log-likelihood evaluated at MLE
[1] -627.1672
> AIC(zicmp.out)                 ## AIC evaluated at MLE
[1] 1266.334
> BIC(zicmp.out)                 ## BIC evaluated at MLE
[1] 1290.085
> coef(zicmp.out)                ## Estimates of theta as a flat vector
X:(Intercept)   X:EDUCATION     X:ANXIETY S:(Intercept) W:(Intercept)
  -0.16047147   -0.06776507    0.02257241          -Inf    0.41722279
  W:EDUCATION     W:ANXIETY
```

```
  -0.38633736   -0.52450590
> coef(zicmp.out, type = "list")  ## Estimates of theta as a named list
$beta
X:(Intercept)   X:EDUCATION     X:ANXIETY
  -0.16047147   -0.06776507    0.02257241

$gamma
S:(Intercept)
         -Inf

$zeta
W:(Intercept)   W:EDUCATION     W:ANXIETY
    0.4172228    -0.3863374    -0.5245059
> vcov(zicmp.out)                 ## Estimated covariance matrix of theta hat
              X:(Intercept)   X:EDUCATION     X:ANXIETY W:(Intercept)
X:(Intercept)  0.0003528180 -2.909813e-04 -1.246030e-04  0.0005767772
X:EDUCATION   -0.0002909813  1.057615e-03  2.583975e-05 -0.0005084108
X:ANXIETY     -0.0001246030  2.583975e-05  2.016361e-04 -0.0001489969
W:(Intercept)  0.0005767772 -5.084108e-04 -1.489969e-04  0.0255741783
W:EDUCATION   -0.0005266101  2.146314e-03  7.074582e-05 -0.0255267345
W:ANXIETY     -0.0001695705  1.830298e-04  3.848049e-04 -0.0006753149
                W:EDUCATION     W:ANXIETY
X:(Intercept) -5.266101e-04 -0.0001695705
X:EDUCATION    2.146314e-03  0.0001830298
X:ANXIETY      7.074582e-05  0.0003848049
W:(Intercept) -2.552673e-02 -0.0006753149
W:EDUCATION    7.169741e-02  0.0009802763
W:ANXIETY      9.802763e-04  0.0178471340
> sdev(zicmp.out)                 ## Standard deviations from vcov(...) diagonals
X:(Intercept)   X:EDUCATION     X:ANXIETY W:(Intercept)   W:EDUCATION
   0.01878345    0.03252099    0.01419986    0.15991929    0.26776372
    W:ANXIETY
   0.13359317
> sdev(zicmp.out, type = "list")  ## Standard deviations as a named list
$beta
[1] 0.01878345 0.03252099 0.01419986

$gamma
[1] NA

$zeta
[1] 0.1599193 0.2677637 0.1335932
> equitest(zicmp0.out)            ## Likelihood ratio test for H_0: gamma = 0
$teststat
[1] 350.5662

$pvalue
[1] 3.190326e-78

$df
[1] 1
> tryCatch({                      ## An error is thrown for model with fixed gamma
+   equitest(zicmp.out)
```

