

# The Poisson Multinomial Distribution and Its Applications in Voting Theory, Ecological Inference, and Machine Learning

Zhengzhi Lin, Yueyao Wang, and Yili Hong

Department of Statistics, Virginia Tech, Blacksburg, VA 24061

September 27, 2021

## Abstract

The Poisson Multinomial Distribution (PMD) is the sum of  $n$  independent indicators, in which each indicator is an  $m$  element vector that follows a multinomial distribution. The PMD is useful in many areas such as, machine learning, uncertainty quantification, and voting theory. The distribution function (e.g., the probability mass function) has been studied by many authors for a long time, but there is no general computing method available for its distribution functions. In this paper, we develop algorithms to compute the probability mass function for the PMD, and we develop an R package that can calculate the probability mass function efficiently. We also study the accuracy of different methods. We illustrate the use of the PMD with three applications from voting theory, ecological inference, machine learning. We also provide examples to demonstrate the use of the R package.

**Key Words:** Binomial distribution; Classification; Poisson Binomial Distribution; Machine Learning; Uncertainty Quantification; Voting Theory

# 1 Introduction

## 1.1 Motivation

The Poisson Multinomial Distribution (PMD) is defined as sum of different independent Multinomial distributions. It has applications in game theory (?), digital imaging (?), machine learning (?), etc. One intriguing area that involves PMD is political science. It would be good to illustrate PMD a little bit further via a very simple example in election scenario. Suppose a committee with  $n$  independent members needs to elect a chairman between  $m$  candidates. Each member has a different voting behavior. Mostly, people pay attention to the result of an election. Statisticians focus on the probability for each electoral outcome to happen. An  $(n, m)$  PMD is a perfect model for this example by seeing members as independent random variables following different Multinomial distributions. The probability vector of the Multinomial distribution regards to the probabilities for a member voting for each candidates. By this way, we will be allowed to compute the probability of each electoral result which is referred as probability mass function of PMD. Without a doubt, computing the probabilities of electoral results will be of greatest concern. Enumeration could be a fine method when  $n \times m$  is small. However, when we have more members (large  $n$ ) or more candidates (large  $m$ ), the computing will be a dilemma that is impossible to overcome by enumeration. Thus arises the need of methods that are computationally cheap and precise at the meantime. Additionally, in machine learning, especially in classification context, when we classify  $n$  samples one by one into  $m$  categories, then the total number of samples assigned to each category follows a PMD. People who study the probability of classification outcomes also encounters same dilemma. Not only in the field of classification and political science, computing pmf is frequently required in scenarios that PMDs are involved. However, there is no existing algorithm for computing pmf for PMD. Hereby it is a necessity to build one that can calculate pmf of PMDs efficiently.

## 1.2 Related Literature and Contribution of This Work

Some former studies have uncover PMD's structure and properties, (?) shows the Fourier transformation of PMD is sparse and provide a theorem that there exist algorithms to calculate PMD's density. (?) also prove us PMD is  $\epsilon$ -cover and Central Limit Theory is valid for PMD. Other papers such as (?) illustrates us some interesting applications of PMD and its sparsity property. Motivated by the huge practical value of PMD, we develop an algorithm based on prior studies to compute probability mass function of PMDs. Our algorithm contains three methods. They are DFT-CF which is based on multi-dimensional Fourier transformation, SIM that is a simulation method and NA which applies C.L.T to approximate the PMD. The algorithm enables researchers to calculate pmf of PMDs under certain requirements of

accuracy and timing even when  $n$  and  $m$  gets large. We also construct a R package to perform our algorithm, the functions contained in the package are able to compute pmf and cdf of PMD using all methods we mentioned in this paper as well as generate random samples from given PMD.

### 1.3 Overview

The rest of the paper is organized as follows. In the second part, we describe the mathematical definition of Poisson Multinomial distribution and list some of its useful properties. In the third part, we introduce three methods that we include in the algorithm to compute PMDs' pmf by details. The fourth part of this article studies the accuracy and time efficiency characteristics of the three methods. In the fifth part, we illustrate application of our algorithm via three examples respectively in the fields of voting theory, statistical inference for aggregated data and classification. In the sixth part, we introduce our R package that implemented with the algorithm and the seventh part of this paper draws the conclusion and prospective research areas.

## 2 Poisson Multinomial Distribution

### 2.1 Definition of the Distribution

Let  $\mathbf{I}_i = (I_{i1}, \dots, I_{im})', i = 1, \dots, n$  be a random indicator vector that follows multinomial distribution with associated probabilities  $\mathbf{p}_i = (p_{i1}, \dots, p_{im})'$  and for a fixed  $i$ ,  $\sum_{j=1}^m I_{ij} = 1$ . Then the sum  $\mathbf{X} = (X_1, \dots, X_m)' = \sum_{i=1}^n \mathbf{I}_i$  follows a Poisson Multinomial Distribution with corresponding probability matrix  $\mathbf{P}_{n \times m} = (\mathbf{p}_1, \dots, \mathbf{p}_n)'$ , denoted it as

$$\mathbf{X} \sim \text{PMD}(\mathbf{P}_{n \times m}).$$

where  $X_j = \sum_{i=1}^n I_{ij}, j = 1, \dots, m$ . The matrix  $\mathbf{P}$  is called Success Probability Matrix (SPM).

$$\mathbf{P}_{n \times m} = \begin{pmatrix} p_{11} & \cdots & p_{1m} \\ \vdots & \ddots & \vdots \\ p_{n1} & \cdots & p_{nm} \end{pmatrix}.$$

It is trivial to find that the random variables,  $X_1, \dots, X_m$  actually have to be constrained under linear equation

$$\sum_{j=1}^m X_j = n$$

. Hence we can replace one of them, for instance,  $X_m$  with  $n - \sum_{j=1}^{m-1} X_j$ .

Let vector  $\mathbf{x} = (x_1, \dots, x_m)'$  be a realization of a PMD random variable  $\mathbf{X}$ , the probability mass function (pmf) of PMD,

$$\Pr(\mathbf{X} = \mathbf{x}) = \Pr\left(X_1 = x_1, \dots, X_m = x_{m-1}, X_m = n - \sum_{i=1}^m x_i\right)$$

is of interest.

**Example 1** Suppose there are three candidates and four voters, the result of voting is a random variable  $\mathbf{X} \sim \text{PMD}(\mathbf{P})$ . Based on prior information,

$$\mathbf{P}_{4 \times 3} = \begin{pmatrix} 0.1 & 0.2 & 0.7 \\ 0.5 & 0.2 & 0.3 \\ 0.4 & 0.5 & 0.1 \\ 0.8 & 0.1 & 0.1 \end{pmatrix}.$$

There are 15 distinct outcomes of the election, that is, the pmf is composed of 15 distinct non-zero mass points. It is trivial to compute the pmf of  $\mathbf{X}$  through enumeration. For instance, the probability of a result that the first candidate gets 4 votes and others gets 0 vote, that is,  $\mathbf{x} = (4, 0, 0)'$ .

$$P\{\mathbf{X} = \mathbf{x}\} = 0.1 \times 0.5 \times 0.4 \times 0.8 = 0.016.$$

Also, the probability of  $\mathbf{X} = (1, 3, 0)'$  is

$$\begin{aligned} P\{\mathbf{X} = (1, 3, 0)\} &= 0.1 \times 0.2 \times 0.5 \times 0.1 + 0.5 \times 0.2 \times 0.5 \times 0.1 \\ &+ 0.4 \times 0.2 \times 0.2 \times 0.1 + 0.8 \times 0.2 \times 0.2 \times 0.5 = 0.0236. \end{aligned}$$

□

When dimension of  $\mathbf{P}$  is small, enumeration is an exact way to calculate the probability mass function. However, as  $n \times m$  gets larger enumeration becomes impossible since we will have to compute  $\binom{n+m-1}{m-1}$  possible outcomes which is calculated by the number of non-negative integer solution for equation  $x_1 + \dots + x_m = n$ .

Note that when the SPM is identical across all rows, that is,  $I_i, i = 1, \dots, n$  are identically distributed, the distribution of  $X$  can be simplified as multinomial distribution. Hence, the PMD is a generalization of the multinomial distribution. When  $m = 1$ , the PMD is reduced to the Poisson binomial distribution as in ?).

In related literature, ?) consider the exact and approximate methods for computing the pmf of the Poisson binomial distribution. ?) introduce the general Poisson binomial distribution and develop an algorithm to compute its distribution functions.

Now let's take a look of some basic properties of Poisson Multinomial Distributions.

## 2.2 Properties of the Distribution

**Property 1** Given random variable  $\mathbf{X}$  that follows a Poisson Multinomial distribution with  $\mathbf{P}_{n \times m}$ , the mean of  $\mathbf{X}$  is

$$E(\mathbf{X}) = \boldsymbol{\mu} = (p_{\cdot 1}, \dots, p_{\cdot, m-1}, p_{\cdot m})'$$

where  $p_{\cdot k} = \sum_{i=1}^n p_{ik}$ .

