THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Efficient Computation of Mean-Parametrized Conway–Maxwell–Poisson Models via Bilinear Interpolation & Extrapolation

Wilson Lorensyah - 44838711 - MATH6020

BMath(Statistics)/BCom(Finance)

Wilson Lorensyah's ORCID

A Thesis Submitted For The Degree of

Bachelor of Mathematics (Honours) in Statistics at

The University of Queensland in Semester 1, 2022

School of Mathematics and Physics

Contents

Contents **ii**

Abstract iii

1 Introduction **1**

1.1 Background and Introduction 1

1.2 Count Regression Model 2

2 Investigating Bilinear Interpolation/Extrapolation Behaviour **5**

2.1 Behaviour of $\log(\lambda)$ and $\log(Z)$ for Large μ and/or ν 5

2.2 Bilinear Interpolation 13

2.3 Pre-computation of $\log(\lambda)$ and $\log(Z)$ on a Grid 15

2.4 Bilinear Extrapolation 15

3 Speed and Accuracy Bilinear Interpolation/Extrapolation **17**

3.1 Comparison of Numerical Accuracy and Computational Speed 17

4 Bilinear Interpolation/Extrapolation Applications **23**

4.1 Bilinear Interpolation & Extrapolation on Bayesian Setting 23

4.2 Underdispersed Takeover Bids 24

4.3 Underdispersed Cotton Bolls 28

4.4 Overdispersed Class Attendance 30

4.5 Overdispersed Fish Dataset 32

5 Conclusion **35**

5.1 Conclusion & Further Research 35

5.2 Acknowledgement 35

A Appendix **37**

A.1 Code and Tables 37

Bibliography **41**

Abstract

The mean-parametrized Conway–Maxwell–Poisson (CMP) model (Huang, 2017) has seen a recent surge in popularity for the analysis of dispersed counts. However, the main hindrance in its wider use in practice in both frequentist and Bayesian settings is the computational bottleneck when evaluating the intractable normalizing constant and canonical parameter of the distribution, which in turn affects the computation speed for calculating the variance and other higher order moments. This project examines the accuracy and efficiency of pre-computing these intractable values at a grid of parameter values and using bi-linear interpolation (and extrapolation) to approximate points in between (and beyond) the grid at a cost of potentially trading accuracy with some gain in computational speed. Mathematical results giving bounds on the functional behaviour of these intractable quantities provide a theoretical justification for such an approach. The proposed approximation is compared to direct computation via an extensive numerical study in a Bayesian framework, finding that in underdispersed cases it can be over one order of magnitude faster.

Chapter 1

Introduction

1.1 Background and Introduction

The classical Poisson distribution has been widely used for modelling count data. However, this also has the property of that mean and variance are the same (known as equidispersion) which is just the rate parameter itself (mostly denoted $\lambda \geq 0$). While this is a simple mathematical property, it is restrictive from an applied modelling point of view. To recap, the Poisson distribution $X \sim \text{Pois}(\lambda)$ is characterised by the properties:

$$\text{P}(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}, \quad \mathbb{E}[X] = \lambda, \quad \text{Var}[X] = \lambda.$$

Real world data often does not satisfy the equidispersion property. An extension of Poisson model reintroduced by Shmueli et al. (2005) with two input parameters, known as Conway-Maxwell-Poisson distribution or CMP (named after Conway & Maxwell (1962)), is characterised by a rate parameter $\lambda \geq 0$, that is similar to that of the Poisson distribution but with an additional dispersion parameter $\nu \geq 0$). The CMP distribution with parameters λ and ν has the properties:

$$\begin{aligned}
\text{P}(X = x|\lambda, \nu) &= \frac{\lambda^x}{(x!)^\nu} \frac{1}{Z(\lambda, \nu)}, \quad \text{where } Z(\lambda, \nu) = \sum_{x=0}^{\infty} \frac{\lambda^x}{(x!)^\nu}, \\
\mathbb{E}[X] &= \sum_{x=0}^{\infty} \frac{x\lambda^x}{(x!)^\nu Z(\lambda, \nu)} \\
\text{Var}[X] &= \sum_{x=0}^{\infty} \frac{x^2\lambda^x}{(x!)^\nu Z(\lambda, \nu)} - [\mathbb{E}[X]]^2.
\end{aligned}$$

More recently, Huang (2017) came up with the CMP$_\mu$ model, which is similar in spirit to the original CMP model but using the mean $\mu \geq 0$ instead of rate parameter λ as an input. This mean reparametrisation of CMP was established on the basis that if a distribution belongs to a canonical exponential family, then such distributions can always be parameterised by its mean. In addition, the main hindrance to using CMP model in practice was the inability to directly model the mean of counts, making it incompatible with other competing count regression models, including log-linear Poisson, generalised Poisson, and Negative-Binomial. Therefore, CMP$_\mu$ is expected to boost interpretability aspects compared to CMP and be a competitive candidate for count regression modelling. The CMP$_\mu$ distribution corresponding to random variable $X \sim \text{CMP}_\mu(\mu, \nu)$ is characterised by the following properties:

$$\text{P}(X = x|\mu,\nu) = \frac{\lambda(\mu,\nu)^x}{(x!)^\nu}\frac{1}{Z(\lambda(\mu,\nu),\nu)}, \quad \text{where } Z(\lambda(\mu,\nu),\nu) = \sum_{x=0}^{\infty}\frac{\lambda(\mu,\nu)^x}{(x!)^\nu}, \qquad (\star)$$

$$\mathbb{E}[X] = \mu$$

$$\text{Var}[X] = \sum_{x=0}^{\infty}\frac{(x-\mu)^2\lambda(\mu,\nu)^x}{(x!)^\nu Z(\lambda(\mu,\nu),\nu)},$$

where $\lambda(\mu,\nu)$ can be found by the solution of:

$$0 = \sum_{x=0}^{\infty}(x-\mu)\frac{\lambda^x}{(x!)^\nu}. \qquad (\dagger)$$

Note that for both CMP and CMP$_\mu$ model, the variance Var$[X]$ could be smaller or larger than the expectation $\mathbb{E}[X]$. Also, Poisson is the special case of CMP and CMP$_\mu$; one can see this when the dispersion parameter is equal to 1 ($\nu = 1$) as the reciprocal of the normalizing constant $Z(\lambda,\nu)$ in CMP model or $Z(\lambda(\mu,\nu),\nu)$ in CMP$_\mu$ will equal to $e^{-\lambda}$. If $\nu < 1$, the model is conditionally overdispersed. Conversely, if $\nu > 1$, the model is conditionally underdispersed. Moreover, both CMP or CMP$_\mu$ pass through other discrete distributions as special cases; as $\nu \to \infty$ they become an underdispersed Bernoulli distribution with $\text{P}(X = 1) = \frac{\lambda}{1+\lambda}$ and for $\nu = 0$, they coincide with an overdispersed geometric distribution with $\text{P}(X = x) = \lambda^x(1-\lambda)$ for $x \geq 0$.

There are also several count regression models compared recently developed and discussed with more detail in Huang (2020) but they have some limitations on the parameter space. For example, one critique on generalised Poisson (GP) model is that its parameters are functionally dependent, such that for some combinations of the parameters, it is not a proper probability mass function (pmf) because its sum does not converge to 1. Sellers & Shmueli (2010) also argue that the underdispersion achieved by GP was through unnatural truncation of support of the distribution where the support and parameter space are also functionally dependent which then restricts parameter ranges. The Poisson-Tweedie model is another popular model, whose maximum likelihood estimate (MLE) does not exist when underdispersed, resorting to quasi-likelihood instead, which technically cannot handle underdispersion. Thus, the CMP$_\mu$ model is one of the most flexible models that can handle both underdispersion and overdispersion while preserving the data generating capabilities of being a pmf without any parameter restrictions.

1.2 Count Regression Model

The default choice of fitting generalised linear model (GLM) for count response is to use Poisson regression. Consider the famous `warpbreaks` dataset (Tukey, 1977) with 54 data points that count the number of warp breaks per loom, where a loom corresponds to a fixed length of yarn. Two types of wools and 3 level of tensions (Low, Medium, High) are considered in the study. The mean-model chosen by default is the log link for count data ($\mu = \exp(X^T\beta)$). Upon fitting the log link Poisson model, using the number of breaks as response variable (Y), and wool types (X_1) plus tension levels (X_2) and its interaction (X_1X_2) as explanatory variables, the code is shown in Appendix A.1. In order to check goodness-of-fit, one can use an overdispersion test, which is done by testing the deviance of the residuals, i.e. by looking at the χ^2 with null-hypothesis H_0 that the conditional variance $\mathrm{Var}(Y|X)$ equal to conditional expectation $\mathbb{E}(Y|X) = \mu$. Using the the deviance of the residual, the p-value is 0, it rejects the H_0, and shows that there is strong evidence of conditional overdispersion.

An alternative (and easier way) is to explicitly estimate the dispersion using a quasiPoisson model that is characterised by the formula below:

$$\mathrm{Var}(Y|X) = \phi\,\mathbb{E}(Y|X) \iff \phi = \frac{\mathrm{Var}(Y|X)}{\mathbb{E}(Y|X)},$$

where ϕ is dispersion parameter that is constant for all data points. Re-arranging this, ϕ could be represented as a ratio of conditional variance divided by the conditional mean as above, which looks at the deviation around the fitted line. Thus, after replacing the family Poisson with quasiPoisson on GLM, whether a model is overdispersed $\phi > 1$ or underdispersed $\phi < 1$ can be observed through the output of dispersion parameter upon fitting GLM. Overdispersion can be detected by calculating the conditional mean given 3 levels of tensions 36.39, 26.39, 21.67 for Low, Medium, High groups respectively, and comparing to the conditional variances which are much larger at 270.49, 83.19, 69.76 for Low, Medium, High respectively. Upon fitting log link quasiPoisson model using the number of breaks as response, and simple additive model of wool types plus tension levels and its interactions, estimated dispersion parameter from `glm` is 3.76, which confirms overdispersion in the data. An appropriate model that can handle overdispersion well is to use Negative Binomial due to its flexibility without any parameter restrictions, i.e. the probability of success $p \in [0,1]$ and number of of successes $r > 0$ are functionally independent. CMP_μ has also functionally independent parameters μ and ν, which could be use for count regression models with overdispersion.

An example of conditionally underdispersed dataset would be the CD4 count dataset for HIV-Positive patients (Arel-Bundock, 2019). CD4 cells carried in the blood (part of the human immune system) would die as a side effect of HIV virus. The count of CD4 cells is used in determining the onset of full-blown AIDS in a patient. The experiment was to measure the effectiveness of a new anti-viral drug on 20 HIV-positive patients, having their CD4 counts recorded while they were put on a continuous treatment with this drug for a year. After using the drug for one year, their CD4 counts were again recorded using the same measurement (in 100's). From this dataset, the analysis can be done using GLM to fit a log link quasiPoisson model using CD4 counts estimated in one year as the response variable and the baseline CD4 counts as the explanatory variable. The dispersion parameter ϕ from the `glm` output is 0.1730 (much less than 1) which indicates conditionally underdispersed. Even though the log link is the default, one could argue the usage of identity link ($\mu = X^T\beta$) in this case due to non-negative linear mapping of baseline counts to one year counts. From the summary output, this also gave an estimated dispersion of 0.1673, also indicating conditionally underdispersed counts. Arguably, CMP$_\mu$ adds value as a popular full probability model for underdispersed count data (Huang, 2020). In terms of application, without a full probability distribution, one cannot do bayesian analysis because there is no proper likelihood, and can only make mean prediction from posterior of μ and ν. One main benefit for having full probability model is the capability of making future value prediction of (actual) integer count response (also can obtain the 95% prediction interval) from posterior distributions of explanatory variables μ and dispersion parameter ν of Markov Chain Monte Carlo (MCMC) samples.

Using the CMP$_\mu$ distribution, in a Bayesian framework, the dispersion test can be done through fitting the dataset with GLM (using `glm.cmp` from `mpcmp` package giving the output for the dispersion parameter) after having prior belief about distribution of ν, then looking at the posterior distribution of ν for the updated belief (detailed in Chapter 4). It will have posterior mean and 95% credible interval which shows whether the posterior distribution of ν contains one (equidispersed), $\nu < 1$ (overdispersed) or $\nu > 1$ (underdispersed). Then one can make a probability statement: posterior probability that $\nu > 1$ is (or estimated to be) the area under the curve of the posterior distribution of ν from 1 onward. In a Frequentist framework, one can fit dataset with GLM using Poisson model and CMP$_\mu$ model. Then using the likelihood from both models, dispersion test can be carried through using likelihood-ratio test because Poisson is a special case of CMP$_\mu$ when $\nu = 1$.

Based on direct calculation of the CMP$_\mu$ model (Huang, 2017), implemented in the R package `mpcmp` (Fung et al., 2020), high accuracy in getting numerical result is guaranteed for ”small” parameter values of μ and/or ν. However, in the case of ”large” parameter values of μ and/or ν, this tends to be computationally expensive, slower and even out of reach of the package. This is true for both evaluating Z, which deals with infinite sum, given by the quantifiable specification of the `summax` argument, and this is **even harder for evaluating λ since the solution of (†) evaluates the root of an infinite sum**. Section 3.1 will examine computational performance, accuracy and time for computing λ and Z. An alternative solution is to provide a different method that can achieve approximate results with higher computational efficiency. To do this, it is important to check its pattern and behaviour of parameter of interest $\log(\lambda)$ and $\log(Z)$[1] for large μ and/or ν, which is detailed in Section 2.1. The implementation of Bilinear Interpolation is described in Section 2.2, and the conditions for chosen μ and ν outside the grid can be handled using Bilinear Extrapolation, as detailed in Section 2.4. The grid evaluation of matrix consisting of different combinations of μ and ν is detailed in Section 2.3. One then uses the index of the grid to find the location of chosen μ and ν in order to make a pre-computation for $\log(\lambda)$ and $\log(Z)$. Subsequently, the output and its application will be integrated into bayesian framework, such as Metropolis Hastings Markov Chain Monte Carlo (MH-MCMC) which is discussed in Section 4.1.

[1] $\log(\lambda)$ and $\log(Z)$ can be exponentiated to get Z and λ respectively. Using the log scale is more computationally tractable.

Chapter 2

Bilinear Interpolation & Extrapolation Probe

2.1 Behaviour of $\log(\lambda)$ and $\log(Z)$ for Large μ and/or ν

Firstly, one can start by plotting the behaviour of $\log(\lambda)$ vs μ for different ν values, and vice versa. One can also consider the relationship between μ and ν with $\log(Z)$. Figure 2.1 depicts the behaviour of these relationships where the parameter μ and ν is defined as:

$$\mu = 2^x, \quad \text{with } x = \{0, 0.05, 0.1, ..., 8\}, \qquad \nu = \{1, 4, 7, 10, ..., 112\},$$

where the colours from Red to Green correspond to from small values to large.

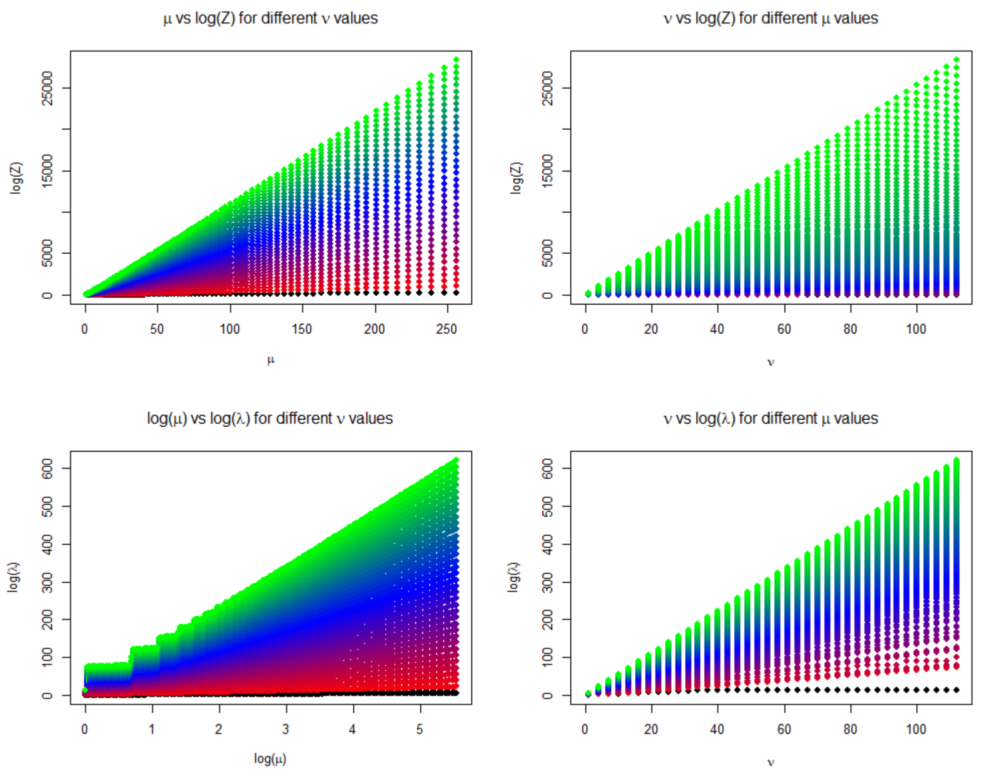

Figure 2.1: Different Combinations of μ $(\log(\mu))$, ν and its behaviour against log(Z) and $\log(\lambda)$

After observing the plots, $\log(\lambda)$ seems to be linear in ν for different values of μ, when μ and/or ν are relatively large. However, the values of $\log(\lambda)$ are not increasing linearly when ν values are large and μ values are small, as seen clearly in green points in the bottom left plot in Figure 2.1 (approximately when ν are between 80 and 100). However, for big enough values of ν, $\log(\lambda)$ is linear in $\log(\mu)$ as $\log(\mu)$ increases.

