The implementation of the Conway-Maxwell (COM) Poisson model in **brms**

Wellington J. Silva & Luiz Max Carvalho
School of Applied Mathematics, Foundation Getulio Vargas, Brazil.

April 29, 2025

Contents

1 Introduction

The Conway-Maxwell Poisson distribution (COMP, Conway and Maxwell (1962)) is a popular model for count data, mainly due to its ability to accommodate under- as well as over-dispersed data - see Sellers et al. (2012) for a survey. For $\lambda > 0$ and $\nu > 0$, the COMP probability mass function (p.m.f.) can be written as

$$p_{\lambda,\nu}(n) := \Pr(X = n \mid \lambda, \nu) = \frac{\lambda^n}{Z(\lambda,\nu)(n!)^\nu},$$

where

$$Z(\lambda,\nu) := \sum_{n=0}^{\infty} \frac{\lambda^n}{(n!)^\nu} \tag{1}$$

is the normalizing constant. The sum in (1) is not usually known in closed-form for most values of (λ, ν) and thus needs to be computed approximately. In Carvalho et al. (2025) we explore distributions whose normalization constant is a series that passes the ratio test, for example the COMP, and how we can evaluate these series with guaranteed error. Here, we will focus on COMP, how this guaranteed-error method works, and its application within the **brms** library.

In the following section, we will show and prove the method. In Section 3, we will show some examples of how the method behaves in simple and more complex parameter cases. In Section 4, we will show how the method interacts in practice with the **brms** library. And finally, some suggestions in Section 5.

2 Proof of stopping criterion for guaranteed error

Our goal is to find a function $U_k(\lambda, \nu)$, such that

$$|Z(\lambda,\nu) - Z_k(\lambda,\nu)| \leq U_k(\lambda,\nu) \tag{2}$$

where $Z_k(\lambda,\nu) := \sum_{i=0}^{k} \frac{\lambda^i}{(i!)^\nu}$ (the partial sum of $Z(\lambda,\nu)$). To achieve this, we can use the following result (Braden (1992)):

Proposition 1 (Bounding a series that pass in ratio test). *Let $S_k := \sum_{i=0}^{k} a_i$. Under the assumptions that $(a_k)_{k\geq 0}$ is positive, decreasing and passes the ratio test, then for every $0 \leq k < \infty$ the following holds:*

$$S_k + a_k\left(\frac{L}{1-L}\right) < S < S_k + a_k\left(\frac{1}{1-\frac{a_k}{a_{k-1}}}\right), \tag{3}$$

if $\frac{a_{k+1}}{a_k}$ ***decreases*** *to* L *and*

$$S_k + a_k\left(\frac{1}{1-\frac{a_k}{a_{k-1}}}\right) < S < S_k + a_k\left(\frac{L}{1-L}\right), \tag{4}$$

if $\frac{a_{k+1}}{a_k}$ ***increases*** *to* L.

Proof. See Appendix A. □

To obtain the result of Equation 2 using Proposition 1, we will prove that the series defined in Equation 5 satisfies the ratio test and has a decreasing ratio of terms. First, for the ratio test, we have

$$\lim_{i\to\infty}\frac{\left|\frac{\lambda^{i+1}}{(i+1)!^{\nu}}\right|}{\left|\frac{\lambda^{i}}{(i)!^{\nu}}\right|} = \lim_{i\to\infty}\frac{\lambda^{i+1}}{(i+1)!^{\nu}}\frac{(i)!^{\nu}}{\lambda^{i}} = \lim_{i\to\infty}\frac{\lambda}{(i+1)^{\nu}} = 0.$$

Thus, it passes the ratio test with a limit of 0. Also, note that $\frac{\lambda}{(i+1)^{\nu}}$ is a decreasing function of i (considering $\lambda, \nu > 0$). Then, by Proposition 1, it follows that

$$S < S_k + a_k\left(\frac{1}{1-\frac{a_k}{a_{k-1}}}\right)$$

where $a_k = \frac{\lambda^k}{k!^{\nu}}$, and since the terms are positive, we have

$$|Z(\lambda,\nu) - Z_k(\lambda,\nu)| \leq a_k\left(1-\frac{a_k}{a_{k-1}}\right)^{-1}.$$

2.1 Algorithm

We can write the previous result as an algorithm as follows:

Checking $a_k \geq a_{k-1}$ is important because the series may start increasing, but after a point, it always decreases.