The variance-covariance matrix of  $\mathbf{X}$  is an  $m \times m$  matrix  $\boldsymbol{\Sigma}$  that has entries  $\Sigma_{ij}$ ,  $i = 1, \dots, m$ ,  $j = 1, \dots, m$  defined as

$$\Sigma_{ij} = \begin{cases} \sum_{k=1}^n p_{ki}(1 - p_{ki}) & \text{if } i = j \\ -\sum_{k=1}^n p_{ki}p_{kj} & \text{if } i \neq j \end{cases}$$

The CF for the PMD is

$$\phi_{\mathbf{X}}(t_1, \dots, t_{m-1}) = \phi_{(X_1, \dots, X_m)'}(t_1, \dots, t_{m-1}) = \sum_{x_1=0}^n \cdots \sum_{x_{m-1}=0}^n p(x_1, \dots, x_{m-1}) \exp \left( \mathbf{i} \sum_{j=1}^{m-1} t_j x_j \right).$$

where  $\mathbf{i} = \sqrt{-1}$ . □

The derivations of mean and characteristic function are as trivial as following the definitions. The  $\boldsymbol{\Sigma}$  can be calculated by observing that for any fix  $i = 1, \dots, n$ ,  $I_{ij}$  and  $I_{ik}$  has covariance  $-p_{ij}p_{ik}$ ,  $j = 1, \dots, m$ ,  $k = 1, \dots, m$ . One important thing is that the covariance matrix  $\boldsymbol{\Sigma}$  is singular because the elements of  $\mathbf{X}$  are linear dependent.

Let  $\mathbf{X}^* = (X_1, \dots, X_{m-1})'$  with corresponding  $\mathbf{P}^*$  equals to the first  $m - 1$  columns of  $\mathbf{P}$ , then it has a non-singular  $(m - 1) \times (m - 1)$  covariance matrix  $\boldsymbol{\Sigma}_0$ . Similar to  $\boldsymbol{\Sigma}$ , the covariance matrix

$$\text{Var}(\mathbf{X}^*) = \boldsymbol{\Sigma}^* = \sum_{i=1}^n [\text{Diag}(\mathbf{p}_i) - \mathbf{p}_i \mathbf{p}_i']$$

where  $\mathbf{p}_i$  is the  $i$ th row of  $\mathbf{P}^*$ . Also,

$$\mathbb{E}(\mathbf{X}^*) = \boldsymbol{\mu}^* = (p_{\cdot 1}, \dots, p_{\cdot, m-1})'$$

We call  $\mathbf{X}^*$  a reduced PMD of  $\mathbf{X}$  and  $\mathbf{P}^*$  a reduced SPM of  $\mathbf{P}$ . Respectively,  $\boldsymbol{\Sigma}^*$  is the reduced covariance matrix of  $\boldsymbol{\Sigma}$  and  $\boldsymbol{\mu}^*$ .

**Property 2** If the SPM  $\mathbf{P}$  can be written as a combination of diagonal matrices  $\mathbf{P}_1, \mathbf{P}_2, \dots, \mathbf{P}_K$ ,  $\mathbf{P} = \text{Diag}(\mathbf{P}_k)$ ,  $k = 1, \dots, K$ . Then the outcome of  $\mathbf{P}$ ,  $\mathbf{x}$  can hereby be decomposed in to outcomes of the diagonal matrices,  $\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_K)$ . There will also have random variables  $\mathbf{X}_i \sim \text{PMD}(\mathbf{P}_i)$ , respectively. The pmf can computed as the product of the corresponding marginal pmfs. That is

$$\Pr(\mathbf{X} = \mathbf{x}) = \Pr(\mathbf{X}_1 = \mathbf{x}_1, \dots, \mathbf{X}_K = \mathbf{x}_K) = \prod_{k=1}^K \Pr(\mathbf{X}_k = \mathbf{x}_k).$$

□

To show this, we assume  $\mathbf{P}$  is  $n \times m$  and  $\mathbf{P}_k$  is  $n_k \times m_k$ .  $\sum_{k=1}^K n_k = n$  and  $\sum_{k=1}^K m_k = m$ . Suppose there are  $n$  voters to vote  $m$  candidates, certain groups of voters only vote for certain groups of candidates and there are no overlaps. Heuristically, we can separate candidates and voters into independent groups, in each group  $k$ ,  $k = 1, \dots, K$ , voters voting for corresponding candidates described by probability matrix  $\mathbf{P}_k$ . Strict proof can be done by decomposition of characteristic function of  $\mathbf{P}$  into characteristic functions of  $\mathbf{P}_k$ 's.

### 3 Computation of The Probability Mass Function

We introduce three methods for computing the pmf, which are the method based on multi-dimensional discrete Fourier transform (DFT-CF), the normal approximation (NA) method, and simulation based method (SIM). DFT-CF is an exact method while the other two are approximate methods.

#### 3.1 The DFT-CF Method

Although PMD is of great importance, its pmf is obscure and no straightforward form has been found so far. However, its characteristic function(cf) can be wrote down in an accessible form. The cf is just a Fourier transformation of pmf, thus we can obtain pmf via Fourier transformed cf. Notice the pmf is discrete, so the discrete Fourier transformation will be used. In this section we provide an exact method to compute the pmf of the PMD using multidimensional discrete Fourier transformation. To speed up the computing, we implement Fast Fourier Transformation algorithm(FFT).

Suppose a random variable  $\mathbf{X} = (X_1, \dots, X_m)' \sim \text{PMD}(\mathbf{P})$ , where  $\mathbf{P} = (p_{ij}), i = 1, \dots, n, j = 1, \dots, m$ . Consider  $\mathbf{X}^* = (X_1, \dots, X_{m-1})'$  will be equivalent since the last element of  $\mathbf{X}$  is redundant. The cf of  $\mathbf{X}^*$  is

$$\phi(t_1, \dots, t_{m-1}) = \mathbb{E} \left[ \exp \left( \mathbf{i} \sum_{j=1}^{m-1} t_j X_j \right) \right] = \mathbb{E} \left[ \exp \left( \mathbf{i} \sum_{i=1}^n \sum_{j=1}^{m-1} t_j I_{ij} \right) \right]. \quad (1)$$

Here  $\mathbf{i} = \sqrt{-1}$ . We notice

$$\mathbb{E} \left[ \exp \left( \mathbf{i} \sum_{j=1}^{m-1} t_j X_j \right) \right] = \sum_{x_1=0}^n \cdots \sum_{x_{m-1}=0}^n p(x_1, \dots, x_{m-1}) \exp \left( \mathbf{i} \sum_{j=1}^{m-1} t_j x_j \right). \quad (2)$$

By definition,

$$\exp \left( \mathbf{i} \sum_{j=1}^{m-1} t_j X_j \right) = \exp \left( \mathbf{i} \sum_{i=1}^n \sum_{j=1}^{m-1} t_j I_{ij} \right) \quad (3)$$

The expectation of RHS of (3) can be expressed as

$$\begin{aligned} \mathbb{E} \left[ \exp \left( \mathbf{i} \sum_{i=1}^n \sum_{j=1}^{m-1} t_j I_{ij} \right) \right] &= \mathbb{E} \left[ \exp \left( \mathbf{i} \sum_{j=1}^{m-1} t_j I_{1j} + \cdots + \mathbf{i} \sum_{j=1}^{m-1} t_j I_{nj} \right) \right] \\ &= \prod_{i=1}^n \mathbb{E} \left[ \exp \left( \mathbf{i} \sum_{j=1}^{m-1} t_j I_{ij} \right) \right] = \prod_{i=1}^n \left[ \left( 1 - \sum_{j=1}^{m-1} p_{ij} \right) + \sum_{j=1}^{m-1} p_{ij} \exp(\mathbf{i} t_j) \right]. \end{aligned} \quad (4)$$

We know (4) is equivalent to (2). Therefore we get

$$\sum_{x_1=0}^n \cdots \sum_{x_{m-1}=0}^n p(x_1, \dots, x_{m-1}) \exp \left( \mathbf{i} \sum_{j=1}^{m-1} t_j x_j \right) = \prod_{i=1}^n \left[ \left( 1 - \sum_{j=1}^{m-1} p_{ij} \right) + \sum_{j=1}^{m-1} p_{ij} \exp(\mathbf{i} t_j) \right].$$

Let  $t_j = \omega l_j$ ,  $l_j = 0, \dots, n$ ,  $\omega = 2\pi/(n+1)$ . Then the equation becomes

$$\frac{1}{(n+1)^{m-1}} \sum_{x_1=0}^n \cdots \sum_{x_{m-1}=0}^n p(x_1, \dots, x_{m-1}) \exp \left( \mathbf{i} \omega \sum_{j=1}^{m-1} l_j x_j \right) = \frac{1}{(n+1)^{m-1}} q(l_1, \dots, l_{m-1}), \quad (5)$$

where

$$q(l_1, \dots, l_{m-1}) = \prod_{i=1}^n \left[ \left( 1 - \sum_{j=1}^{m-1} p_{ij} \right) + \sum_{j=1}^{m-1} p_{ij} \exp(\mathbf{i} \omega l_j) \right].$$