Conversely, for different values of μ, $\log(Z)$ seems to be linear in ν, and for different values of ν, $\log(Z)$ seems to be linear in μ. From here, the idea of interpolation / extrapolation using the grid will be more valuable when both $\log(\mu)$ and ν are large as the function of interest is relatively bi-linear in the two parameters. Notably, the behaviour of $\log(\lambda)$ and $\log(Z)$ are also still relatively linear in ν within the range $0 < \nu \leq 1$ for different μ values.

Theoretical justification or motivation to do bilinear interpolation/extrapolation are based on Lemma 1 and Lemma 2 below. When ν is large, $\log(\lambda)$ has the asymptotic behaviour, which was first stated in (Huang, 2020), showing bounds on the solution to the mean constraint (†). To derive this, first note that the behaviour of $f(x) = \frac{\mu^x}{x!}$ for any $\mu \geq 0$ can be established by looking at integer $\mu \geq 1$ (see Result 1) and non-integer $\mu > 0$ (see Result 2) cases, respectively:

Result 1. *For integer $\mu \geq 1$, the function $f(x) = \frac{\mu^x}{x!}$ is strictly increasing when $0 \leq x \leq \mu - 1$, strictly decreasing when $x \geq \mu$, and strictly less than 1 when $x \geq 2\mu^2$.*

Proof. First, note that the behaviour of $f(x) = \frac{\mu^x}{x!}$ and its bound can be found by looking at its derivative because it is a smooth function in x. In terms of calculations, the result is more easily derived on the log-scale first:

$$\begin{aligned} \log\left(\frac{\mu^x}{x!}\right) &= \log(\mu^x) - \log(x!) = \log(\mu^x) - \log(\Gamma(x+1)) = x\log(\mu) - \log(\Gamma(x+1)) \\ \therefore \frac{\partial \log(\frac{\mu^x}{x!})}{\partial x} &= \log(\mu) - \frac{\Gamma'(x+1)}{\Gamma(x+1)} = \log(\mu) - \psi(x+1), \end{aligned}$$

where $\Gamma(n) = (n-1)!$ and $\Gamma(n+1) = n\Gamma(n)$ and $\psi(x) = \frac{\Gamma'(x)}{\Gamma(x)}$ is the digamma function. By Sterling's inequality and Equation 3 of Diamond & Straub (2016), it is known that $\forall x \geq 0$:

$$\begin{aligned} \log(x+1/2) \leq \psi(x+1) &\leq \log(x+1) - \frac{1}{2(x+1)} \\ \implies -\log(x+1/2) \geq -\psi(x+1) &\geq \frac{1}{2(x+1)} - \log(x+1) \\ \implies \log(\mu) - \log(x+1/2) \geq \log(\mu) - \psi(x+1) &\geq \frac{1}{2(x+1)} - \log(x+1) + \log(\mu). \end{aligned}$$

Thus, using this inequality, it is deduced that the derivative is positive in the region $0 \leq x \leq \mu - 1$ and negative for $x \geq \mu$ for integer $\mu \geq 1$. To establish the conditions $\mu < 1$ with the bound of $x \geq 2\mu^2$, by setting $x = 2\mu^2$, this produces:

$$\begin{aligned}
(2\mu^2)! &> \frac{(2\mu^2)!}{(\mu-1)!} \\
\implies x! &> \frac{(2\mu^2)!}{(\mu-1)!} \\
\implies \log(x!) &> \log\Big(\frac{(2\mu^2)!}{(\mu-1)!}\Big) \\
&> \underbrace{\log(\mu)+\log(\mu+1)+...+\log(\mu^2-1)}_{\mu^2-\mu \ \text{ terms}} \quad \underbrace{+\log(\mu^2)+\log(\mu^2+1)+...+\log(2\mu^2)}_{\mu^2+1 \ \text{ terms}} \\
&> (\mu^2-\mu)\log(\mu)+(\mu^2+1)\log(\mu^2) \\
&= (3\mu^2-\mu+2)\log(\mu),
\end{aligned}$$

for $\mu \geq 1$. Note that from the fourth line to the fifth line, each subsequent term from $\log(\mu+1)$ until $\log(\mu^2-1)$ will always be greater than $\log(\mu)$. Counting the number of terms in between $\log(1)$ to $\log(\mu^2-1)$, there are μ^2-1 terms. Then, summing from $\log(1)$ to $\log(\mu-1)$, there are $\mu-1$ terms. Therefore, the number of terms from $\log(\mu)$ to $\log(\mu^2-1)$ would be the difference, i.e. $(\mu^2-1)-(\mu-1) = (\mu^2-\mu)$ terms. Similarly, each subsequent term from $\log(\mu^2+1)$ until $\log(2\mu^2)$ will always be greater than $\log(\mu^2)$. Also, the number of the terms would be the difference between summing the $\log(1)$ to $\log(2\mu^2)$ minus summing the $\log(1)$ to $\log(\mu^2-1)$, i.e. $2\mu^2-(\mu^2-1) = (\mu^2+1)$ terms. Using $x = 2\mu^2$ and modifying the terms, it would look like:

$$\begin{aligned}
-\log(x!) &< -(3\mu^2-\mu+2)\log(\mu) \\
\implies x\log(\mu)-\log(x!) &< 2\mu^2\log(\mu)-(3\mu^2-\mu+2)\log(\mu) \\
&= -(\mu^2-\mu+2)\log(\mu) \\
&\leq 0.
\end{aligned}$$

Since the function $x\log(\mu)-\log(x!) \leq 0$ for any integer $\mu \geq 1$ and with $x = 2\mu^2$, re-arranging leads to $\frac{\mu^x}{x!} < 1, \forall x \geq 2\mu^2$ using the monotonicity property. This is because the function $\frac{\mu^x}{x!}$ is strictly decreasing from $x=\mu$ to $x=\infty$, so that it is also strictly decreasing in the range $x \geq 2\mu^2$. This completes the proof of Result 1. $\square$

Result 2. *For non-integer $\mu > 0$, The function $f(x)$ is strictly increasing when $0 \leq x \leq \lfloor \mu \rfloor$, strictly decreasing when $x \geq \lceil \mu \rceil$, and strictly less than 1 when $x \geq 2\lceil \mu \rceil^2$.*

Proof. Similar to Result 1 proof, for non-integer μ, one exploits the ceiling and floor functions and apply the same idea. Looking at the behaviour of $\frac{\mu^x}{x!}$ from its derivative, $\frac{\partial \log(\frac{\mu^x}{x!})}{\partial x} = \log(\lceil \mu \rceil) - \psi(x+1)$, it is seen that the value of the derivative $\frac{\partial \log(\frac{\mu^x}{x!})}{\partial x}$ can also be bounded, similar to the proof of Result 1, but replacing μ with $\lceil \mu \rceil$:

$$\log(\lceil \mu \rceil) - \log(x+1/2) \geq \log(\lceil \mu \rceil) - \psi(x+1) \geq \frac{1}{2(x+1)} - \log(x+1) + \log(\lceil \mu \rceil).$$

Using the inequality above, the derivative is positive in the region $0 \leq x \leq \lfloor \mu \rfloor$ and negative for $x \geq \lceil \mu \rceil$ since $\mu > 0$ is non-integer. To establish the conditions strictly less than 1 in the bound $x \geq 2\lceil \mu \rceil^2$, by setting $x = 2\lceil \mu \rceil^2$, it produces:

$$\begin{aligned}
(2\lceil \mu \rceil^2)! &> \frac{(2\lceil \mu \rceil^2)!}{(\mu - 1)!} \\
\implies x! &> \frac{(2\lceil \mu \rceil^2)!}{(\mu - 1)!} \\
\implies \log(x!) &> \log\left(\frac{(2\lceil \mu \rceil^2)!}{(\lceil \mu \rceil - 1)!}\right) \\
&> \underbrace{\log(\lceil \mu \rceil) + \log(\lceil \mu \rceil + 1) + \ldots + \log(\lceil \mu \rceil^2 - 1)}_{\lceil \mu \rceil^2 - \lceil \mu \rceil \ \text{ terms}} + \\
&\quad \underbrace{\log(\lceil \mu \rceil^2) + \log(\lceil \mu \rceil^2 + 1) + \ldots + \log(2\lceil \mu \rceil^2)}_{\lceil \mu \rceil^2 + 1 \ \text{ terms}} \\
&> (\lceil \mu \rceil^2 - \lceil \mu \rceil)\log(\lceil \mu \rceil) + (\lceil \mu \rceil^2 + 1)\log(\lceil \mu \rceil^2) \\
&= (3\lceil \mu \rceil^2 - \lceil \mu \rceil + 2)\log(\lceil \mu \rceil),
\end{aligned}$$

for non-integer $\mu > 0$. Thus, this will imply

$$\begin{aligned}
-\log(x!) &< -(3\lceil \mu \rceil^2 - \lceil \mu \rceil + 2)\log(\lceil \mu \rceil) \\
\implies x\log(\lceil \mu \rceil) - \log(x!) &< 2\lceil \mu \rceil^2 \log(\lceil \mu \rceil) - (3\lceil \mu \rceil^2 - \lceil \mu \rceil + 2)\log(\lceil \mu \rceil) \\
&= -(\lceil \mu \rceil^2 - \lceil \mu \rceil + 2)\log(\lceil \mu \rceil) \\
&\leq 0.
\end{aligned}$$

Since the function $x\log(\lceil \mu \rceil) - \log(x!) \leq 0$ for non-integer $\mu > 0$ with $x = 2\mu^2$, re-arranging leads to $\frac{\lceil \mu \rceil^x}{x!} < 1, \forall x \geq 2\lceil \mu \rceil^2$ using the monotonicity property. This is because the function $\frac{\lceil \mu \rceil^x}{x!}$ is strictly decreasing from $x = \lceil \mu \rceil$ to $x = \infty$, so that it is also strictly decreasing in the range $x \geq 2\lceil \mu \rceil^2$. This completes the proof of Result 2. □

Results 1 and Result 2 can then be used to prove the following two lemmas concerning upper and lower bounds on the solution of $\lambda = \lambda(\mu, \nu)$ of (†).

Lemma 1. *For any $\mu \geq 0$, the solution $\lambda = \lambda(\mu, \nu)$ to the mean constraint (†) satisfies $\lambda/\mu^\nu \to \infty$ and $\lambda/(\mu + 1)^\nu \to 0$ as $\nu \to \infty$. In other words, for any constants $a, b > 0$, $\log(\lambda)$ is always bounded via $a\nu\log(\mu) < \log(\lambda) < b\nu\log(\mu + 1)$ for large enough $\nu > 0$.*

Proof. It is first required to show that the mean constraint (†) for finding the solution of $\lambda = \lambda(\mu, \nu)$ has (at most) one solution. Shmueli et al. (2005) showed that CMP distribution is a linear exponential family with canonical parameter $\log(\lambda)$ for each ν. By properties of exponential families, the mean is a monotonic function of the canonical parameter and therefore a monotonic function of λ too (thus λ only has a solution to the root). Next, to find the solution of $\lambda = \lambda(\mu, \nu)$, one can consider writing it as a root finding problem $m(\lambda) = \sum_{x=0}^{\infty}(x - \mu)\frac{\lambda^x}{(x!)^\nu} = 0$. However, this is not easy to solve analytically and it has to use other method(s) rather than direct evaluation. The aim is to show that the root lies between two functions of λ_1 and λ_2 such that $m(\lambda_1(\mu,\nu)) < 0$ and $m(\lambda_2(\mu,\nu)) > 0$.

First, consider substituting $\lambda = a\mu^\nu$ into the equation $m(\lambda) = 0$, for any $a > 0$. Writing out the sum $m(a\mu^\nu)$ term by term gives the following expression (since x value would be discrete, the cases can be broken down into 3 parts from $0 \le x \le \mu - 1$, $x \ge 2\mu^2$, and the remainder $\mu + 1 \le x \le 2\mu^2 - 1$ since $m(\lambda) = 0$ when $x = \mu$):

$$m(a\mu^\nu) = \underbrace{\sum_{x=0}^{\mu-1}(x-\mu)a^x\left[\frac{\mu^x}{(x!)}\right]^\nu}_{\text{negative component}} + \underbrace{\sum_{x=\mu+1}^{2\mu^2-1}(x-\mu)a^x\left[\frac{\mu^x}{(x!)}\right]^\nu}_{\text{positive component}} + \underbrace{\sum_{x=2\mu^2}^{\infty}(x-\mu)a^x\left[\frac{\mu^x}{(x!)}\right]^\nu}_{\text{remainder component}}$$

One can show that the remainder component above will go to 0. To see this, note that for $x \ge 2\mu^2$, $\frac{\mu^x}{x!}$ would be strictly less than 1. For fixed value a (no matter how big a is), one can choose ν such that $\nu \to \infty$, which makes $\left[\frac{\mu^x}{x!}\right]^\nu$ dominate the remainder terms (including a^x term) and each remainder term becomes 0. Subsequently, to make sure that the whole sum of the remainder term goes to 0, the monotone convergence theorem is used because the terms are strictly decreasing and bounded; here each individual sum tends to 0, so the whole sum must converge to 0 when $\nu \to \infty$.
Using Result 1, the $\frac{\mu^x}{x!}$ term in the negative component would be strictly increasing in x. Also, note that $(x-\mu)$ is always negative in the range $x = 0$ to $x = \mu - 1$ Thus, for arbitrarily large ν, this means that the dominating term (the biggest negative term) would be the last term (i.e. when $x = \mu - 1$), which is:

$$-a^{\mu-1}\left[\frac{\mu^{\mu-1}}{(\mu-1)!}\right]^\nu \tag{2}$$

Similarly, the $\frac{\mu^x}{x!}$ of positive term would be strictly decreasing (here it excluded when $x = \mu$ because it will give $m(\lambda) = 0$). Again, $(x-\mu)$ is always positive in the range $x = \mu + 1$ to $x = 2\mu^2 - 1$. Thus, for arbitrarily large ν, this means the dominating term (the biggest positive term) would be the first term (i.e. when $x = \mu + 1$), which is:

$$+a^{\mu+1}\left[\frac{\mu^{\mu+1}}{(\mu+1)!}\right]^\nu \tag{3}$$

Combining both results, the determining factor of the sign of $m(a\mu^\nu)$ would be to see whether (3) is smaller or larger than (2) for a sufficiently large ν. One way to evaluate this is to look at the ratio of these two:

$$\frac{a^{\mu+1}\left[\frac{\mu^{\mu+1}}{(\mu+1)!}\right]^\nu}{a^{\mu-1}\left[\frac{\mu^{\mu-1}}{(\mu-1)!}\right]^\nu} = a^2\left[\frac{\mu}{\mu+1}\right]^\nu. \tag{4}$$

Then, it can be seen that when setting $a > 0$ fixed, one can choose sufficiently large ν, such that the ratio would be less than one which means the negative component dominates more than the positive component. Thus, $m(a\mu^\nu)$ is negative for large enough ν.

Similarly, one might want to check the behaviour for $\lambda = b(\mu+1)^{\nu}$ to get a positive bound for the root of $m(\lambda) = 0$.
Consider $\lambda = b(\mu+1)^{\nu}$ for any $b > 0$. Using the same partition as before, it produces:

$$m(b(\mu+1)^{\nu}) = \underbrace{\sum_{x=0}^{\mu-1}(x-\mu)b^{x}\left[\frac{(\mu+1)^{x}}{(x!)}\right]^{\nu}}_{\text{negative component}} + \underbrace{\sum_{x=\mu+1}^{2\mu^{2}-1}(x-\mu)b^{x}\left[\frac{(\mu+1)^{x}}{(x!)}\right]^{\nu}}_{\text{positive component}} + \underbrace{\sum_{x=2\mu^{2}}^{\infty}(x-\mu)b^{x}\left[\frac{(\mu+1)^{x}}{(x!)}\right]^{\nu}}_{\text{remainder component}}$$

Using similar argument as the case for $\lambda = a\mu^{\nu}$, the negative component dominates when $x = \mu - 1$, the positive component dominates when $x = \mu + 1$, and the remainder term will approach 0. Then, it remains to check whether the negative or positive component is bigger to determine the sign of $m(b(\mu+1)^{\nu})$. Substituting the dominant terms, the equation is:

$$b^{\mu+1}\left[\frac{(\mu+1)^{\mu+1}}{(\mu+1)!}\right]^{\nu} - b^{\mu-1}\left[\frac{(\mu+1)^{\mu-1}}{(\mu-1)!}\right]^{\nu}$$

Finding the ratio of these two terms, similar to the idea in (4), it produces:

$$\frac{b^{\mu+1}\left[\frac{(\mu+1)^{\mu+1}}{(\mu+1)!}\right]^{\nu}}{b^{\mu-1}\left[\frac{(\mu+1)^{\mu-1}}{(\mu-1)!}\right]^{\nu}} = b^{2}\left[\frac{\mu+1}{\mu}\right]^{\nu}.$$

Then, it can be seen that when setting $b > 0$ fixed, one can choose sufficiently large ν, such that the ratio would be greater than one which means the increase of ratio bigger than 1 being exponentiated is faster than squaring value of b. Thus, $m(b(\mu+1)^{\nu})$ is positive for large enough ν.