```
+ }, error = print_warning)
[1] Error in equitest.zicmpfit(zicmp.out): Some elements of gamma were
[2] fixed, chi-squared test for equidispersion not defined
```

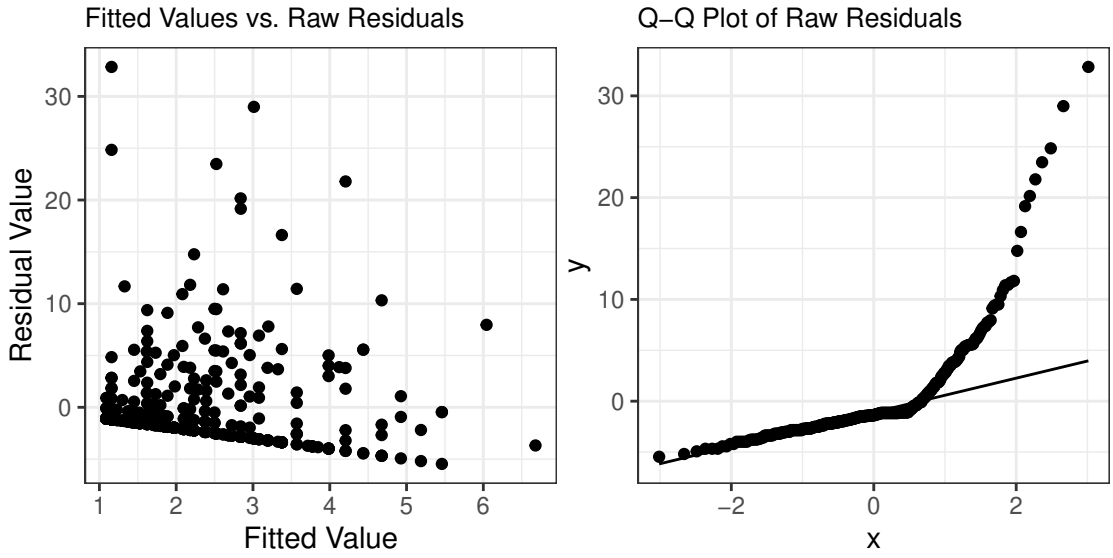
Because we fixed $\gamma = -\infty$ to obtain `zicmp0.out`, the `equitest` function throws an error instead of proceeding with an equidispersion test.

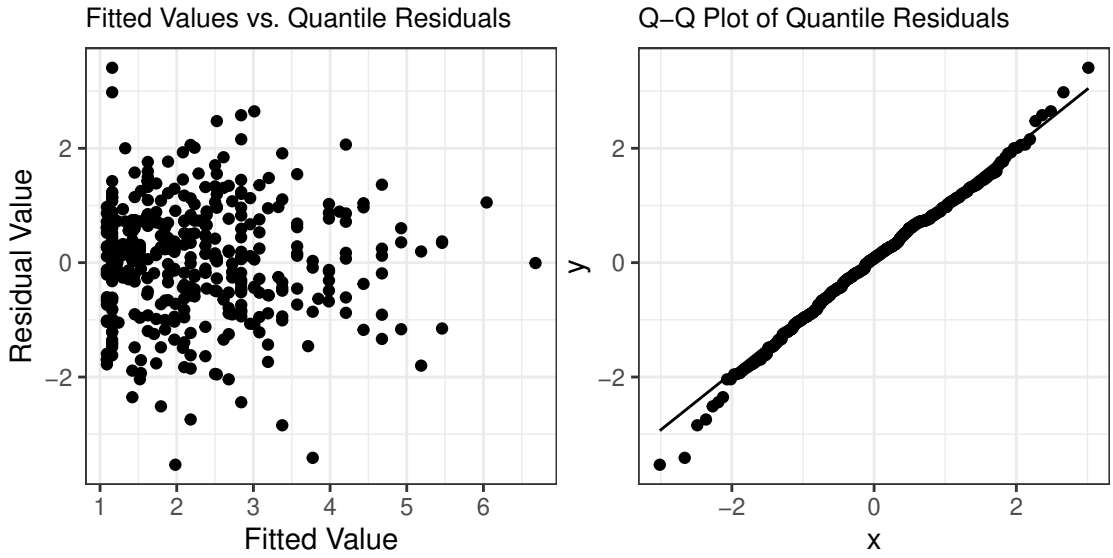
The `predict` function behaves similarly as in CMP regression; however, the `link` type here also includes a column with the estimated p_i.

```
> y.hat = predict(zicmp.out)  ## Fitted values based on ecmp
> link.hat = predict(zicmp.out, type = "link")
> head(y.hat)
[1] 3.570765 1.622937 1.451592 2.391102 2.522830 3.713332
> head(link.hat)
     lambda nu         p
1 0.8712909  0 0.4725120
2 0.8383254  0 0.6870063
3 0.7833983  0 0.5986451
4 0.8069894  0 0.4281048
5 0.8559186  0 0.5753131
6 0.8337620  0 0.2596150
```

In this example, we can see the benefit of using quantile residuals rather than raw residuals for diagnostic plots. The functions `plot.fit.res` and `plot.qq.res` have been defined in Section 4.1.

```
res.raw = residuals(zicmp.out, type = "raw")
res.qtl = residuals(zicmp.out, type = "quantile")

plot.fit.res(y.hat, res.raw) +
    ggtitle("Fitted Values vs. Raw Residuals")
plot.qq.res(res.raw) +
    ggtitle("Q-Q Plot of Raw Residuals")

plot.fit.res(y.hat, res.qtl) +
    ggtitle("Fitted Values vs. Quantile Residuals")
plot.qq.res(res.qtl) +
    ggtitle("Q-Q Plot of Quantile Residuals")
```





Here is an example of computing fitted values for new covariate data.