Note that  $q(l_1, \dots, l_{m-1})$  can be computed directly. The left side of equation (5) is the inverse multi-dimensional discrete Fourier transform of the sequence  $p(x_1, \dots, x_{m-1})$ ,  $x_i = 0, \dots, n$ . Therefore we can apply Multi Dimensional Discrete Fourier Transformation(MD-DFT) on both sides to recover the sequence, the pmf can be obtained as

$$p(x_1, \dots, x_{m-1}) = \frac{1}{(n+1)^{m-1}} \sum_{l_1=0}^n \cdots \sum_{l_{m-1}=0}^n q(l_1, \dots, l_{m-1}) \exp \left( -\mathbf{i} \omega \sum_{j=1}^{m-1} l_j x_j \right) \quad (6)$$

Let  $\ell = (l_1, \dots, l_{m-1})$ , then we will have  $(n+1)^{m-1}$  different  $\ell$  as  $l_i$  values from 0 to  $n$ . For example, if we have  $n = 4$ ,  $m = 4$ , then  $\ell$  can be  $(0, 0, 0), (0, 0, 1), \dots, (4, 4, 4)$ , 125 different vectors in total. Now we have  $q(l_1, \dots, l_{m-1}) = q(\ell)$ . We design to use these vectors to generate respective  $p(x_1, \dots, x_{m-1})$ . For each  $\ell$ , we get a  $p(x_1, \dots, x_{m-1})$ .

To speed up the computing, we apply the Fast Fourier transformation (FFT) algorithm from GSL Scientific Library. The FFT algorithm is C language based, we implement it with R and make a new algorithm named DFT-CF algorithm. Our DFT-CF algorithm can calculate all values of distribution function as long as we input our  $P_{n \times m}$  matrix.

### 3.2 Normal-Approximation Based Method

Since covariance matrix of any  $\mathbf{X} \sim \text{PMD}$  is singular, we use the reduced PMD  $\mathbf{X}^*$  to establish the normal approximation. From ?) we know that central limit theory can be applied to PMDs, the following theorem studies the error bound, or converge rate of normal approximation of PMD.

**Theorem 1** *For a Poisson-Multinomial random variable  $\mathbf{X} = (X_1, \dots, X_m)'$  that has a reduced mean vector  $\boldsymbol{\mu}^* = (p_1, \dots, p_{m-1})'$  and non-singular reduced covariance matrix  $\boldsymbol{\Sigma}^*$ . For each possible outcome  $\mathbf{x}_r$ , and its neighbourhood interval  $\mathcal{N}_{\mathbf{x}_r} = [\mathbf{x}_r - 0.5, \mathbf{x}_r + 0.5]$ . There exists a non-singular matrix  $\mathbf{C}$  such that  $\boldsymbol{\Sigma}^* = \mathbf{C}\mathbf{C}'$  and the error bound of Central Limit Theory approximation is*

$$|P(\mathbf{X} \in \mathcal{N}_{\mathbf{x}_r}) - P(\mathbf{Z} \in \mathcal{N}_{\mathbf{x}_r - \boldsymbol{\mu}^*})| \leq b(m-1)^{\frac{1}{4}} \sum_{i=1}^n \mathbb{E}|C^{-1}\mathbf{I}_i|^3$$

where  $\mathbf{Z}$  is normal with mean 0 and covariance matrix  $\boldsymbol{\Sigma}^*$ .

By central limit theorem (CLT),

$$\left(\frac{\mathbf{X}^*}{n} - \mathbf{p}\right) \dot{\sim} \mathcal{N}\left(\mathbf{0}, \frac{1}{n}\boldsymbol{\Sigma}^*\right).$$

where  $\mathbf{p} = \frac{1}{n} \sum_{i=1}^n \mathbf{p}_i$ . To show **Theorem1**, notice

$$\mathbf{X}^* = (X_1, \dots, X_{m-1})' = \sum_{i=1}^n \mathbf{I}_i^* = \sum_{i=1}^n (I_{i1}, \dots, I_{i,m-1})'$$

Intuitionally,  $\mathbf{I}_i^* - \mathbb{E}(\mathbf{I}_i^*)$  has mean 0. Hereby  $\mathbf{X}^* - \mathbb{E}\mathbf{X}^* = \sum_{i=1}^n (\mathbf{I}_i^* - \mathbb{E}\mathbf{I}_i^*)$  also has mean 0. The covariance of  $\mathbf{X}^*$  is  $\boldsymbol{\Sigma}^* = \mathbf{C}\mathbf{C}'$ .

By extended Berry-Esseen theory (V. Yu. Bentkus, A Lyapunov-type bound in  $\mathbb{R}^d$ ), there existing a constant  $c$  such that

$$|\Pr(\mathbf{X} \in \mathcal{N}_{\mathbf{x}_i}) - \Pr(\mathbf{Z} \in \mathcal{N}_{\mathbf{x}_i - \boldsymbol{\mu}_0})| \leq c(m-1)^{\frac{1}{4}} \sum_{i=1}^n \mathbb{E}|C^{-1}\mathbf{I}_i|^3$$

Where  $\mathbf{Z}$  is CLT multivariate normal distribution of with 0 mean and covariance  $\boldsymbol{\Sigma}^*$ .

### 3.3 Simulation-Based Method

One can simulate  $\mathbf{I}_i$  from multinomial distribution and then compute  $\mathbf{X} = \sum_{i=1}^n \mathbf{I}_i$ . Repeat this many times to generate enough samples for  $\mathbf{X}$ . Then use the sample distribution to approximate the true distribution. To be specific, let  $\mathbf{x}$  be a given realization of  $\mathbf{X}$ , the probability  $\Pr(\mathbf{X} = \mathbf{x})$  can be calculated by following steps,



Step 1 for each  $i = 1, \dots, n$  randomly generate  $\mathbf{I}_i$  once with given  $\mathbf{p}_i$  using multinomial distribution, calculate  $\mathbf{X} = \sum_{i=1}^n \mathbf{I}_i$ .

Step 2 repeat step 1 B times to get  $\mathbf{X}^{(1)}, \dots, \mathbf{X}^{(B)}$ .

Step 3 count the frequency of  $\mathbf{x}$  showing up in  $\mathbf{X}^{(1)}, \dots, \mathbf{X}^{(B)}$  as the desired probability.

Consequently, we can calculate each point one by one and eventually the pmf can be calculated by this way. Alternatively, another scheme to do this is by

Step 1 generate  $\mathbf{X}^{(1)}, \dots, \mathbf{X}^{(B)}$ .

Step 2 calculate the frequency of each probability mass point showing up in  $\mathbf{X}^{(1)}, \dots, \mathbf{X}^{(B)}$  as the pmf.

We design a C++ based algorithm to perform this simulation process. Our algorithm uses the first scheme to calculate probabilities of  $\mathbf{x}$ s as user input and use the second scheme to calculate pmf if  $\mathbf{x}$  is no specified.

The following theorem studies the expected error bound of the simulation method by given  $\mathbf{P}$  and  $\mathbf{B}$ .

**Theorem 2** *Given repeating time B and  $n \times m$  matrix  $\mathbf{P}$ , there will be totally  $N = \binom{n+m-1}{m-1}$  different results, denote them as  $\mathbf{x}_r, r = 1, \dots, N$ . Let the probability for a specific result  $\mathbf{x}_r$  be  $p_{\mathbf{x}_r}, i = 1, \dots, N$ . The estimate of  $p_{\mathbf{x}_r}$  using simulation method is  $\hat{p}_{\mathbf{x}_r}$ . We have the following expected error given by Central Limit Theory,*

$$\mathbb{E}|p_{\mathbf{x}_r} - \hat{p}_{\mathbf{x}_r}| = \sqrt{\frac{2p_{\mathbf{x}_r}(1 - p_{\mathbf{x}_r})}{\pi B}}$$

*The expected total absolute error,*

$$\sum_{r=1}^N \mathbb{E}|p_{\mathbf{x}_r} - \hat{p}_{\mathbf{x}_r}| \leq \sqrt{\frac{2(N-1)}{\pi B}}$$

For any result  $\mathbf{x}_r, i = 1, \dots, N$ . Consider a Bernoulli  $\mathbf{r.v}$   $Y_r$  with probability  $p_{\mathbf{x}_r}$  to be 1, and  $1 - p_{\mathbf{x}_r}$  to be 0. By repeating the trail for B times, we get **i.i.d** random variables  $Y_r^{(1)}, \dots, Y_r^{(B)}$ . By WLLN

$$\bar{Y}_r - p_{\mathbf{x}_r} \xrightarrow{d} N(0, \sigma^2)$$

where  $\sigma^2 = \frac{p_{\mathbf{x}_r}(1-p_{\mathbf{x}_r})}{B}$ . Then the expectation of absolute error for a single  $\mathbf{x}_r$

$$\mathbb{E}|\bar{Y} - p_{\mathbf{x}_r}| = \frac{\sqrt{2}}{\sqrt{\pi}}\sigma = \sqrt{\frac{2}{\pi B}p_{\mathbf{x}_r}(1 - p_{\mathbf{x}_r})}$$