In conclusion, for any fixed $a, b > 0$, the solution (root) of $\lambda = \lambda(\mu, \nu)$ with mean constraint $m(\lambda) = 0$ is between $a\mu^{\nu}$ (negative value) and $b(\mu+1)^{\nu}$ (positive value) for sufficiently large ν. Hence, this must be $a = \frac{\lambda}{\mu^{\nu}} \to \infty$ and $b = \frac{\lambda}{(\mu+1)^{\nu}} \to 0$ for integer μ. This conclude the proof of Lemma 1. $\square$

Lemma 2. *If μ is non-integer, the bounds on the solution $\lambda = \lambda(\mu, \nu)$ to the mean constraint (†) can be tightened to $\lambda > \Delta\lceil\mu\rceil^{\nu}$ and $\lambda < (1-\Delta)^{-1}\lceil\mu\rceil^{\nu}$ for sufficiently large ν. In other words, for non-integer μ and large ν, the bound of* $\log(\lambda)$ *is* $\nu\log\lceil\mu\rceil + \log(\Delta) < \log(\lambda) < \nu\log\lceil\mu\rceil - \log(1-\Delta)$

Proof. Using Result 2, the idea would be similar to Lemma 1, but for non-integer μ. Again, it requires $\lambda = \lambda(\mu, \nu)$ to have (at most) one solution under the mean constraint (†). Consider when $\lambda = \Delta\lceil\mu\rceil^{\nu}$, and $\Delta = \mu - \lfloor\mu\rfloor$ and $1 - \Delta = \lceil\mu\rceil - \mu$. With a similar partition idea in Lemma 1 proof, the bound can have 3 terms:

$$m(\Delta\lceil\mu\rceil^{\nu}) = \underbrace{\sum_{x=0}^{\lfloor\mu\rfloor}(x-\mu)\Delta^{x}\left[\frac{\lceil\mu\rceil^{x}}{(x!)}\right]^{\nu}}_{\text{negative component}} + \underbrace{\sum_{x=\lceil\mu\rceil}^{2\lceil\mu\rceil^{2}-1}(x-\mu)\Delta^{x}\left[\frac{\lceil\mu\rceil^{x}}{(x!)}\right]^{\nu}}_{\text{positive component}} + \underbrace{\sum_{x=2\lceil\mu\rceil^{2}}^{\infty}(x-\mu)\Delta^{x}\left[\frac{\lceil\mu\rceil^{x}}{(x!)}\right]^{\nu}}_{\text{remainder component}}$$

The negative component is strictly increasing (based on Result 2) and has a negative value in the range $x = 0$ to $x = \lfloor\mu\rfloor$, so the dominant term would be when $x = \lfloor\mu\rfloor$ for large enough ν. The positive component is strictly decreasing (based on Result 2) and has a positive value in the range $x = \lceil\mu\rceil$ to $x = 2\lceil\mu\rceil^2 - 1$, so the dominant term would be when $x = \lceil\mu\rceil$ for large ν. The remainder component will be 0 because for $x \geq 2\lceil\mu\rceil^2$, $\frac{\mu^x}{x!}$ would be strictly less than 1. Next, as $\nu \to \infty$, $\left[\frac{\lceil\mu\rceil^x}{(x!)}\right]^\nu$ will dominate the other terms in remainder component and become 0. Also, using monotone convergence theorem shows the whole sum must converges to 0. Now evaluating the negative and positive components' dominating term (substituting when $x = \lfloor\mu\rfloor$ and $x = \lceil\mu\rceil$):

$$(\lfloor\mu\rfloor - \mu)\Delta^{\lfloor\mu\rfloor}\left[\frac{\lceil\mu\rceil^{\lfloor\mu\rfloor}}{(\lfloor\mu\rfloor)!}\right]^\nu + (\lceil\mu\rceil - \mu)\Delta^{\lceil\mu\rceil}\left[\frac{\lceil\mu\rceil^{\lceil\mu\rceil}}{(\lceil\mu\rceil)!}\right]^\nu = (1-\Delta)\Delta^{\lceil\mu\rceil}\left[\frac{\lceil\mu\rceil^{\lceil\mu\rceil}}{(\lceil\mu\rceil)!}\right]^\nu - \Delta\Delta^{\lfloor\mu\rfloor}\left[\frac{\lceil\mu\rceil^{\lfloor\mu\rfloor}}{(\lfloor\mu\rfloor)!}\right]^\nu$$

Evaluating this ratio, this would give:

$$\frac{(1-\Delta)\Delta^{\lceil\mu\rceil}\left[\frac{\lceil\mu\rceil^{\lceil\mu\rceil}}{(\lceil\mu\rceil)!}\right]^\nu}{\Delta\Delta^{\lfloor\mu\rfloor}\left[\frac{\lceil\mu\rceil^{\lfloor\mu\rfloor}}{(\lfloor\mu\rfloor)!}\right]^\nu} = 1 - \Delta < 1$$

since $\Delta^{\lfloor\mu\rfloor+1} = \Delta^{\lceil\mu\rceil}$. This implies $m(\Delta\lceil\mu\rceil^\nu)$ is negative for large ν.

By the "reverse-engineered" idea, when μ is non-integer, there should be probability mass in both $\lceil\mu\rceil$ and $\lfloor\mu\rfloor$ (which later correspond to Proposition 2). Thus, one might want to check the behaviour for $(1-\Delta)^{-1}$ to get an intuition on the capability to have the bound that give the root of $m(\lambda) = 0$. Consider $\lambda = (1-\Delta)^{-1}\lceil\mu\rceil^\nu$. Using similar argument when $\lambda = \Delta\lceil\mu\rceil^\nu$, one needs to compare the dominating terms, i.e. when $x = \lfloor\mu\rfloor$ and $x = \lceil\mu\rceil$ of negative and positive components respectively. This is given by:

$$\begin{aligned}&(\lfloor\mu\rfloor - \mu)(1-\Delta)^{-\lfloor\mu\rfloor}\left[\frac{\lceil\mu\rceil^{\lfloor\mu\rfloor}}{(\lfloor\mu\rfloor)!}\right]^\nu + (\lceil\mu\rceil - \mu)(1-\Delta)^{-\lceil\mu\rceil}\left[\frac{\lceil\mu\rceil^{\lceil\mu\rceil}}{(\lceil\mu\rceil)!}\right]^\nu \\ &= (1-\Delta)(1-\Delta)^{-\lceil\mu\rceil}\left[\frac{\lceil\mu\rceil^{\lceil\mu\rceil}}{(\lceil\mu\rceil)!}\right]^\nu - \Delta(1-\Delta)^{-\lfloor\mu\rfloor}\left[\frac{\lceil\mu\rceil^{\lfloor\mu\rfloor}}{(\lfloor\mu\rfloor)!}\right]^\nu\end{aligned}$$

Evaluating this ratio, this gives:

$$\frac{(1-\Delta)(1-\Delta)^{-\lceil\mu\rceil}\left[\frac{\lceil\mu\rceil^{\lceil\mu\rceil}}{(\lceil\mu\rceil)!}\right]^\nu}{\Delta(1-\Delta)^{-\lfloor\mu\rfloor}\left[\frac{\lceil\mu\rceil^{\lfloor\mu\rfloor}}{(\lfloor\mu\rfloor)!}\right]^\nu} = \frac{1}{\Delta} > 1$$

Since the ratio is greater than 1, so $m((1-\Delta)^{-1}\lceil\mu\rceil^\nu)$ will be positive for large enough ν. In conclusion, for non-integer μ, the solution of $\lambda = \lambda(\mu,\nu)$ with mean constraint $m(\lambda) = 0$ is between $\Delta\lceil\mu\rceil^\nu$ (the negative term) and $(1-\Delta)^{-1}\lceil\mu\rceil^\nu$ (the positive term) for sufficiently large ν.

□

There is no direct theoretical justification / asymptotic proof yet for large μ, but empirical results (see Figure 2.1 and Figure 2.2) show that the bilinear interpolation/extrapolation still works well using the argument of asymptotic results for large ν. In fact, when μ is relatively big, using bilinear interpolation with $\nu \geq 2$ is good enough (accuracy up to two decimal points) as the μ parameter is still involved in Lemma 1 and Lemma 2 in some sense. To increase the accuracy when $\nu < 2$, one could use built-in functions in `mpcmp` or make a finer μ and ν grid which is detailed in Section 2.3 up to $\nu = 2$. Moreover, functions in `mpcmp` evaluation are still accurate and fast enough when μ is large and ν is relatively small up to the combination of (μ, ν) that could be handled by the package. Thus, using the combination of evaluation using `mpcmp` and interpolation/extrapolation works well as thoroughly checked in Section 2.4.

Additionally, another way to characterise the CMP/CMP$_\mu$ distribution is via the ratio of subsequent probabilities:

$$\begin{aligned} \frac{\mathrm{P}(X=x+1)}{\mathrm{P}(X=x)} &= \frac{\lambda}{(x+1)^\nu} \\ \iff \frac{\mathrm{P}(X=x)}{\mathrm{P}(X=x-1)} &= \frac{\lambda}{(x)^\nu} \end{aligned}$$

Using Lemma 1 and Lemma 2, one can prove the proposition 1(i) and 1(ii) in Huang (2020) (denoted Proposition 1 and Proposition 2 respectively below) :

Proposition 1. *As $\nu \to \infty$, the mpcmp distribution with mean $\mu \geq 0$ converges to a unit point mass at μ if μ is integer, i.e. $P(X=\mu) \to 1$*

For integer case, if $\frac{\mathrm{P}(X=\mu+1)}{\mathrm{P}(X=\mu)} = \frac{\lambda}{(\mu+1)^\nu} \to 0$ as $\nu \to \infty$, this means the numerator is infinitely smaller than denuminator. Conversely, If $\frac{\mathrm{P}(X=\mu)}{\mathrm{P}(X=\mu-1)} = \frac{\lambda}{(\mu)^\nu} \to \infty$ as $\nu \to \infty$, this means the numerator is infinitely larger than the denuminator. Using both of them, then the limit is unit point mass at μ with probability 1. Thus, this justifies the usage of bilinear interpolation because the value of λ is sandwiched between two bilinear functions (this becomes trivial after having Lemma 1).

Proposition 2. *As $\nu \to \infty$, the mpcmp distribution with mean $\mu \geq 0$ converges to a shifted Bernoulli on the two integers $\lfloor \mu \rfloor$ and $\lceil \mu \rceil$, if μ is non-integer, with probabilities equal to the fractional parts of μ, i.e. $P(X = \lfloor \mu \rfloor) \to 1 - \Delta$ and $P(X = \lceil \mu \rceil) \to \Delta$ where $\Delta = \mu - \lfloor \mu \rfloor$*

For non-integer case, there are 2 values when we find the values of floor and ceiling of μ, i.e. $\lfloor \mu \rfloor$ and $\lceil \mu \rceil = \lfloor \mu \rfloor + 1$. If $\frac{\mathrm{P}(X=\lceil \mu \rceil+1)}{\mathrm{P}(X=\lceil \mu \rceil)} \to 0$ as $\nu \to \infty$, this means that the probability mass in $\lceil \mu \rceil$ will be infinitely higher than (or dominates) $\lceil \mu \rceil + 1$ to the right. Similarly, If $\frac{\mathrm{P}(X=\lfloor \mu \rfloor)}{\mathrm{P}(X=\lfloor \mu \rfloor-1)} \to \infty$, this means that the probability mass in $\lfloor \mu \rfloor$ is infinitely higher than (or dominates) $\lfloor \mu \rfloor$-1 to the left.

Using Lemma 2, λ is bounded between 2 values, i.e.

$$\begin{aligned} \Delta(\lceil \mu \rceil)^\nu < \lambda &< \frac{(\lceil \mu \rceil)^\nu}{1-\Delta} \\ \implies \Delta < \frac{\lambda}{(\lceil \mu \rceil)^\nu} &< \frac{1}{1-\Delta} \end{aligned}$$

Then, using the characterisation of CMP/CMP$_\mu$:

$$\frac{\mathrm{P}(X=\lceil \mu \rceil)}{\mathrm{P}(X=\lfloor \mu \rfloor)} = \frac{\mathrm{P}(X=\lfloor \mu \rfloor+1)}{\mathrm{P}(X=\lfloor \mu \rfloor)} = \frac{\lambda}{(\lfloor \mu \rfloor+1)^\nu} = \frac{\lambda}{(\lceil \mu \rceil)^\nu}$$

This will give

$$\Delta < \frac{\mathrm{P}(X=\lceil \mu \rceil)}{\mathrm{P}(X=\lfloor \mu \rfloor)} < \frac{1}{1-\Delta}$$

Thus, as $\nu \to \infty$, this means that the ratio must converge to some constant, i.e. $P(X = \lceil \mu \rceil) \to \Delta$ and $P(X = \lfloor \mu \rfloor) \to 1 - \Delta$ for the fixed μ and inequalities from Lemma 2 to hold, and this ratio probability will achieve $\frac{\Delta}{1-\Delta}$. Combining the above results, there can be **at most two non-zero values** in the pmf which are at $\lfloor \mu \rfloor$ and $\lceil \mu \rceil$ which are weighted according to the magnitude to nearest integer, i.e. $P(X = \lfloor \mu \rfloor) \to 1 - \Delta$ and $P(X = \lceil \mu \rceil) \to \Delta$, where $\Delta = \mu - \lfloor \mu \rfloor$ (This becomes trivial after having Lemma 2).

After establishing this result, how to **justify the usage of bilinear interpolation**? This can be seen from modifying expression of Lemma 1 and Lemma 2 with $a, b = 1$ gives the bound for $\log(\lambda)$, $\nu \log(\mu) < \log(\lambda) < \nu \log(\mu + 1)$, i.e. $\log(\lambda)$ will be sandwiched between two linear functions in ν and two linear functions in $\log(\mu)$ and $\log(\mu + 1)$ as can be seen in Figure 2.2, hence suggesting that bilinear interpolation/extrapolation in ν and in $\log(\mu)$ yield a good approximation. Also, when using bigger scaling factor $a, b > 0$, the gap between the upper bound (red line) and the lower bound (blue line) of $\log(\lambda)$ will be wider, but $\log(\lambda)$ values are still within this bound.

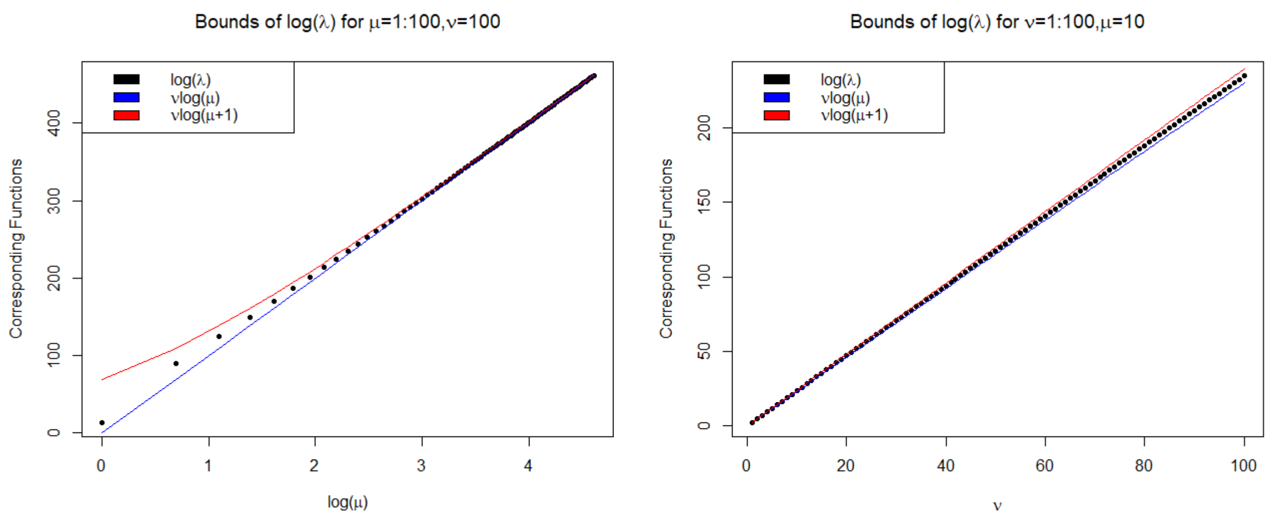

Figure 2.2: Visual Representation of $\log(\lambda)$ bounds, i.e. $\nu \log(\mu) < \log(\lambda) < \nu \log(\mu + 1)$

2.2 Bilinear Interpolation

In general, simple case interpolation for finding y_3 given x_3 between 2 known points (x_1, y_1) and (x_2, y_2) can be obtained by using the following formula:

$$y_3 = y_1 + \frac{x_3 - x_1}{x_2 - x_1} \times (y_2 - y_1),$$

or taking proportion within the bounds

$$y_3 = \left(1 - \frac{x_3 - x_1}{x_2 - x_1}\right) \times y_1 + \left(\frac{x_2 - x_3}{x_2 - x_1}\right) \times y_2,$$

where $x_1 < x_3 < x_2$ and $y_1 < y_3 < y_2$.