Algorithm 1 Adaptive truncation via ratio test

```
Initialize $a_0$, $a_1$ and $k = 0$
while $a_k \geq a_{k-1}$ or $a_k \left(1 - \frac{a_k}{a_{k-1}}\right)^{-1} \geq \varepsilon$ do
    Set $k = k + 1$
    Evaluate $a_k$
end while
Evaluate $S_k = \sum_{i=0}^{k} a_i$
Return $S_k$
```

3 Examples with new approach

Here, we do some tests with the p.m.f. of the COMP. The *COMP_brms_updated* is the new version for the normalization constant of COM-Poisson; The *COMP_brms* corresponds to the current implementation in the **brms** repository; The *COMPoissonReg* is the version from the library **brms**; The *COMP_true* evaluates the series with at least 1 million terms (much more than necessary in these cases).

parameters	COMP_brms_updated	COMP_brms	COMPoissonReg	COMP_true
$\mu = 0.50, \nu = 2.00$	1.2660	1.0634	1.2660	1.2660
$\mu = 1.00, \nu = 1.50$	2.4309	2.4309	2.4309	2.4309
$\mu = 1.10, \nu = 1.40$	2.7816	2.9266	2.7816	2.7816
$\mu = 2.00, \nu = 1.30$	8.1933	14.4379	8.2008	8.2008
$\mu = 3.00, \nu = 1.20$	25.0584	59.2945	25.0669	25.0669

Table 1: Normalising constant Z of COM-Poisson with an updated version.

And here, we have the errors in relation to the *COMP_true*

parameters	error_COMP_brms_updated	error_COMP_brms	error_COMPoissonReg
$\mu = 0.50, \nu = 2.00$	0.0000	0.2025	6.8290e-08
$\mu = 1.00, \nu = 1.50$	0.0000	0.0000	1.2823e-07
$\mu = 1.10, \nu = 1.40$	0.0000	0.1450	1.0944e-06
$\mu = 2.00, \nu = 1.30$	0.0074	6.2370	2.9324e-06
$\mu = 3.00, \nu = 1.20$	0.0080	34.2276	9.0006e-06

Table 2: Normalising constant Z of COM-Poisson with errors.

Notice that the last two rows, where the error is not zero, fall under the Gaunt approximation Gaunt et al. (2019), which raises some discussion on whether the speed gain justifies the error (see Section 5.1).

We also test for some more complicated cases, here the number of terms necessary to achieve the error is between $\approx$ 150 terms in the first line and $\approx$ 160000 terms in the last line.

For the current version of COM-Poisson, the repository version had a bug in the list index that was fixed so that the more demanding tests (with this one) would be fair. The change proposed in Section 2 proves to be robust even in complicated cases, (the error given is the machine-epsilon $\varepsilon = 2^{-52} \approx 2.220\text{e-}16$):

parameters	error_COMP_brms_updated	error_COMP_brms	error_COMPoissonReg
$\mu = 10, \nu = 0.1$	7.9905e-17	1.9569e-14	2.4317e-04
$\mu = 100, \nu = 0.01$	2.0508e-16	7.4670e-13	40.0716
$\mu = 1000, \nu = 0.001$	2.1836e-16	1.1505e-10	412.105
$\mu = 10000, \nu = 0.0001$	2.1972e-16	1.7554e-08	4136.83

Table 3: Normalising constant Z of COM-Poisson with errors.

4 MCMC

Here, we will show how the COMP in the **brms** works with a simple example[1]:

```
library(brms)

zinb <- read.csv("https://paul-buerkner.github.io/data/fish.csv")

fit <- brm(count ~ 1,
           cores=4,
           iter=1000,
           data = zinb,
           family = "com_poisson",
           backend = "cmdstanr",
           control = list(adapt_delta = 0.99))

summary(fit)
```

For the repository version of **brms**, we have a problem [2]:

```
Compiling Stan program...
Start sampling
Running MCMC with 4 parallel chains...

Warning: Chain 1 finished unexpectedly!

Warning: Chain 2 finished unexpectedly!

Warning: Chain 3 finished unexpectedly!

Warning: Chain 4 finished unexpectedly!

Warning: Use read_cmdstan_csv() to read the results of the failed chains.
Error: Fitting failed. Unable to retrieve the metadata.
```

[1]Example from https://github.com/paul-buerkner/brms/issues/607#issuecomment-944024534.
[2]Installing with `devtools::install_github("paul-buerkner/brms")`

```
In addition: Warning messages:
1: All chains finished unexpectedly! Use the $output(chain_id) method for
more information.

2: No chains finished successfully. Unable to retrieve the fit.
```