```
new.df = data.frame(EDUCATION = round(1:20 / 20), ANXIETY = seq(-3,3, length.out = 20))
X.new = model.matrix(~ EDUCATION + ANXIETY, data = new.df)
S.new = model.matrix(~ 1, data = new.df)
W.new = model.matrix(~ EDUCATION + ANXIETY, data = new.df)
new.data = get.modelmatrix(X.new, S.new, W.new)

# For model fit using raw interface, use get.modelmatrix to prepare new design
# matrices, offsets, etc
y.hat.new = predict(zicmp.out, newdata = new.data)

# For models fit with the formula interface, pass a data.frame with the same
# structure as used in the fit.
y.hat.new = predict(zicmp0.out, newdata = new.df)
```

```
> print(y.hat.new)
 [1] 0.4699009 0.5622034 0.6711344 0.7991174 0.9487440 1.1227246 1.3238519
 [8] 1.5549321 1.8187435 2.1180079 2.0359259 2.2797300 2.5382952 2.8108725
[15] 3.0968672 3.3959399 3.7081292 4.0339807 4.3745609 4.7316167
```

As with CMP regression, a `parametric.bootstrap` function is provided for convenience to obtain a bootstrap sample $\hat{\boldsymbol{\theta}}^{(r)}$ of $\boldsymbol{\theta}$ based on the estimate $\hat{\boldsymbol{\theta}}$. Because it is too time consuming to run this example within the vignette, we show the code without output below. As in Section 4.1, we consider using the bootstrap samples to construct a 95% confidence interval for each of the coefficients.

```
> zicmp.boot = parametric.bootstrap(zicmp.out, reps = 100)
> head(zicmp.boot)
> apply(zicmp.boot, 2, quantile, c(0.025,0.975))
```

Acknowledgements

We acknowledge Thomas Lotze for significant contributions to the initial development of the `COMPoissonReg` package and for service as initial maintainer on CRAN. We thank Darcy Steeg Morris, Eric Slud, and Tommy Wright for reviewing the manuscript. We are grateful to the users of `COMPoissonReg` for their interest and for bringing to light some of the issues which have been considered in this work.

References

Fraser Daly and Robert E. Gaunt. The Conway-Maxwell-Poisson distribution: Distributional theory and approximation. *ALEA Latin American Journal of Probability and Mathematical Statistics*, 13:635–658, 2016. URL https://doi.org/10.30757/ALEA.v13-25.

Peter K. Dunn and Gordon K. Smyth. Randomized quantile residuals. *Journal of Computational and Graphical Statistics*, 5(3):236–244, 1996. URL https://doi.org/10.1080/10618600.1996.10474708.

Dirk Eddelbuettel. *Seamless R and C++ Integration with Rcpp*. Springer, New York, 2013. URL https://doi.org/10.1007/978-1-4614-6868-4.

Robert E. Gaunt, Satish Iyengar, Adri B. Olde Daalhuis, and Burcin Simsek. An asymptotic expansion for the normalizing constant of the Conway–Maxwell–Poisson distribution. *Annals of the Institute of Statistical Mathematics*, 71(1):163–180, 2019. URL https://doi.org/10.1007/s10463-017-0629-6.

Paul Gilbert and Ravi Varadhan. *numDeriv: Accurate Numerical Derivatives*, 2019. URL https://CRAN.R-project.org/package=numDeriv. R package version 2016.8-1.1.

Steven B. Gillispie and Christopher G. Green. Approximating the Conway-Maxwell-Poisson distribution normalization constant. *Statistics*, 49(5):1062–1073, 2015. URL https://doi.org/10.1080/02331888.2014.896919.

Michael H. Kutner, Christopher J. Nachtsheim, and John Neter. *Applied Linear Regression Models*. McGraw-Hill, 4th edition, 2003.

Tom Loeys, Beatrijs Moerkerke, Olivia De Smet, and Ann Buysse. The analysis of zero-inflated count data: Beyond zero-inflated Poisson regression. *British Journal of Mathematical and Statistical Psychology*, 65(1):163–180, 2012. URL https://doi.org/10.1111/j.2044-8317.2011.02031.x.

Herbert Robbins. A remark on Stirling's formula. *The American Mathematical Monthly*, 62(1):26–29, 1955. URL http://www.jstor.org/stable/2308012.

Kimberly F. Sellers and Andrew Raim. A flexible zero-inflated model to address data dispersion. *Computational Statistics & Data Analysis*, 99:68–80, 2016. URL https://doi.org/10.1016/j.csda.2016.01.007.

Kimberly F. Sellers and Galit Shmueli. A flexible regression model for count data. *The Annals of Applied Statistics*, 4(2):943–961, 2010. URL https://doi.org/10.1214/09-AOAS306.

Galit Shmueli, Thomas P. Minka, Joseph B. Kadane, Sharad Borle, and Peter Boatwright. A useful distribution for fitting discrete data: Revival of the Conway-Maxwell-Poisson distribution. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 54(1):127–142, 2005. URL http://www.jstor.org/stable/3592603.

Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL https://ggplot2.tidyverse.org.