Now suppose that we want to find the value of the unknown function f (which corresponds to $\log(\lambda)$) at an arbitrary point $(\tilde{\mu_3}, \nu_3)$ where $\tilde{\mu_1} < \tilde{\mu_3} < \tilde{\mu_2}$ and $\nu_1 < \nu_3 < \nu_2$ such that:

$$\tilde{\mu_j} = \log(\mu_j), \qquad \text{for } j = 1,2,3.$$

Then there will be four different points, $(\tilde{\mu_1}, \nu_1)$,$(\tilde{\mu_1}, \nu_2)$,$(\tilde{\mu_2}, \nu_1)$, and $(\tilde{\mu_2}, \nu_2)$ that function f are weighted from. This means interpolation can happen for both $\tilde{\mu}$ and ν at the same time, which is called **Bilinear Interpolation**. Using the linear combination of weight and evaluating the function from each of 4 points would give relatively accurate approximation value of function f at point $(\tilde{\mu_3}, \nu_3)$ as shown in a compact form in the equation ($\bigoplus$). Then, this is geometrically visualized in Figure 2.3.

$$f(\tilde{\mu_3}, \nu_3) \approx w_{11} f(\tilde{\mu_1}, \nu_1) + w_{12} f(\tilde{\mu_1}, \nu_2) + w_{21} f(\tilde{\mu_2}, \nu_1) + w_{22} f(\tilde{\mu_2}, \nu_2) \qquad (\bigoplus)$$

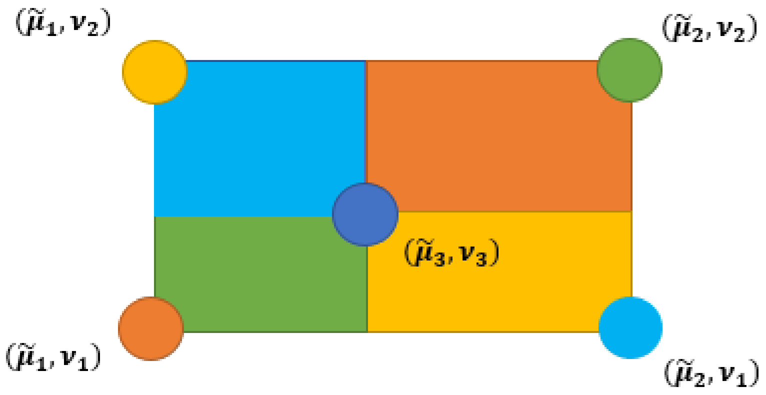

Figure 2.3: Geometric Illustration of Bilinear Interpolation on $\log(\lambda)$

where the weight is defined as:

$$\begin{bmatrix} w_{11} \\ w_{12} \\ w_{21} \\ w_{22} \end{bmatrix} = \begin{bmatrix} (1 - \frac{\tilde{\mu_3} - \tilde{\mu_1}}{\tilde{\mu_2} - \tilde{\mu_1}})(1 - \frac{\nu_3 - \nu_1}{\nu_2 - \nu_1}) \\ (1 - \frac{\tilde{\mu_3} - \tilde{\mu_1}}{\tilde{\mu_2} - \tilde{\mu_1}})(\frac{\nu_3 - \nu_1}{\nu_2 - \nu_1}) \\ (\frac{\tilde{\mu_3} - \tilde{\mu_1}}{\tilde{\mu_2} - \tilde{\mu_1}})(1 - \frac{\nu_3 - \nu_1}{\nu_2 - \nu_1}) \\ (\frac{\tilde{\mu_3} - \tilde{\mu_1}}{\tilde{\mu_2} - \tilde{\mu_1}})(\frac{\nu_3 - \nu_1}{\nu_2 - \nu_1}) \end{bmatrix}$$

The idea above can be extended to the unknown function of interest, i.e. $\log(Z)$ by replacing $\tilde{\mu_j}$ with μ_j for $j = 1,2,3$ as the bilinearity of $\log(Z)$ happens on μ and ν instead of $\log(\mu)$ and ν as seen in Figure 2.1. Now take a look on a simple example, if we are given the value of $f(\mu, \nu)$ the grid at four points (μ, ν): (2,7), (2,8), (3,7), and (3,8). Then now, one wants to find $f(\mu, \nu)$ at (2.2,7.1). Then this can be evaluated:

$$\begin{aligned} f(2.2, 7.1) = (1-0.2)(1-0.1)f(2,7) + (1-0.2)(0.1)f(2,8) \\ + (0.2)(1-0.1)f(3,7) + (0.2)(0.1)f(3,8). \end{aligned}$$

Then, apply the function of $f(\tilde{\mu}, \nu)$ or $f(\mu, \nu)$ as a method for calculating $\log(\lambda)$ or $\log(Z)$ respectively, and use the capability to interpolate in order to achieve similar results with the true data generating mechanism using a faster algorithm. Then, the reliability of this method can be measured using the relative error from the true value and the interpolation method in Section 3.1.

Subsequently, the calculation for $\log(\lambda)$ and $\log(Z)$ has been stored in a grid for each combination of $(\tilde{\mu}, \nu)$ and (μ, ν) respectively. Thus, it would need to find the index of the grid to do the interpolation. The index of the grid would be formulated depending on how the grid was set up which is discussed in Section 2.3.

2.3 Pre-computation of $\log(\lambda)$ and $\log(Z)$ on a Grid

Firstly, one wants to set up fine enough grid for our μ and ν to maintain computation accuracy. In this case, the chosen grid for μ is linear starting from 1 to 256 with an incremental increase of 0.2. Similarly, the chosen grid for ν is linear starting from 1 to 112 with incremental increase of 3. The important part here is to make the log-scale interpolation $\tilde{\mu}$ for $\log(\lambda)$ and linear-scale interpolation μ for $\log(Z)$ while using linear-scale interpolation ν for both $\log(\lambda)$ and $\log(Z)$. Linearity in the log-scale will still imply in the linear-scale when the relationship is linear. Then this can be simulated for evaluating objective functions (in this case $\log(\lambda)$ and $\log(Z)$) in a double loop for each possible combination of μ and ν on the grid. These bounds on the grid has been chosen as such due to empirical result Figure 2.1. Of course, minimization of approximation error can be done by finer grid (having smaller increments in the grid) but the main consideration was running time to evaluate the function at each combination of the grid, which in the long term can be done on cloud computing system or high performance computer.

For **bilinear interpolation method**, the construction from Section 2.2 give us the condition that for datapoint of interest, $(\tilde{\mu_3}, \nu_3)$, we have $\tilde{\mu_1} < \tilde{\mu_3} < \tilde{\mu_2}$ (interchangably with μ) and $\nu_1 < \nu_3 < \nu_2$. The formula used to track the indices from the grid are:

$$\text{lower\_ind } \mu_1 = \lfloor 1 + \frac{\mu_3 - 1}{\Delta\mu} \rfloor, \qquad \text{lower\_ind } \nu_1 = \lfloor \frac{\nu_3 + 2}{\Delta\nu} \rfloor,$$

$$\text{higher\_ind } \mu_2 = \text{lower\_ind } \mu_1 + 1, \qquad \text{higher\_ind } \nu_2 = \text{lower\_ind } \nu_1 + 1,$$

where $\Delta\mu$ is the jump increment between μ grid values which is 0.2 in this case and $\Delta\nu$ is the jump increment between ν grid values which is 3 in this case. The index of higher value in the grid is always one plus the index of the lower value in the grid (this is more computationally efficient as calculating +1 will be more feasible rather than finding the upper index by using ceiling function).

2.4 Bilinear Extrapolation

What happens when the chosen μ and ν are outside the grid? the idea of bilinear interpolation can be extended to bilinear extrapolation using a similar method, but always taking the biggest and second biggest index from the grid setup from Section 2.3 for both $\tilde{\mu}$ (or μ) and ν. Then extrapolate to the desired μ and ν value. How does one define the parameter? Looking at Table 2.1 would show which method to use for every μ and ν combination.

μ	ν	Method
$\mu \leq$ grid	$\nu \leq$ grid	`mpcmp`
$\mu \leq$ grid	$\nu \in$ grid	`mpcmp`
$\mu \leq$ grid	$\nu >$ grid	`mpcmp`
$\mu \in$ grid	$\nu \leq$ grid	`mpcmp`
$\mu \in$ grid	$\nu \in$ grid	interp
$\mu \in$ grid	$\nu >$ grid	extrap.nu
$\mu >$ grid	$\nu \leq$ grid	`mpcmp`
$\mu >$ grid	$\nu \in$ grid	extrap.mu
$\mu >$ grid	$\nu >$ grid	extrap.both

Table 2.1: Possible combinations of μ and ν and method used for all conditions

Compiling the results from above, it is decided that the default (considering accuracy and computation time) option for each combination would be `mpcmp` because it can handle most calculations accurately and quickly when μ or ν is small, then if both μ and ν are inside the grid, the function should interpolate (this was due to a computational gain but still maintaining accuracy while interpolating compared to evaluation using built-in functions of `mpcmp`). Then, one can extrapolate if μ or ν is beyond the grid (mainly this was due to computational limitation in the original package `mpcmp` for some combination of μ and ν). For example, take a look on the evaluation of $\log(\lambda)$ in Table 2.2. From here, it can be seen that `mpcmp` fails to converge even though adding 1 on ν, so the superiority of using bilinear extrapolation when μ and ν are outside the grid are justified. However, a slight modification by extending computation speed can be done using only bilinear interpolation/extrapolation without direct `mpcmp` evaluation, and different ways of writing the code would affect its performance (see Chapter 3 for details).

μ	ν	$\log(\lambda)$ (using `mpcmp`)	$\log(\lambda)$ (using extrap.both)
$\mu = 100$	$\nu = 150$	691.5193	691.5193
$\mu = 100$	$\nu = 151$	not converge	696.1295

Table 2.2: Combination of μ and ν that does not converge in `mpcmp` but converges in bilinear extrapolation

Chapter 3

Accuracy and Speed of The Bilinear Interpolation & Extrapolation

3.1 Comparison of Numerical Accuracy and Computational Speed

Accuracy can be tested by comparing built-in functions from `mpcmp` against the interpolation/extrapolation methods used from different conditions of μ and ν stated in Section 2.4. Suppose define a sequence of value(s) to evaluate $\log(\lambda)$ and $\log(Z)$:

$T_1 = \{1.5, 2, 2.5, ..., 99.5\}$	$T_7 = \{100, 99, 98, ..., 2, 1\}$
$T_2 = \{13, 13.05, 13.1, ..., 23\}$	$T_8 = \{40, 41, 42, ..., 80\}$
$T_3 = \{120, 120.05, 120.1, ..., 130\}$	$T_9 = \{2\}$
$T_4 = \{256, 256.05, 256.1, ..., 266\}$	$T_{10} = \{0.1\}$
$T_5 = \{113, 113.05, 113.1, ..., 123\}$	$T_{11} = \{60\}$
$T_6 = \{1, 2, 3, ..., 100\}$	$T_{12} = \{20\}$

Table 3.1: Sequence of values for testing accuracy and computational speed.

These values have been chosen such that they satisfied the condition for performing bilinear interpolation/ extrapolation method with approximately sequence of 200 values as shown in Table 3.2:

μ	ν	Method	Mean of Error $\log(\lambda)$ against `mpcmp`	Mean of Error $\log(Z)$ against `mpcmp`:
$\mu = T_1$	$\nu = T_1$	interp	2.06e-4	3.69e-4
$\mu = T_2$	$\nu = T_3$	extrap.nu	2.33e-5	6.67e-5
$\mu = T_4$	$\nu = T_2$	extrap.mu	9.17e-8	2.25e-7
$\mu = T_4$	$\nu = T_5$	extrap.both	1.23e-7	7.21e-7

Table 3.2: Testing accuracy for $\log(\lambda)$ and $\log(Z)$ for different μ and ν values.

where the Mean of Error is calculated by finding the mean of (values from direct mpcmp divided by values from Method used evaluation minus 1). As it is seen from Mean of Error test, its error using interpolate/extrapolate compared to the original or true value is only seen at roughly 4 decimal points (fairly negligible) for the points that are not exactly on the grid. Obviously, if the points chosen are exactly on the grid, the relative error should be 0 as it is compared to functions used in mpcmp directly. Here, the summax specifications for mpcmp are accurate enough as a baseline comparison because by default, summax are the maximum of 100 or the upper 20 standard deviations away from the largest mean parameter values to ensure sum terms are large enough for capturing the full pmf. For example, if one has CMP$_\mu$ model with $\mu = 30$ and $\nu = 2$. This means 20 standard deviation away from the mean suggests that the package will use summax$= 30 + 20 \times \sqrt{30/2} \approx 108$ by default. Since the package also allows this to be user-defined, choosing summax to be more than 108 or large enough, e.g. summax $= 300$ would ensure high accuracy of mpcmp.

However, for small values of μ and/or ν, the accuracy is not quite satisfactory due to the pattern of ”step function” behaviour (very high increase followed by small increase) as seen in Figure 2.1, particularly obvious in the plot $\log(\mu)$ against $\log(\lambda)$ for different ν values. In other words, the bigger values of μ and/or ν, the more accurate the interpolation/extrapolation method against direct mpcmp. At this point, it stops at the chosen grid values above in Section 2.3 for getting at least 2-3 decimal points accurate and significant improvement in computation time compared to functions in mpcmp.

The computational speed test was carried through using the function system.time from the base function in R, where elapsed time is the time charged to the CPU(s) for the expression which is commonly used as comparison metric for time, user time is the wall clock time (the actual human time taken to evaluate the methods), and system time gives the CPU time spent by the kernel (the operating system - operations that involve resources that many processes must share) on behalf of the current process. All time measures are denoted in seconds. Even though these numerical performance could vary due to machinery processing error, this is handled by including results with standard deviation from 10 times simulation. The results of the simulation repeated 10 million times (to capture differences between methods) are visually shown in Figure 3.1, while the numerical representation for each mean and standard deviation is shown Table A.1:

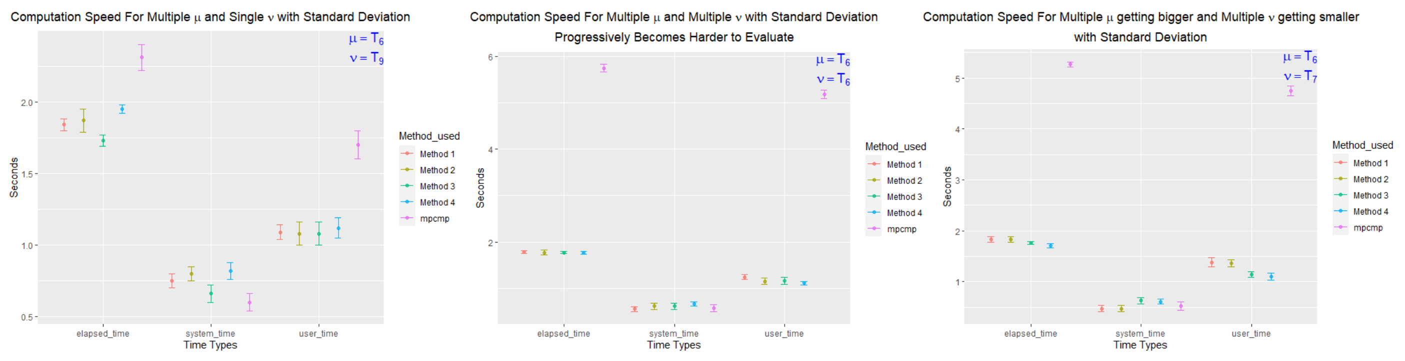

Figure 3.1: Computation speed $\log(\lambda)$ for different μ and ν values using 5 different methods

There are several ways to implement the coding for comparing computational speed to obtain the objective functions $\log(\lambda), \log(Z)$, and the pmf as denoted in ($\star$). For $\log(\lambda)$ and $\log(Z)$, the proposed methods are:

- Method 1: Writing out a vectorized function for all 5 different conditions (i.e. direct `mpcmp`[1], interp, extrap.nu, extrap.mu, or extrap.both).
- Method 2: Similar to Method 1, but creating a function for single value evaluation of μ and ν, then incorporate default R function `Vectorise` for multiple values.
- Method 3: Similar to Method 1, but condensing the Interpolation/Extrapolation method into 1 condition, thus only evaluating 2 conditions (direct `mpcmp`[1] or Interpolation/Extrapolation).
- Method 4: only Interpolation/Extrapolation all the way without direct `mpcmp` evaluation.

From Figure 3.1 and Table A.1, one can suggest to calculate the all combinations of $\mu = T_6$ and $\nu = T_6$, i.e. $\mu = 1$ with $\nu = 1:100$, $\mu = 2$ with $\nu = 1:100$ and so on. However, the trajectory to do so is represented by using $\mu = T_6$ and $\nu = T_7$ instead (producing the same patterns that there is significant improvement in using 4 proposed methods compared to `mpcmp`) due to computation time limitation of this project. Method 1 shows a computational improvement compared to `mpcmp`, but still slightly slower compared to the other methods. Method 2 was the simplest to start with, but the built-in `Vectorise` function uses many `for` loop and `if` functions which slows down the computation speed. Method 2 and Method 3 show very competitive results when evaluating easy cases to hard (i.e. when μ and ν is evaluated at T_6). However, adding slight complexity by changing the value of ν to T_7, Method 3 is faster than Method 2, and the fastest compared to Method 1 and Method 2. Method 4 seems to be the fastest due to the fixed time taken for performing linear operations evaluation, whereas the variability of the computational speed of direct `mpcmp` would depend on large value of μ and/or ν.

Method 3 would be the most preferable method due to the ease of evaluating 2 conditions than Method 1. Secondary preference would be to use Method 4 to take full advantage of fixed time evaluation. After evaluating these methods, it is shown that calculation for getting $\log(\lambda)$ is **faster about approximately 4-4.5 times** than direct evaluation using functions in `mpcmp`.

Similarly, given the values of $\log(\lambda)$, the computational time for $\log(Z)$ is shown in Table A.2 and visually represented in Figure 3.2. Notice that all of the proposed methods are fairly producing similar performance, i.e. not much computation time improvement (slight improvement between 1 to 2 decimal points). This was due to direct evaluation of $\log(Z)$ will be a summation that depends on $\log(\lambda)$ and ν (see ($\star$)). Hence, due to this concern, using built-in `mpcmp` for evaluating $\log(Z)$ would

[1] direct `mpcmp` is not 100% identical to how the built-in functions `mpcmp` do the coding for evaluating parameter of interest, instead calculation is done by pre-processing / taking only relevant or useful subset of the original code, particularly known as variable 'exact' and 'exact3' (see github page in Appendix A.1) for evaluating $\log(\lambda)$ and $\log(Z)$ respectively

be the first preference, but if one would want to focus on computation speed, Method 3 and Method 4 is preferred.