We can find in the code an index error; for test, we fix this and run again [3]

```
Compiling Stan program...
Start sampling
Running MCMC with 4 parallel chains...

...

All 4 chains finished successfully.
Mean chain execution time: 163.9 seconds.
Total execution time: 185.9 seconds.

>
> summary(fit)
 Family: com_poisson
  Links: mu = log; shape = identity
Formula: count ~ 1
   Data: zinb (Number of observations: 250)
  Draws: 4 chains, each with iter = 1000; warmup = 500; thin = 1;
         total post-warmup draws = 2000

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept  -104.19     87.15  -371.47   -28.22 1.03      172      186

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
shape     0.00      0.00     0.00     0.01 1.03      170      180

Draws were sampled using sample(hmc). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

But if we check with another library with `glm.cmp` (of **COMPoissonReg** library) we can see that the result is about -0.265 instead -104 (remembering that **COMPoissonReg** also does not have a guaranteed low error for the normalization constant). Now looking to our method with $\varepsilon \approx 10^{-16}$ (machine-epsilon for 64-bits)[4]

```
Compiling Stan program...
Start sampling
Running MCMC with 4 parallel chains...

...

All 4 chains finished successfully.
Mean chain execution time: 980.2 seconds.
```

[3]Installing with `devtools::install_github("wellington36/brms@original_fixed")`
[4]Can be installed with `devtools::install_github("wellington36/brms")`

```
Total execution time: 1160.9 seconds.

Warning: 17 of 2000 (1.0%) transitions ended with a divergence.
See https://mc-stan.org/misc/warnings for details.

>
> summary(fit)
 Family: com_poisson
  Links: mu = log; shape = identity
Formula: count ~ 1
   Data: zinb (Number of observations: 250)
  Draws: 4 chains, each with iter = 1000; warmup = 500; thin = 1;
         total post-warmup draws = 2000

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept    -0.27      0.02    -0.30    -0.23 1.01      445      522

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
shape     0.00      0.00     0.00     0.00 1.01      198      394

Draws were sampled using sample(hmc). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
Warning message:
There were 17 divergent transitions after warmup. Increasing adapt_delta
above 0.99 may help. See
http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

Since each evaluation of the normalization constant has an error smaller than the machine epsilon, we guarantee that there is no error in the calculation of the normalization constant. Some possible adjustments to improve the time are discussed below in the Section 5.

5 Suggestions

Suggestions to be introduced into the code.

5.1 Remove Gaunt's approximation

As seen from Table 3, Gaunt's approximation adds error to the normalization constant. Based on some empirical tests, I believe that the small gain in speed in the evaluation does not compensate for the error. Moreover, doing all of the approximate computations *via* the proposed algorithm simplifies the code and improves maintainability.

5.2 Make the parametrization clearer

We noticed two ways of parametrization the COM-Poisson (or COMP):

- For $\lambda > 0$ and $\nu > 0$, the COMP probability mass function (p.m.f.) can be written as

$$p_{\lambda,\nu}(n) := \Pr(X = n \mid \lambda, \nu) = \frac{\lambda^n}{Z(\lambda,\nu)(n!)^\nu},$$

  where

$$Z(\lambda,\nu) := \sum_{n=0}^{\infty} \frac{\lambda^n}{(n!)^\nu} \tag{5}$$

  is the normalising constant.

- And

$$\tilde{p}_{\mu,\nu}(n) = \frac{\mu^{\nu n}}{\tilde{Z}(\mu,\nu)(n!)^\nu},$$

  where

$$\tilde{Z}(\mu,\nu) := \sum_{n=0}^{\infty} \left(\frac{\mu^n}{n!}\right)^\nu. \tag{6}$$

In the **brms** we use the first one, but the name of the parameter is `mu`. I believe this creates confusion, perhaps it would be interesting to change the name or the parametrization.

5.3 Make the code fast

In this report we present a method to evaluate the normalising constant of the COMP with guaranteed error, but we not discuss if the code is fast or not. In fact, the version presented in the repository can be improved in terms of speed. Instead of checking the stopping criterion in each term, we can do it every K terms, which has proven to be more efficient in tests. Another approach is to increase the value of epsilon, which on the one hand can reduce the time by half, but on the other hand, in very difficult problems, it can generate errors in the final result.

References

Braden, B. (1992). Calculating sums of infinite series. *The American mathematical monthly*, 99(7):649–655.

Carvalho, L. M., Silva, W. J., and Moreira, G. A. (2025). Adaptive truncation of infinite sums: applications to statistics.

Conway, R. W. and Maxwell, W. L. (1962). A queuing model with state dependent service rates. *Journal of Industrial Engineering*, 12(2):132–136.

Gaunt, R. E., Iyengar, S., Daalhuis, A. B. O., and Simsek, B. (2019). An asymptotic expansion for the normalizing constant of the Conway–Maxwell–Poisson distribution. *Annals of the Institute of Statistical Mathematics*, 71(1):163–180.