Let  $c = \sqrt{\frac{2}{\pi B}}$  for simplicity, then

$$\begin{aligned}
\sum_{r=1}^N \mathbb{E}|\bar{Y}_r - p_{\mathbf{x}_r}| &= c \sum_{r=1}^N \sqrt{p_{\mathbf{x}_r}(1 - p_{\mathbf{x}_r})} = cN \sum_{r=1}^N \frac{\sqrt{p_{\mathbf{x}_r}(1 - p_{\mathbf{x}_r})}}{N} \\
&\leq cN \sqrt{\sum_r p_{\mathbf{x}_r}(1 - p_{\mathbf{x}_r})/N} = c\sqrt{N} \sqrt{1 - \sum p_{\mathbf{x}_r}^2} \\
&\leq c\sqrt{N} \sqrt{1 - 1/N} = c\sqrt{N - 1} \\
&= \sqrt{\frac{2(N - 1)}{\pi B}}
\end{aligned}$$

The equality will be achieved if  $p_{\mathbf{x}_1} = \dots = p_{\mathbf{x}_N}$ . For a fixed B, because of the sparsity of PMD, the expected total absolute error goes up that as the dimension of  $\mathbf{P}$  goes up. Therefore for higher dimensional  $\mathbf{P}$  we need higher B to maintain accuracy regardless of time efficiency.

## 4 Method Comparisons

In this section, we compare the three methods in terms of numerical accuracy and the time efficiency, we also give out recommendations of under what condition which method will be preferred. We employ multiple criterions to show the accuracy, the major one is maximum absolute error(MAE). Suppose we have an PMD matrix with  $n \times m$  dimension. Denote the number of possible outcome as  $N$  and the set of them as  $\chi = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ . MAE is defined as following,

$$\text{MAE} = \max_{\chi} |p(\mathbf{x}) - p_{\text{true}}(\mathbf{x})|$$

which is the max value of the differences between probability densities calculated by our methods and true ones. We also use total absolute error (TAE) where

$$\text{TAE} = \sum_{\mathbf{x} \in \chi} |p(\mathbf{x}) - p_{\text{true}}(\mathbf{x})|$$

We also compare the differences between the max, 0.95 and 0.90 quantiles of the true pmf and the pmf computed by simulation method. The machine we use is Tinkercliff, which is a Linux server belongs to Virginia Tech ARC scientific computing center. The processor is AMD EPYC 7702(128 cores, 2GHz) and 256GB RAM. We first test the accuracy of our exact method DFT-CF with the true pmf small dimensional PMD calculated by enumeration. Due to the limitation of enumeration method, we compare the accuracy of DFT-CF using Binomial distribution and Poisson Binomial Distribution. After we justify the accuracy of DFT-CF, we consider the pmf calculated by this method as true pmf to study the accuracy of NA and

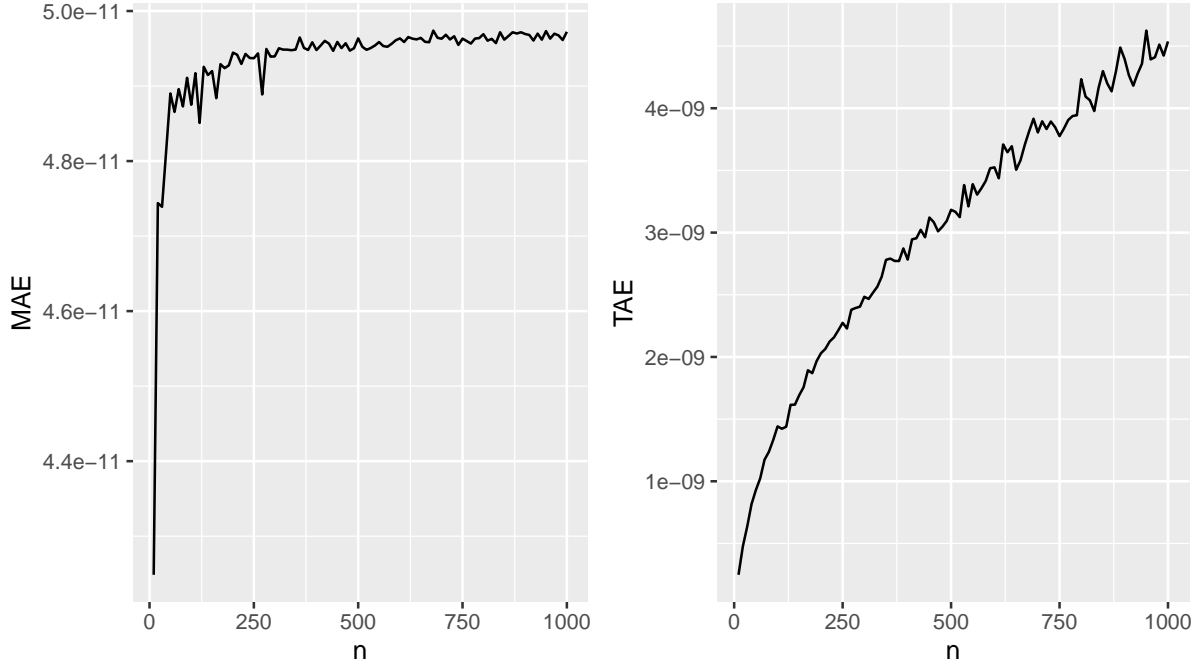


Figure 1: As  $n$  increasing to 1000, the MAE is around  $10^{-11}$  and the TAE is around  $10^{-9}$

SIM. The detailed procedure is to generate thousands of  $\mathbf{P}$ s randomly for a given  $(n, m)$  and calculate the mean accuracy criterions except when using results computed by enumeration method.

#### 4.1 Accuracy of DFT-CF

As we know already, DFT-CF is an analytic proved method. It will be impossible to get the true pmf for large  $n$  and  $m$ , but some special cases can be considered. First of all we look at the Binomial distribution which is actually a partical case of PMD when all rows of  $\mathbf{P}$  are same and  $m = 2$ . For large  $n$  and  $m = 2$  and no constrain on  $\mathbf{P}$ , which is a more general case called Binomial Passion Distribution, a method based on one dimensional discrete Fourier transformation is justified by ?), thus we can use it to test the accuracy of DFT-CF. For  $m \geq 3$ , let  $n$  be relatively small so that enumeration can be applied here. In this case, we can work out their probability densities by hand, and compare them with the results computed by DFT-CF method.

To prove DFT-CF is able to calculate pmf exactly under the situation when  $(n, m)$  is small enough to be calculated by enumeration, we randomly generate multiple  $\mathbf{P}$ s from dimension  $2 \times 2$  to  $6 \times 4$  with smallest probability 0.007 and largest probability 0.97. The detailed result is not listed since DFT-CF computes the pmfs exactly same as those computed by enumeration.