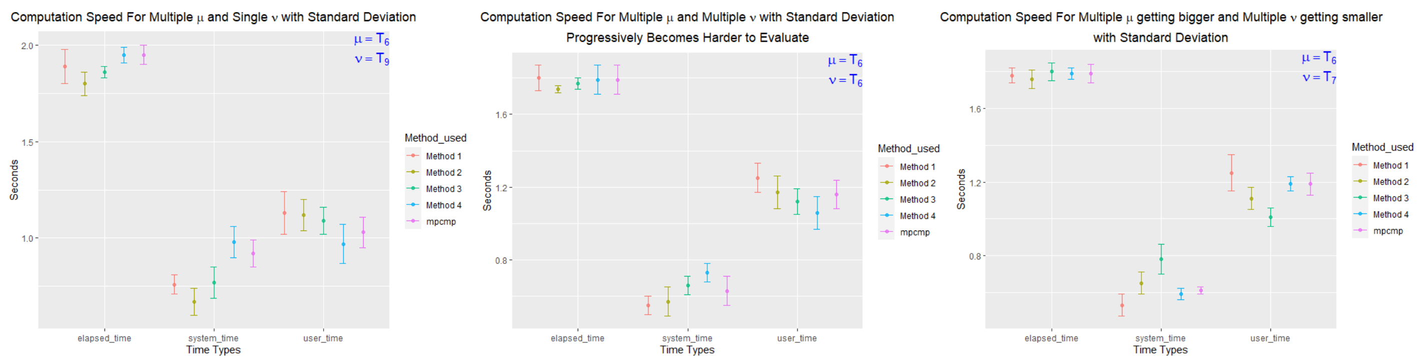

Figure 3.2: Computation speed $\log(Z)$ for different μ and ν values using 5 different methods

Deducing the speed and accuracy for $\log(\lambda)$ and $\log(Z)$ above, the proposed methods for calculating pmf are:

- Method a: combination of Method 1 $\log(\lambda)$ evaluation and Method 1 $\log(Z)$ evaluation
- Method b: combination of Method 1 $\log(\lambda)$ evaluation and `mpcmp` $\log(Z)$ evaluation
- Method c: combination of Method 3 $\log(\lambda)$ evaluation and Method 3 $\log(Z)$ evaluation
- Method d: combination of Method 3 $\log(\lambda)$ evaluation and `mpcmp` $\log(Z)$ evaluation

For accuracy purpose, one only needs to compare for 2 cases (then compare them with direct `mpcmp` using `dcomp` function): making bilinear interpolation/extrapolation for both $\log(\lambda)$ and $\log(Z)$ (can be done using Method a), and only interpolating / extrapolating $\log(\lambda)$ but normalizing $\log(Z)$ using `mpcmp` (can be done using Method b):

μ	ν	x	Method	Mean of Error against `mpcmp dcomp`	$\sum_x P(X=x)$:
$\mu = T_{11}$	$\nu = 100 \times T_{10}$	$x = T_6$	Method a	0	1
$\mu = T_{11}$	$\nu = 100 \times T_{10}$	$x = T_6$	Method b	0	1
$\mu = T_{11}$	$\nu = T_{12}$	$x = T_8$	Method a	1.21e-3	0.9988
$\mu = T_{11}$	$\nu = T_{12}$	$x = T_8$	Method b	3.61e-12	1
$\mu = T_{11}$	$\nu = T_9$	$x = T_6$	Method a	1.21e-1	0.8915
$\mu = T_{11}$	$\nu = T_9$	$x = T_6$	Method b	1.10e-4	1

Table 3.3: Testing accuracy for different methods of evaluating pmf `dcomp`

where the Mean of Error is calculated by finding the mean of (direct `mpcmp` divided by evaluation of Method used minus 1). As seen in Table 3.3, Method a) and Method b) have no difference for some combination of $\mu = 60, \nu = 10, x = 1:100$, but it is captured that when $\nu = 20, x = 40:80$ and $\nu = 2, x = 1:100$, the bilinear interpolation of normalization constant $Z(\lambda(\mu,\nu),\nu)$ would not produce proper pmf (since $\sum_x \mathrm{P}(X = x) \neq 1$), but maybe useful to get rough estimations of pmf (which is useful in some cases for trading off accuracy for computation speed). To make a valid pmf and for generalisation purposes, this shows Method b) is preferred and might be deemed as more stable to preserve the validity of pmf condition $\sum_x \mathrm{P}(X = x) = 1$ (even though the mean error is relatively small). The computational speed test for pmf can be seen in Figure 3.3 whose numerical representation for each mean and standard deviation (in brackets) is shown in Table A.3:

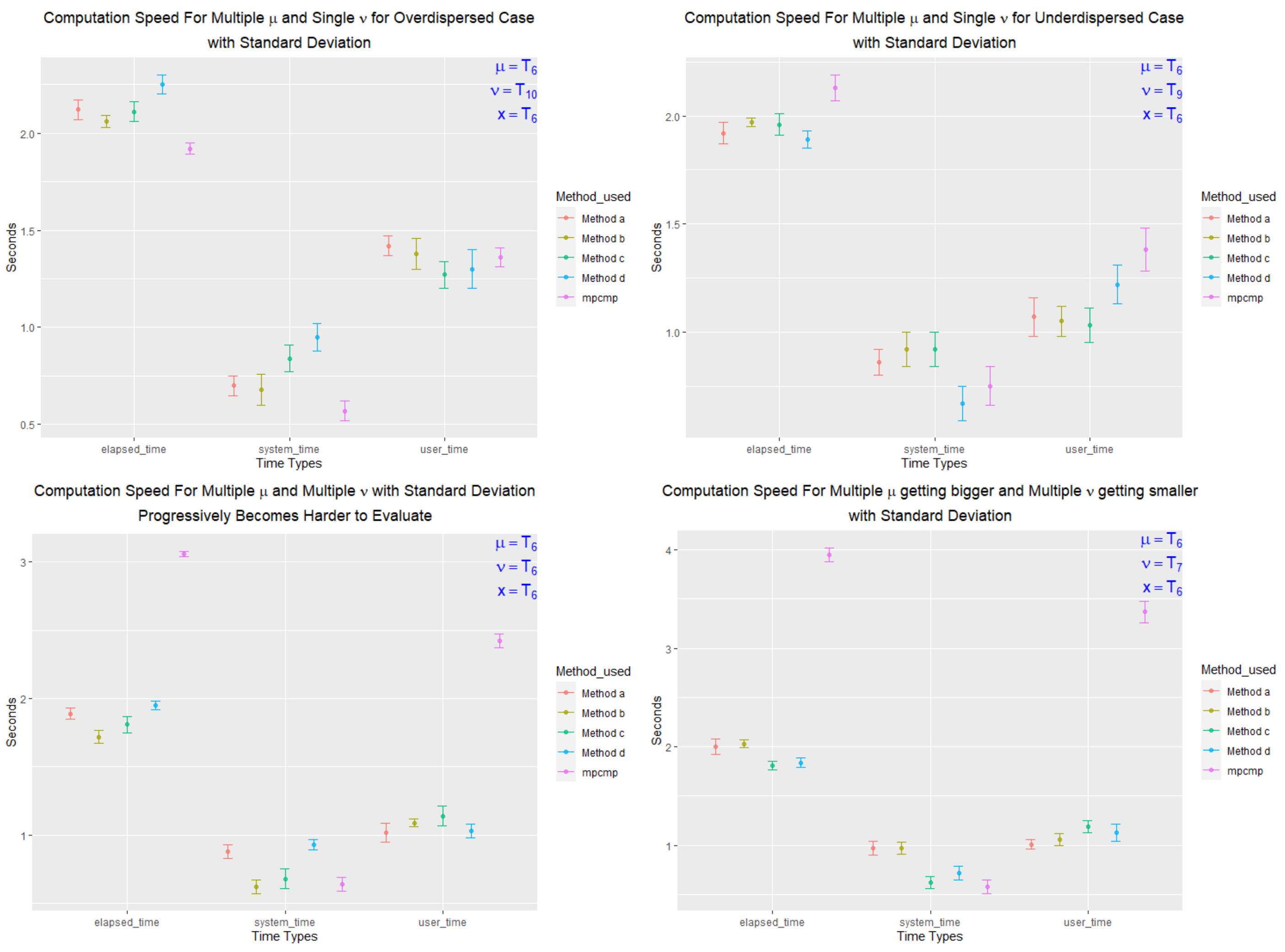

Figure 3.3: Computation speed pmf for different μ and ν values using 5 different methods

The pmf computation speed results carried forward from the computation speed of $\log(\lambda)$ and $\log(Z)$. For overdispersed case, functions in `mpcmp` can compete well with the other proposed methods, but it is not the case after trying different combinations for larger μ and ν. Method a) and Method c) in general would be slightly faster since both $\log(\lambda)$ and $\log(Z)$ are interpolating/extrapolating, but Method b) and Method d) would preserve the capability of having valid pmf as it normalises well using direct `mpcmp`. Since the computational speed comparison for $\log(\lambda)$ shows preference for Method 3, so overall **Method d) is preferred to have balance for computational speed and accuracy**. In Chapter 4, only Method d) and Method c) will be used to show the contrast with original `dcomp` function in `mpcmp`.

Overall, notice that the evaluation time for direct `mpcmp` is exponentially slower as μ and ν increase, whereas the time to do interpolation/extrapolation in the calculated grid is fixed. In terms of speed, the bilinear interpolation and direct `mpcmp` using relatively low values of μ and ν might have similar performance. Notwithstanding, a couple of seconds of efficiency in each line would save a lot of time when running on thousands or millions iterations. One can ensure the grid setup in Section 2.3 is fine enough to ”control” the relative error to be in certain threshold, and faster calculation speed using bilinear interpolation/extrapolation method is preferred.

Chapter 4

Applications

4.1 Bilinear Interpolation & Extrapolation on Bayesian Setting

Once more efficient computation using bilinear interpolation/extrapolation are established, this can be applied to different statistical properties, such as looking at the posterior density (likelihood at each support approximately). Subsequently, the main difficulty of root-finding problem with $\log(\lambda)$ has been solved, so the remaining task is to test whether computational improvement is feasible for different datasets. Given the value of $\log(\lambda)$, the other statistical properties (e.g. the normalizing constant (Z), expectation ($\mathbb{E}$), variance (Var)) are trivial in the sense it can be found under summation whose limit could be determined by `summax` variables as seen in Chapter 3.

Commonly used bayesian framework used for sampling to obtain posterior distribution would be Metropolis-Hastings (MH) Markov Chain Monte Carlo (MCMC) (also known as MH-MCMC). The full sampling procedures are shown in Algorithm 1. In a GLM framework, one could have a prior belief for each explanatory variable and dispersion parameter before looking at the data which is specified by some prior distributions. Then, fit the dataset using corresponding GLM framework (specifically `glm.cmp` in this case) whose parameter estimates $\hat{\beta}_{\text{MLE}}$ and $\hat{\nu}_{\text{MLE}}$ are used as starting value of means β_0 and its covariance matrices S_β, and dispersion ν_0. These will be used to update the belief, and the updated belief can be seen from the posterior distributions. Non-informative prior distribution chosen were log-Normal distribution with $(\mu, \sigma^2) = (0, 10^5)$ for ν, and $N(\mathbf{0}, 10^5\mathbf{I})$ for β. Then, the acceptance probability of β is given by the likelihood ratio as follows:

$$\alpha_\beta = \frac{p(\beta_{\mathbf{1}}, \nu_0 | \mathbf{y}, \mathbf{X})}{p(\beta_{\mathbf{0}}, \nu_0 | \mathbf{y}, \mathbf{X})},$$

where ν_0 is the current value of ν. Then, consider exponential sample proposals for ν with each mean is given by the current value ν_0, i.e., $\nu_1 \sim \text{Exp}(1/\nu_0)$[1]. Then, having β_0 as the current value of β and

[1] Note that here we can use different sample proposals, but exponential simple to work with because there is no secondary "variance"-type parameters to select from (dependent only one parameter)

β_1 is the drawn sample depending on β_0, the corresponding acceptance probability for ν is given by:

$$\alpha_\nu = \frac{p(\beta_0, \nu_1|\mathbf{y}, \mathbf{X})\nu_1}{p(\beta_0, \nu_0|\mathbf{y}, \mathbf{X})\nu_0} \exp\left(\frac{\nu_1}{\nu_0} - \frac{\nu_0}{\nu_1}\right)$$

Algorithm 1 MH-MCMC algorithm for sampling from posterior density p(β,ν |**y**,**X**)

```
Require: Inputs data $\mathbf{X}$ and $\mathbf{y}$
  Initialize $\beta$ and $\nu$ at some $(\beta_0, S_\beta)$ and $\nu_0$ from its MLE (i.e. the fitted values from regression)
  Cycle through for both $\beta$ and $\nu$:

  Draw a sample $\beta_1$ from $\beta_1 \sim N(\beta_0, S_\beta)$

  if probability is within $min(1, \alpha_\beta)$ then
      Accept $\beta_1$ and update $\beta_0 \leftarrow \beta_1$
  else
      reject $\beta_1$ and keep $\beta_0$
  end if

  Draw a sample $\nu_1$ from $\nu_1 \sim \exp(\frac{1}{\nu_0})$

  if probability is within $min(1, \alpha_\nu)$ then
      Accept $\nu_1$ and update $\nu_0 \leftarrow \nu_1$
  else
      reject $\nu_1$ and keep $\nu_0$
  end if
  until $K$ MCMC samples are generated
```

This framework would be used for subsequent Sections for some datasets that are Underdispersed (Section 4.2 and 4.3) and Overdispersed (Section 4.4 and 4.5). All simulations were carried out in R version 4.0.3 on desktop computer AMD Ryzen 7 3700X 8-Core Processor. Using original `dcomp` from `mpcmp` package with specification that density plots produced are thinned every 10 samples to reduce samples autocorrelation, and using $K = 10000$ MCMC samples to make sure chain (including the trace plot) is stable and is running long enough. In terms of running time, the standard deviation are obtained from running the above procedures 10 times to see its volatility.

4.2 Underdispersed Takeover Bids

The motivating example would be to look at the implementation in conditionally underdispersed ($\nu > 1$) takeover bids data (Huang & Kim, 2021). The dataset was looking at number of bids `numbids` as a response variable of 126 US firms that were successful targets of tender offers during 1978-1985 with 9 explanatory variables whose description are taken from Sáez-Castillo & Conde-Sánchez (2013):

- Defensive actions taken by management of target firm: indicator variable for legal defense by lawsuit (`leglrest`), proposed changes in asset structure (`rearest`), proposed change in ownership structure (`finrest`) and management invitation for friendly third-party bid (`whtknght`).

- Firm-specific characteristics:bid price divided by price 14 working days before bid (`bidprem`), percentage of stock held by institutions (`insthold`), total book value of assets in billions of dollars (`size`) and book value squared (`sizesq`).
- Intervention by federal regulators: indicator variable for Department of Justice intervention (`regulatn`).

Consider fitting this dataset with GLM provided using CMP$_\mu$ from `mpcmp` package with an argument: `glm.cmp(numbids ∼ leglrest + rearest + finrest + whtknght+ bidprem + insthold + size + sizesq + regulatn, data = takeoverbids)`. Preliminary results upon fitting GLM shows that $\nu = 1.75$. The posterior densities are displayed in Figure 4.1, which has resemblance to Figure 2 in Huang & Kim (2021). Using Method c) and Method d) from Section 3.1, the posterior densities produced for β and ν values give similar results compared to `dcomp` which are shown in Figure 4.2 and Figure 4.3 respectively.