Sellers, K. F., Borle, S., and Shmueli, G. (2012). The COM-Poisson model for count data: a survey of methods and applications. *Applied Stochastic Models in Business and Industry*, 28(2):104–116.

A Proofs

Proof of Proposition 1:

Proof. First define the series $r_k = \frac{a_{k+1}}{a_k}$. Now define the remainder $R_k = S - S_k = \sum_{i=k+1}^{\infty} a_i$. Now assume that r_k decreases to L. Then

$$\begin{aligned}
R_k &= a_{k-1}\left(\frac{a_{k+1}}{a_{k-1}} + \frac{a_{k+2}}{a_{k-1}} + \frac{a_{k+3}}{a_{k-1}} + \dots\right) \\
&= a_{k-1}\left(\frac{a_k}{a_{k-1}}\frac{a_{k+1}}{a_k} + \frac{a_k}{a_{k-1}}\frac{a_{k+1}}{a_k}\frac{a_{k+2}}{a_{k+1}} + \frac{a_k}{a_{k-1}}\frac{a_{k+1}}{a_k}\frac{a_{k+2}}{a_{k+1}}\frac{a_{k+3}}{a_{k+2}} + \dots\right) \\
&= a_{k-1}\left(r_{k-1}r_k + r_{k-1}r_k r_{k+1} + r_{k-1}r_k r_{k+1} r_{k+2} + \dots\right) \\
&< a_{k-1}\left(r_{k-1}r_{k-1} + r_{k-1}r_{k-1}r_{k-1} + r_{k-1}r_{k-1}r_{k-1}r_{k-1} + \dots\right) \\
&= a_{k-1}r_{k-1}^2\left(1 + r_{k-1} + r_{k-1}^2 + \dots\right) \\
&= a_k r_{k-1}\sum_{i=0}^{\infty} r_{k-1}^i = a_k \frac{r_{k-1}}{1 - r_{k-1}} \\
&= a_k \frac{\frac{a_k}{a_{k-1}}}{1 - \frac{a_k}{a_{k-1}}} = a_k \frac{\frac{a_k}{a_{k-1}}}{\frac{a_{k-1}-a_k}{a_{k-1}}} \\
&= a_k \frac{a_k}{a_{k-1} - a_k} = a_k\left(\frac{1}{1 - \frac{a_k}{a_{k-1}}}\right).
\end{aligned} \tag{7}$$

On the other hand, since $r_k > L$ for all n,

$$\begin{aligned}
R_k &= a_k\left(\frac{a_{k+1}}{a_k} + \frac{a_{k+2}}{a_k} + \frac{a_{k+3}}{a_k} + \dots\right) \\
&= a_k\left(\frac{a_{k+1}}{a_k} + \frac{a_{k+1}}{a_k}\frac{a_{k+2}}{a_{k+1}} + \frac{a_{k+1}}{a_k}\frac{a_{k+2}}{a_{k+1}}\frac{a_{k+3}}{a_{k+2}} + \dots\right) \\
&= a_k r_k\left(1 + r_{k+1} + r_{k+1}r_{k+2} + k_{n+1}r_{k+2}r_{k+3} + \dots\right) \\
&< a_k L\left(1 + L + L^2 + L^3 + \dots\right) \\
&= a_k L \sum_{i=0}^{\infty} L^i = a_k \frac{L}{1 - L}.
\end{aligned} \tag{8}$$

For the case in which r_k increases to L, the proof is analogous, with inequality signs reversed. □

B Additional results

The method with a low epsilon $\approx 10^{-7}$ (here, machine-epsilon to 32bits)[5]

```
Model executable is up to date!
Start sampling
Running MCMC with 4 parallel chains...

...

All 4 chains finished successfully.
Mean chain execution time: 306.9 seconds.
Total execution time: 320.4 seconds.

Warning: 38 of 2000 (2.0%) transitions ended with a divergence.
See https://mc-stan.org/misc/warnings for details.

>
> summary(fit)
 Family: com_poisson
  Links: mu = log; shape = identity
Formula: count ~ 1
   Data: zinb (Number of observations: 250)
  Draws: 4 chains, each with iter = 1000; warmup = 500; thin = 1;
         total post-warmup draws = 2000

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept    -0.27      0.02    -0.30    -0.24 1.01      413      458

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
shape     0.00      0.00     0.00     0.00 1.02      211      258

Draws were sampled using sample(hmc). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
Warning message:
There were 38 divergent transitions after warmup. Increasing adapt_delta
above 0.99 may help. See
http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

[5] Install with `devtools::install_github("wellington36/brms@bounding_low_epsilon")`