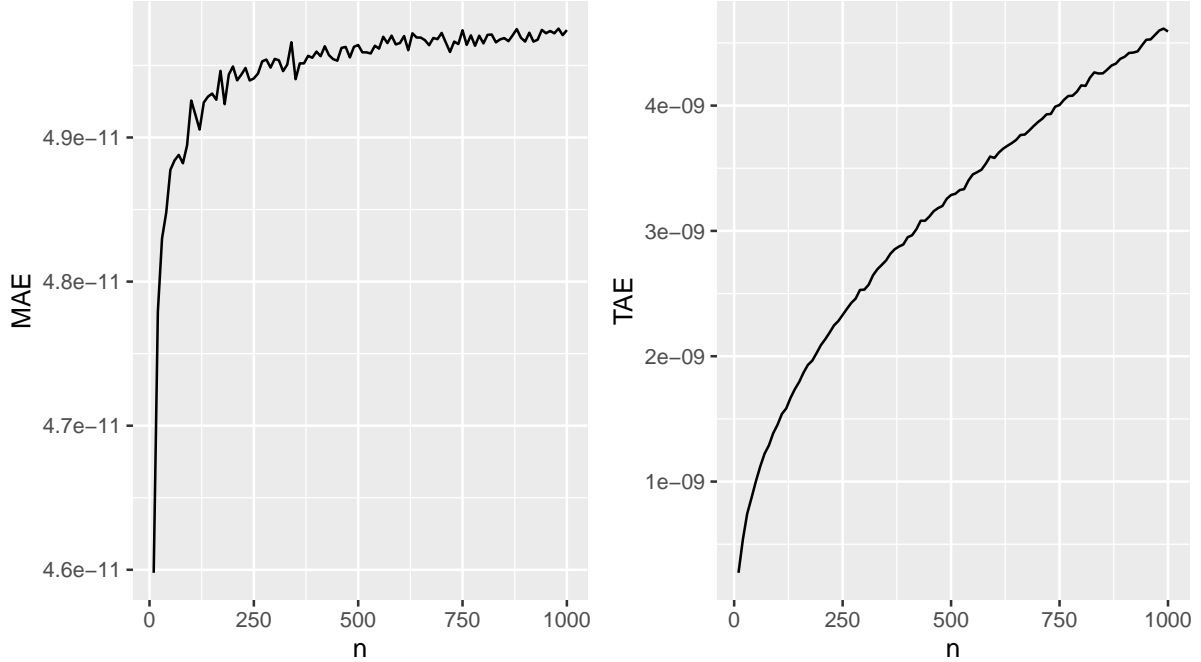


Figure 2: Accuracy result of DFT-CF under Poisson Binomial senario

From Figure 1, one can tell DFT-CF is reliable. The error is well controlled and smaller than  $10^{-9}$  generally. The possible source of error is the machine error from inside of the C++ interface of Multi-Dimensional Fast Fourier Transformation algorithm. As dimension increases, the Fast Fourier transformation algorithm become imprecise. This is also the why compare with the result from ?) our algorithm is less accurate. Figure 2 shows us similar results when the Binomial is replaced by Poisson Binomial, the error is small and due to FFT algorithm itself. Overall, it suffices to conclude that our DFT-CF algorithm is accurate and it can be used to compute true pmf.

## 4.2 Accuracy of Normal Approximation and Simulation Method

Among the three computing algorithms we proposed, one of them is able to calculate the exact probability while the other two are approximation methods. Therefore, it is necessary to explore accuracy of the approximation methods under different circumstances. The accuracy metric for NA is MAE. For SIM we only consider the maximum probability and probability mass points of 0.95, 0.90 quantiles due to computation capability of device. Probability mass points computed by DFT-CF are considered as true probabilities, denote it as  $p_{\text{true}}$ . MAE and TAE for normal approximation are given by

$$\text{MAE} = \max_{\mathbf{x} \in \chi} |p_{\text{NA}}(\mathbf{x}) - p_{\text{true}}(\mathbf{x})|,$$

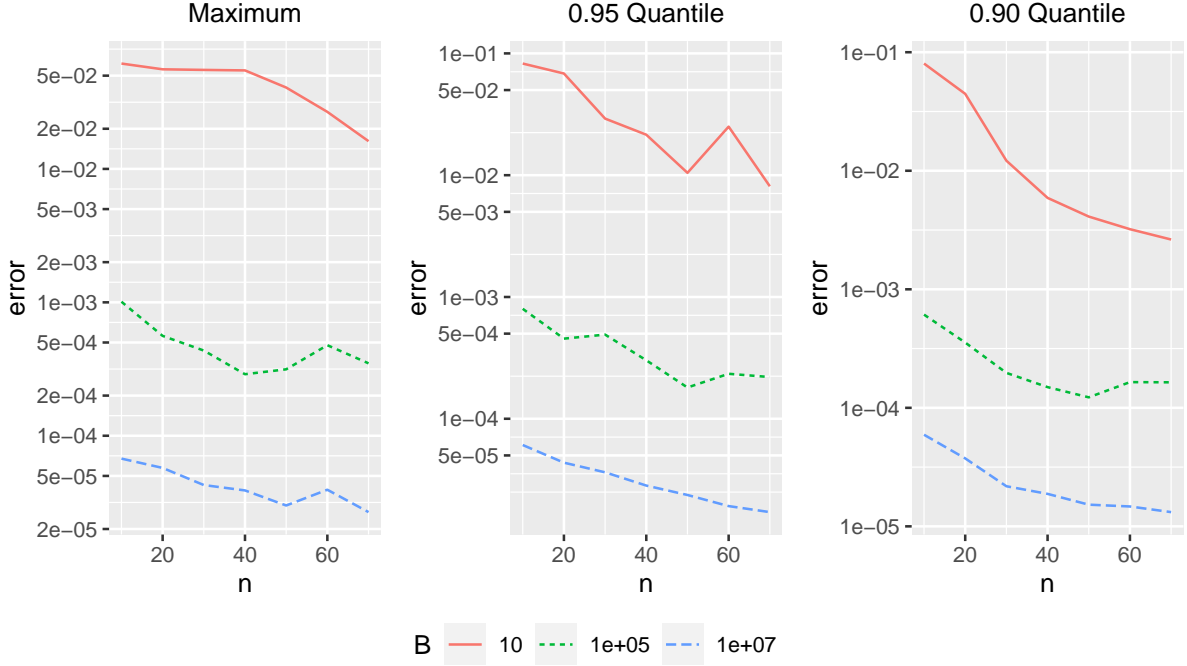


Figure 3: Accuracy of SIM method based on quantile of PMD when  $m = 3$

For simulation method the error metric is given via the difference between the selected probability mass points(maximum, 0.95 and 0.90 quantiles) calculated by SIM and DFT-CF.

For fixed  $n$  from 1 to 75 and  $m = 3$ , randomly generate 5000  $\mathbf{P}$ s to exclude the noise effect followed by calculating error metrics with respect to each methods. The results are shown via Figure 3 and Figure 4. In Figure 3, we mainly study the relationship between accuracy and simulation times  $B$ . The top lines in three plots are of  $B = 10$ , the middle lines are of  $B = 10^5$  and the rest are of  $B = 10^7$ . Observe that the gap between lines reduces as  $n$  increases possibly due to sparsity. The accuracy performance when  $B$  is evidently better than when  $B$  is small. We can tell from the plots that  $B = 10^7$  might be a good choice for user since it provide an accuracy error as small as  $10^{-5}$ .

Specifically, for NA we introduce a baseline called **Original** to express the sparsity of PMD as  $n$  and  $m$  increase. The **Original** is just to estimate all probability mass points by 0s. We can see through the plot the baseline is decreasing which is resulting from sparsity. From Figure 3, for given  $m$ , as  $n$  increase the solid line which represent NA is always under the dashed baseline. The gap between the two lines grows wider as  $n$  gets larger for fixed  $m$  indicates that NA is more accurate when  $n$  is large regardless of sparsity.

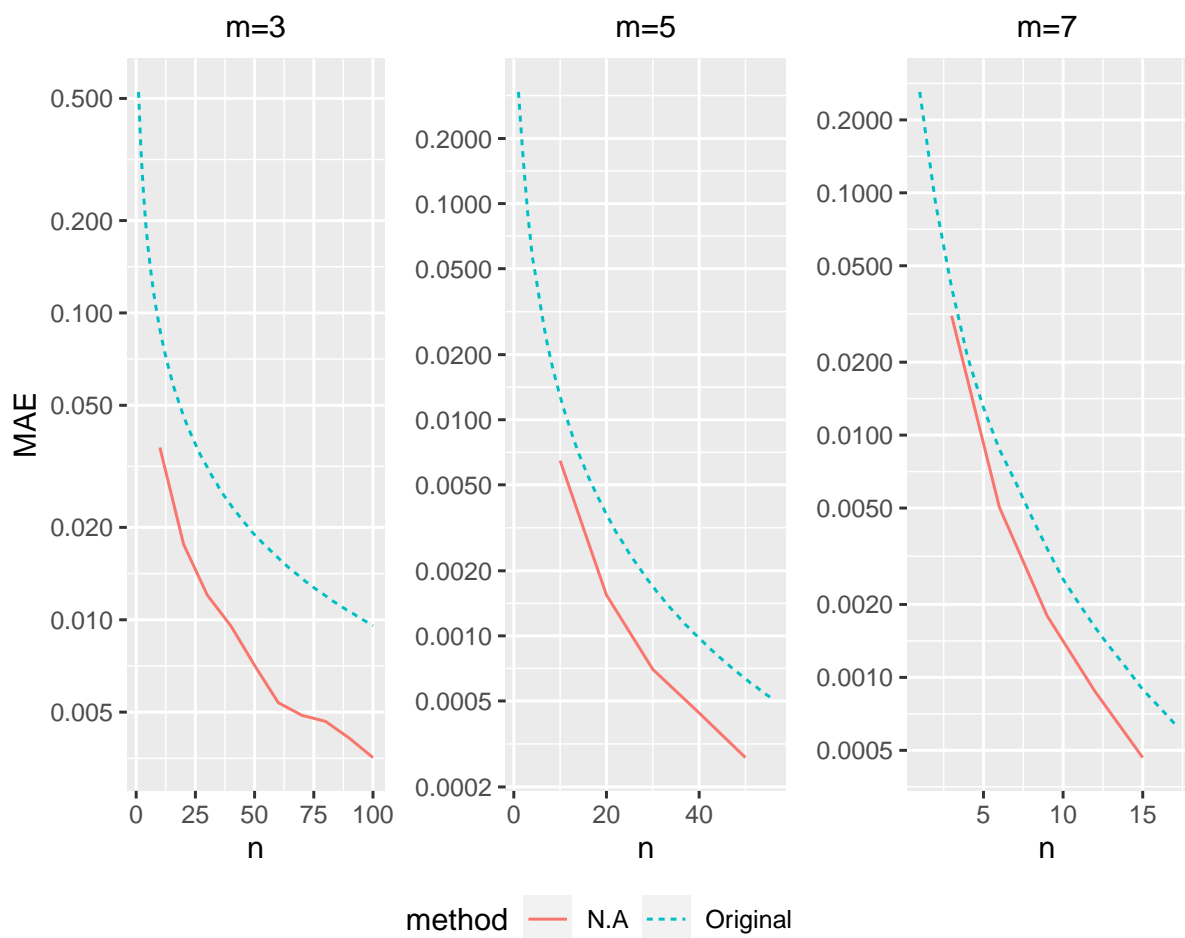


Figure 4: MAE of NA compared with baseline(Original)