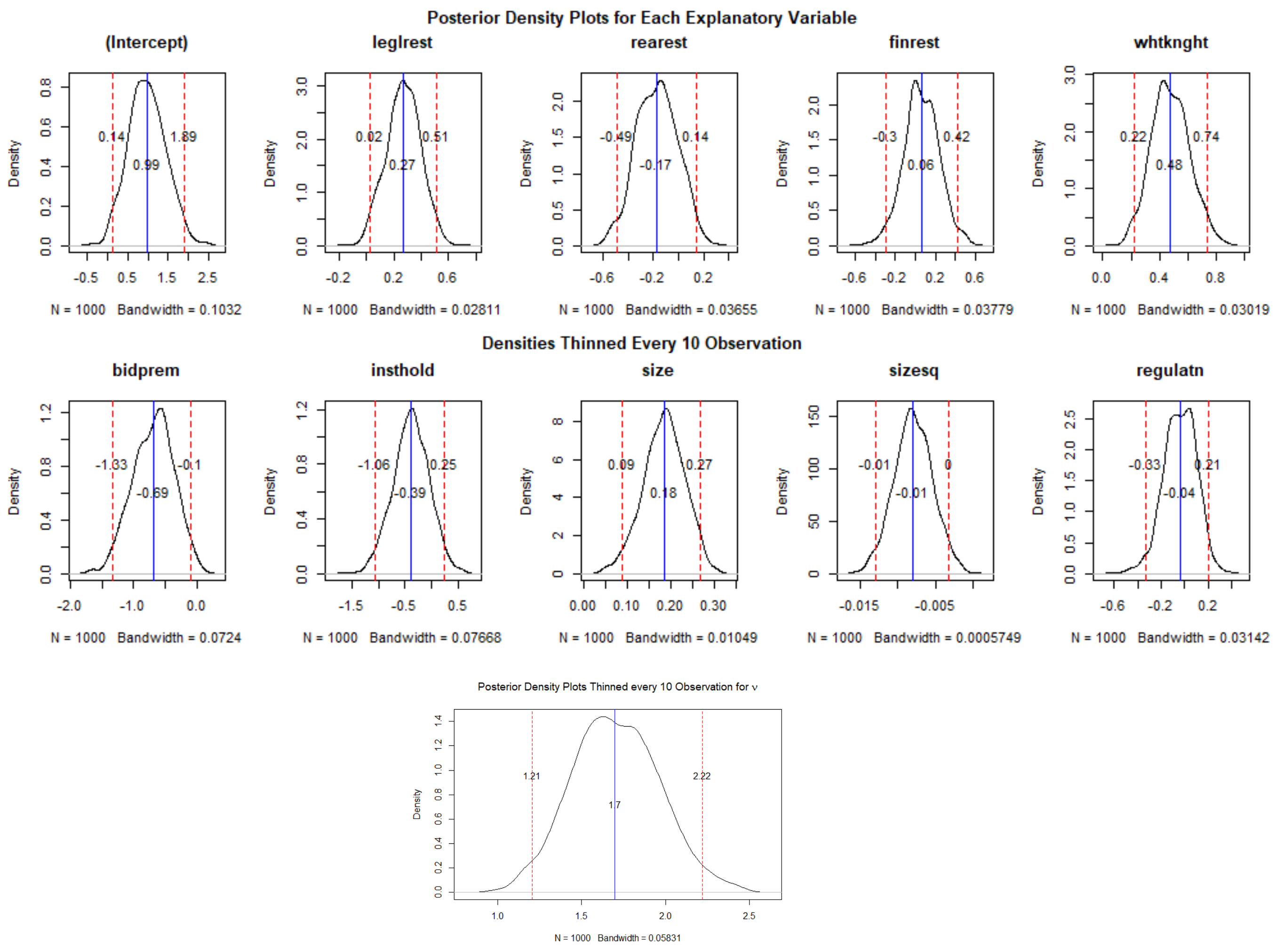

Figure 4.1: Posterior densities for β and ν with posterior means denoted in blue line, and 95% credible interval given in red-dotted line using `dcomp`

From Figure 4.1, the posterior distributions of β are relatively small and mostly centered around 0. Here, the 95% credible interval means that there are 95% probability that the true parameter estimate is between the red-dotted line given evidence provided by observed data. Also, the posterior distributions of ν does not contain 1, which shows that takeover bids example is severely underdispersed and unlikely to be a Poisson case. From Figure 4.2, the posterior distributions are mostly normalised using bilinear interpolation because β and ν are inside the grid setup in Section 2.3. The posterior distributions of β seems to be close enough compared to Figure 4.1 (with 2 or 1 decimal point(s) difference in the posterior mean and 95% credible interval compared to using `dcomp`), but the posterior mean for ν is 1.06 instead of 1.7 which are going to affect the model perception of the dispersion. Furthermore, its lower bound of 95% credible interval(=0.86) indicating overdispersed case instead. This was due to improper normalisation of $\log(Z)$, where pmf is not summing to one. This will disrupt the correctness of the likelihood evaluated and affect the acceptance probability. From Figure 4.3, the posterior distributions are normalised using built-in `mpcmp`, so the posterior distributions are closer to Figure 4.1 than those are estimated in Figure 4.2. More importantly, using Method d) preserves the claim that the model is still underdispersed in the posterior distribution of ν.

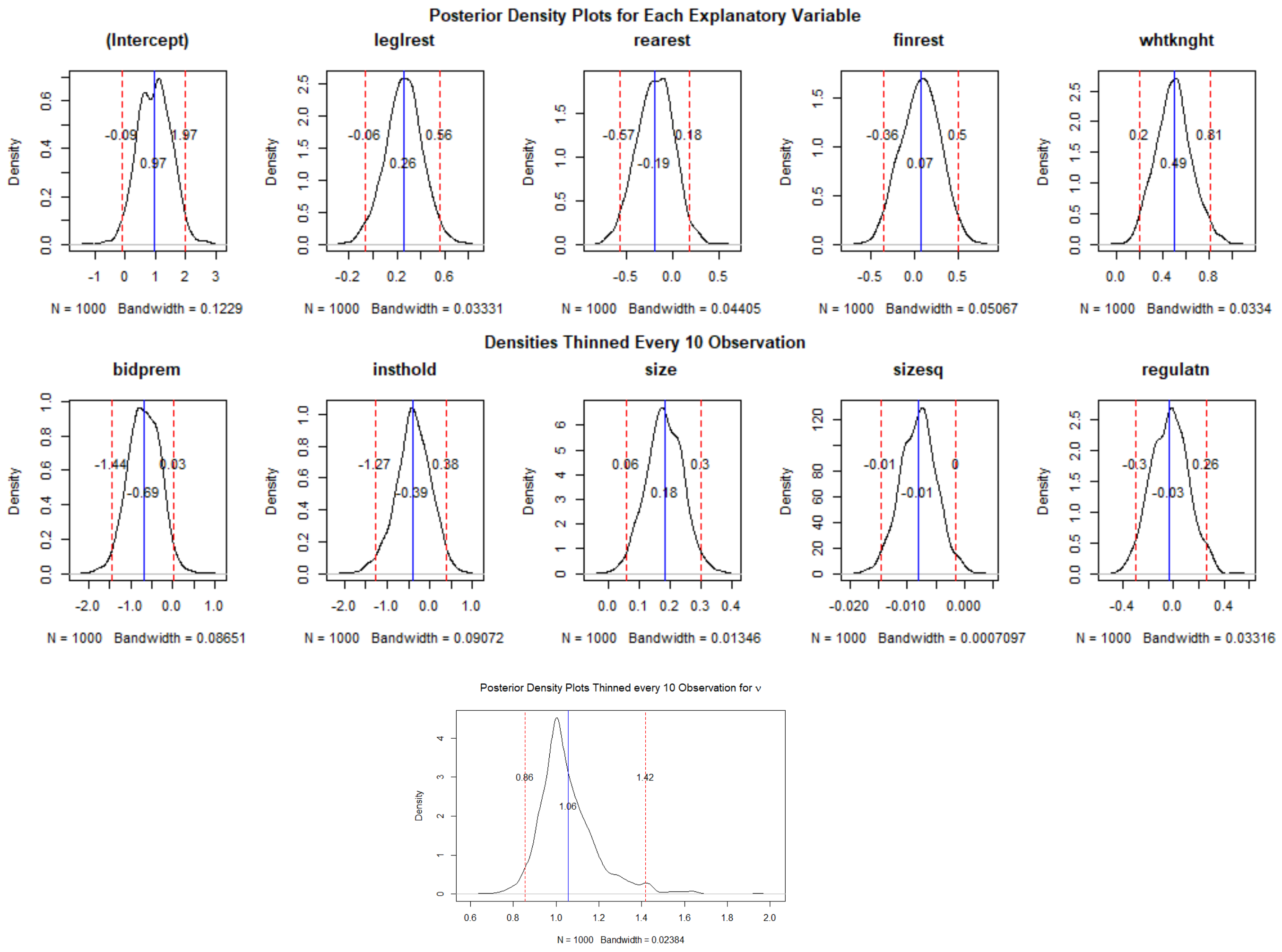

Figure 4.2: Posterior densities for β and ν with posterior means denoted in blue line, and 95% credible interval given in red-dotted line using Method c) from Section 3.1

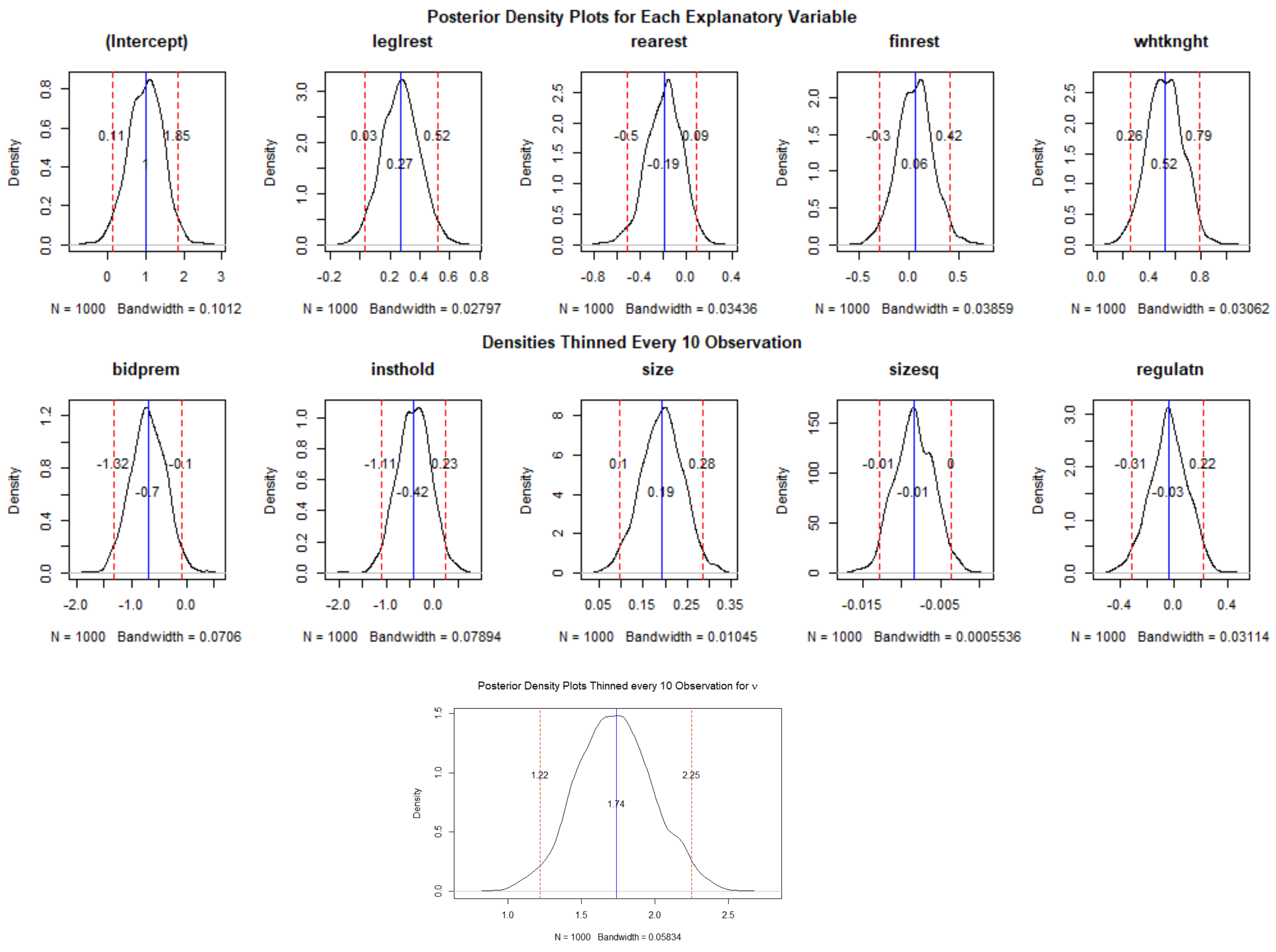

Figure 4.3: Posterior densities for β and ν with posterior means denoted in blue line, and 95% credible interval given in red-dotted line using Method d) from Section 3.1

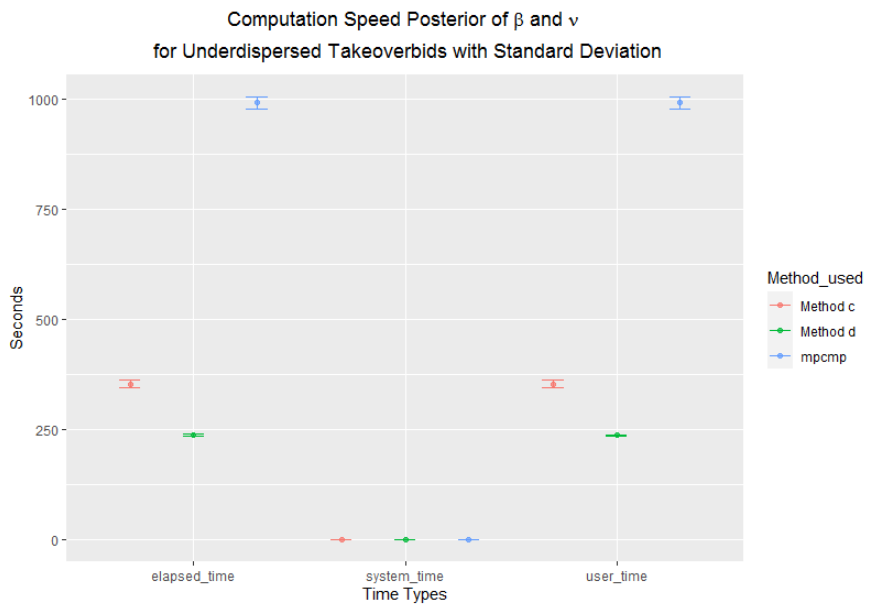

Figure 4.4: Computation speed for calculating Takeover Bids dataset β and ν using 3 different methods denoted in seconds and its standard deviation

In terms of computational speed, Method c) and Method d) **reduces computational time by 3 to 4 times faster** which are graphically represented in Figure 4.4 and numerically in Table A.4. Both Method c) and Method d) has been successful in creating more efficient calculation than the original `dcomp`, as shown in both user and elapsed time, especially with a preference of Method d) because of its stability (lower standard deviation than Method c)). Notably, the effect of not having a valid pmf using Method c) (due to not normalizing properly) affects the performance of the acceptance probability α_ν which can cause disruption whether or not to accept the sample proposals even though the posterior distributions of β are still reasonably close compared to `dcomp`. Also, this aligns with the results obtained in Table A.3 that Method d) would be most preferred to have balance in maintaining good accuracy while enhancing the computational speed significantly.

4.3 Underdispersed Cotton Bolls

Another underdispersed example would be a greenhouse experiment using 125 observations on cotton plants for examining the effects of 5 defoliation (`def`) levels (0%,25%,50%,75%,100%) applied at 5 growth stages (`stages`) (vegetative, flower-bud, blossom, fig and cotton boll), then observing the number of cotton bolls (`nc`) produced (Zeviani et al., 2014). Consider fitting this dataset with GLM provided in CMP$_\mu$ with argument: `glm.cmp(nc` $\sim$ `1 + stages:def + stages:def2, data = cottonbolls)`. Preliminary results upon fitting GLM shows that $\nu = 4.85$ (more underdispersed than takeover bids dataset). Here, it includes the interaction term and squared of the variable to increase model complexity, and see if the bilinear interpolation/extrapolation still could handle it well. In this case, the fitted parameters for β_{MLE} and ν_{MLE} are bigger than the takeover bids example.

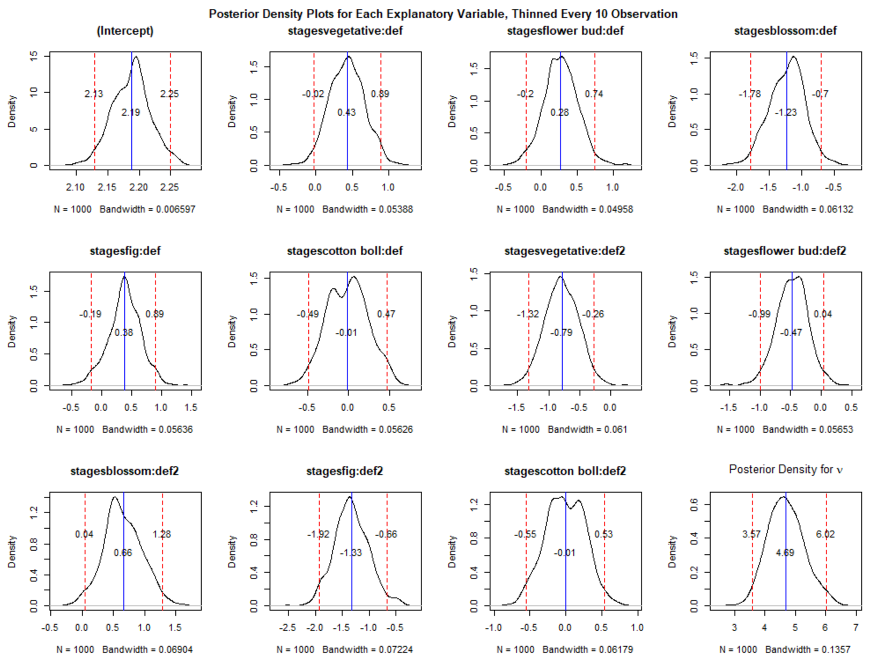

Figure 4.5: Posterior densities for β and ν with posterior means denoted in blue line, and 95% credible interval given in red-dotted line using `dcomp`