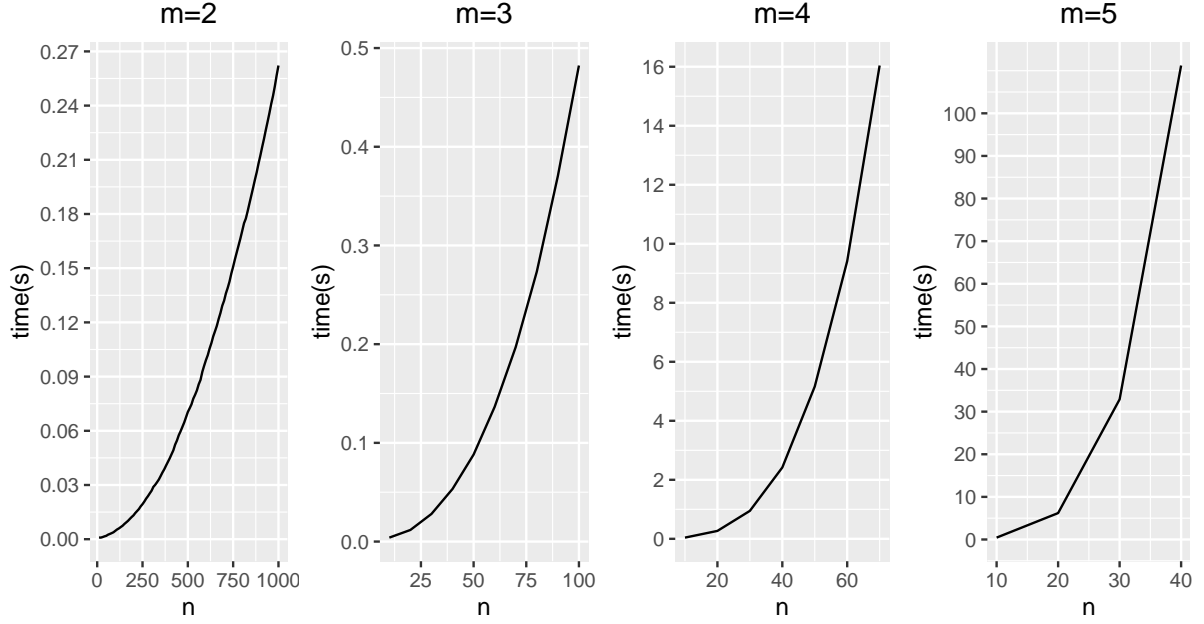


Figure 5: Time efficiency study of DFT-CF

### 4.3 Time Efficiency of DFT-CF method

As a method to compute the whole pmf, we need to consider DFT-CF's time efficiency especially when the dimension is large. The method actually calculates  $(n+1)^{m-1}$  probability mass points that includes the possible outcomes of PMD which is  $\binom{n+m-1}{m-1}$ . Because it takes every  $x_j, j = 1, \dots, m-1$  from 0 to  $n$ . The number  $(n+1)^{m-1}$  increases drastically as  $m$  gets larger. When  $m$  is small, the result is showed by Figure 5. When  $m$  is moderate ( $8 \leq m \leq 20$ ) or larger, using DFT-CF can be problematic since  $(n+1)^{m-1}$  can be enormous such that the device either takes too long to compute nor unable to compute due to exhausted usage of RAM.

We use the same device as for accuracy study to test the time efficiency of DFT-CF. Figure 5 plots the average computing time of 1000 randomly generated  $\mathbf{P}$ s using DFT-CF. The scale of time is second(s) and the computing is done on the same device as for the accuracy study. The following plot (Figure 5) shows that the when  $m$  is small (less or equal to 4), it is good to use DFT-CF to calculate the entire pmf. As we can see through the plot the computing time for  $n = 60, m = 4$  is 16 seconds which is affordable. Even when  $m = 5$  and  $n = 40$  the time is around 100 seconds which is still acceptable. However, when  $m = 6$  or higher, increasing  $n$  can make the computing time huge.

## 4.4 Recommendations

According to the results of accuracy and efficiency study. We hereby give out our recommendations of which method to choose under what circumstances. For small  $m$  and moderate  $n$ , the DFT-CF can be used. Since the computing time is affordable and the method is an exact method that can compute the entire pmf automatically. However when  $m$  is moderate or large, this method can be very time consuming and will need a huge RAM. Thus for moderate  $m$  ( $5 \leq m \leq 20$ ) and small  $n$ , we encourage users to use simulation method with  $B$  around  $10^6$  to calculate partial pmf that is of users' interests. By this way the accuracy will be maintained at a satisfied level and computing time will be affordable. For large  $n$  including when  $m$  is also large, the NA is recommended because the accuracy is backed up by central limit theory. If only calculate partial pmf, computing time will not be a concern.

## 5 Applications

### 5.1 Calculation of Voting Probability

In voting scenarios, people always pay attention to the election result. The most thing we usually care about is who will win the election and how many chances each candidate has to win the election. A Poisson Multinomial distribution can fit the situation perfectly under some assumptions.

Suppose a election has  $n$  voters and  $m$  candidates, there will be  $N = \binom{n+m-1}{m-1}$  possible outcomes denoted as  $\mathbf{x}_r, r = 1, \dots, N$ , respectively. Each  $\mathbf{x}_r$  is a 3 dimension vector that has three elements denoting the number of votes each candidate gets. Assume we know the  $\mathbf{P}$  based on prior polls, for example, we can always estimate the approval rate of each candidate in a certain constituency from the monthly polls or exit polls. Then we are able to compute the notional result.

To demonstrate that, suppose we have ten electoral voters and three candidates with  $\mathbf{P}$  matrix that has means of column one, two and three to be 0.3631, 0.3405 and 0.2964, the first five rows are as following,

$$\begin{pmatrix} 0.071 & 0.589 & 0.340 \\ 0.365 & 0.195 & 0.440 \\ 0.445 & 0.505 & 0.050 \\ 0.353 & 0.382 & 0.265 \\ 0.620 & 0.111 & 0.269 \end{pmatrix}$$

To compute the probability of each candidate winning the election, just need to introduce



constraints. For instance, under the constraint  $\chi_1 = \{\text{the first element is the largest one}\} = \{x_1 > x_2, x_1 > x_3; x_1, x_2, x_3 \in \{0, \dots, 10\}\}$ , we are able to compute the winning rate of the first candidate is

$$\Pr(\text{the 1st candidate wins}) = \sum_{\mathbf{x} \in \chi_1} p(\mathbf{x}) = 0.3429$$

Similarly, the probabilities for the second candidate and the third candidate to win are 0.2745 and 0.2001. Additionally, the most possible result is  $\mathbf{x} = (4, 3, 3)$ , which has probability 0.08546.

## 5.2 Statistical Inference for Aggregated Data

First introduce the “Logistic-like” model to fit a given aggregated dataset with selected features and a categorical response variable that has  $m$  categories. By dividing the rows of our data into  $H$  groups  $G_1, \dots, G_S$  via some given criterion, the group size of each group is  $s_i, i = 1, \dots, S$ . Each  $s_i$  are positive integer but not necessarily equal to each other. Let the quantity for each category of group  $G_i$  be  $\mathbf{x}^{(i)} = (x_1^{(i)}, \dots, x_m^{(i)})'$ ,  $m$  is the category number. Denote  $\mathbf{H}^{(i)}$  to be the covariate matrix of  $G_i$ , then  $\mathbf{H}^{(i)} = (\mathbf{1}, \mathbf{h}_1^{(i)}, \dots, \mathbf{h}_{s_i}^{(i)})'$  is a  $s_i \times v$  matrix with first column being  $\mathbf{1}$ , where  $v$  equals to the number of covariates plus one. Let  $\mathbf{P}^{(i)} = (p_{jk}^{(i)})$  be the SPM for group  $G_i, i = 1, \dots, S, j = 1, \dots, s_i$  and  $k = 1, \dots, m$ . Let the probability of getting  $\mathbf{x}^{(i)}$  for group  $G_i$  be  $p(\mathbf{x}^{(i)})$ . The total log-likelihood for all groups can be computed as

$$\mathcal{L} = \sum_{i=1}^S \mathcal{L}_i = \sum_{i=1}^S \log p(\mathbf{x}^{(i)})$$

Where  $p(\mathbf{x}^{(i)})$  can be computed via Poisson Multinomial distribution with SPM  $\mathbf{P}^{(i)}$ . Set category  $m$  as baseline and use softmax function to form the  $\mathbf{P}^{(i)}$  for each group through parameter  $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \dots, \boldsymbol{\beta}_{m-1})$  as

$$p_{jk}^{(i)} = \frac{\exp(\mathbf{h}_j^{(i)} \boldsymbol{\beta}_k)}{1 + \sum_{k=1}^{m-1} \exp(\mathbf{h}_j^{(i)} \boldsymbol{\beta}_k)} \quad k \neq m \quad \text{and} \quad p_{i,m}^{(i)} = \frac{1}{1 + \sum_{k=1}^{m-1} \exp(\mathbf{h}_j^{(i)} \boldsymbol{\beta}_k)}$$

Then we are able to estimate our parameters on the direction of minimizing total log-likelihood and finally get our estimate  $\hat{\mathbf{P}}^{(i)}$ .

In the rest of this part, we apply the model to the dataset “ai4i”. The dataset “ai4i” is a synthetic machine failure dataset that reflects real predictive maintenance data encountered in industry. The data consists of 10000 products(rows) and 14 features(columns) including Product type, Air temperature, Process temperature, Rotational speed and others.

For demonstration purpose, here we only use the first 1000 rows and Product type as response variable, Air temperature and Process temperature as covariates. The feature Product

type is categorical and has three levels, “M”, “L”, “H”, we denote them as category 1, 2, 3 for simplicity. The number of products that fall in category 1, 2, and 3 are 285, 601, and 114. The other two features are continuous.

We randomly divide the dataset into 100 groups  $G_1, \dots, G_{100}$ , the smallest group has size of three rows and the largest one has 18 rows. Note that readers can use other criterion to divide the dataset by their own. Notice the covariate number is three including intercept and the response has three categories, thus the dimension of our parameter matrix  $\beta$  will be  $3 \times 2$  if we set category 3 as baseline.

The estimates of the parameters are

$$\hat{\beta} = \begin{pmatrix} 1.07484986 & 2.2820922 \\ 1.62342045 & 1.9108976 \\ -0.06277732 & -0.8455964 \end{pmatrix}$$

The corresponding  $\hat{\mathbf{P}}^{(1)}$  and  $\hat{\mathbf{P}}^{(5)}$  for group 1 and group 5 are

$$\hat{\mathbf{P}}^{(1)} = \begin{pmatrix} 0.11914 & 0.63310 & 0.24776 \\ 0.12326 & 0.57268 & 0.30406 \\ 0.14809 & 0.47560 & 0.37631 \\ 0.14504 & 0.56971 & 0.28525 \\ 0.12451 & 0.54095 & 0.33454 \\ 0.04170 & 0.55944 & 0.39886 \\ 0.03559 & 0.53568 & 0.42873 \\ 0.04890 & 0.54668 & 0.40442 \end{pmatrix}, \quad \hat{\mathbf{P}}^{(5)} = \begin{pmatrix} 0.15604 & 0.70766 & 0.13630 \\ 0.14469 & 0.73976 & 0.11555 \\ 0.11246 & 0.67372 & 0.21382 \\ 0.12036 & 0.64885 & 0.23079 \\ 0.11263 & 0.61304 & 0.27433 \\ 0.11563 & 0.55029 & 0.33408 \\ 0.11774 & 0.61672 & 0.26554 \\ 0.15071 & 0.52510 & 0.32419 \\ 0.13878 & 0.56645 & 0.29477 \\ 0.08194 & 0.54561 & 0.37245 \\ 0.06307 & 0.58191 & 0.35502 \end{pmatrix}$$

For group 1 and 5,  $\mathbf{x}^{(1)} = (1, 5, 2)'$  and  $\mathbf{x}^{(5)} = (2, 6, 3)'$ , so  $p(\mathbf{x}^{(1)}) = 0.070$ ,  $p(\mathbf{x}^{(5)}) = 0.067$ .

### 5.3 Uncertainty Quantification in Classification

In machine learning classification problem with multiple labels, for each unit in the test set, the probability of the unit belongs to each class is computed. Usually, the predicted class is assigned as the highest probability. In the classifiers (i.e., soft classification), the unit class is randomly assigned according to the predicted probabilities, leading to randomness in the confusion matrix. The PMD can be used to characterize the distribution of the counts in the confusion matrix.

In this section, we consider an Electroluminescence (EL) image classification example to illustrate the usage of PMD in machine learning classification problems. In the photovoltaic (PV) reliability study, the EL image is an important data type that reveal information about the PV health status. Because disconnected parts do not irradiate, the darker areas in EL

Table 1: Partitioning of solar cells into functional and defective, with an additional self-assessment on the rater’s confidence after visual inspection. Non-confident decisions obtain a weight lower than 100% for the evaluation of the classifier performance.

Condition	Confident?	Label $p$	Weight $w$	Class
functional	Yes	functional	0%	A
	No	defective	33%	B
defective	No	defective	67%	C
	Yes	defective	100%	D

images indicate defective cells. The EL imaging provide visual inspection of solar panels and is a non-destructive technology for failure analysis of PV modules (?).

The work of (?), (?), and (?) provide a public dataset of solar cells extracted from high resolution EI images of PV modules (<https://github.com/zae-bayern/elpv-dataset>). In total there are 2624 images. All images are preprocessed with respect to size and are eliminated distortion induced by the camera lens used to capture the EL images. Each image is manually labeled with its degree of defectiveness. The degree of defectiveness is determined by two questions. The first is how do evaluators think the status of the solar cells, functional or defective; the second is how they are confident about their assessments. In total there are four labels as shown in Table 1 and we marked them as Class A-D.

We split our data to training data (80%) and test data (20%), then train a CNN model on the training set. In CNN model, we use Relu activation function and set the kernel size  $3 \times 3$  and stride  $1 \times 1$ . For each image in the testing set, the model provides a probability that the prediction belongs to each class. Table 2 provides a subset of the CNN model output. For true class for the first sample in Table 2 is A. If we predict the class of the first sample using the highest probability, then the prediction is A and there’s no randomness. If we allow the model to make predictions based on the probability vector as shown in the first row then there are randomness in the confusion matrix. We can consider the confusion matrix to follow a PMD distribution then we can quantify the uncertainty in confusion matrix using PMD.

Figure 6 shows the marginal probability of each cell. For example, in the first row first column panel cell in Figure 6, we know the possible counts that the model correctly predict class A as well as the corresponding probability. In this way, we have a uncertainty quantification in the confusion matrix.