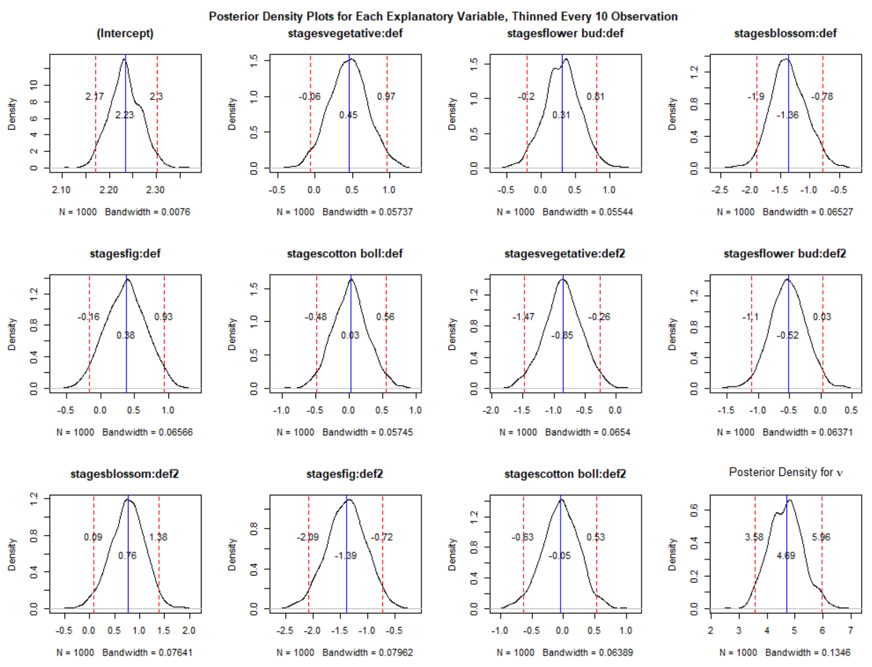

Figure 4.6: Posterior densities for β and ν with posterior means denoted in blue line, and 95% credible interval given in red-dotted line using Method d) from Section 3.1

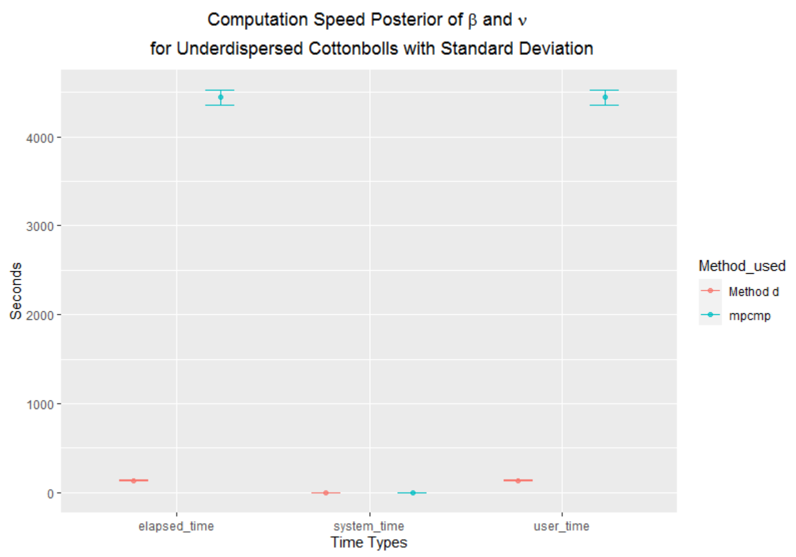

Figure 4.7: Computation speed for calculating Cottonbolls dataset β and ν using 2 different methods denoted in seconds and its standard deviation

The posterior distributions using `dcomp` are displayed in Figure 4.5. Unfortunately, there is a limitation on using Method c) for this dataset due to improper normalization when interpolating which makes the probability mass at each point is not correct as pmf does not sum to 1. This causes the algorithm to always accept the sample proposal ν_1. Once accepting the proposal, it will update the value of ν_0, which then be used to sample ν_1 for the subsequent iteration. However, when the updated value of ν_0 becomes too big, computation becomes intractable, i.e. ν_1 explode returning NA and cannot sample ν_1, so the algorithm diverges. The effect on not normalizing properly is more severe here than the takeover bids example.

Also, Method d) give similar posterior distributions for β and ν as shown in Figure 4.6. (i.e. posterior means only differs up to 2 decimal points, and some posterior means for β are identical). Subsequently, some interaction terms have two peaks using `dcomp` but become one peak when using Method d), or vice versa. This might indicate slight potential to be bimodal distribution (although it really is not, instead only due to sampling variability), as seen in posterior densities of `stagescotton boll:def` on Figure 4.5 and ν on Figure 4.6. However, as long as the posterior means and the coverage of 95% credible interval is roughly the same, it should not be a big issue. More importantly, the posterior mean for ν using `dcomp` and Method d) are the same, i.e. $\nu = \mathbf{4.69}$ indicating strong underdispersion. After looking at the numerical speed, it **reduces computational time by more than 1 order magnitude faster**, to be precise this can be **up to 30-35 times faster than the original** `mpcmp` using Method d) as shown graphically in Figure 4.7 and numerically in Table A.5. This proves the returns of the bilinear interpolation / extrapolation method are indeed more efficient in higher ν and Method d) provides the model parsimony and stability to maintain relatively high accuracy while improving computational time significantly which will be the highlight of this thesis.

4.4 Overdispersed Class Attendance

The class attendance dataset (UCLA, 2013) examines the relationship between the number of days absent from high school and the gender, maths score (standardized score out of 100) and academic program ("General", "Academic" and "Vocational") of 314 students sampled from two urban high schools. An insightful plot of the data, along with some summary statistics, can be found on the UCLA website. Consider fitting this dataset with GLM provided in CMP$_\mu$ using an argument: `glm.cmp(daysabs ~ gender + math + prog, data = attendance)`. Upon fitting the GLM, this indicates that the number of days absent exhibit strong overdispersion ($\nu = 0.020$).

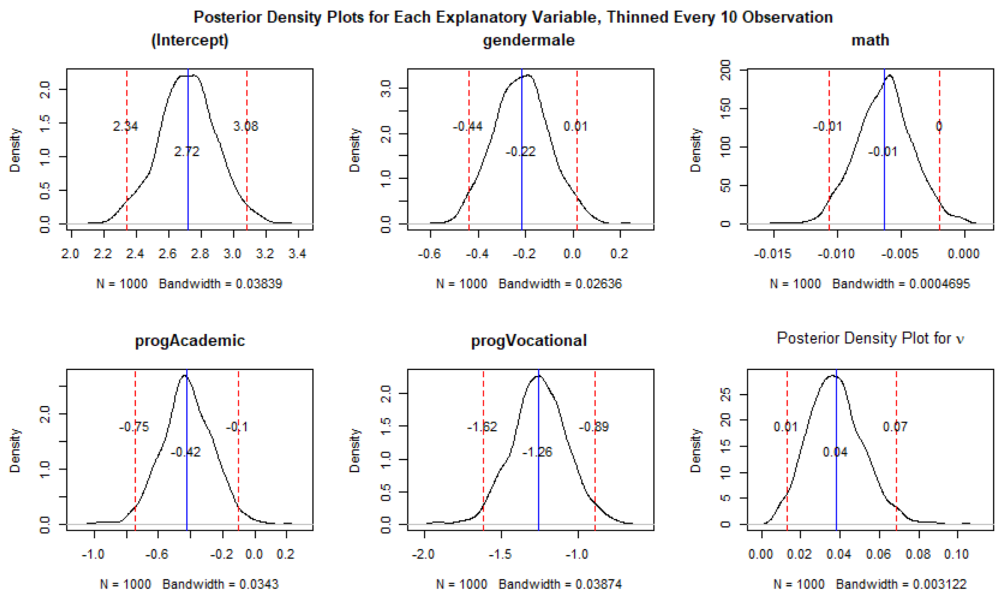

Figure 4.8: Posterior densities for β and ν with posterior means denoted in blue line, and 95% credible interval given in red-dotted line using `dcomp`

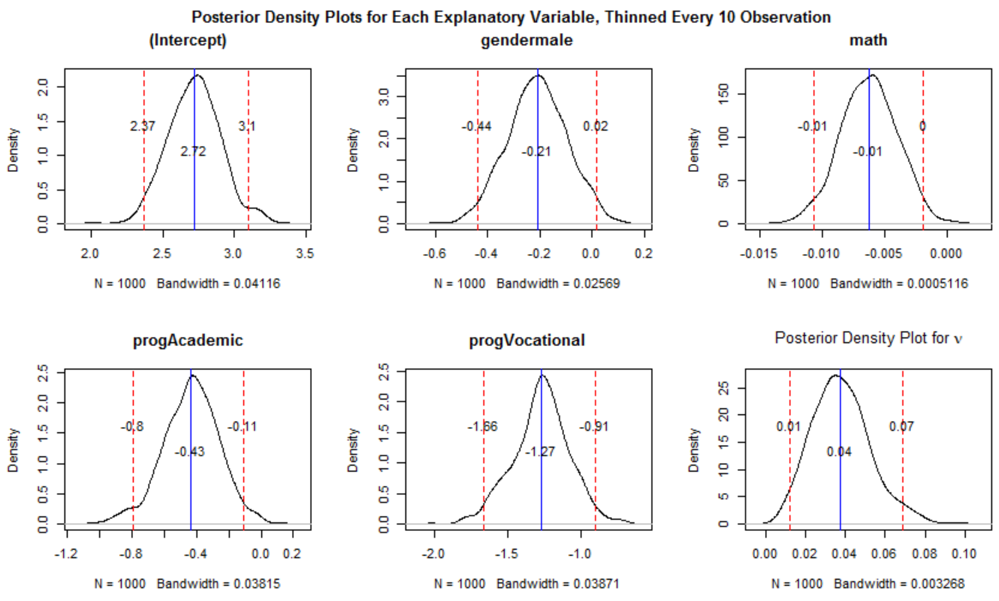

Figure 4.9: Posterior densities for β and ν with posterior means denoted in blue line, and 95% credible interval given in red-dotted line using Method c) from Section 3.1

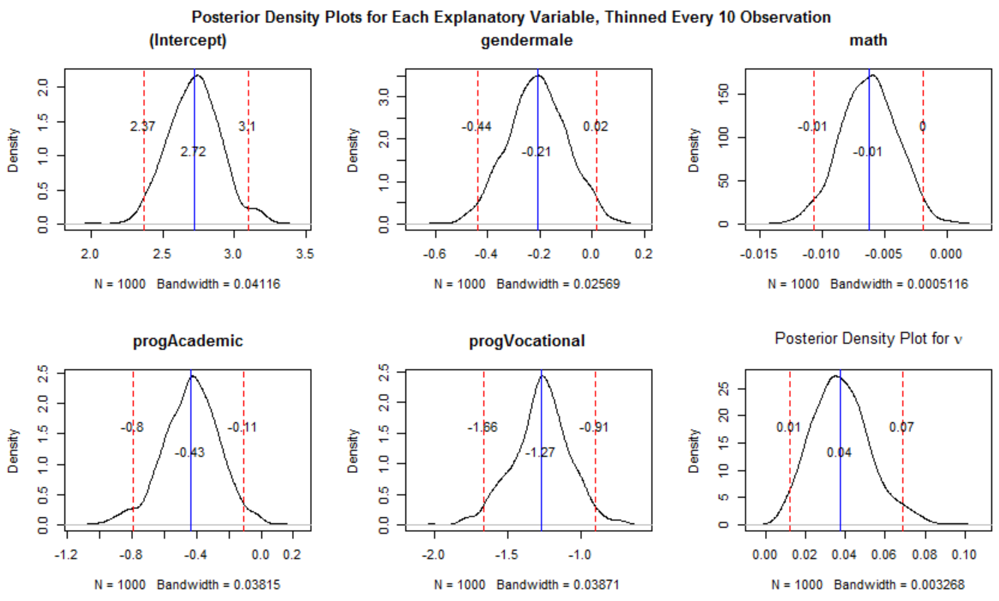

Figure 4.10: Posterior densities for β and ν with posterior means denoted in blue line, and 95% credible interval given in red-dotted line using Method d) from Section 3.1

The posterior distributions using `dcomp` are displayed in Figure 4.8. Method c) and Method d) also give similar posterior distribution plots for β and ν as shown in Figure 4.9 and 4.10 respectively. This result is expected because anytime ν values are less than the lower bound of the grid in Section 2.3, the pmf will always be evaluated using direct `mpcmp`[1] (in this case is true as the ν grid starts from 1 but posterior distributions shows ν values are smaller than 1) to evaluate both β and ν which are used for obtaining the pmf based on the decision rule in Table 2.1. Thus, the posterior distribution plots using all methods are similar and only varied due to randomness.

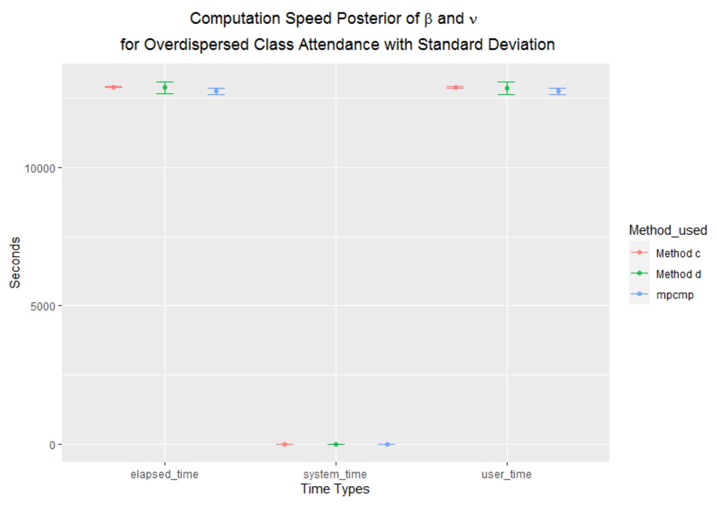

Figure 4.11: Computation speed for calculating Class Attendance dataset β and ν using 3 different methods denoted in seconds and its standard deviation

After running the MH-MCMC simulations, it shows that there is not much computational time improvement using Method c) or Method d) as graphically shown in Figure 4.11. Instead, `mpcmp` `dcomp` is the fastest on average. One possible explanation is Method c) and Method d), which use direct `mpcmp`[1] across all MCMC samples produced, have to evaluate the conditions whether to use bilinear interpolation/extrapolation or direct `mpcmp`[1] beforehand. However, since the time difference between `mpcmp` and (Method c) or Method d)) to obtain the posterior distributions are relatively small, any numerical discrepancies on calculation speed between these methods shown in Table A.6 are deemed to be a stochastic calculation error.

4.5 Overdispersed Fish Dataset

The fish dataset looks at 70 observations on 4 explanatory variables, the number of fish species (`species`) in lakes of the world with different surface area (in km squared) (`area`) and its latitude (`latitude`) (Barbour & Brown, 1974). Now, consider fitting the model with log of area and latitude, `glm.cmp(species ~ 1 + log(area) + latitude, data = fish)`. Preliminary results using GLM shows $\nu = 0.018$ only considering log(area), indicating severely overdispersed (more overdispersed than class attendance dataset), but adding latitude as a regressor explains more variability of the response variable giving $\nu = 0.027$ which will be used in the subsequent analysis.

The posterior distributions for β and ν using `dcomp` are displayed in Figure 4.12. Method c) and Method d) also give similar posterior distribution plots for β and ν as shown in Figure 4.13 and 4.14 respectively. Having similar argument to class attendance dataset, this was due to the decision rule that Method c) and Method d) evaluate both β and ν (which are used to evaluate posterior distributions) using direct `mpcmp`[1] since the ν values are less than ν grid (again, less than 1 in this case).