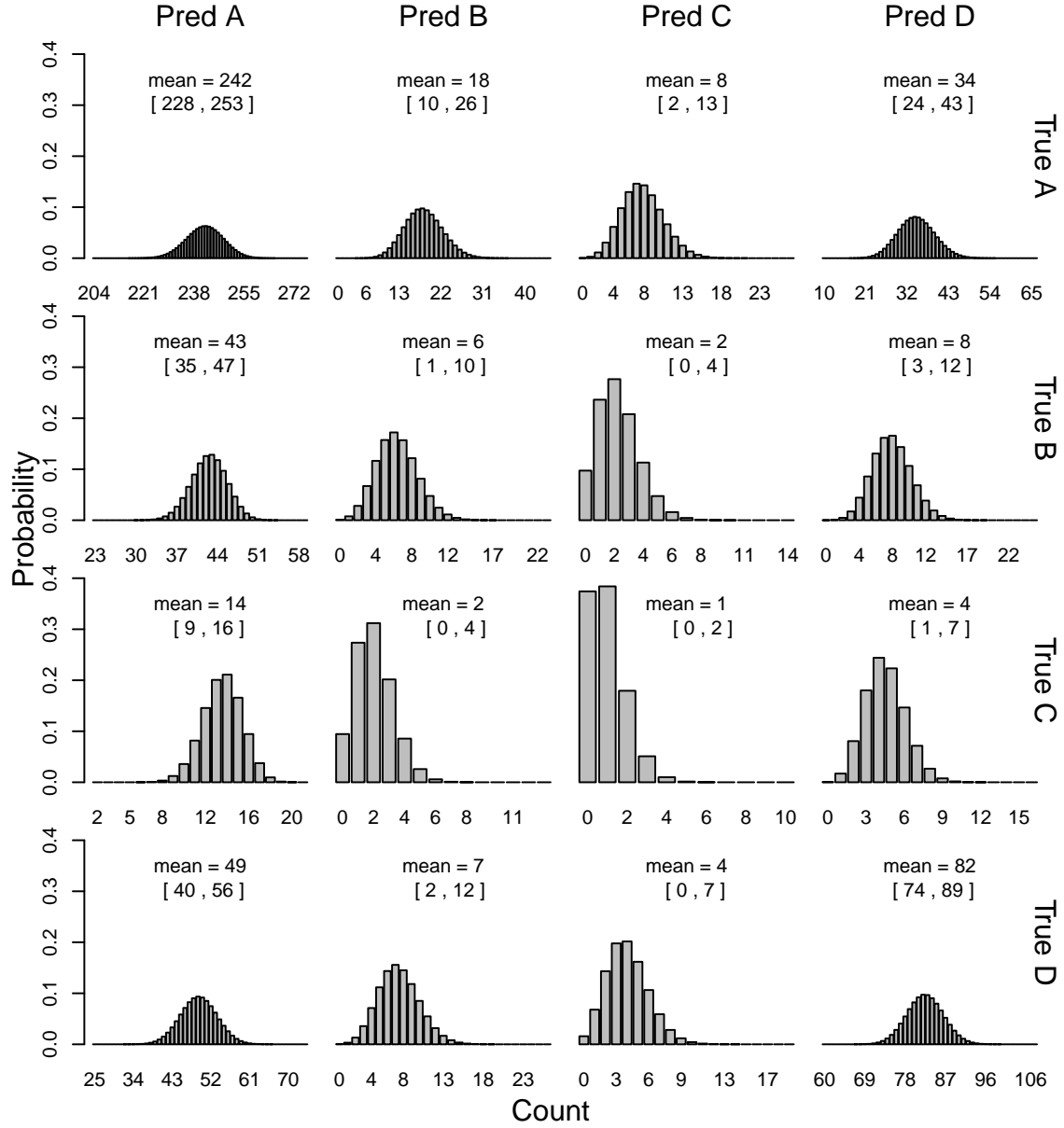


Figure 6: The barplot shows the confusion matrix prediction. For each panel cell, the plot shows the corresponding prediction counts fall in this cell as well as their probability. We also present the mean and 95% naive prediction interval.

Table 2: An example of the probability vectors of a subset in testing set from the trained CNN model.

	A	B	C	D
1	0.9230	0.0366	0.0107	0.0297
2	0.0736	0.0802	0.0513	0.7950
3	0.0000	0.0016	0.0006	0.9978
4	0.9170	0.0537	0.0062	0.0231
5	0.9579	0.0239	0.0070	0.0112
6	0.8991	0.0347	0.0132	0.0530

## 6 Illustrations of the R Package

Our R package PoissonMultinomial specifically computes probability density function for PMD with corresponding  $\mathbf{P}$  using the methods described by this paper. The fast Fourier transformation algorithm are implemented in C.

### 6.1 Examples

There are three major functions in the package. `dpmd` is a function to compute pmf, `ppmd` is for cdf and one can use `rpmd` to generate PMD random samples. User has to specify  $\mathbf{P}$  so that a PMD can be determined. With unspecified  $\mathbf{x}$ , `dpmd(P)` will automatically calculate the whole pmf and the output will be a multi-dimensional array. If user inputs  $\mathbf{x}$ , the output of `dpmd(P, x)` will be partial pmf chosen by  $\mathbf{x}$ . User can also select all methods we list in this paper to compute the pmf or cdf. Notice only DFT-CF can automatically compute the whole pmf and it is the most efficient way for this job as well. The function `ppmd` uses same method as `dpmd` to compute the probability  $\Pr(\mathbf{X} < \mathbf{x})$ . Examples of using `dpmd` is following,

```
P <- matrix(c(0.1, 0.1, 0.8, 0.1, 0.3, 0.6, 0.5, 0.2, 0.3), nrow=3, byrow=T)
x <- c(0, 1, 2)
dpmd(P)          # calculate whole pmf
dpmd(P, x)       # calculate the probability mass point of x
```

The output of `dpmd` is a matrix as following, and we can see the output of `dpmd(P, x)` is 0.186 which is the [1, 2] element of the output.

$$\begin{pmatrix} 0.144 & 0.186 & 0.069 & 0.006 \\ 0.282 & 0.190 & 0.023 & 0.000 \\ 0.073 & 0.022 & 0.000 & 0.000 \\ 0.005 & 0.000 & 0.000 & 0.000 \end{pmatrix}$$

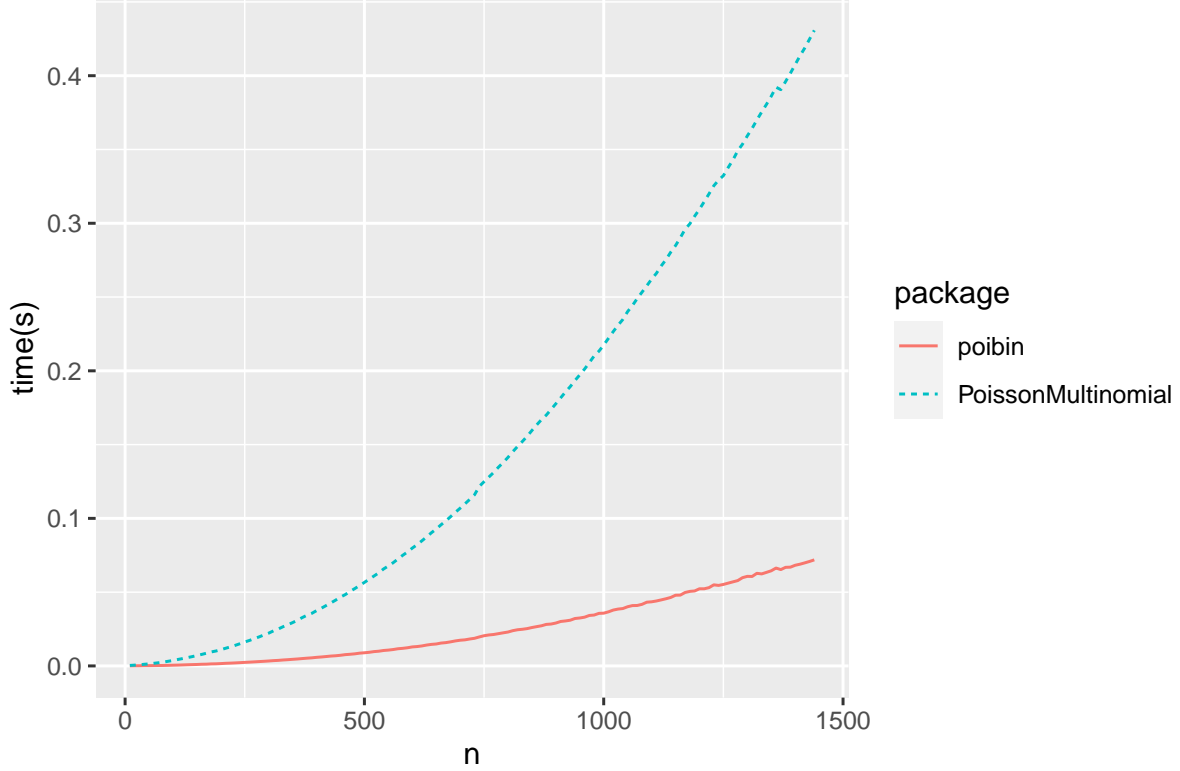


Figure 7: Benchmark comparison of PoissonMultinomial and poibin

## 6.2 Benchmark of R Packages for Poisson Binomial Distribution

We compare the time efficiency of our package and an existing package that computes pmf of Poisson binomial distribution, which is a special case of Poisson multinomial distribution. The result Figure 7 shows when  $m = 2$  which is the special case of Poisson multinomial, `poibin` is much faster than our package `PoissonMultinomial`. This is due to the fact that 2 dimensional fast Fourier transformation is faster than multi dimensional Fourier transformation in general.

## 7 Conclusions and Areas for Future Research

We develop algorithms that can be useful for computing the pmf of the PMD distribution, which is challenging to compute but useful in many application scenarios. We design three methods that are implemented in our algorithms. Among the methods, DFT-CF is an exact method, SIM is a simulation method and NA is a method using approximation theories. The accuracy and efficiency of the methods are guaranteed under given circumstances. We also recommend users to use DFT-CF when  $m$  is small, use SIM when  $m$  is moderate and  $n$  is small, use NA when  $n$  is large. However, there are still some fields remain to be explored.

The computing speed of DFT-CF can be improved using more efficient Fourier transformation algorithms or we could find a way to calculate only the  $\binom{n+m-1}{m-1}$  possible outcomes rather than  $(n+1)^{m-1}$  probability mass points which contains a large amount of points have values equal to 0. SIM method is still time consuming although it can compute some cases that DFT-CF are unable to, once we develop more efficient exact algorithms, SIM could be replaced. Untill now, we are still unable to compute the pmf of PMD when  $m$  is large, this area remains to be disclosed.