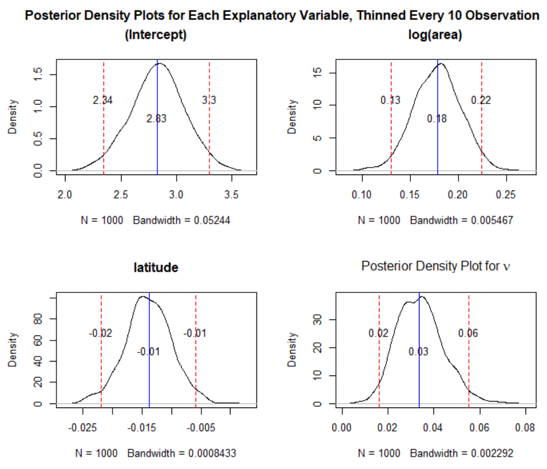

Figure 4.12: Posterior densities for β and ν with posterior means denoted in blue line, and 95% credible interval given in red-dotted line using `dcomp`

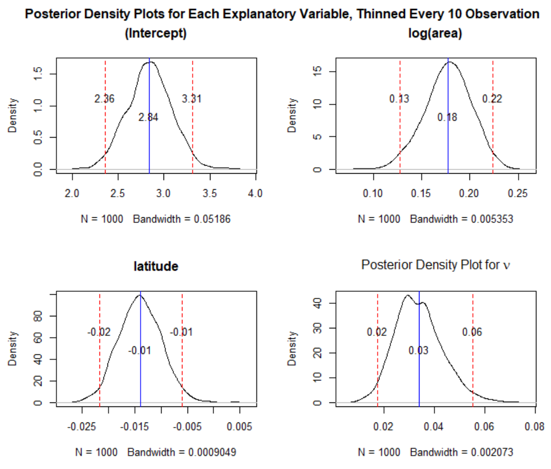

Figure 4.13: Posterior densities for β and ν with posterior means denoted in blue line, and 95% credible interval given in red-dotted line using Method c) from Section 3.1

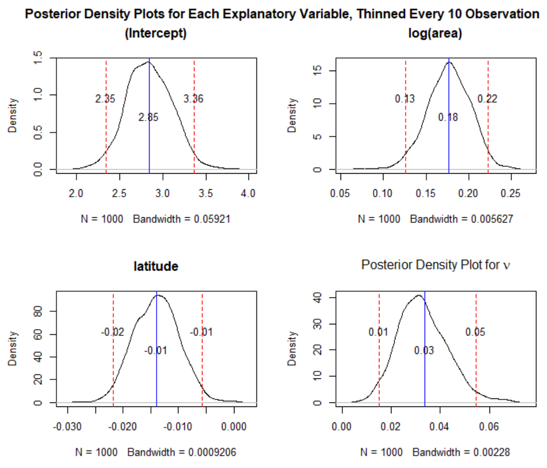

Figure 4.14: Posterior densities for β and ν with posterior means denoted in blue line, and 95% credible interval given in red-dotted line using Method d) from Section 3.1

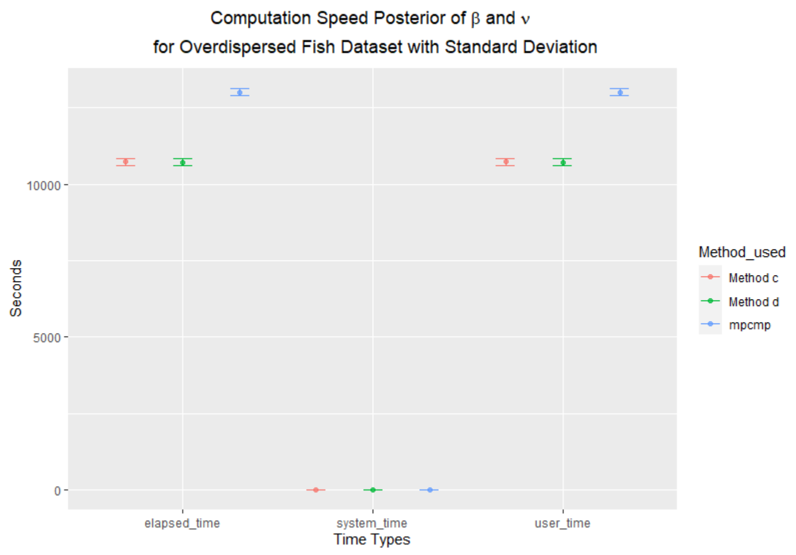

Figure 4.15: Computation speed for calculating Fish dataset β and ν using 3 different methods denoted in seconds and its standard deviation

In terms of computational speed, Method c) and Method d) are surprisingly faster than `mpcmp` as shown graphically in Figure 4.15 and numerically in Table A.7. Upon inspection, this is possibly due to the time to evaluate conditions in `dcomp` (see how `dcomp` are evaluated in (Fung et al., 2020)), which might be more expensive (due to many 'if' statements to ensure stability conditions for obtaining pmf are satisfied) than Method c) and Method d) even though both Method c) and Method d) still have to evaluate the conditions whether to use bilinear interpolation/extrapolation or direct `mpcmp`[1] beforehand. Thus, this still promotes the usage of Method c) and/or Method d) because both can obtain computational time improvement with similar model generating capabilities in overdispersed dataset due to simpler functions defined for obtaining $\log(\lambda)$ and/or $\log(Z)$ (recall direct `mpcmp`[1] are simplified version of the built-in functions in `mpcmp` are used).

Chapter 5

Further Research and Conclusion

5.1 Conclusion & Further Research

While the exploitation of the bilinear interpolation/extrapolation method could be optimized for computational purpose, its usefulness will depend on the extent to which the dataset is underdispersed. The more underdispersed the dataset is, the greater the computational gain for calculating the CMP_μ pmf. In terms of bayesian framework application (namely, MH-MCMC), our methods achieve up to 30-35 times computationally faster than time calculated using original `mpcmp`. For some overdispersed datasets, it is still possible to get computational gain (against the original `mpcmp`) because of the coding modification (taking the relevant subset of the original code) on evaluating the mean parameter μ and dispersion parameter ν, although the computational gain might not be as robust as in the underdispersed case. Moreover, our method might not work for other distributions as this result follows from both the theoretical justification that the solution of λ is bounded between μ^ν and $(\mu+1)^\nu$, and is supported by an empirical observations of the bilinearity behaviour in both $\log(\mu)$ and ν in $\log(\lambda)$ that enables bilinear interpolation or bilinear extrapolation in CMP_μ distribution.

For future research, one could test performance comparison with other approximation methods, including measuring the computation speed and accuracy using truncated Taylor expansions for different combinations of μ and ν. The idea of bilinear interpolation/ extrapolation can be applied to different bayesian frameworks and can be extended to a frequentist problem, such as applying iterative methods like the gradient descent or Newton-Raphson which utilizes the variance of the CMP_μ distribution which can also be pre-computed and interpolated/extrapolated.

5.2 Acknowledgement

The author thanks supervisor Alan Huang (UQ) and Ian Wood (UQ) for ideas, comments, guidance and consultation times during the project leading to the writing of the thesis and Jane Kosasih (RMIT) and 2 anonymous UQ friends for constructive comments that improved the flow of the thesis.

Appendix A

Appendix

A.1 Code and Tables

R codes for the projects are available at github page here

Supporting tables used of various computational time:

μ	ν	Method	user time(s)	system time(s)	elapsed time(s)
Multiple μ and Single ν					
$\mu = T_6$	$\nu = T_9$	Method 1	1.09 (0.05)	0.75 (0.05)	1.84 (0.04)
$\mu = T_6$	$\nu = T_9$	Method 2	1.08 (0.08)	0.80 (0.05)	1.87 (0.08)
$\mu = T_6$	$\nu = T_9$	Method 3	1.08 (0.08)	0.66 (0.06)	1.73 (0.04)
$\mu = T_6$	$\nu = T_9$	Method 4	1.12 (0.07)	0.82 (0.06)	1.95 (0.03)
$\mu = T_6$	$\nu = T_9$	`mpcmp`	1.70 (0.10)	0.60 (0.06)	2.31 (0.09)
Multiple μ and Multiple ν progressively becomes harder to evaluate					
$\mu = T_6$	$\nu = T_6$	Method 1	1.24 (0.05)	0.55 (0.05)	1.78 (0.03)
$\mu = T_6$	$\nu = T_6$	Method 2	1.15 (0.07)	0.61 (0.07)	1.77 (0.05)
$\mu = T_6$	$\nu = T_6$	Method 3	1.16 (0.08)	0.61 (0.07)	1.77 (0.02)
$\mu = T_6$	$\nu = T_6$	Method 4	1.11 (0.04)	0.66 (0.04)	1.76 (0.03)
$\mu = T_6$	$\nu = T_6$	`mpcmp`	5.19 (0.09)	0.57 (0.08)	5.75 (0.08)
Multiple μ getting bigger and Multiple ν getting smaller					
$\mu = T_6$	$\nu = T_7$	Method 1	1.38 (0.09)	0.47 (0.06)	1.83 (0.06)
$\mu = T_6$	$\nu = T_7$	Method 2	1.36 (0.07)	0.47 (0.06)	1.83 (0.06)
$\mu = T_6$	$\nu = T_7$	Method 3	1.14 (0.06)	0.63 (0.06)	1.76 (0.03)
$\mu = T_6$	$\nu = T_7$	Method 4	1.10 (0.07)	0.61 (0.05)	1.71 (0.04)
$\mu = T_6$	$\nu = T_7$	`mpcmp`	4.75 (0.09)	0.52 (0.08)	5.27 (0.05)

Table A.1: Computation speed $\log(\lambda)$ for different μ and ν values using 5 different methods

μ	ν	Method	user time(s)	system time(s)	elapsed time(s)
Multiple μ and Single ν					
$\mu = T_6$	$\nu = T_9$	Method 1	1.13 (0.11)	0.76 (0.05)	1.89 (0.09)
$\mu = T_6$	$\nu = T_9$	Method 2	1.12 (0.08)	0.67 (0.07)	1.80 (0.06)
$\mu = T_6$	$\nu = T_9$	Method 3	1.09 (0.07)	0.77 (0.08)	1.86 (0.03)
$\mu = T_6$	$\nu = T_9$	Method 4	0.97 (0.10)	0.98 (0.08)	1.95 (0.04)
$\mu = T_6$	$\nu = T_9$	`mpcmp`	1.03 (0.08)	0.92 (0.07)	1.95 (0.05)
Multiple μ and Multiple ν progressively becomes harder to evaluate					
$\mu = T_6$	$\nu = T_6$	Method 1	1.25 (0.08)	0.55 (0.05)	1.80 (0.07)
$\mu = T_6$	$\nu = T_6$	Method 2	1.17 (0.09)	0.57 (0.08)	1.74 (0.02)
$\mu = T_6$	$\nu = T_6$	Method 3	1.12 (0.07)	0.66 (0.05)	1.77 (0.03)
$\mu = T_6$	$\nu = T_6$	Method 4	1.06 (0.09)	0.73 (0.05)	1.79 (0.08)
$\mu = T_6$	$\nu = T_6$	`mpcmp`	1.16 (0.08)	0.63 (0.08)	1.79 (0.08)
Multiple μ getting bigger and Multiple ν getting smaller					
$\mu = T_6$	$\nu = T_7$	Method 1	1.25 (0.10)	0.53 (0.06)	1.78 (0.04)
$\mu = T_6$	$\nu = T_7$	Method 2	1.11 (0.06)	0.65 (0.06)	1.76 (0.05)
$\mu = T_6$	$\nu = T_7$	Method 3	1.01 (0.05)	0.78 (0.08)	1.80 (0.05)
$\mu = T_6$	$\nu = T_7$	Method 4	1.19 (0.04)	0.59 (0.03)	1.79 (0.03)
$\mu = T_6$	$\nu = T_7$	`mpcmp`	1.19 (0.06)	0.61 (0.02)	1.79 (0.05)

Table A.2: Computation speed $\log(Z)$ for different μ and ν values using 5 different methods

μ	ν	x	Method	user time(s)	system time(s)	elapsed time(s)
Multiple μ and Single ν for Overdispersed Case						
$\mu = T_6$	$\nu = T_{10}$	$x = T_6$	Method a	1.42 (0.05)	0.70 (0.05)	2.12 (0.05)
$\mu = T_6$	$\nu = T_{10}$	$x = T_6$	Method b	1.38 (0.08)	0.68 (0.08)	2.06 (0.03)
$\mu = T_6$	$\nu = T_{10}$	$x = T_6$	Method c	1.27 (0.07)	0.84 (0.07)	2.11 (0.05)
$\mu = T_6$	$\nu = T_{10}$	$x = T_6$	Method d	1.30 (0.10)	0.95 (0.07)	2.25 (0.05)
$\mu = T_6$	$\nu = T_{10}$	$x = T_6$	`mpcmp`	1.36 (0.05)	0.57 (0.05)	1.92 (0.03)
Multiple μ and Single ν for Underdispersed Case						
$\mu = T_6$	$\nu = T_9$	$x = T_6$	Method a	1.07 (0.09)	0.86 (0.06)	1.92 (0.05)
$\mu = T_6$	$\nu = T_9$	$x = T_6$	Method b	1.05 (0.07)	0.92 (0.08)	1.97 (0.02)
$\mu = T_6$	$\nu = T_9$	$x = T_6$	Method c	1.03 (0.08)	0.92 (0.08)	1.96 (0.05)
$\mu = T_6$	$\nu = T_9$	$x = T_6$	Method d	1.22 (0.09)	0.67 (0.08)	1.89 (0.04)
$\mu = T_6$	$\nu = T_9$	$x = T_6$	`mpcmp`	1.38 (0.10)	0.75 (0.09)	2.13 (0.06)
Multiple μ and Multiple ν progressively becomes harder to evaluate						
$\mu = T_6$	$\nu = T_6$	$x = T_6$	Method a	1.02 (0.07)	0.88 (0.05)	1.89 (0.04)
$\mu = T_6$	$\nu = T_6$	$x = T_6$	Method b	1.09 (0.03)	0.62 (0.05)	1.72 (0.05)
$\mu = T_6$	$\nu = T_6$	$x = T_6$	Method c	1.14 (0.07)	0.68 (0.07)	1.81 (0.06)
$\mu = T_6$	$\nu = T_6$	$x = T_6$	Method d	1.03 (0.05)	0.93 (0.04)	1.95 (0.03)
$\mu = T_6$	$\nu = T_6$	$x = T_6$	`mpcmp`	2.42 (0.05)	0.64 (0.05)	3.06 (0.02)
Multiple μ getting bigger and Multiple ν getting smaller						
$\mu = T_6$	$\nu = T_7$	$x = T_6$	Method a	1.01 (0.05)	0.97 (0.07)	2.00 (0.08)
$\mu = T_6$	$\nu = T_7$	$x = T_6$	Method b	1.06 (0.06)	0.97 (0.06)	2.03 (0.04)
$\mu = T_6$	$\nu = T_7$	$x = T_6$	Method c	1.19 (0.06)	0.62 (0.06)	1.81 (0.04)
$\mu = T_6$	$\nu = T_7$	$x = T_6$	Method d	1.13 (0.09)	0.72 (0.07)	1.84 (0.05)
$\mu = T_6$	$\nu = T_7$	$x = T_6$	`mpcmp`	3.37 (0.11)	0.58 (0.07)	3.95 (0.07)

Table A.3: Computation speed pmf for different μ and ν values using 5 different methods

Method	user time(s)	system time(s)	elapsed time(s)
Method c	352.67(8.59)	0.56(0.09)	353.25(8.59)
Method d	236.85(1.67)	0.51(0.09)	237.35(1.68)
`mpcmp dcomp`	992.12(13.73)	0.08 (0.15)	992.27(13.70)

Table A.4: Computation speed for calculating Takeover bids dataset β and ν using 3 different methods denoted in seconds and standard deviation in brackets

Method	user time(s)	system time(s)	elapsed time(s)
Method c	not converge ()	not converge ()	not converge ()
Method d	136.78(2.86)	0.01(0.01)	136.8(2.84)
`mpcmp dcomp`	4440.81(85.55)	0.346(0.18)	4441.29(85.70)

Table A.5: Computation speed for Cottonbolls dataset calculating β and ν using 3 different methods denoted in seconds and standard deviation in brackets

Method	user time(s)	system time(s)	elapsed time(s)
Method c	12903.36(31.77)	3.85(0.45)	12908.29(31.88)
Method d	12859.89(217.99)	2.66(0.44)	12881.08(211.56)
`mpcmp dcomp`	12752.53(116.38)	0.14 (2.36)	12753.16(117.77)

Table A.6: Computation speed for Class Attendance dataset calculating β and ν using 3 different methods denoted in seconds and standard deviation in brackets

Method	user time(s)	system time(s)	elapsed time(s)
Method c	10723.05(120.97)	1.81 (0.85)	10725.24 (121.37)
Method d	10716.73 (124.48)	1.26 (1.23)	10718.92 (124.79)
`mpcmp dcomp`	13006.37 (118.44)	6.11 (4.83)	13013.54 (122.17)

Table A.7: Computation speed for Fish dataset calculating β and ν using 3 different methods denoted in seconds and standard deviation in brackets

Bibliography

AREL-BUNDOCK, V. (2019). https://vincentarelbundock.github.io/Rdatasets/datasets.html

BARBOUR, C. D. & BROWN, J. H. (1974). Fish species diversity in lakes. *The American Naturalist* 108, 473–488.

CONWAY, R. W. & MAXWELL, W. L. (1962). A queuing model with state dependent service rates. *Journal of Industrial Engineering 1962.* 12, 132-–136.

DIAMOND H.G. & STRAUB A. (2016). *Bounds for the logarithm of the Euler gamma function and its derivatives.* *Journal of Mathematical Analysis and Applications* https://doi.org/10.1016/j.jmaa.2015.08.034

FUNG, T., ALWAN, A., WISHART, J. & HUANG, A. (2020). `mpcmp`: Mean-Parametrized Conway-Maxwell Poisson (COM-Poisson) Regression. *R package version 0.3.5.* https://CRAN.R-project.org/package=mpcmp.

HUANG, A. (2017). Mean-parametrized Conway-Maxwell-Poisson regression models for dispersed counts. *Statistical Modelling.* **17**, 359–380.

HUANG, A. (2020). On arbitrarily underdispersed Conway-Maxwell-Poisson distributions. *arXiv: Statistics Theory.*

HUANG, A. & KIM, A. S. I. (2021). Bayesian Conway–Maxwell–Poisson regression models for overdispersed and underdispersed counts. *Communications in Statistics - Theory and Methods.* **50:13**, 3094–3105. https://doi.org/10.1080/03610926.2019.1682162

SÁEZ-CASTILLO, A. J. & CONDE-SÁNCHEZ, A. (2013). A hyper-Poisson regression model for overdispersed and underdispersed count data. *Computational Statistics and Data Analysis.* **61**, 148–157.

SELLERS, K. F. & SHMUELI, G. (2010). A flexible regression model for count data. *Annals of Applied Statistics.* **4**, 943–961.

SELLERS, K. F. & SHMUELI, G. (2013). Data dispersion: now you see it . . . now you dont. *Communications in Statistics – Theory and Methods.* **42**, 3134–3147.

SHMUELI, G., MINKA, T. P., KADANE, J. B., BORLE, S. & BOATWRIGHT, P. (2005). A useful distribution for fitting discrete data: revival of the Conway–Maxwell-Poisson distribution. *Appl. Statist.* **54**, 127–142.

TUKEY, J. W. (1977). warpbreaks: The Number of Breaks in Yarn during Weaving. *R dataset version 3.6.2*. https://www.rdocumentation.org/packages/datasets/versions/3.6.2/topics/warpbreaks

UCLA (2013). Negative Binomial Regression — R Data Analysis Examples, http://www.ats.ucla.edu/stat/stata/dae/nb_data.dta

ZEVIANI, W. M., RIBERIO JR, P. J., BONAT, W. H., SHIMAKURA, S. E., AND MUNIZ, J. A. (2014). The Gamma-count distribution in the analysis of experimental underdispersed data. *Journal of Applied Statistics.* **41**, 2616–2626.